2007-07-02

Obstacle Avoidance and Path Traversal Using Interactive Machine Learning

Jonathan M. Turner
Brigham Young University - Provo

OBSTACLE AVOIDANCE AND PATH TRAVERSAL USING INTERACTIVE MACHINE LEARNING

by

Jonathan Milton Turner

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jonathan Milton Turner

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

_____	_____
Date	Dan R. Olsen, Jr., Chair
_____	_____
Date	Michael A. Goodrich
_____	_____
Date	Dennis Ng

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jonathan Milton Turner in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

_____	_____
Date	Dan R. Olsen, Jr. Chair, Graduate Committee

Accepted for the Department

_____	_____
Date	Parris Egbert Graduate Coordinator

Accepted for the College

_____	_____
Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

OBSTACLE AVOIDANCE AND PATH TRAVERSAL

USING INTERACTIVE MACHINE LEARNING

Jonathan Milton Turner

Department of Computer Science

Master of Science

Recently there has been a growing interest in using robots in activities that are dangerous or cost prohibitive for humans to do. Such activities include military uses and space exploration. While robotic hardware is often capable of being used in these types of situations, the ability of human operators to control robots in an effective manner is often limited. This deficiency is often related to the control interface of the robot and the level of autonomy that control system affords the human operator. This thesis describes a robot control system, called the safe/unsafe system, which gives a human operator the ability to quickly define how the system can cause the robot to automatically perform obstacle avoidance. This definition system uses interactive machine learning to ensure that the obstacle avoidance is both easy for a human operator to use and can perform well in different environments. Initial, real world tests show that system is effective at automatic obstacle avoidance.

TABLE OF CONTENTS

Chapter 1 – Introduction

There has been significant focus in recent years on using robots for tasks typically done by humans. There are advantages to using robots instead of humans in many situations. One of the primary sources of difficulty in using robots is enabling them to act in a manner that reduces and augments human effort rather than increasing it. This thesis introduces a system designed to reduce human effort while navigating robots. Part of this system in operation, shown in figure 1, demonstrates obstacle recognition.

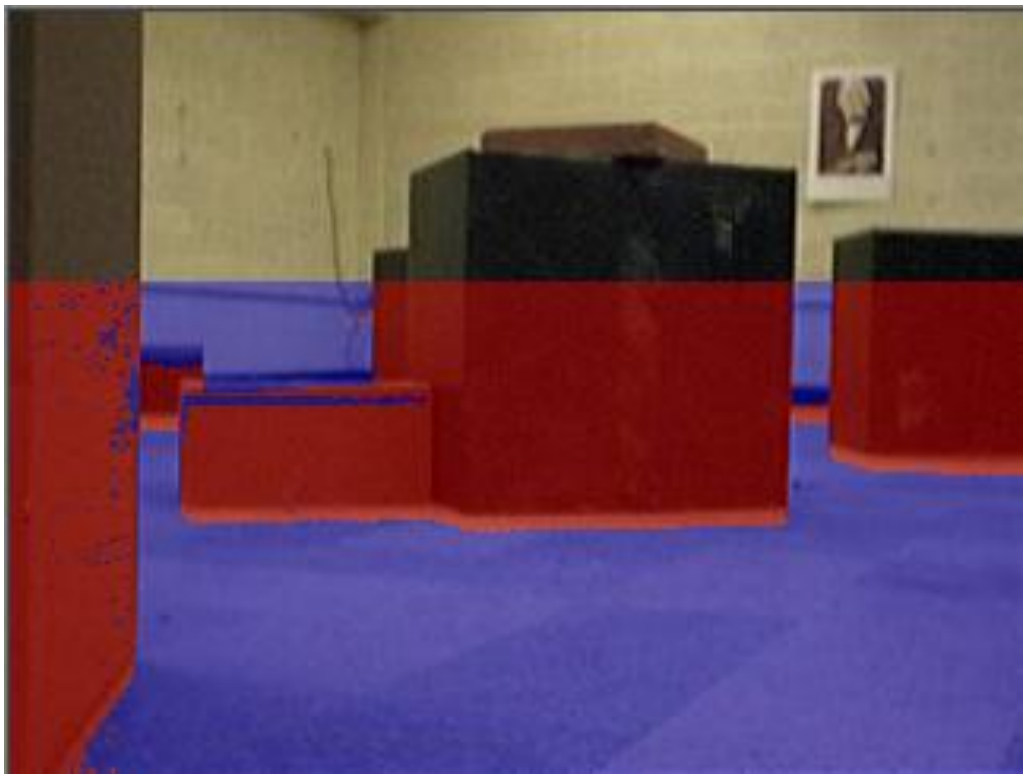

Figure 1 – System has marked obstacles with red and safe areas with blue.

There are various situations where using robots instead of humans is desirable because of the level of risk involved. There are many things that humans have to do even

though they involve a great deal of risk to the human doing them. One such example is military reconnaissance. This activity is dangerous because usually the areas of most strategic importance are the areas where it is most likely that hostile enemy forces will be located. If humans perform this activity they run a serious risk of being wounded or killed. If a robot goes into enemy territory the worst that will happen is that it will be destroyed and have to be replaced. There are various other military applications for robots that help reduce the risk to human life.

Another example of a situation where having a robot perform a given task would reduce the risk to human life is when the environment where the task is being performed is itself dangerous. Such an environment might be inside a nuclear power plant. Maintenance has to be performed regularly to ensure proper operation of the facility. Exposure to radioactive environments is always risky to humans, so any potential exposure that can be avoided by using robots would reduce the risk to humans.

In addition to making tasks less dangerous for humans, there are also situations where it is simply easier or more cost effective to use robots. An excellent example of that is space exploration. Take for example the recent Mars rovers, one of which is pictured in figure 2. Sending two robotic rovers to Mars cost approximately $800 million [BRAI04], whereas sending a manned spacecraft to Mars is estimated to cost anywhere from tens of billions to hundreds of billions of dollars (some estimates are even as high as a trillion dollars) [OBER04]. This extra cost is directly attributable to the extra materials and effort necessary to keep humans alive on such a journey. Robots are much easier and less expensive to transport.

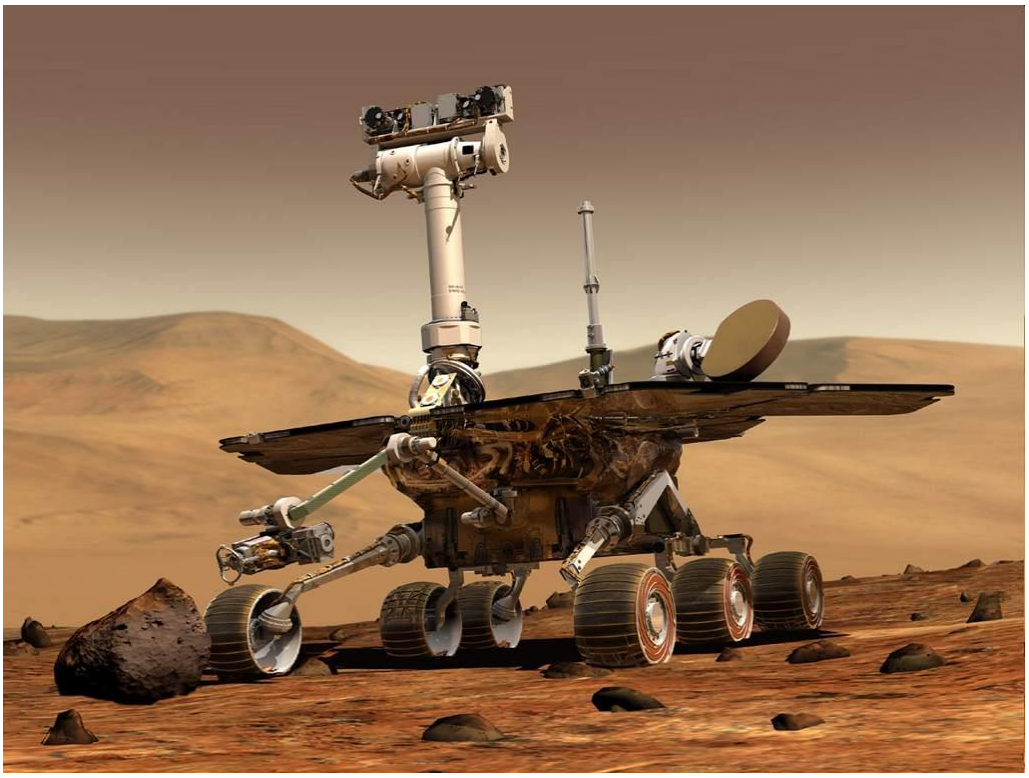

Figure 2 – One of the Mars rovers [STEN02].

Despite the many advantages of using robots there are disadvantages as well. One of the main obstacles in using robots is getting them to do something useful. In the past, operating a robot has been a very tedious and manual process. The majority of the responsibility for operating the robot, navigating it through its environment, and interpreting the situation around it has been placed primarily on a human who directly controls the robot. This is not only difficult for humans to do, it is also imprecise and time consuming. Fortunately, not all robot control systems are of this limited, teleoperated type. Some control systems used information available to the system (usually from sensors located on the robot) to make some simple, but useful decisions that help a user more easily navigate an environment. The system described in this thesis is one such system. Some others are discussed in chapter two.

1.1 The Safe/Unsafe System

This thesis details a system, referred to here as the safe/unsafe system, that addresses some of the difficulties related to traditional robot navigation. The heart of this system is a user-trainable, vision-based obstacle detection mechanism based on the Crayons system [FAIL03]. With this Crayons-based system, a user may specify which areas are safe for a robot to travel and which areas are unsafe. After the user has told the system which areas are safe and which areas are unsafe, the user is then able to give commands like "go over there" or "move to the other side of that obstacle" as opposed to lower level commands that might include "drive forward thirty-two inches" or "turn negative six degrees." Instead, the robot is responsible for these low-level commands. This is a type of system known as shared control. Shared control systems are discussed in more detail in chapter two. The robot performs basic obstacle avoidance, driving around the unsafe areas, to arrive at a location the user specifies. Real-time, interactive trainability allows the system to be used in a wide variety of situations. The purpose of the system is to make the robot responsible for many low level navigational commands thus allowing a human user to focus on higher level, strategic tasks. The reason for doing this is to make navigating a robot easier for the human user.

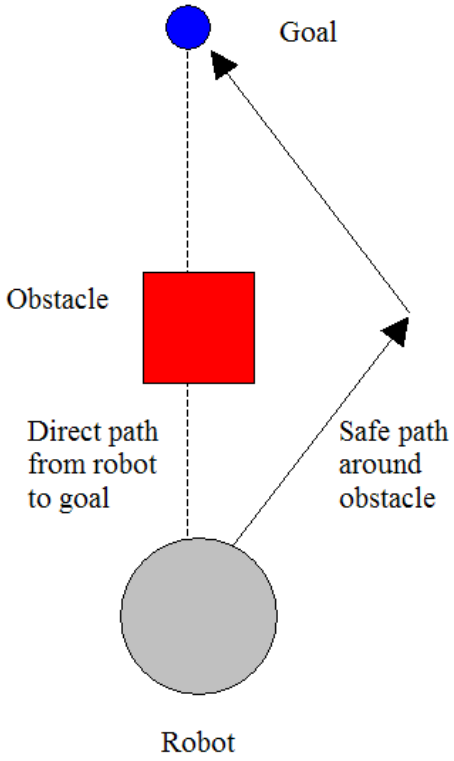

Figure 3 – Obstacle blocking path between robot and desired goal location.

A basic example can help demonstrate the difference between the safe/unsafe system and a traditional robot control system. Suppose there is a robot with a single obstacle directly in front of it, as shown in figure 3. Next assume that the human user controlling the robot wants to move the robot to the goal location as marked in the figure. The robot will obviously have to go around the obstacle in order to be able to get to the desired goal. This maneuvering can either be performed by the user or by the system. An explanation of the difference between these two options follows.

1.2 Approaches to Robot Control

In a traditional control system, the user would have to turn the robot so that driving it forward will no longer cause it to collide with the obstacle, as shown in figure

3. The user must then drive the robot forward far enough so that it can get past the obstacle. Since the obstacle will likely no longer be visible to the user when the robot is in the correct location, the user must guess at when the robot has gone far enough. An example of a situation where an obstacle is no longer visible to the robot but is still in the robot's path is shown in figure 4.

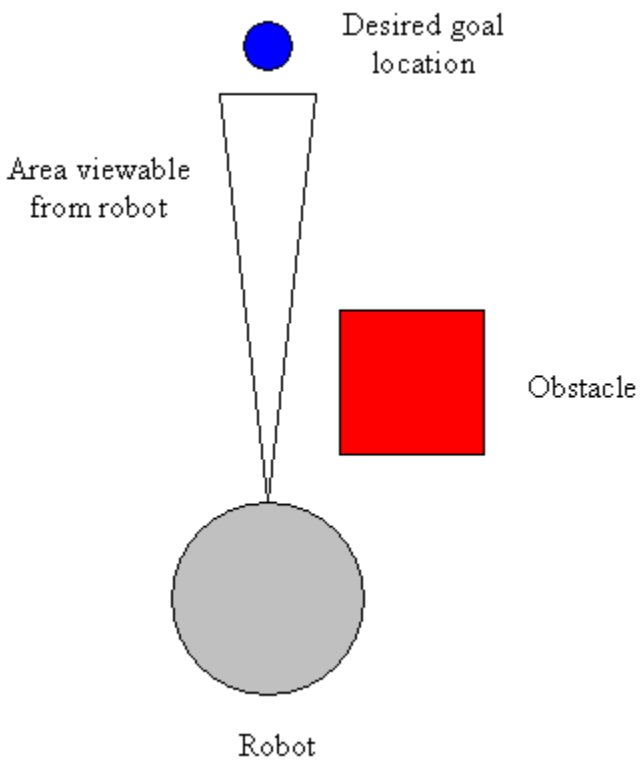

Figure 4 – Situation where an obstacle is not visible to robot, but potentially still in its path.

The user will have to perform additional movements, like turning the robot to face the obstacle, to determine if the robot has traveled far enough or not. If the robot has not traveled far enough or has traveled too far, even more movements might be necessary. Once the robot has moved far enough, the user must tell it to move to the goal. The user

must determine where the goal is in relation to the robot's new position, as opposed to from where the robot originally was when the goal was identified, and how to get there.

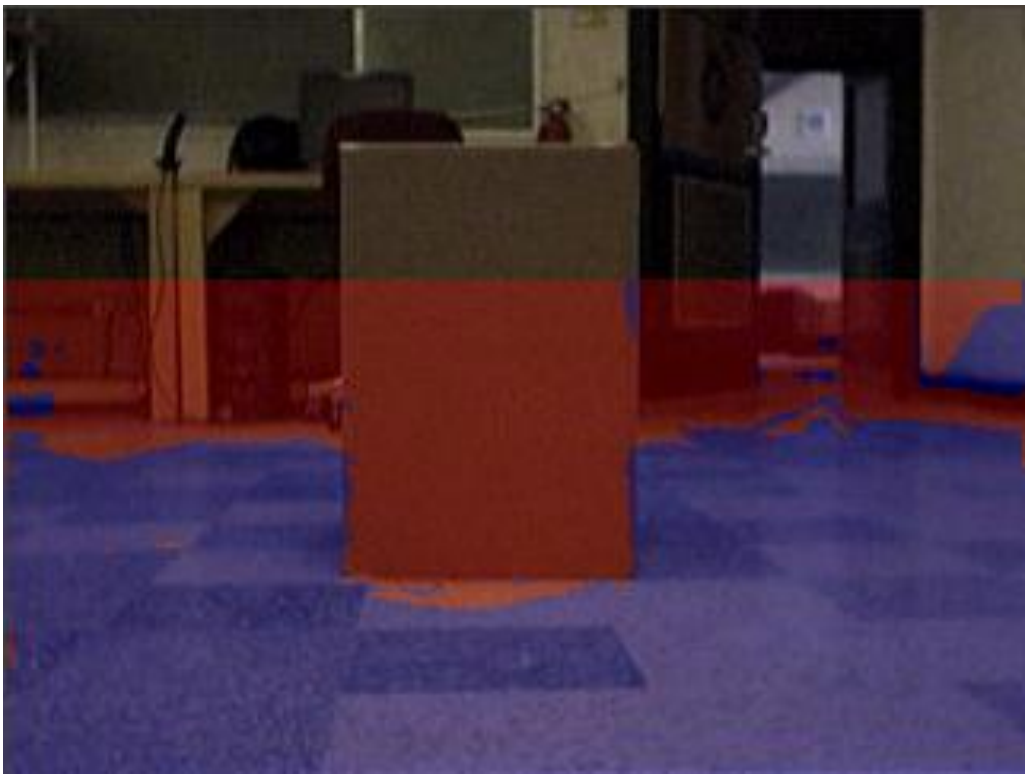

Figure 5 – An obstacle is marked as unsafe (red).

Next, take the same situation, but instead of a traditional robot control system the user has available the safe/unsafe system. An obstacle is still directly in front of the robot and is blocking a direct path to the desired goal position. At this point the user specifies what things are unsafe (the obstacle) and what things are safe (the area around the obstacle). The system now knows that the obstacle should be avoided. An example of such a classification is shown in figure 5. The red indicates areas that are unsafe, while the blue indicates areas that are safe. With this safe/unsafe classification, the user can now simply tell the robot to go straight to the desired goal location. As the robot is

driving forward, the system will determine that the obstacle (the unsafe area) is in the robot's path. The system will turn the robot to avoid the obstacle, drive the robot around the obstacle, and then correct the robot's path to take it to the originally specified goal location. The user does not have to give any additional movement instructions beyond the initial "go to the goal location" command. The classification that the user specified will be remembered by the system.

As long as the robot is in an area that has obstacles that look similar to the original obstacle and safe areas that look like the original safe areas the user need not add any additional safe/unsafe information in order to get the robot to avoid the new obstacles. If new types of obstacles or new types of safe areas are encountered, the user need only specify the new items as such and the safe/unsafe system will be able to avoid the new obstacles, as well as the old one, and know to travel in the new safe areas, as well as the old one. Figure 6 shows the original example situation with the path the system causes the robot to travel in order to avoid the obstacle.

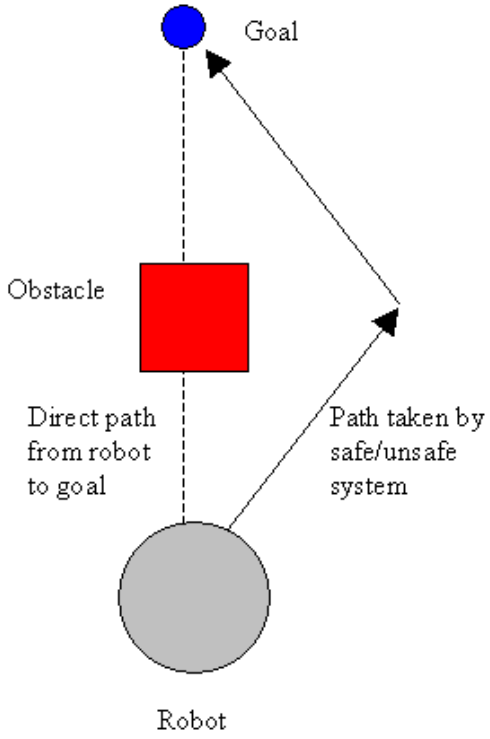

Figure 6 – Obstacle blocking path to desired goal and the path determined by safe/unsafe system around obstacle.

With traditional robot control systems the control is typically very direct. A user controls a single robot. The user has a joystick or similar control device. When the user is manipulating the control device then the robot moves. For example, when the user pushes forward on the joystick the robot moves forward. When the user pushes left on the joystick, the robot turns left. If the user doesn't manipulate the control device the robot doesn't move. The system takes no initiative. Also, with traditional robot control systems the user is in charge of maintaining a complete mental model of the situation. The system will report information back to the user about its environment, but will not try to interpret that information. That is up to the user. All navigation tasks are the responsibility of the user. The user must remember virtually everything about the

environment, including the location and nature of obstacles and the desired destination. The user must also navigate the environment using relatively low level, rather unnatural commands, the above mentioned "move forward" or "turn left" type commands. These factors combined make a traditional robot control experience difficult and inefficient for the user.

The safe/unsafe system addresses some of the drawbacks of traditional robot control systems by increasing the level of autonomy the robot has. Care must be taken when adding automation to a system, as it is possible to make a system less effective by adding automation to it [BAIN83]. However, if proper automation is added in an effective way then the system can be easier to operate and the user's efforts can be made more efficient. Following are some ways in which automation can improve the effectiveness of a system.

1.2.1 Robot Perspective vs. User Perspective

One area in which increasing autonomy in a robotic system can improve the effectiveness of the system is in regards to difference between user perspective and robot perspective. An example of this is a problem often encountered when using the robots on which the system in this thesis was implemented. This is the problem of corner clipping. This problem was almost invariably encountered with inexperienced users, but not infrequently encountered with even relatively experienced users. The difficulty stemmed from the fact that from where the camera is situated on the robot, to the user it appears as though the robot may have passed an obstacle when it fact it hadn't. The user, assuming the robot was past the obstacle would turn and continue driving. Since the robot was not

actually past the obstacle it would collide with the obstacle, causing the robot not to move or to move in a manner unexpected by the user. This obviously caused great frustration for users and increased the difficulty of navigating the robots. However, in the safe/unsafe system, the automation can take into account the size of the robot when determining when and how much it should turn and will consequently not make a turn that is too close to an obstacle.

A second area in which automation can take advantage of the difference between robot perspective and user perspective is that it is possible for a robotic system to handle more simultaneous inputs than a human user can. For example, a human user might have a difficult time interpreting data from several cameras, several sonar sensors and odometry sensors all at the same time. Depending on the hardware being used, a robotic system could handle all the above sources of information without being confused by trying to interpret all the data in unnecessary ways. The difference between robot and user perspective can also be taken advantage of in the case of high latency control situations. For example, with the Mars rovers it can take several minutes for a complete feedback cycle to occur. The robot can travel a great distance and encounter a wide variety of situations in that amount of time if it is traveling autonomously.

1.2.2 Driving With Distractions

Another area in which increasing autonomy in a robotic system can improve the effectiveness of the system is in situations where the user is attempting to control a robot while being distracted. An example of such a situation might be a military environment where a soldier is controlling a robot to gain reconnaissance information while trying to

complete other, non-related orders. The soldier could focus on the non-related orders while only occasionally focusing on controlling the robot. When the soldier is not focusing on the robot it can continue to perform useful work autonomously.

A related benefit of increased autonomy is the ability to control multiple robots. This is similar to the example given above where a user performs non-related tasks while a robot moves autonomously. However, in this case the "non-related tasks" involve controlling one or more other robots. So a user gives a command or set of commands to a robot. While the robot is performing those actions the user can give commands to another robot. If the first robot is still performing the commands given when the user finishes giving commands to the second robot then the user could start giving commands to a third robot. The number of robots that can be controlled depends on how autonomous the robots are and the limits of the user's ability to focus on multiple tasks [OLSE04].

One of the main reasons for controlling multiple robots with a single user is to increase user efficiency. Since fully autonomous robots have not yet proven effective in most situations, the most important part of a robotic control system is arguably the human user. The robot can perform useful work, but only when so instructed by the user. Thus it makes sense to utilize the most important part of the system, the user, in the most efficient manner possible. When using multiple robots, the user's time and talents can be more efficiently utilized by generating new commands for one robot while waiting for another robot to finish commands already given. This allows the user to perform useful work instead of simply watching idly while waiting for a single robot to perform a set of commands the user has given it.

How a user's abilities can be more effectively used with multiple robots is illustrated in a search example. Suppose that an area of fixed size must be searched. The search could be for a person, such as in a search and rescue operation, or for land mines in a military setting. Given one robot, a user could search the entire area in a certain amount of time. However, if the user were given five robots, the area could be searched in approximately one-fifth the time. This, of course, assumes that the robots are sufficiently autonomous and the user sufficiently skilled so that all five robots may be controlled at once.

Chapter 2 – Related Work

The safe/unsafe system has similarities to many previous works. This chapter discusses some of these various works which include shared control systems, sonar and laser-based navigations systems and vision-based systems.

2.1 Shared Control

The safe/unsafe system is similar to some shared control navigation approaches. Shared control systems, also called collaborative control systems, have been a topic of much research [FONG98] [BRUE03]. In general, shared control systems allow the system to control some part of the robot navigation, thus allowing a user to control it more easily and focus on higher-level goals [CRAN02]. Exactly what the robot does is determined partly by commands given by the user and partly by commands generated by the safe/unsafe system. Rosenblatt [ROSE95] describes a general shared control system. Various components generate desired behavior (or commands). These desired commands are then combined to determine the actual action that will be taken. The safe/unsafe system essentially has only two such components, the obstacle avoidance module and commands from the user. Röfer and Lankenau detail several different classes of shared control systems [ROFE99], one of which is similar to how the safe/unsafe system combines user commands and automated adjustments. Different types of shared control systems can be more effective than others, depending on the situation [BRUE04]. Some shared control systems allow for automatic preemption of user commands. In these situations the system is able to modify or nullify a user command if it is deemed

inappropriate. In the safe/unsafe system, user commands are modified so that the robot avoids obstacles, but the system will respond immediately to any new user commands. This is similar to the first shared control mode (the speed controller supervisory mode) discussed by Röfer and Lankenau. However, these shared control systems are based on sonar and only provide limited obstacle avoidance whereas the safe/unsafe system is based on vision and provides path traversal capabilities in addition to obstacle avoidance.

2.2 Sonar/Laser

Many systems have been implemented that use sonar for robot navigation [BORE89], [BORE91], [JENS99], [LEVI99], [MING00], [OHYA98]. These systems use sonar sensors to get the distance and heading to obstacles in the robot's environment. This information is used to create a representation of the obstacles in the robot's environment. The systems then use this obstacle information to modify or override user commands that would cause the robot to collide with an obstacle according to the information obtained from the sensors. Some systems [JENS99] use laser range finders in place of sonar sensors. The primary differences between sonar and laser sensors are that the laser sensors are more accurate, less susceptible to noise and generally more expensive.

Sonar and laser-based systems are effective at navigating many types of environments. The computational requirements necessary for such a system are relatively low. However, such systems have difficulty navigating environments that do not consist of positive space obstacles (environments where the obstacles are holes or flat delineators) or if the environment is such that the obstacles cannot easily be detected by

the sensors. Some examples of environments where obstacles would be hard to detect would be environments where the navigation surface is not level (if the robot is tipped forward or backward the sensors might not detect an obstacle) or environments containing obstacles too short to be detected by the sensors or not solid enough to reflect the sonar or laser (an example of this might be a bush or small tree).

Another potential disadvantage of sonar and laser-based systems is that there is a significant amount of information in an environment that they are not able to collect or use. Sonar and laser sensors are not able to detect or use any visual information (such as color). In many navigational tasks this type of information is not critical to successful navigation of the environment, but in some situations such information is critical.

2.3 Vision-Based

Other systems use visual information for navigation instead of information from sonar or laser sensors. For the purposes of this thesis these types of systems have been divided into two general categories: fixed and trainable. Fixed vision-based systems employ no learning aspect. The ways in which new images received are interpreted and environmental information extracted from them is always the same. The algorithm used is always the same regardless of past experience or use input. Trainable vision-based systems involve some form of learning. The system can learn how to process new images in different ways or how to interpret the information differently. In order for the system to learn how to deal with the images it receives it must be trained. The training process can greatly affect the performance of the system. The safe/unsafe system is a type of trainable vision-based navigation system. The differences between the

safe/unsafe system and other trainable vision-based systems are discussed below in the trainable vision-based section.

2.3.1 Fixed Vision-Based

Fixed vision-based systems have a set algorithm used to process new images and extract information from them. A common element of fixed vision-based systems is the use of feature areas. These are areas that have some visual characteristic that can be fairly easily recognized and can be used to distinguish the area from other areas in the image. These same characteristics can be checked for in other images. Such characteristics can be determined for an image at a known location. If the characteristics in a new image are the same as those computed for the area in the image whose location is known then it is assumed that the new area is the same location in the environment as specified in the original image. This way the same environmental location may be identified in two different images, whether these images were seen consecutively or one a great deal of time after the other. Since visual information is used, fixed vision-based systems are able to draw from a much richer set of information than sonar or laser-based systems, as discussed above. However, fixed vision-based systems suffer from two major drawbacks.

The first drawback to fixed vision-based systems has to do with general environment navigation. Systems such as those discussed by Davison [DAVI99] and Ohya et al [OHYA98] use the above mentioned feature areas to determine where the robot is in a previously created environment map. A map of the area in which the robot is going to navigate is created. In this map, several points of interest are identified.

Feature information about these points is generated and stored. As the robot is navigating it looks for areas in the images it receives that match the features for these points. When such a point is detected with a given degree of certainty the system uses that information to determine where the robot is in the environment. The environment map can then be used to generate a path to other areas in the environment.

The obvious disadvantage of such an approach is that a detailed map of the environment must be generated before the robot can effectively navigate said environment. In some situations, like an office space or warehouse, the robot will never leave its environment or encounter significantly different environmental characteristics. In such a situation relying on a detailed, previously generated map might be acceptable. However, in many situations the environment that the robot is to navigate changes very drastically during navigation. In other situations the robot's environment cannot be mapped adequately a priori, meaning a detailed environment map cannot be generated before navigation. Such examples might include space exploration (the purpose of sending a robot being to investigate a previously unknown environment) or a war zone (where buildings or geographic landmarks can be changed significantly or even destroyed while a robot is navigating).

Other fixed vision-based systems do not require such a previously generated environment map. These types of systems, such as those discussed by Davison et al [DAVI95], Lang et al [LANG99], Murray et al [MURR96], and Zhang [ZHAN92] are similar to the fixed vision-based systems discussed above. They find areas with features that are identifiable and discernable from other areas in an image. Corresponding areas are then found in subsequent images that the system receives. The location of

corresponding areas in two or more images may be compared to determine information about the robot's environment and how the robot is moving in that environment.

However, there is another drawback to vision-based systems, whether or not they require a pre-generated environment map. This second drawback is the fixed nature of such systems. If the algorithm for processing new images and extracting information from them is always the same, then there is a very real possibility that such a system would perform adequately in one environment and fail completely in another environment. Take, for example, the system described by Lang et al [LANG99] that uses information gleaned from the perpendicular lines in a suspension ceiling to aide the system during navigation. Not even taking into consideration environments that do not have a ceiling, it can be seen that such a system could perform very well in some environments and fail in others. This is because the algorithm would be expecting to find information about perpendicular lines on the ceiling regardless of what type of ceiling (or lack thereof) exists in the environment. It will always generate the same kinds of information, whether or not that type of information exists in or is relevant to the environment the robot is in. The other fixed vision-based systems mentioned here suffer from the same drawback.

2.3.2 Trainable Vision-Based

Trainable vision-based systems are similar to fixed vision-based systems in that they receive new images and extract information from these images about the environment that the robot is in. The difference between trainable vision-based systems and fixed vision-based systems is that the interpretation of a given set of input images

when using a trainable system can be different depending on how the system was trained. The interpretation of a given set of input images will always be the same for a fixed vision-based system. Trainable vision-based systems can learn to recognize trees [LECU05], soccer balls [MITR05], people and faces [PAPA00] or places the robot has already been to [CHAN99] as well as other items in a given environment.

A trainable vision-based system, as its name implies, requires some type of training. The system does not do any useful interpretation of images until it has been trained to some degree. Training consists of a human user giving some type of interpretation of an image or a part of an image. This interpretation can range from identifying an object in the image [MITR05], [PAPA00] to identifying which way a robot should turn [CHAN99], [LECU05]. The sample images combined with the interpretation of those images is often called training data.

The safe/unsafe system is a trainable vision-based with some significant differences from a normal trainable system. One of the greatest strengths of the safe/unsafe system, when compared to other trainable systems, is in regards to how it is trained. Most trainable systems require a distinct training phase that must happen independent from and prior to any kind of navigation phase. Once the training phase is complete and the navigation phase started no more training can occur unless the robot is taken out of the field and additional training data created for it. This process could take hours or days or weeks. It is something that most systems try very hard to avoid doing. However, with the safe/unsafe system the user is able to add more training data as the robot is navigating. The process only takes seconds. The way incoming images are interpreted changes as soon as the new training data is added. This is one of the greatest

advantages of the safe/unsafe system in general and is its single greatest advantage over other trainable vision-based systems.

2.4 Interface Support

Many robot control systems use the system's user interface in order to help the operator more effectively navigate a robot. The control interface can make robot operation more effective in the way it displays sensory information [MURP96]. Robots are often equipped with sensors that are not directly analogous to a human sense (such as sonar). The way in which the interface displays information from these types of sensors can greatly aid or greatly confuse the user. The interface may also interpret this information (to build a map for example) and combine it with other sensory information such as video [NIEL06] to give the user the benefit of using multiple types of sensors. Additionally, the interface may let the user change the level of autonomy the system exhibits based on the situation [YANC05]. The system's current model of the world may also be displayed by the interface in a way that is relatively easy for the user to understand and interpret [DRUR03]. The safe/unsafe system uses several of these ideas (displaying an interpretation of sensory input and the system's current model of the world) or variations on them (displaying sensor data and interpreted sensor data as though they were from two different types of sensors) to aid users in performing effective robot navigation.

Chapter 3 – Overview

This chapter provides details about the robotic platform used to implement and test the system. The overall interaction between different components of the system is also detailed. The details of the individual components are provided in subsequent chapters.

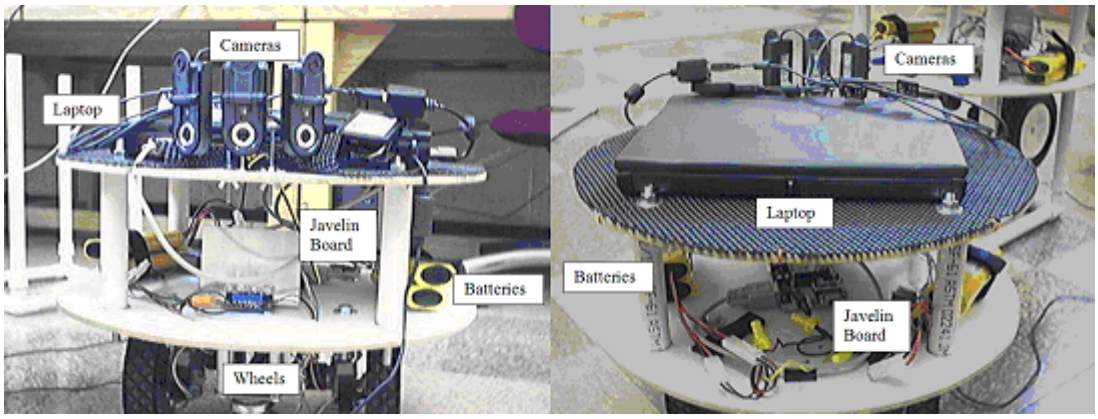

Figure 7 – Two views of robot, with different components marked.

3.1 System Components

The robots used to implement and test this system consist of a transport platform, a low-level control board, a laptop and a camera. The transport platform is custom made for these robots and is shown in figure 7. It is capable of holding and transporting all of the other robot's components. It includes two powered wheels plus two additional, smaller wheels for stability. The low-level control board is called a Javelin board. It contains a limited Java virtual machine. The Javelin board accepts commands over a serial connection to the laptop. It can also send back status information to the laptop.

The Javelin board has been programmed to accept commands of the type "drive forward *x* inches" or "turn left *y* degrees". These commands are then translated into actual wheel movements by the Javelin board. The Javelin board keeps track of the command it is currently executing, how much of the command has been executed and when the wheels should stop moving because the command is complete. The Javelin board is paired with an additional control board, which regulates the exact voltage applied to the wheel motors. Two pairs of 7.2V rechargeable batteries power the Javelin board and motors.

The laptop communicates with other computers. There is a controlling, desktop computer that is connected to the laptop through a wireless network (802.11g). The user can operate the robot with the desktop computer. All commands sent from the desktop to the laptop are translated into simpler commands that can be given to the Javelin board. Also all image processing and path generation functions (both of which will be described in more detail shortly) occur on the laptop. The camera is attached to and operated by the laptop. The basic robot control system is capable of using one or three cameras. However, the obstacle avoidance system currently only uses one of the cameras.

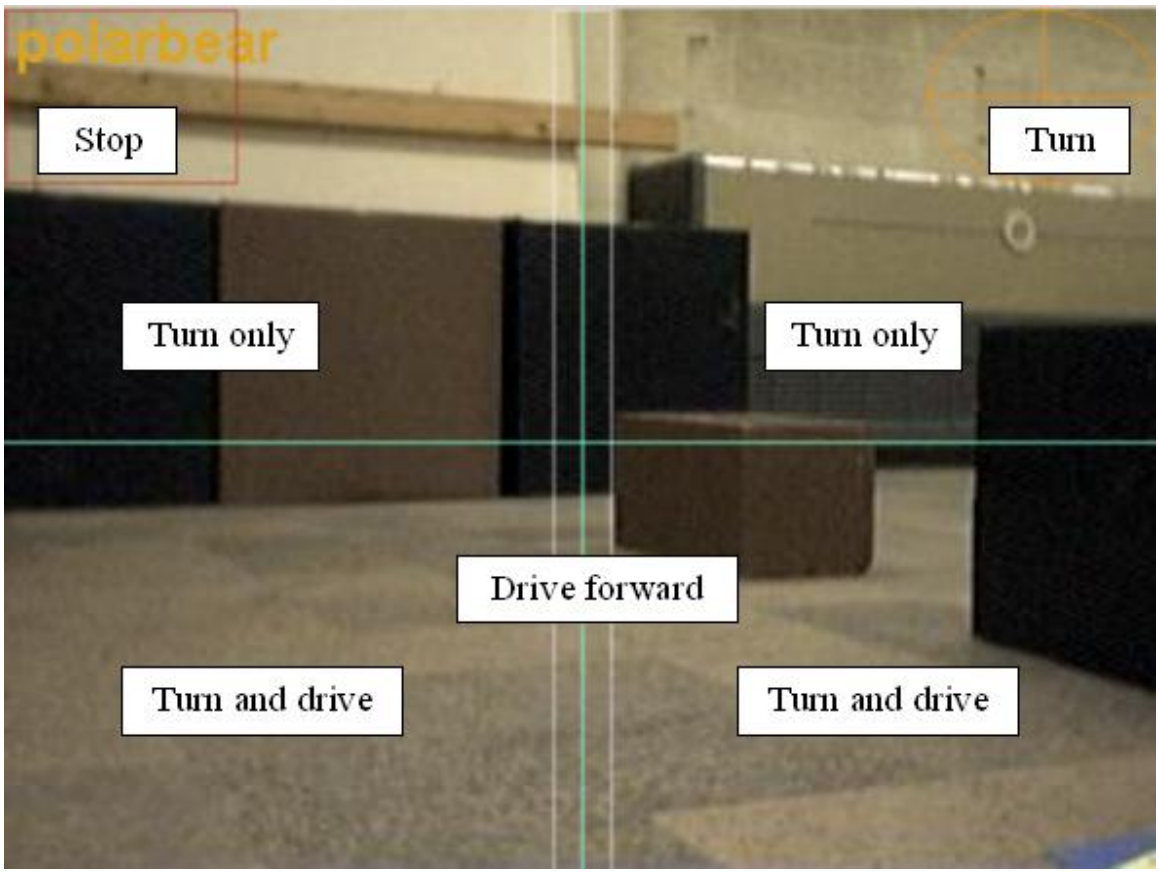

Figure 8 – Navigation interface with different areas labeled by functionality.

3.2 Navigation Interface

The robot control interface is based on a "click where you want to go" mentality. An image from the point of view of the robot is shown to the user. The location the user clicks on the image determines the type of command that is issued. The interface is shown in figure 8. If the user clicks on the lower half of the image, the robot will turn until it is facing that point and then drive forward until it is on top of the point where the user clicked. If the user wishes the robot to turn but not drive forward, the upper half of the image can be clicked. The robot will turn to face the point the user clicked, but will

not drive forward. Due to the limited field of view of the cameras being used, the user sometimes desires to turn farther than is visible by the camera. For these situations the user may click on the circle in the upper right-hand corner. One can imagine the robot being at the center of the circle facing up. How far around the circle the user clicks determines how far the robot will turn. For example, if the user were to click at three o'clock on the circle, the robot would turn ninety degrees to the right. If the user were to click at six o'clock the robot would turn one hundred eighty degrees. For situations where the user wants to simply drive forward, there is a small section in the middle of the view where the user can click to accomplish this. For safety and convenience purposes, the user may click on the rectangle in the upper left-hand corner to stop the robot at any time. When the user does this any movement commands the user has previously given or that the robot is currently following are immediately terminated and the robot will take no further action until more commands are given.

The system is able to translate where a user clicks on the screen into movement commands for the robot by translating pixels in camera space into an angle from the robot's center of view and into a distance from the robot. This process is shown in figures 9 and 10. The system must be calibrated to accurately map pixels in image space to angles and distances. This is done by measuring angles and distances for a small number of pixels. The angle from the center of the robot's view and the distance from the robot for these few pixels are then known. However, it is impractical to take such measurements for all the pixels in the image viewable from the robot's camera. For the pixels that do not have explicit measurements linear interpolators are used, one for distance and another for the angle. An interpolator interpolates, or estimates, an

unknown value that is between two known values. For example, if a pixel at height 100 is manually mapped to 50 inches and a pixel at height 200 is manually mapped to 80 inches then the linear interpolator for distance might give a pixel at height 150 a distance of 65 inches. This interpolation does not result in perfectly precise distances and angles, but given a sufficient number of manually measured points it does give adequate and effective results. This algorithm also assumes that the world in which the robot is traveling is flat. The effects of this assumption and why it only has minimal impact on the safe/unsafe system are discussed in chapter 5.

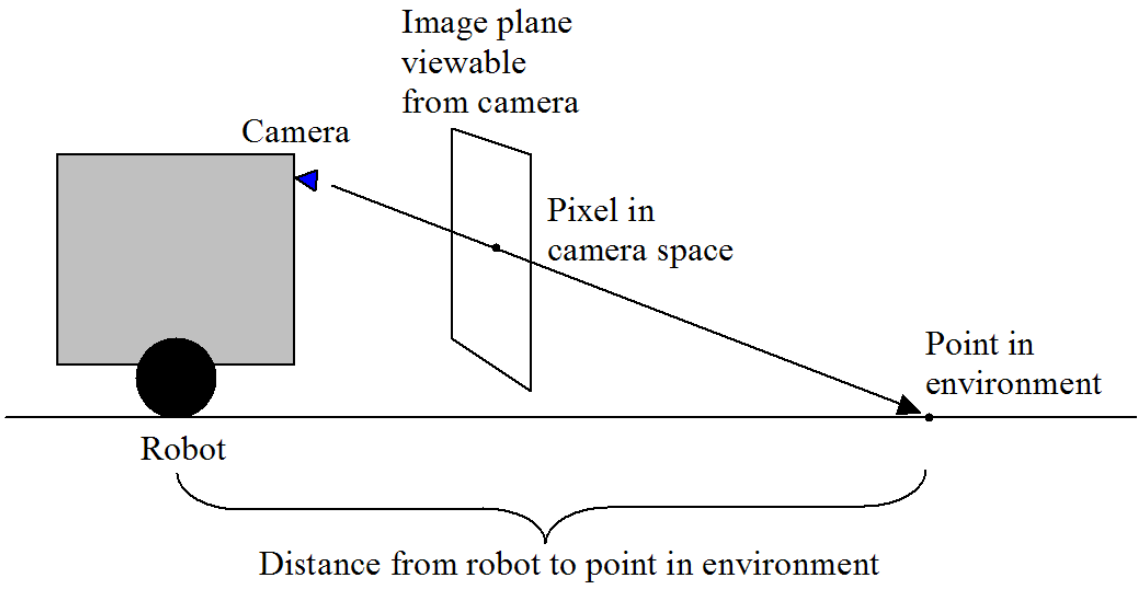

Figure 9 – Mapping a pixel seen to a distance from the robot (side view).

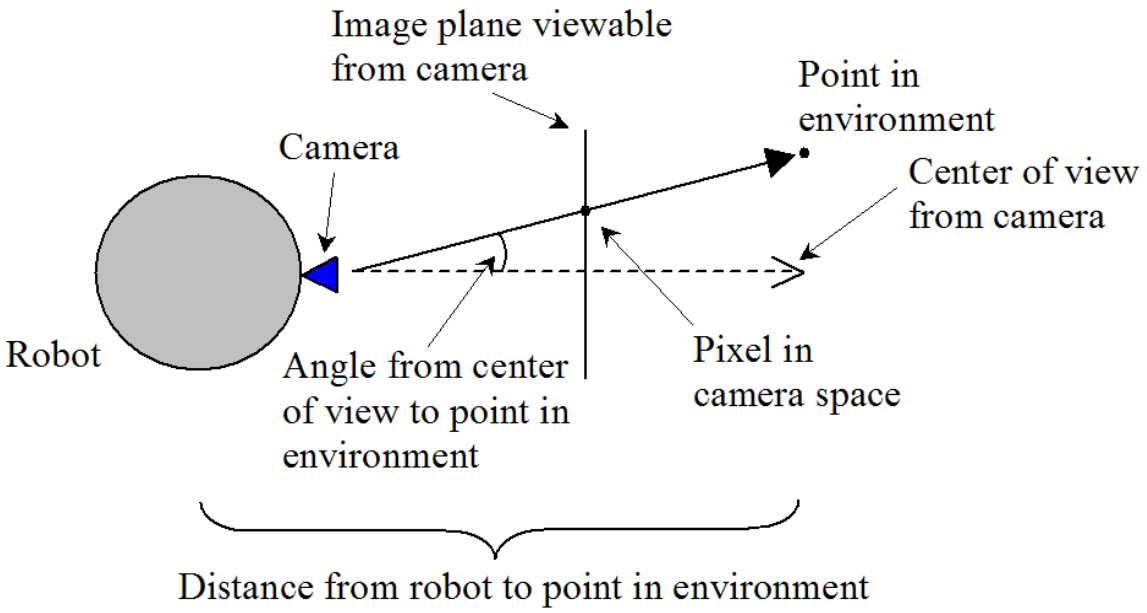

Figure 10 – Mapping a pixel seen to a distance from the robot and an angle from the center of view (top view).

3.3 Safe/Unsafe Specification

The system described above was implemented previously to and is not the focus of this thesis. Rather this thesis focuses on an addition to the above system whose purpose is to increase the efficiency with which a user may control the robot. The basic system combined with the addition is what is referred to here as the safe/unsafe system. This section describes how the various parts of this system work and interact. Extensive details about each part are covered in subsequent chapters.

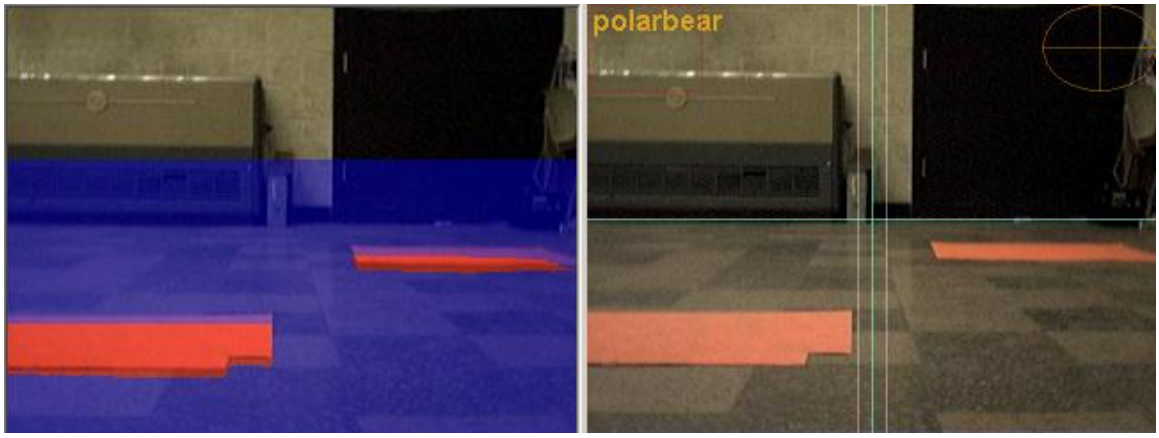

Figure 11 – Navigation and safe/unsafe specification components of user interface.

On the user interface for controlling the robot, the user is shown two images from the camera on the robot. An example of this interface is shown in figure 11. One of these images is used for navigation, as described above. The other is used to specify obstacle avoidance information. The user may draw on this second image with the mouse. The left mouse button draws in one color while the right mouse button draws in another color. The colors that were chosen were blue for the left button and red for the right button. The purpose of this is to specify areas that are safe for the robot and areas that are unsafe for the robot. The blue areas represent areas that are safe, while the red areas represent areas that are unsafe. Using the left mouse button will denote safe areas (blue) and using the right mouse button denotes unsafe areas (red). After the user finishes drawing on the image, that information is sent to the robot. The systems on the robot use that information (combined with any such previously received information) to train a classifier. The classifier is trained to classify images in terms of what areas in the image are safe and what areas are unsafe. When new images are received from the camera on the robot they are classified by the safe/unsafe classifier. This classification is sent back to the user interface and displayed over the image that the user draws on. The

transparency of the layer may be modified by the user. This allows the user to see what is currently being classified as safe and unsafe. The user will be able to immediately find areas that are misclassified and supply more training data to correct the classification.

Figures 12 through 15 show these classifier training principles. In figure 12, no training data has yet been provided (no classification overlay on the left-hand image). The user is drawing in red, specifying that the orange area on the floor is unsafe. Figure 13 shows the resulting classification. Everything is classified as unsafe. This is because the system does not yet have any counter examples. The user draws on an area of floor in blue to specify that the floor is safe. The classification after adding these two pieces of training data is shown in figure 14. With only minimal training data the system is already classifying the orange areas correctly as unsafe and the floor correctly as safe. The orange area on the right is correctly classified as unsafe, even though it was not specifically marked as unsafe. Figure 15 shows an incorrect classification. Part of the floor is being classified as unsafe. The user is drawing in blue on the incorrectly classified part of the floor to specify that is it safe. A new classifier will be created that will correctly classify the floor as safe. These principles are discussed in more detail in the next chapter.

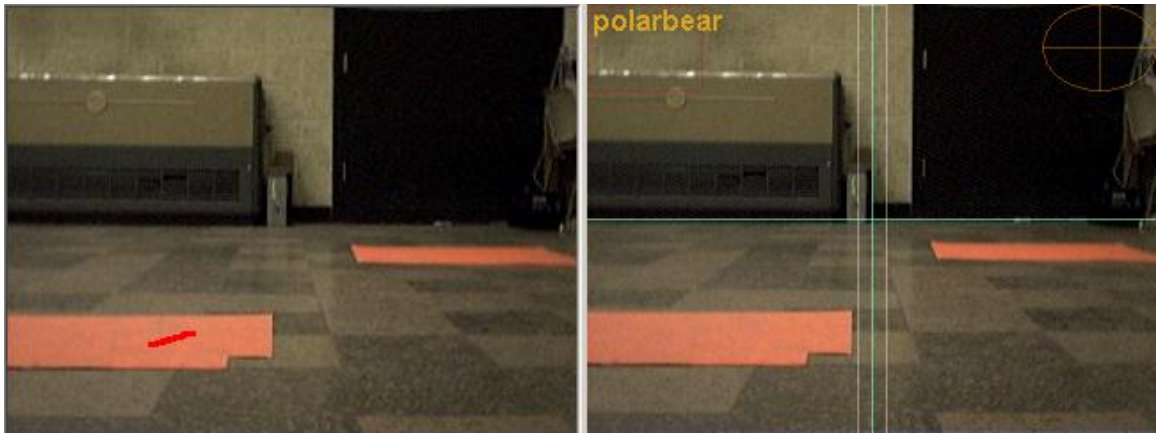

Figure 12 – User specifying that the orange area is unsafe by marking on it with red.

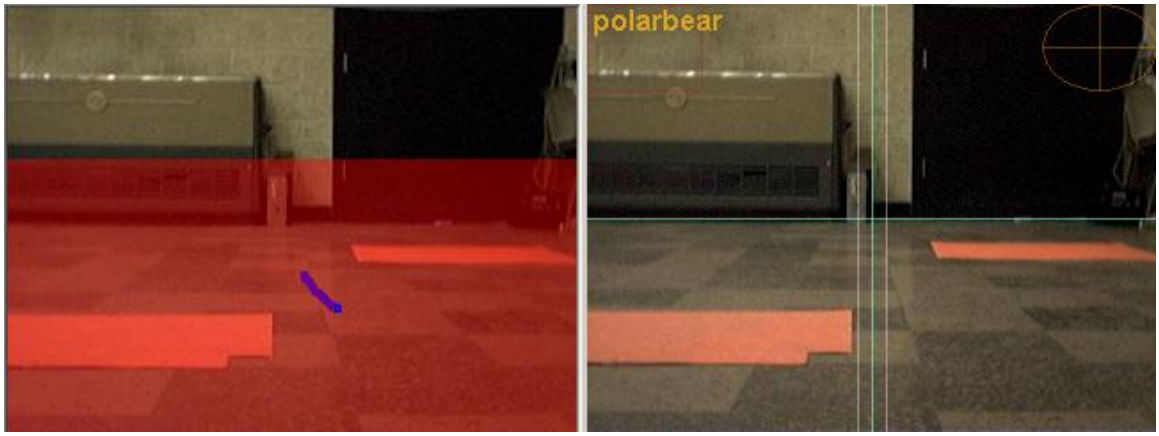

Figure 13 – Everything is now classifying as unsafe (red). User specifies that floor is safe by marking on it with blue.

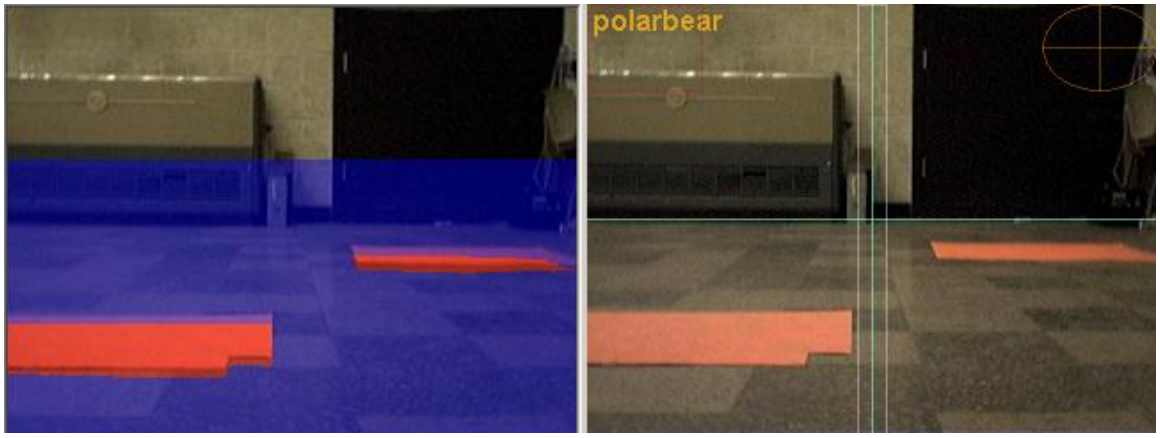

Figure 14 – Orange areas are now properly classified as unsafe (red), and floor is properly classified as safe (blue).

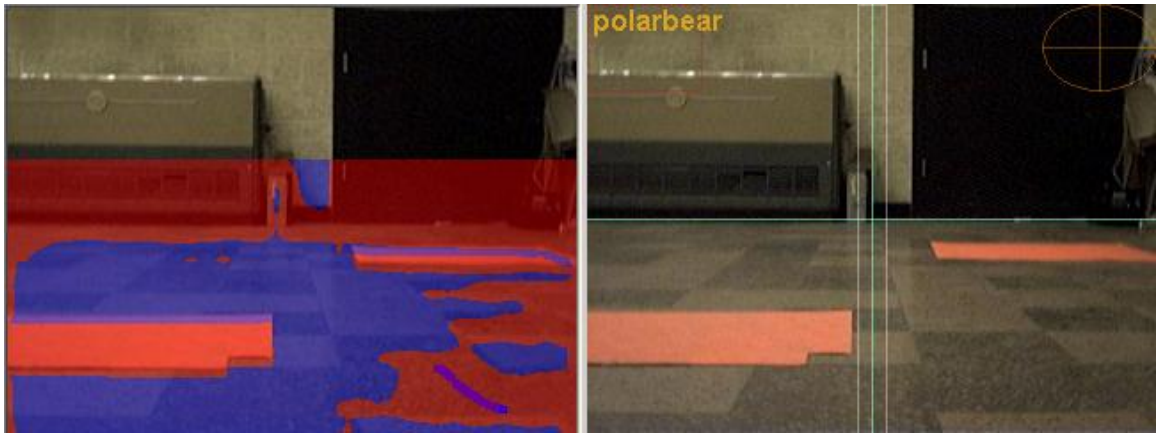

Figure 15 – Use correcting a misclassification of the floor by marking on it with blue.

3.4 Map Generation

In addition to sending the classification to the user interface, the robot uses the classification to build a map of its environment. By using the same mechanism employed in the navigation component, a point on an image from the camera (camera space) may be translated into point in the robot's environment (environment or map space), by

getting the angle from the center of view and distance to the robot. Since each pixel in the camera image is classified as safe or unsafe and each pixel in the camera image can be mapped to a point in the robot's environment, a map of the obstacles in the robot's environment can be built. An example of such a map is shown in the left-most pane of figure 16. The process will be discussed in more detail in chapter 5.

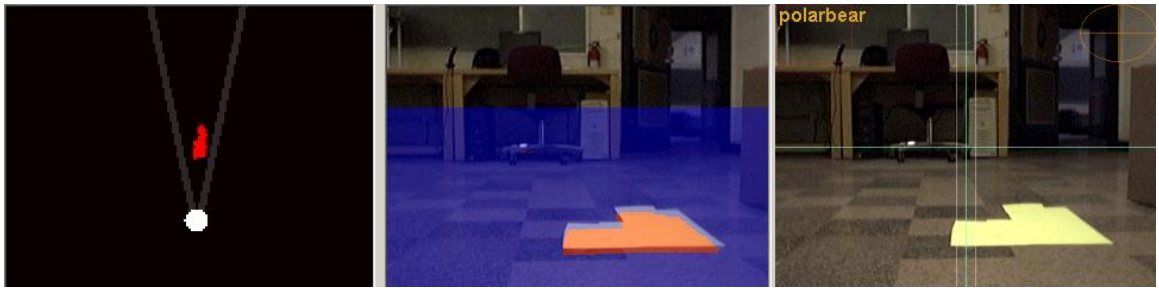

Figure 16 – The navigation, safe/unsafe and path traversal components of the user interface.

3.5 Path Traversal

This environment map is sent back to the user interface for display to the user. The environment map is also used to help the robot avoid obstacles while navigating. Suppose that the user tells the robot to drive forward, but that there is an area between the robot and the user-specified destination that is being classified as unsafe. The robot can use the environment map to first detect that such an obstacle exists and not run into it and second to plot a safe path around the obstacle to end up where the user originally told it to go. When the robot is given a movement command it generates a safe path to the specified location using the environment map. It then starts moving, making sure to follow the path it has generated. As the robot moves, the environment map is translated and rotated based on the forward movement and turning of the robot. Also as the robot

moves and its camera receives new images and they are classified, the environment map is updated with the new classification information. Thus the robot will always have current information about the safeness and unsafeness of what is directly in front of it but will also be able to "remember" how safe and unsafe the areas are that are no longer directly visible. This process will be discussed in more detail in chapter 6.

Chapter 4 – Classification

The classification portion of the safe/unsafe system is the first step in allowing the system to perform obstacle avoidance. Without classification, the system would not be able to interpret the images the robot receives from its camera. Once what the robot is seeing is interpreted, or classified, information can start being gathered about the robot's environment. In the case of the safe/unsafe system it is the map generation module (described in the next chapter) that takes the interpretation of what the robot sees and extracts information from it about the environment the robot is in.

The classification employed in the safe/unsafe system uses machine learning to create a classifier which interprets new incoming images. This classifier must be trained. Training a classifier requires adequately labeled training data. The training data is obtained by having the user "draw" what areas are safe and what areas are unsafe. The image the user has drawn on is considered the data, while the "ink" the user has drawn on the image is considered the classification of that data.

4.1 Decision Trees

While various types of machine learning could be used to create the safe/unsafe classifier, for this system decision trees [MITC97] were chosen for several reasons. The basic idea of a decision tree is to build a tree where each node in the tree asks a question about the data being classified. The answer to that question determines how the tree is traversed and eventually what classification the incoming data will receive. A simple example of a decision tree would be one with a single node. The question at the node

would be, “Is the pixel being classified completely black?” If the answer is yes then the pixel is classified as unsafe. Otherwise the pixel is classified as safe. In more complex decision trees, instead of a node having classifications for its children it has other nodes. All the leaves in the tree, of course, have to be classifications. Decision trees do not have to be binary, but the ones implemented for this system are.

In order to be able to have more useful “questions” at each node, features are generated for each pixel. These features are used to determine qualities of a given pixel and how it relates to the pixels around it. This allows information beyond the simple RGB values of a given pixel to be used in determining its classification. In the system as it is currently implemented, there are over 200 features generated for each pixel. Some of these features are as simple as the RGB values for the pixel being classified, while others take into account many of the pixels around the pixel in question. Some of these more complex features include the sum of the red component of all the pixels in a 3x3 square around the pixel in question, the sum of the blue component of all the pixels in a 9x9 square around the pixel in question and the difference between the sum of the red component of all the pixels in a 9x9 square and sum of the green component of all the pixels in a 3x3 square. The areas dealt with range in size from 1x1 to 225x225. There are also features that deal with the minimum or maximum red, green, or blue value over an area, the average red, green or blue value over an area, the difference between averages, and the difference between an average over an area and a sum over an area. Most differencing features have both areas centered around the pixel in question. Other features calculate the difference between two areas to either side of the pixel in question,

both horizontally and vertically in order to detect edges. All features are listed in the appendix.

All 200 features must be generated for each pixel when the classifier is being trained and at least some of the features must be generated when a pixel is classified. Since most of the features have to do with color values over an area, a technique known as integral images [VIOL01] is used to calculate these values quickly. An integral image is created for each color plane (red, green and blue) on each image. Each pixel in an integral image is the sum of the pixels above and to the left of it. Once the integral image has been created, the sum of the values over any arbitrary rectangle in the original image can be calculated in constant time by using the values in the integral image at the corners of the rectangle. This allows sums of areas to be calculated and compared quickly and in constant time. Thus the system can use the information about a large area around a given pixel without incurring a high calculation time.

Decision trees offer several benefits as they are used in the system. For the safe/unsafe system, the training process must be able to happen in real time. Decision trees have very fast classification times compared to most machine learning algorithms. Decision tree also have a fast training time compared to most machine learning algorithms. For the safe/unsafe system the classification must be fast, both in order to allow the robot to navigate effectively while traveling at a reasonable speed and to give the user fast feedback about the quality of the classifier currently being used. The training times must be fast so that the user can quickly add new data and correct misclassifications. If the system took hours or even minutes to create a new classifier every time the user added more data, the system would be considerably less useful. The

classification times of the system must be fast because each image that is received must be classified. To do this most of the pixels on the image must be classified. A separate classification is generated for each pixel. If the system takes too long classifying pixels it may not be able to recognize obstacles in its environment quickly enough, and run the risk of colliding with them. For the purposes of this system, decision trees offer a good balance between training times and classification times.

Another benefit of decision trees is that they can typically use a small working set of the total available features for classification. For example, in the system implemented there are over 200 features. When the classifier is being trained, the features that best classify the training data are used to create the decision tree. It is not uncommon for a good classifier to only use ten of the available 200 features to build a decision tree, but depending on the environment a different set of ten features will be used. This allows for smaller decision trees, since each node of the tree can be more "descriptive" in how it classifies or categorizes the data. A smaller tree in turn leads to faster classification times since fewer nodes have to be traversed in order to classify a given pixel.

There are many variations on the basic decision tree algorithm. The safe/unsafe system was implemented in such a way that different variations can be used relatively easily. However, one particular implementation was used the most often as it provided the best overall classification quality and execution time in the situations in which the system was used. The quality and execution time of the implementation were compared empirically to other implementations. This implementation is a binary decision tree that uses subsampling and standard deviation split points.

In a decision tree, a choice must be made at each node as to which feature should be used at that node. The training examples whose classification would use a particular node in the tree are used to determine which feature should be used at that node. A standard decision tree used all the training examples available to it to determine which feature to use at any given node. In the safe/unsafe system the number of training examples can quickly grow into the thousands or even millions. In order to reduce the impact on training time that all these examples have, a subsample of the examples at each node is used. This can greatly reduce the training time while only slightly impacting the classification time (since each node is only looking at a subset it is not as effective at dividing the remaining data as it could be, thus resulting in a slightly increased tree depth).

A second optimization to the standard decision tree algorithm that is used in the decision tree algorithm employed in the safe/unsafe system is the use of standard deviation split points. At each node of the tree a feature of the data being classified must be inspected. The node splits the tree depending on the value of that feature. This value is called a split point. In the standard decision tree algorithm, the value of each training example is used as a potential split point. The algorithm finds the best of all possible split points. There are different metrics for determining which split point is best, but they all deal with how that split point would divide up the training examples. With the standard deviation optimization, instead of checking the value of each training example for a potential split point, the mean and standard deviation are used to find "clusters" of training examples. The values of these "clusters" are used instead of the values of the individual examples. This allows for potentially superior split points. This results in a

smaller tree which translates into decreased classification times. Again, the size of the tree was compared empirically with other implementations.

4.2 Classifier Training

The safe/unsafe system uses a decision tree as the basis for its classification process. But this decision tree must be built using data that has been labeled as safe and unsafe by the user. This labeled data is called training data. Each time the user supplies new training data the previously constructed decision tree is discarded and a new one is built using both the new training data and any previously received training data. The classification portion of the safe/unsafe system uses a decision tree to classify each pixel in an image received from the robot's camera, marking each as safe or unsafe. The following images show an example of how a safe/unsafe classifier might be trained. The images are discussed below.

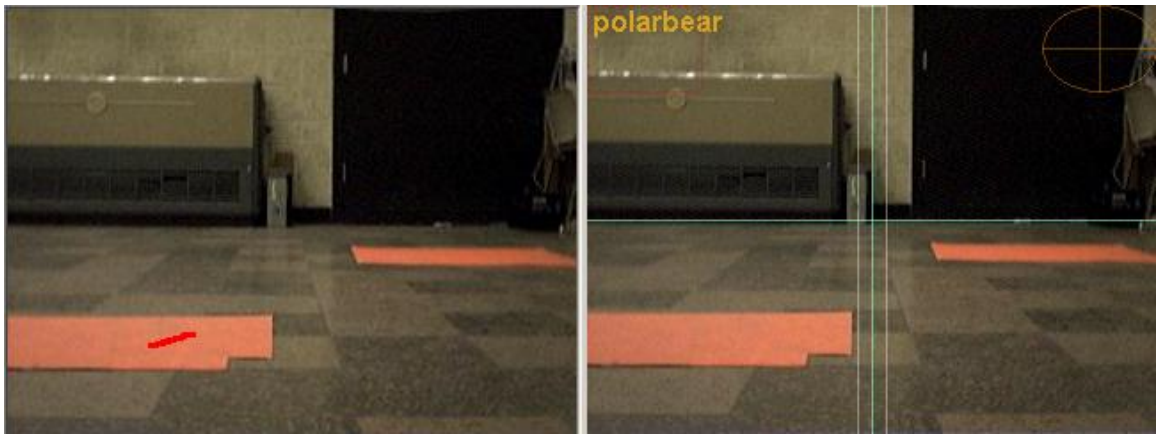

Figure 17 – User specifying that the orange area is unsafe by marking on it with red.

Figure 18 – Everything is now classifying as unsafe (red). User specifies that floor is safe by marking on it with blue.

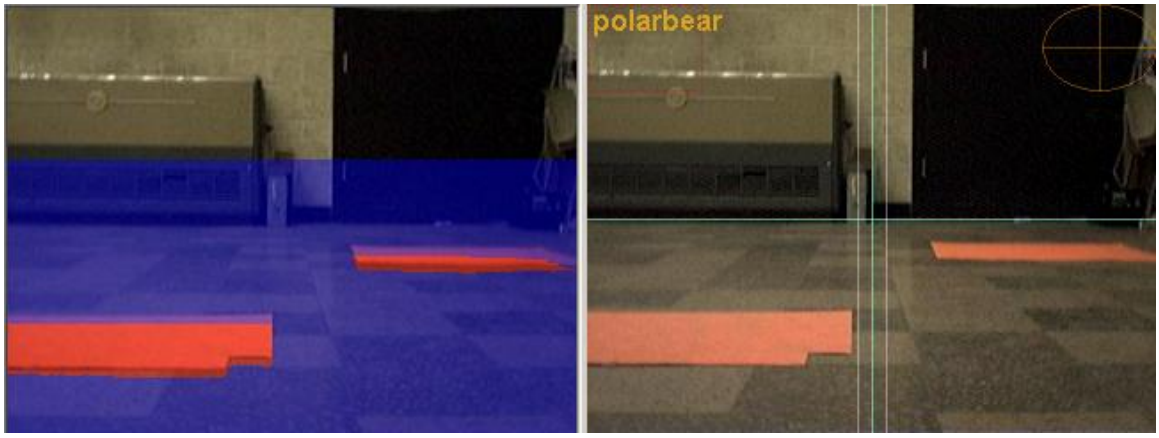

Figure 19 – Orange areas are now properly classified as unsafe (red), and floor is properly classified as safe (blue).

Figure 17 shows a view from the robot's camera before the user specifies any training data. In the view on the left the user is adding an unsafe (red) specification to the orange area on the floor. This is to specify that the orange area is an unsafe area and that the robot should not drive on it. When the user presses the right mouse button down, red ink (specifying unsafe) begins to be drawn. Anywhere the user moves the cursor on the image is marked as unsafe. When the user releases the right mouse button the current

image from the camera is saved with the ink the user has drawn on it. This information is passed to the classifier. The classifier uses each pixel from the camera image that has ink associated with it. Camera image pixels that the user did not mark as safe or unsafe are ignored since it is not known whether they should be safe or unsafe. Features are generated for each pixel with a classification. Since no previous training data exists, a decision tree is built using the pixels and classification the user just specified.

The resulting classification of this decision tree is shown in figure 18. In the left view in figure 18 everything is overlaid with red, indicating it was classified as unsafe. This is incorrect, but is as correct a classification as the machine learning can produce. At this point in the process the training data consists of examples of only one classification (unsafe). There are no examples of any other classification. In a sense, the classifier doesn't yet know that anything can be classified as safe, only unsafe, so everything is classified as the only classification known to exist. In order for the classifier to be able to classify pixels as safe, it must have some examples of what safe pixels look like. This is what the user is specifying by drawing blue ink in the middle of the left view in figure 18. The user is specifying that the area of floor in the image is safe and that the robot should be allowed to travel there.

The specification of a safe area works much the same as the specification of an unsafe area. To specify a safe area the user presses the left mouse button down. While the button is pressed, anywhere on the camera image that the user moves the cursor is marked as safe (indicated by blue). When the user releases the left mouse button the current camera image and the user-specified classification are sent to the classifier as new training data, just like when an unsafe area was specified. In the situation shown in the

figures above some training data already existed (when the user indicated that the orange area on the floor was unsafe), so the current training data is added to the existing training data. A new decision tree is created that correctly classifies both the original training data and the new training data.

The classification results of this new decision tree can be seen in figure 19. It can be seen that the orange areas on the floor are correctly classified as unsafe while the floor itself is correctly classified as safe. Even the orange area farther away is correctly classified as unsafe, even though it was never explicitly marked as unsafe by the user. This is one of the great strengths of the classification mechanism used in the safe/unsafe system – generalization. Since one orange area was specified as unsafe, all orange areas will be classified as unsafe, assuming that the features for the pixels are similar enough to those of the pixels that were originally marked as unsafe.

If a situation is ever encountered where an area is classified incorrectly, the user simply has to add more training data and a new classifier will be created that will correctly classify the area. Adding the new training data and creating a new decision tree can be done in a matter of seconds (or less depending on how much area is specified by the user and how much previous training data there is). The user gets feedback about how the classifier is working almost immediately and can correct errors almost as quickly. This is one of the greatest advantages of the safe/unsafe system.

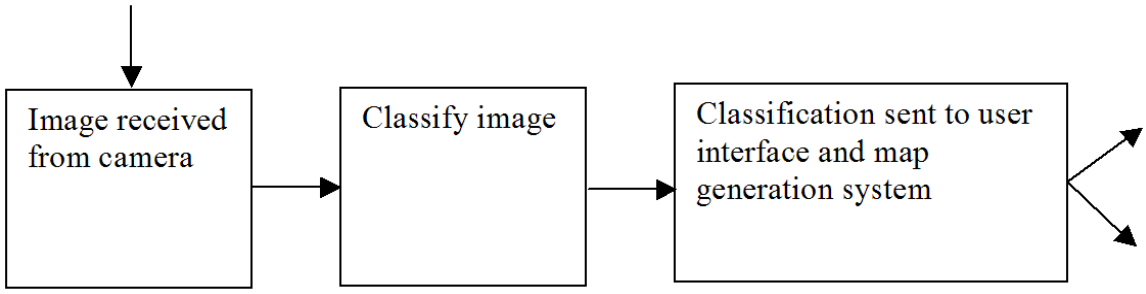

Figure 20 – Basic classification process.

4.3 Classifying Images

This section describes how the classification process works in the safe/unsafe system. The process is summarized in figure 20. First, an image is obtained from the camera. This image is passed to the classifier. The classifier sends each pixel of the image that is to be classified to the decision tree. The decision tree returns a class (safe or unsafe) for that pixel, depending on the past training data it was created with. Once all the pixels have been classified, the resulting classified image can be sent to other parts of the system to determine what should be done based on which areas in the image are safe and which are unsafe.

The safe/unsafe system as currently implemented does not classify every pixel of every image. This is primarily for speed reasons. The system can classify every pixel of every image, but then most of the computational resources of the system are spent on classification. This leaves few resources available to actually react to those classifications. A balance was found between the amount of information being classified and the system's ability to react. As more computational resources become available in

the same form factor and as additional optimizations are made to the system, this will be less of an issue.

The first optimization taken is that only the bottom two thirds of the image is classified. The top third is always above the horizon. Since the robot is unlikely to encounter many obstacles in the sky, this seems like a valid course to take. The second optimization is that only every other pixel is classified. There are virtually no obstacles that could impede the robot that would show up as only a single pixel in an image received from a camera mounted on the robot. All obstacles are at least several pixels wide. So no information about the general location of obstacles is lost. In fact this can actually improve the accuracy of obstacle detection if there is significant noise in the classified image. The third optimization is that only every other image received from the camera is classified. The camera receives images fast enough when compared to the speed the robot travel at so that this optimization does not hinder the system's ability to avoid obstacles. Enough information is classified far enough in advance that the system is able to make the necessary adjustments in order to avoid obstacles.

Chapter 5 – Map Generation

The map generation module of the safe/unsafe system receives a classified image that has been created by the safe/unsafe classifier. This image indicates what parts of the robot's view are safe and which are unsafe. The map generation module uses this information to create a map of the area around the robot. This environment map shows unsafe areas (potential obstacles) in relation to the robot's position. The map can be used to find a safe path from where the robot is to where the user wants it to go. The path generation module is explained in the next chapter.

The main purpose of the environment map is to give the robot a memory about where it has been and what it has seen. This is important for the following type of situation: the robot is traveling in an area with two obstacles. The robot must turn to avoid the first obstacle. This puts the second obstacle in the robot's path. The robot might turn to avoid the second obstacle and put the first obstacle back in its path. If the obstacles are too close the robot might not be able to see the first obstacle when it makes its second correction. In this type of situation the robot would collide with the first obstacle and never know it. The environment map allows the system to "remember" the location of the first obstacle so as not to collide with it while avoiding the second obstacle.

5.1 Generating the Environment Map

In order to perform the mapping from a classified image to an environment map, the system uses the same algorithms that are employed in the click-and-drive driving

interface explained earlier. The driving interface works by translating a point in camera space (the point where the user clicks) into a distance from the robot and a number of degrees away from where the robot is currently facing. This effectively maps a point in camera space to a point on the floor in the robot's environment, which is what is required for the environment mapping algorithm.

In order for the click-and-drive interface to function correctly, it must be properly calibrated. This calibration also affects how accurately classified images are translated into an environment map. The method of how this is accomplished by using several measured points and then linear interpolators for those points that aren't explicitly measured is detailed in the section on navigation interface in chapter three. One of the figures from chapter three that describes this process is repeated in figure 21. This same method of mapping pixels in the camera space to distances and angles that are used to drive the robot is also used in creating an environment map. This method is not completely accurate but is sufficiently accurate to be not only functional but effective as well.

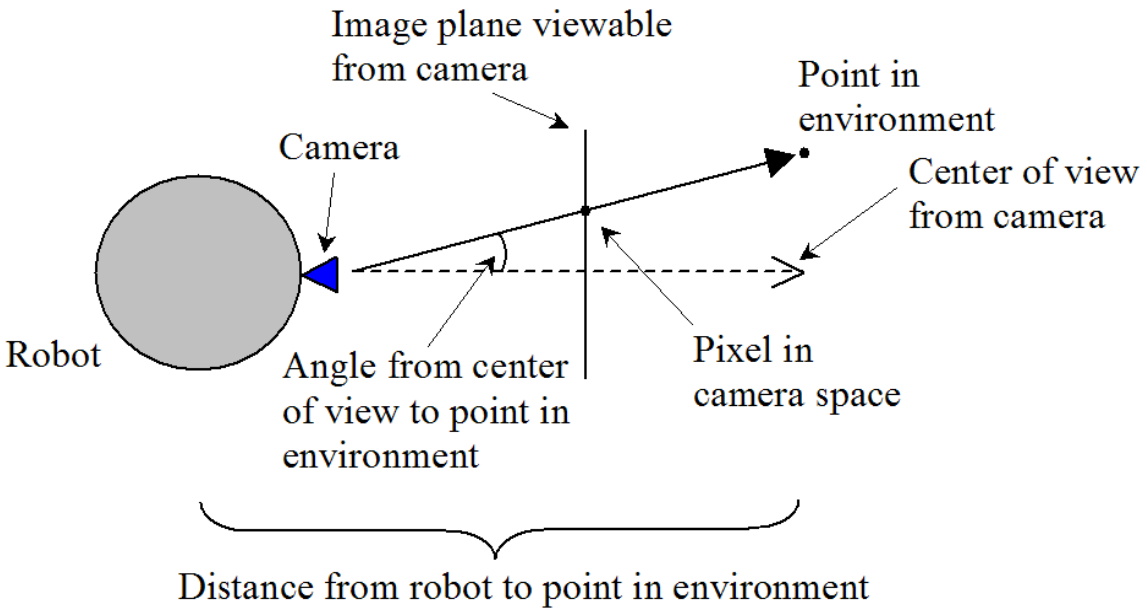

Figure 21 – Mapping a pixel seen to a distance from the robot and an angle from the center of view (top view).

The algorithm for mapping points from a classified image to the environment map is shown below.

- Get x and y coordinates of classified pixel. These coordinates are in camera space
- Use the horizontal linear interpolator to find angle from robot's center of view. This angle is in environment space
- Use the vertical linear interpolator to find distance from robot's current location. This distance is in environment space
- Use trigonometric functions to convert the distance from the robot and the angle from its center of view to forward and horizontal components
- Add these components to the robot's current position to get absolute coordinates on the environment map

One of the benefits of obtaining a safe/unsafe mapping of the robot's environment, other than getting useful obstacle avoidance information, is the ability to give new types of commands. The system allows the user to click on a point on the map to specify a goal location for the robot. First, this allows goal locations to be given that are not currently viewable by the robot's camera. Second, and more importantly, this allows commands of the type "Go to the other side of that obstacle" to be given. The user can give commands with the normal interface that will result in similar behavior of the robot, but this new method results in a more intuitive way to give this type of command. For example, in figure 22 the yellow area on the floor is being classified as unsafe. The environment map is shown in the left-most pane of the figure. The red area in the environment map represents the yellow area on the floor, but in environment space instead of in camera space. The white circle on the environment map represents the robot. The area inside the two gray lines represents the area that is currently viewable by the robot's camera. With the environment map the user is now able to see the entire area around the robot. The user can now click above the red area on the environment map to tell the robot to go to that location. The meaning of this command is "go to the other side of the obstacle". A command that would result in the same behavior could be given in the normal navigation interface, but it would have a different meaning. It would mean "drive forward *x* inches". Driving that distance just happens to be on the other side of the obstacle.

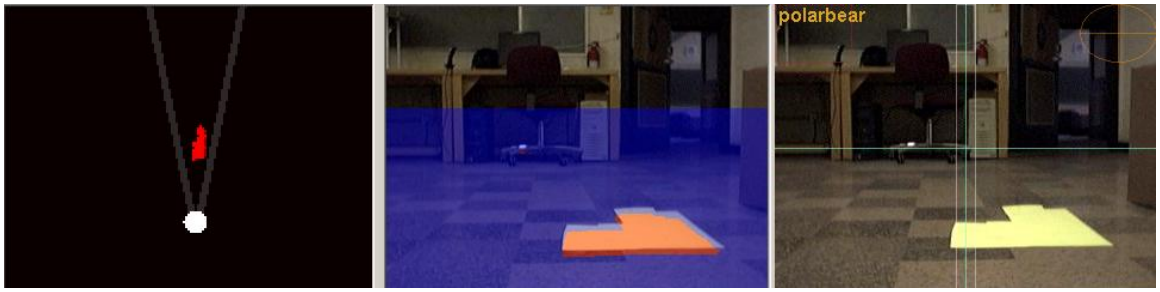

Figure 22 – Simple obstacle classification and accompanying environment map.

It should be noted that the classified image to environment map mapping algorithm cannot distinguish between long, flat objects and tall objects (using a single image). A tall unsafe object will be treated as though it were a long, flat unsafe object thus covering more area in the environment map than it the object does in real life. This is acceptable for two reasons. First, it is a false positive in regards to unsafeness. This may cause the system to avoid an area that in reality doesn't need to be avoided, but it does not cause the system to enter an area that is unsafe. The system errs on the side of caution. Second, when the robot moves past the tall, unsafe object it will be able to see that the area originally classified as unsafe is actually safe and the environment map can be updated with the new information. So typically the false positive is only temporary anyway. An example of such a situation is shown in figure 23. The box is being classified as unsafe. In the environment map this is being translated into a very large unsafe area. However, as the robot moves around the box the area that is being incorrectly classified as unsafe will get correctly classified as safe, and the robot will be able to travel behind the box if the user directs it to.

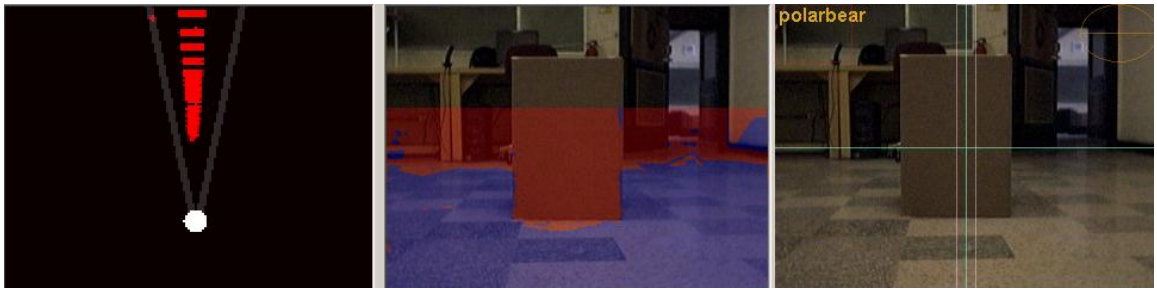

Figure 23 - Classification of tall obstacle.

5.2 Updating the Environment Map

In order for the map to be meaningful it must constantly be updated as the robot gains new information. It must also remember what the robot has already seen but can no longer directly observe. The map is stored as an array of "safeness" values. The value stored at each position on the map can range between 0 and 255. The safer a point is, the lower the number being stored, and thus the higher its "safeness" value. Each time a point on the map is identified as safe the value in the map reduced by 5. To help reduce the effects of noise, the points around the point classified are also reduced (though only by 2 in this case). The same process is followed when a point is identified as unsafe, except the values are increased at the primary point by 10 and at the surrounding points by 5. This essentially gives more weight to unsafe classifications than it does to safe classifications. This was done so that the system can react more quickly to new unsafe areas that are discovered. The map is continually updated as new images are received from the camera and classified.

The algorithm for updating the environment map with data from a new classified image is below.

- For each pixel in the classified image

- Map the pixel to environment space
- If the pixel is safe
  - Decrease the value at the point on the environment map by 5
  - Decrease the values at the four neighbors of the point by 2
- If the pixel is unsafe
  - Increase the value at the point on the environment map by 10
  - Increase the values at the four neighbors of the point by 5

The map must not only be updated as new images are received, but also as the robot moves. The paradigm chosen for this was to have the robot's position on the map remained fixed while the map translates and rotates around the robot as the robot drives and turns. Since with the current control model the robot can only move forwards and since on the map the robot is always facing upwards only down translations of the map must be accounted for. This simply moves values on the map to points lower (higher Y values, since the origin is in the upper left-hand corner) on the map. The system knows when to translate the map and how far to translate it based on feedback it receives from the Javelin board. The Javelin board tells the system how far forward the robot has moved. To rotate the map, a simple backmapping algorithm is used with the robot as the point of rotation. Again, when to rotate and by how much is determined by feedback from the Javelin board. This dead reckoning approach can lead to inaccuracy in terms of reaching the goal position (this is discussed in more detail in chapter 7). However, the

obstacle avoidance parts of the system are affected very little by the inaccuracies of dead reckoning. This is because the environment map is update so frequently with new data about the environment and the obstacles in it.

Chapter 6 – Path Traversal

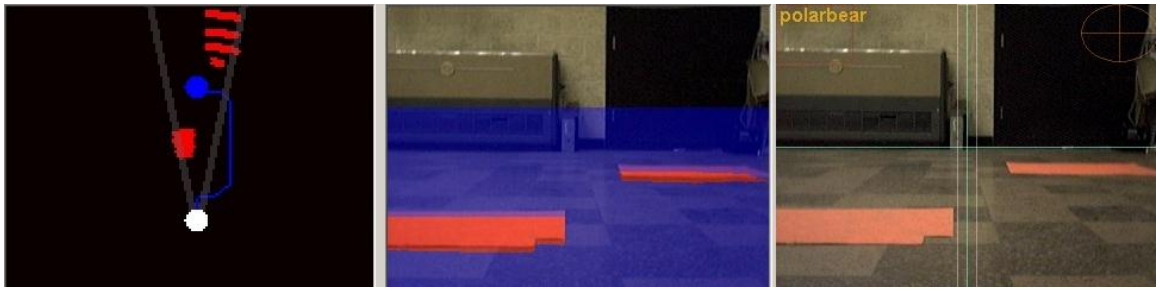

Figure 24 – User interface with path displayed (left-most pane).

The purpose of creating an environment map is so that the location of obstacles relative to the position of the robot may be determined. When the locations of obstacles have been determined then the obstacles may be avoided. Avoiding obstacles is the main goal of the safe/unsafe system. The system uses the environment map to find a path, such as the one shown in figure 19, from where the robot currently is to where the user told it to go, that goes around obstacles rather than through them.

The first step in using the environment map for navigation is to generate a path from the robot's current position to the place it is trying to go. When the user gives the robot a command, the location the user specified is determined in the environment map using the same mapping that is used to map a classified image to the environment map. After finding the goal location, a path between the robot's location on the environment map and the goal location on the environment map can be found. The second step is generating the commands necessary to cause the robot to follow the path that has been generated. If the direction the robot is currently traveling will take it too far off the path the appropriate turn commands must be given to ensure that it stays on the path. If the

robot reaches the goal then the goal is removed. When the user gives the robot a new command, the goal location is recomputed, and the old goal location is discarded. The main components of this process are shown in figure 25.

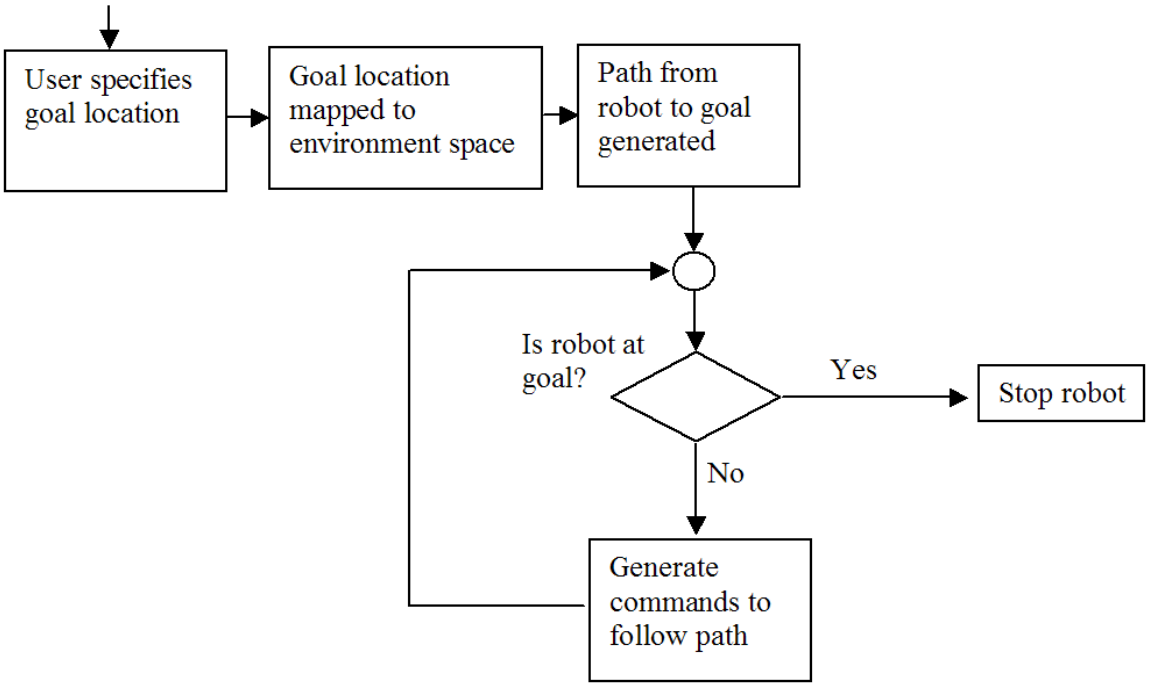

Figure 25 – Basic path traversal algorithm.

6.1 Path Generation

The path from the robot's location to the goal location is computed using a best-first search [PEAR84]. Many search and path planning algorithms exist [RUSS03], but a best-first search was chosen because the information available to the system lends itself well to such a search (the robot and goal locations are known as well as safeness information about all locations between the robot and the goal). Each node in the search represents a position on the environment map. A list of all the nodes on the current

search frontier is kept, sorted according to a heuristic value at each node. The heuristic value is computed from the distance of the node to the goal location combined with the "unsafeness" of that node. The unsafeness of a node is determined by finding the average unsafe value in an area around the node in the environment map. To make computation faster, an integral image of the environment map is generated before finding a path through it. This allows the algorithm to compute an arbitrarily sized unsafeness value at any node in constant time. The unsafeness value over an area is used, as opposed to the value just at the node in question, to take into account the fact that the physical size of the robot takes up an area on the environment map, as opposed to a single location.

When the unsafeness value is found, it is multiplied by a weighting factor and then added to the distance of the node to the goal location. This combined value, distance and unsafeness value, is the heuristic value for the node. The actual equation is $abs(x_{robot} - x_{goal}) + abs(y_{robot} - y_{goal})$ + unsafeness value * weight factor. This weight factor was used to increase importance of the unsafeness of a node when determining whether it should be used as part of the path to the goal or not. Prior to adding a weighting factor the system would sometimes generate a path that led the robot through a very unsafe area because that was the path with the lowest heuristic. Different weight values were not tested extensively, but it was found empirically that a value of 2.0 worked well. A value of 1.0 to 2.0 resulted in the system sometimes still generating a path that led through an unsafe area. A value of 2.0 or greater resulted in the system generating a path that avoided unsafe areas, but often took the robot much farther away from obstacles that it needed to go. One situation where this became especially important was when there were two obstacles fairly close together but with enough room for the robot to travel between

them. In type of situation if the weight factor was set too high then the system would not generate a path that led between the two obstacles, even if there was enough room for the robot to safely pass between them.

Once the heuristic value has been computed for the node, the node is put into the list of frontier nodes, and the list is sorted so that nodes with lower heuristics come first in the list. When a new node is taken from the list to be processed, it is taken from the front of the list (so it has the lowest heuristic). The heuristics for that node's neighbors are computed and the neighbor nodes are added to the list (if the nodes haven't already been visited). The top five neighbors are added to the list (as opposed to all surrounding eight) both to reduce the number of nodes being searched and to keep the algorithm from generating paths that would force the robot to have to backtrack. The search ends when the node containing the goal location is found. To help keep the system responsive, a timeout is also used. If finding the path takes over a certain amount of time, then instead of returning a complete path to the goal, the algorithm returns a path from the robot's location to the node that the algorithm has processed that is closest to the goal. The timeout is not normally used, but can be helpful in keeping the robot moving towards the goal in situations where the environment map is particularly difficult to traverse.

Sometimes the path needs to be regenerated. This is done under several different circumstances. First, if while following a certain path the robot receives a new command from the user, the path is regenerated since the goal location has changed. Second, if the area around the robot has changed significantly in safeness the path is recomputed. Whether or not a significant change has occurred is determined by computing the difference in the sum of the safeness values around the robot when the robot's current

path was initially computed and the current sum of the safeness values around the robot. If the difference between these two sums is great enough (a difference of 8500 was determined empirically for the safe/unsafe system) then the path is recomputed. It could be that when the path was originally computed then an area appeared safe, but as the robot got closer more information was gathered, and it was determined that an area was actually unsafe. If this is the case then the path should be recomputed so the new information can be taken into account. Third, since the map rotation algorithm is imperfect multiple rotation operations can cause the path to slowly degrade. If the path has degraded sufficiently it is regenerated. This is done by determining how many points in the environment map around the robot are marked as part of the path. If the number is too small then the path is regenerated. A special case of this is if there are no path points found around the robot. This can happen if the path has degraded or if the robot has somehow gotten off the path. The path is regenerated in this case also. This will result in a new path from the robot's current location to the goal, using the most up to date information available from the environment map.

6.2 Path Following

After generating a safe path based on the environment map, the robot may use that path to help it navigate. In order to do this the robot must be able to follow the path that has been generated. The part of the safe/unsafe system that is responsible for following the path uses a two window approach. These two windows are fixed areas on the environment map directly in front of the robot's position on the map. If the path is not found in either of these areas then it is regenerated (as discussed above). Since the

start point of path generation is inside the first window this won't result in infinite path regeneration.

The algorithm employed for both windows is the same. The point in the window that is farthest away from the robot and contains part of the path is found. The angle from the robot's current heading to this point is computed. This angle may be used to generate a turn command to ensure the robot stays on the path. The angle from the closest window is always computed first. If the angle for the first window is too large or too small, the angle from the second window will be computed and used. This is to help ensure that the path the robot ultimately takes is relatively smooth. The path generated can contain a lot of small direction changes. Typically all of these small turns do not need to be turned into commands, but rather the overall course of the path must be followed instead. Finding the farthest path point and using the two-windowed approach help the robot to follow the overall course and ignore all the minor variations in the path. An example of a path that would require many small turn commands to follow exactly is shown in figure 26. The jagged part of the path directly in front of the robot changes directions many times in a short distance. The purpose of using a windowed approach is to avoid performing many small turns and instead follow the general direction of the path.

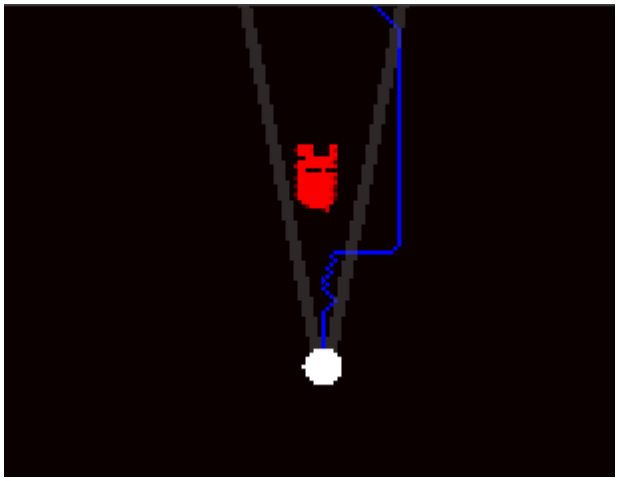

Figure 26 – Path requiring many small turns to follow exactly.

Figures 27 through 29 show examples where the two different windows are used. The windows are shown as green rectangles (they are not part of the normal interface). Figure 20 shows an example of when the angle from the first window will be used. In figure 21 the angle from the second window will be used. This is because the angle from the first window is too small (zero in this case). In this case having the robot turn now instead of waiting until it is further along the path will help it to avoid colliding with the obstacle. Figure 22 shows an example where the angle from the first window is too large. According to the path, the robot should make almost a 90-degree turn to the right in order to stay on the path. However, a smaller angle will result in a more fluid movement around the obstacle. Therefore the angle from the second window will be used.

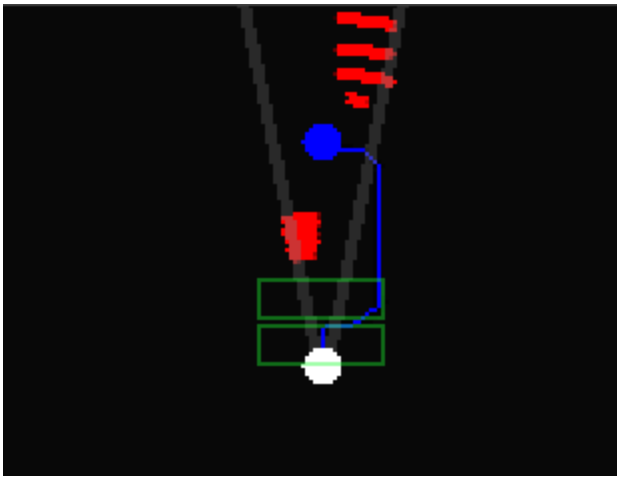

Figure 27 – Situation where first window will be used to determine angle adjustment.

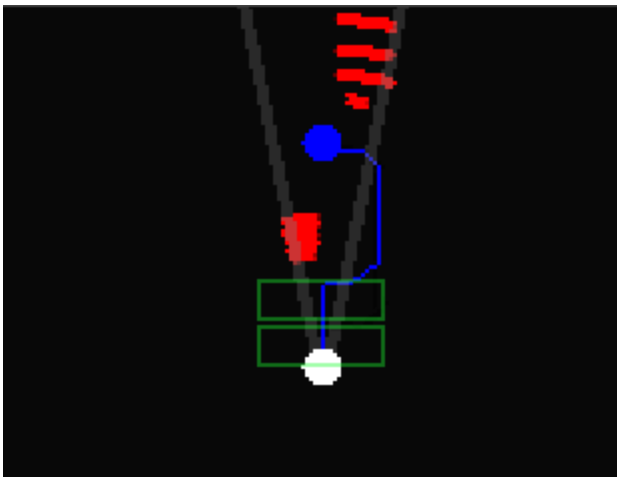

Figure 28 – Situation where second window will be used to determine angle adjustment.

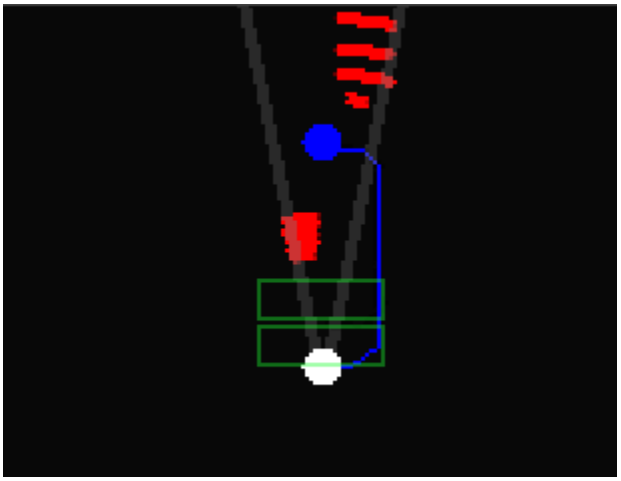

Figure 29 – Situation where second window will be used to determine angle adjustment.

When it has been determined that a turn must be made in order for the robot to stay on the path, a turn command is generated and sent to the Javelin board. After the robot finishes turning a second command (a drive command) is given. This is to make the robot start advancing towards the goal again. If the robot is ever at the goal (a small margin of error (5 inches) is used in determining whether or not the robot is actually at the goal) then a stop command is generated, and no further path generation or move commands take place until the user gives a new command.

Chapter 7 – Evaluation

In evaluating the safe/unsafe system, there are many possible metrics that could be used. For the purposes of this thesis two main areas of evaluation are taken into consideration. The first evaluation metric is whether or not the system can avoid obstacles. The whole purpose of the system is to automatically avoid obstacles, thus allowing the user to concentration on other tasks. If the system cannot avoid obstacles then it is not effective.

The second evaluation metric is how close the robot gets to where the user told it to go. If the user tells the robot to go to a particular point and the robot does not go to that point, but does not run into any obstacles, then the system is still mildly useful, but not nearly as useful as it could be. It is impossible for the robot to go to the exact location the user specified (due to inaccuracies in the goal location specification interface and the inaccuracies of the robot's physical components), but it should get close. Exactly what "close" means can depend on what the robot is to be used for. Traversing a minefield without driving over any of the mines might require a much more restrictive definition of "close" than might a wilderness reconnaissance operation.

7.1 Test Description

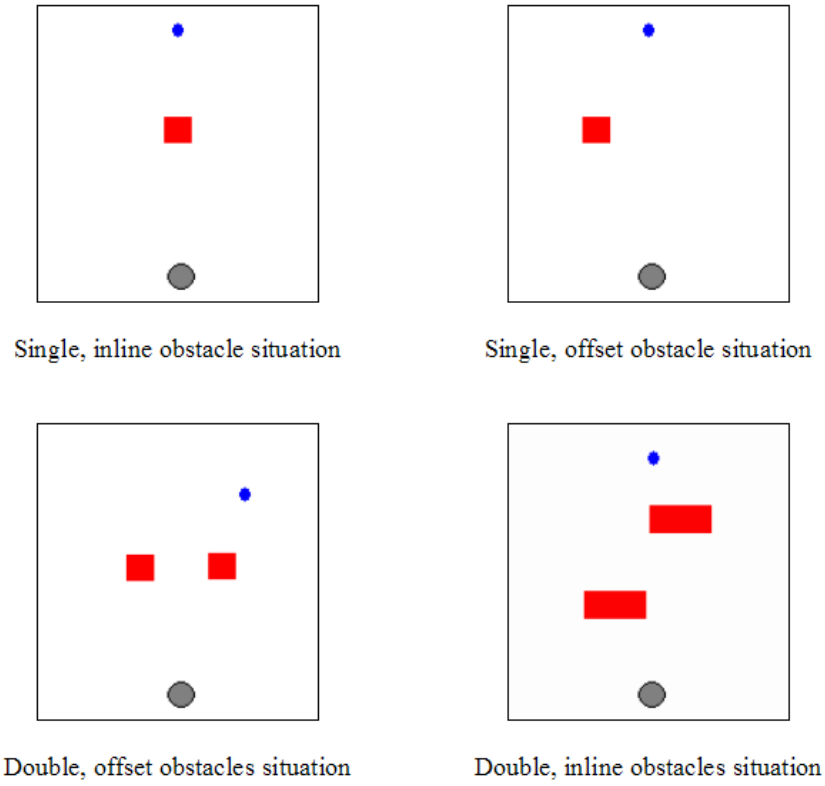

Figure 30 – Overhead representation of four main test situations.

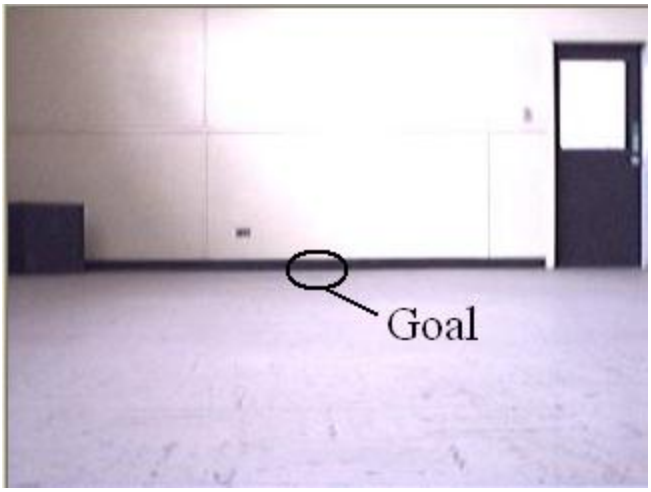

Figure 31 – Situation with no obstacles.

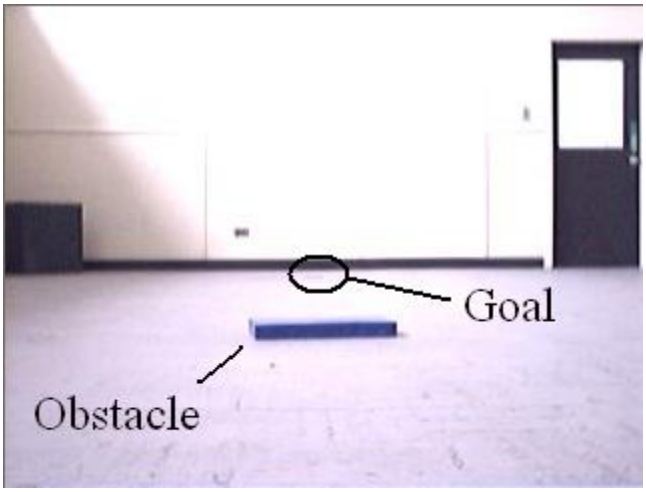

Figure 32 – Situation with single, inline obstacle.

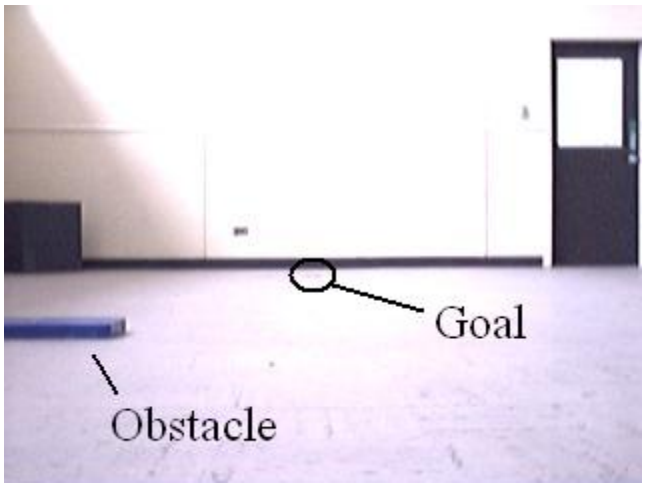

Figure 33 – Situation with single, offset obstacle.

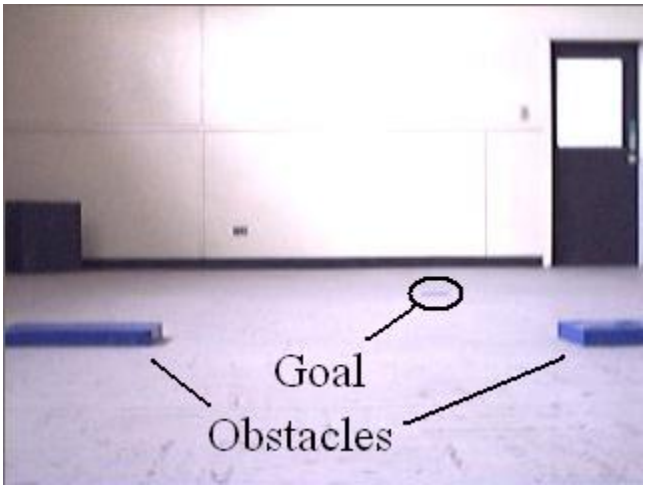

Figure 34 – Situation with double, offset obstacles.

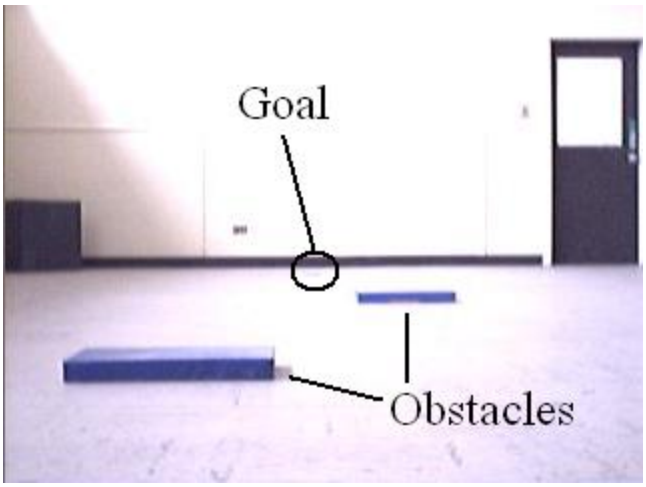

Figure 35 – Situation with double, inline obstacles.

To test the system, four different situations were used. These four situations are diagrammed in figure 30 and pictured in figures 32 through 35. While these test situations don't represent all possible real world situations, they do represent a large percentage of situations commonly encountered. In the first of these test situations the robot is instructed to drive in a straight line, and there is one obstacle in its path. This is the single, inline obstacle situation shown in figure 32. The robot has to avoid the one obstacle to get to the goal specified. In the second situation the robot is again instructed to drive in a straight line, but this time the obstacle is close to the robot's path but not directly in it. This is the single, offset obstacle situation shown in figure 33. The robot doesn't need to make any corrections in order to reach the specified goal. In the third situation there are two obstacles. The goal the robot is given is between the two obstacles and slightly to one side. The robot can't go straight to the goal without running into one of the obstacles. This is the double, offset obstacles situation shown in figure 34. In the fourth situation there are two obstacles, both in the path of the robot. Both of the obstacles must be avoided, but in opposite directions. This is the double, inline obstacles

situation shown in figure 35. In all situations, the distance from the robot's starting position to the desired goal location was 252 inches. In all situations a single drive command was given to the robot telling it to go to the desired goal location. All other movement after that was controlled by the system. Each situation was traversed seven times. The results of each run were recorded and the average was computed.

In order to determine the inherent inaccuracy of the system, a fifth situation was also tested, a situation with no obstacles as shown in figure 31. The robot starting position and goal position were the same distance from each other as specified above. The robot was given a single drive forward command with the intent to drive from the starting position to the goal position. The same measurements were recorded, as with the other situations, though for the no-obstacle situation recording the number of corrections and collisions is not meaningful since there are no obstacles. The results of this situation are in figure 36.

No-Obstacle Situation

	Corrections	Distance to Goal	Collisions
	0	20	0
	0	28	0
	0	29	0
	0	29	0
	0	34	0
	0	34	0
	0	42	0
Avg	0	30.9	0

Figure 36 – Results of no-obstacle situation. Distances are in inches.

The distance to the goal in this situation can be accounted for by several factors. The first is the inaccuracy of the goal specification system. Since the distance to the goal is specified by clicking on a pixel and the number of pixels is limited, especially as the

distance to the goal increases, it is impossible to tell the robot to go exactly 252 inches. The second factor is that while the robot is driving its wheels tend to slip. This causes the system to think that it has driven farther that it really has. This is a cumulative effect, so the farther the distance the greater the effect. The third factor, which is related to wheel slippage, is that the robot does not always drive straight. Sometimes one wheel turns slightly faster than the other. Again, this effect is cumulative over distance, so for short distances it is usually not significant, but can become so over long distances. The distance to the goal for the no-obstacle situation is a measure of all of these factors combined.

To measure the performance of the safe/unsafe system, several measurements were taken. The first, and most important, was how many times the robot collided with obstacles. This measurement tells whether or not the robot is generally effective at avoiding obstacles. The second measurement was how far the robot ended up from the goal originally specified. This measurement gives an indication of how close the robot can get to the specified goal given different levels of complexity in environment and path. The third measurement was how many corrective turns the robot had to make while navigating the situation. This measurement gives a feeling of how complicated the path was and consequently, to a degree, how complicated the environment was.

7.2 Results

Single, Inline Obstacle Situation

	Corrections	Distance to Goal	Collisions
	8	4	0
	3	31	0
	6	22	0
	13	26	0
	12	16	0
	13	28	0
	8	21	0
Avg	9	21.1	0

Single, Offset Obstacle Situation

	Corrections	Distance to Goal	Collisions
	0	12	0
	0	18	0
	0	20	0
	3	10	0
	0	47	0
	0	44	0
	3	13	0
Avg	0.9	23.4	0

Double, Offset Obstacle Situation

	Corrections	Distance to Goal	Collisions
	2	20	0
	2	20	0
	2	26	0
	3	24	0
	1	30	0
	1	27	0
	1	26	0
Avg	1.7	24.7	0

Double, Inline Obstacle Situation

	Corrections	Distance to Goal	Collisions
	4	36	1
	6	30	0
	2	42	0
	5	43	0
	7	30	0
	12	26	0
	6	26	0
Avg	6	33.3	0.1

Figure 37 – Results of four main situations. Total distance for all situations is 252 inches. Distances are in inches.

As can be seen in figure 37, on all the runs of all the situations the robot only collided with one obstacle on one run. This was on one of the runs of situation 4, which is arguably one of the hardest situations. There were no collisions on any other runs of fourth situation or on any runs of any of the other situations. For the primary goal of the safe/unsafe system of avoiding obstacles, this qualifies as a success.

Also shown in figure 37, in the first through third situations the average distance to the specified goal was about twenty inches. In the fourth situation the average distance to the goal was slightly higher at about thirty inches. However, the fourth situation is one of the more complex situations, so slightly less accuracy is to be expected. By comparing figure 36 and figure 37, it can be noted that for the first through third situations, the safe/unsafe system actually got closer to the desired goal location than simply driving

straight to the goal. This result makes sense if the inaccuracy factors discussed in the description of the no-obstacle situation are recalled. Several of the inherent inaccuracies of the system are cumulative over distance. In the no-obstacle situation, the robot had to travel the greatest uninterrupted distance of any of the situations, so the effects of the cumulative inaccuracies were most pronounced in this situation. In the other situations, the total distance was divided up into shorter distances (between corrective turns) so the effect of these inaccuracies was less when using the safe/unsafe system. The distance to the goal in the fourth situation was greater than in the no-obstacle situation, but not by a significant amount. It can be concluded that the accuracy of the system, in terms of reaching the goal location, is generally not worse when using safe/unsafe and can actually be better.

The average number of corrections increases with the complexity of the situation. The only notable exception to this is in the single, inline obstacle situation. More corrective turns were issued in this situation on average than any other situation even though it is not necessarily the most complex environment. Since the system was able to avoid the obstacle successfully and the final distance to the goal specified was acceptable, the increased number of corrections in this situation is not a significant issue. If fewer corrective turns were desired the system could be adjusted.

Chapter 8 – Conclusion

This thesis has detailed the workings and evaluation of the safe/unsafe system, the user interface of which is shown in figure 31. This system uses interactive machine learning to get input from the user, which is used to learn to detect obstacles. User input can be obtained in real time as new environment elements are encountered. The system then uses information about the obstacles to perform effective obstacle avoidance and path following.

The trainable obstacle detection of the system uses a relatively large number of features enabling the system to be used in a wide variety of circumstances. Due to the use of decision trees and integral images, the system can use this large number of features and still have fast training times. And since only the features that are actually used are computed during classification the system is able to remain interactive.

The system uses the safe/unsafe classification to generate a map of the robot's environment. This environment map is generated and updated based on dynamic classifications. The map results in useful data that can be used to effectively avoid obstacles while traveling to a goal specified by the user.

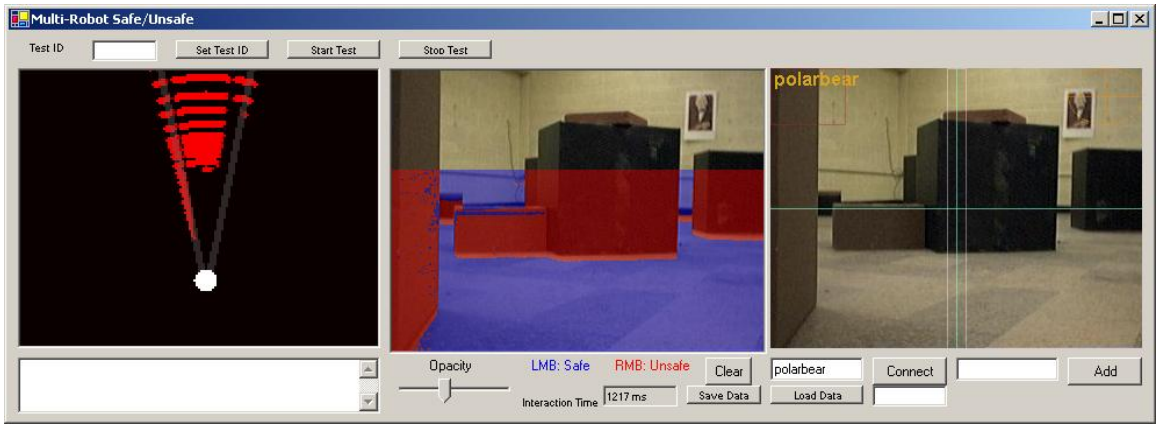

Figure 38 – User interface of the safe/unsafe system showing all three main components (navigation, safe/unsafe specification and environment mapping).

There are many possible areas of further study, based on the work presented in this thesis. A few are listed here. One possible area would be in the use of multiple robots. The ability to control multiple robots and have them share classification information was implemented in the safe/unsafe system but was not tested. The sharing of classification information between robots could result in more robust classifiers. The ability to share classification information might also be more effective, in that training one robot would train all the others, thus making the training of multiple robots less time consuming.

A second possible area of continued study would be the fusing of the obstacle detection portion of the safe/unsafe system with other sensor systems such as sonar. Using two types of sensors could allow for more accurate environment maps and better obstacle avoidance. However, a way to combine the two (or more) sensor systems in an effective manner would need to be developed.

A third area of continued study would be to user test the safe/unsafe system. User testing could help determine if the system helps the user more effectively navigate the robot and if so, how much the system helps. If multiple robots are being tested then fanout tests [OLSE04] [CRAN05] could be used to evaluate how the system affects the user's ability to control multiple robots.

A fourth area would be to add to the system the ability to recognize desirable objects. The system currently only identifies two possible classifications: safe and unsafe. Since the learning algorithm used can use an arbitrary number of possible classifications, a third classification could easily be added. This classification could be used to identify objects that are visually distinct from the rest of the robot's environment that the user would like to be alerted to when the robot sees them. A possible example of this might be a robot performing search and rescue in a forest. Obviously, the trees should be classified as unsafe and the forest floor should be classified as safe. The third classification could be used if it is known that the person being searched for is wearing a red shirt. When a red shirt is seen by the robot, the user could be alerted (audio or visual alert through the user interface) that the robot has seen something that looks like what is being searched for.

A fifth area in which research could continue would be to use alternate map generation techniques. One possibility is in the use of different path planning algorithms. As stated previously, there are many path-planning algorithms. The safe/unsafe system was implemented using a generic best-first search with two values combined (distance to the goal and unsafeness level) to form a single heuristic. The system might benefit from the use of other path-planning algorithms. It is also possible that some algorithms might

work better in some situations, while others work better in other situations. Another possibility would be to use a more sophisticated map generation algorithm, such as particle filters [THRU01].

Traditional robot control systems have limitations. The safe/unsafe system solves some of those problems. It allows a robot to detect obstacles based solely on visual information, thus removing the limitation of only being able to detect "positive space" obstacles. The system is also easily user trainable. This allows the system to be used in a wide variety and situations and to adapt to new situations quickly. Finally, the safe/unsafe system has been shown to be effective at avoiding obstacles while traveling to a user specified goal.

Bibliography

[BAIN83] L. Bainbridge, *Ironies of Automation*, Dept. of Psychology, University College London, 1983

[BORE89] J. Borenstein and Y. Koren, *Real-time Obstacle Avoidance for Fast Mobile Robots*, IEEE Transactions on Systems, Man, and Cybernetics, vol. 19, no. 5, pg. 1179-1187 September/October, 1989

[BORE91] J. Borenstein and Y. Koren, *The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots*, IEEE Journal of Robotics and Automation, vol. 7, no. 3, pg. 278-288, June, 1991

[BRAI04] M. Brain, *How the Mars Exploration Rovers Work*, HowStuffWork, http://science.howstuffworks.com/mars-rover3.htm, 2004

[BRUE03] D. J. Bruemmer, J. L. Marble, D. D. Dudenhoeffer, M. O. Anderson, M. D. McKay, *Mixed-Initiative Control for Remote Characterization of Hazardous Environments*, Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS '03) – Track 5 – Volume 5, 2003

[BRUE04] D. J. Bruemmer, R. L. Boring, D. A. Few, J. L. Marble, M. C. Walton, *"I Call Shotgun!": An Evaluation of Mixed-Initiative Control for Novice Users of a Search and Rescue Robot*, 2004 IEEE International Conference on Systems, Man and Cybernetics, Volume 3, pg. 2847-2852, October, 2004

[CHAN99] P. Chan and G. Wyeth, *Self-Learning Visual Path Recognition*, Proceeding of the Australian Conference on Robotics and Automation, Brisbane, 44-49, 1999

[CRAN02] J. W. Crandall and M. A. Goodrich, *Characterizing Efficiency of Human Robot Interaction: A Case Study of Shared-Control Teleoperation*, IROS '02, 2002

[CRAN05] J. W. Crandall, M. A. Goodrich, D. R. Olsen Jr., C. W. Nielsen, *Validating Human-Robot Interaction Schemes in Multi-Tasking Environments*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 35, Issue 4, pg. 438-449, July, 2005

[DAVI95] A. J. Davison, I. D. Reid, D. W. Murray, *The active camera as a projective pointing device*, Proc. 6th British Machine Vision Conference, 1995

[DAVI99] A. J. Davison, *Mobile Robot Navigation Using Active Vision*, PhD Thesis, Robotics Research Group, University of Oxford, 1999. Available at http://www.robots.ox.ac.uk/~ajd

[DRUR03] J. Drury, L. D. Riek, A. D. Christiansen, Z. T. Eyler-Walker, A. J. Maggi and D. B. Smith, *Evaluating Human-Robot Interaction in a Search-and-Rescue Context*, The MITRE Corporation, 2003

[FAIL03] J. A. Fails and D. R. Olsen, *A Design Tool for Camera-based Interaction*, CHI '03, 2003

[FONG98] T. Fong, *Collaborative Control: A Robot-Centric Model for Vehicle Teleoperation*, Carnegie Mellon University, Pittsburgh, PA, January, 1998

[JENS99] P. Jensfelt and H. I. Christensen, *Laser Based Pose Tracking*, IEEE Intl. Conference on Robotics and Automation, May, 1999

[LANG99] S. Lang, F. Yili and S. K. Tso, *Visual Correction of Orientation Error for a Mobile Robot*, IEEE International Conference on Intelligent Robots and Systems, 1999

[LECU05] Y. LeCun, U. Muller, J. Ben, E. Cosatto and B. Flepp, *Off-Road Obstacle Avoidance through End-to-End Learning*, NIPS 2005

[LEVI99] S. P. Levine, D. A. Bell, L. A. Jaros, R. C. Simpson, Y. Koren and J. Borenstein, *The NavChair Assistive Wheelchair Navigation System*, IEEE Transactions on Rehabilitation Engineering, vol. 7, no. 4, pg. 443-451, December, 1999

[MING00] J. Minguez and L. Montano, *Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach*, Internal Report RR-00-14 University of Zaragoza, pg. 60, February, 2000

[MITC97] T. M. Mitchell, *Machine Learning*, WCB/McGraw-Hill, ISBN 0-07-042807-7, 1997

[MITR05] S. Mitri, S. Frintrop, K. Pervölz, H. Surmann, A. Nüchter, *Robust Object Detection at Regions of Interest with an Application in Ball Recognition*, 2005

[MURP96] R. R. Murphy and E. Rogers, *Cooperative Assistance for Remote Robot Supervision*, Presence, 5(2): 224-240, 1996

[MURR96] D. W. Murray, I. D. Reid and A. J. Davison, *Steering and Navigation Behaviours using Fixation*, Proceedings of the 7th British Machine Vision Conference, pg. 634-644, 1996

[NIEL06] C. W. Nielsen and M. A. Goodrich, *Comparing the Usefulness of Video and Map Information in Navigation Tasks*, Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, pg. 95-101, 2006

[OBER04] J. Oberg, *Bringing Space Costs back down to Earth*, MSNBC, http://www.msnbc.msn.com/id/4031857/, 2004

[OHYA98] A. Ohya, A. Kosaka and A. Kak, *Vision-Based Navigation by a Mobile Robot with Obstacle Avoidance Using Single-Camera Vision and Ultrasonic Sensing*, IEEE Transactions on Robotics and Automantion, vol. 15, no. 6, pg. 969-978, December, 1998

[OLSE04] D. R. Olsen, S. B. Wood and J. Turner, *Metrics for Human Driving of Multiple Robots*, International Conference on Robotics and Automation, April, 2004

[PAPA00] C. Papageorgiou and T. Poggio, *A Trainable System for Object Detection*, International Journal of Computer Vision, vol. 38, no. 1, pg. 15-33, 2000

[PEAR84] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, pg. 48, 1984

[ROFE99] T. Rofer and A. Lankenau, *Ensuring Safe Obstacle Avoidance in a Shared-Control System*, in J. M. Fuertes (Ed.), Proc. Of the 7th Int. Conf. On Emergent Technologies and Factory Automation, pg. 1405-1414, 1999

[ROSE95] J. Rosenblatt, DAMN: A Distributed Architecture for Mobile Navigation, Thesis Summary, The Robotics Institute, Carnegie Mellon University, 1995

[RUSS03] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd Edition*, Pearson Education, Inc., 2003

[STEN02] R. Stenger, *Eight-eyed robot blasts off for Mars*, CNN, http://www.cnn.com/2003/TECH/space/06/10/mars.rover/index.html, 2002

[THRU01] S. Thrun, *Is Robotics Going Statistics? The Field of Probabilistic Robotics*, Journal of the ACM, Carnegie Mellon University, March, 2001

[VIOL01] P. Viola and M. Jones, *Robust Real-time Object Detection*, Second International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing, and Sampling, July 2001

[YANC05] H. A. Yanco, M. Baker, R. Casey, A. Chanler, M. Desai, D. Hestand, B. Keyes and P. Thoren, *Improving Human-Robot Interaction for Remote Robot Operation*, University of Massachusetts Lowell, 2005

[ZHAN92] Z. Zhang, *Iterative Point Matching for Registration of Free-Form Curves*, International Journal of Computer Vision, vol. 13, no. 2, pg. 119-152, October, 1992

Appendix – List of Features

All features are generated for a specific pixel. For the purposes of describing these features, the pixel in question will be referred to by having coordinates of *x* and *y*.

Basic Features

R1 – Red value at *(x, y)*
G1 – Green value at *(x, y)*
B1 – Blue value at (*x, y)*
R3 – Sum of all red values in a 3x3 area with *(x, y)* at its center
G3 – Sum of all green values in a 3x3 area with *(x, y)* at its center
B3 – Sum of all blue values in a 3x3 area with *(x, y)* at its center
R9 – Sum of all red values in a 9x9 area with *(x, y)* at its center
G9 – Sum of all green values in a 9x9 area with *(x, y)* at its center
B9 – Sum of all blue values in a 9x9 area with *(x, y)* at its center
R27 – Sum of all red values in a 27x27 area with *(x, y)* at its center
G27 – Sum of all green values in a 27x27 area with *(x, y)* at its center
B27 – Sum of all blue values in a 27x27 area with *(x, y)* at its center
R81 – Sum of all red values in an 81x81 area with *(x, y)* at its center
G81 – Sum of all green values in an 81x81 area with *(x, y)* at its center
B81 – Sum of all blue values in an 81x81 area with *(x, y)* at its center
A1 – Max of R1, G1 and B1
A3 – Max of R3, G3, and B3
A9 – Max of R9, G9 and B9
A27 – Max of R27, G27 and B27
A81 – Max of R81, G81 and B81
I1 – Min of R1, G1 and B1
I3 – Min of R3, G3 and B3
I9 – Min of R9, G9 and B9
I27 – Min of R27, G27 and B27
I81 – Min of R81, G81 and B81

Averaging Features

C-AVE-*S* – Average *C* (red, green or blue) values over an *S*x*S* area centered on *(x, y)*
Valid values of *S* are 27, 81, 100, 144 and 225 for a total of 15 features

Differencing Features of Same Size

DR1G1 – Difference between R1 and G1
DR3G3 – Difference between R3 and G3
DR9G9 – Difference between R9 and G9
DR27G27 – Difference between R27 andG27
DR81G81 – Difference between R81 and G81

DG1B1 – Difference between G1 and B1
DG3B3 – Difference between G3 and B3
DG9B9 – Difference between G9 and B9
DG27B27 – Difference between G27 and B27
DG81B81 – Difference between G81 and B81
DB1R1 – Difference between B1 and R1
DB3R3 – Difference between B3 and R3
DB9R9 – Difference between B9 and R9
DB27R27 – Difference between B27 and R27
DB81R81 – Difference between B81 and R81
DA1R1 – Difference between A1 and R1
DA3R3 – Difference between A3 and R3
DA9R9 – Difference between A9 and R9
DA27R27 – Difference between A27 and R27
DA81R81 – Difference between A81 and R81
DA1G1 – Difference between A1 and G1
DA3G3 – Difference between A3 and G3
DA9G9 – Difference between A9 and G9
DA27G27 – Difference between A27 and G27
DA81G81 – Difference between A81 and G81
DA1B1 – Difference between A1 and B1
DA3B3 – Difference between A3 and B3
DA9B9 – Difference between A9 and B9
DA27B27 – Difference between A27 and B27
DA81B81 – Difference between A81 and B81
DR1I1 – Difference between R1 and I1
DR3I3 – Difference between R3 and I3
DR9I9 – Difference between R9 and I9
DR27I27 – Difference between R27 and I27
DR81I81 – Difference between R81 and I81
DG1I1 – Difference between G1 and I1
DG3I3 – Difference between G3 and I3
DG9I9 – Difference between G9 and I9
DG27I27 – Difference between G27 and I27
DG81I81 – Difference between G81 and I81
DB1I1 – Difference between B1 and I1
DB3I3 – Difference between B3 and I3
DB9I9 – Difference between B9 and I9
DB27I27 – Difference between B27 and I27
DB81I81 – Difference between B81 and I81

Differencing Features of Different Size

DR1R3 – Difference between R1 and R3
DR1R9 – Difference between R1 and R9
DR1R27 – Difference between R1 and R27

DR1R81 – Difference between R1 and R81
DR3R9 – Difference between R3 and R9
DR3R27 – Difference between R3 and R27
DR3R81 – Difference between R3 and R81
DR9R27 – Difference between R9 and R27
DR9R81 – Difference between R9 and R81
DR27R81 – Difference between R27 and R81
DG1G3 – Difference between G1 and G3
DG1G9 – Difference between G1 and G9
DG1G27 – Difference between G1 and G27
DG1G81 – Difference between G1 and G81
DG3G9 – Difference between G3 and G9
DG3G27 – Difference between G3 and G27
DG3G81 – Difference between G3 and G81
DG9G27 – Difference between G9 and G27
DG9G81 – Difference between G9 and G81
DG27G81 – Difference between G27 and G81
DB1B3 – Difference between B1 and B3
DB1B9 – Difference between B1 and B9
DB1B27 – Difference between B1 and B27
DB1B81 – Difference between B1 and B81
DB3B9 – Difference between B3 and B9
DB3B27 – Difference between B3 and B27
DB3B81 – Difference between B3 and B81
DB9B27 – Difference between B9 and B27
DB9B81 – Difference between B9 and B81
DB27B81 – Difference between B27 and B81
DR1G3 – Difference between R1 and G3
DR1G9 – Difference between R1 and G9
DR1G27 – Difference between R1 and G27
DR1G81 – Difference between R1 and G81
DR3G9 – Difference between R3 and G9
DR3G27 – Difference between R3 and G27
DR3G81 – Difference between R3 and G81
DR9G27 – Difference between R9 and G27
DR9G81 – Difference between R9 and G81
DR27G81 – Difference between R27 and G81
DG1B3 – Difference between G1 and B3
DG1B9 – Difference between G1 and B9
DG1B27 – Difference between G1 and B27
DG1B81 – Difference between G1 and B81
DG3B9 – Difference between G3 and B9
DG3B27 – Difference between G3 and B27
DG3B81 – Difference between G3 and B81
DG9B27 – Difference between G9 and B27
DG9B81 – Difference between G9 and B81

DG27B81 – Difference between G2 and B81
DB1R3 – Difference between B1 and R3
DB1R9 – Difference between B1 and R9
DB1R27 – Difference between B1 and R27
DB1R81 – Difference between B1 and R81
DB3R9 – Difference between B3 and R9
DB3R27 – Difference between B3 and R27
DB3R81 – Difference between B3 and R81
DB9R27 – Difference between B9 and R27
DB9R81 – Difference between B9 and R81
DB27R81 – Difference between B27 and R81
DA3R1 – Difference between A3 and R1
DA9R1 – Difference between A9 and R1
DA27R1 – Difference between A27 and R1
DA81R1 – Difference between A81 and R1
DA9R3 – Difference between A9 and R3
DA27R3 – Difference between A27 and R3
DA81R3 – Difference between A81 and R3
DA27R9 – Difference between A27 and R9
DA81R9 – Difference between A81 and R9
DA81R27 – Difference between A81 and R27
DA3G1 – Difference between A3 and G1
DA9G1 – Difference between A9 and G1
DA27G1 – Difference between A27 and G1
DA81G1 – Difference between A81 and G1
DA9G3 – Difference between A9 and G3
DA27G3 – Difference between A27 and G3
DA81G3 – Difference between A81 and G3
DA27G9 – Difference between A27 and G9
DA81G9 – Difference between A81 and G9
DA81G27 – Difference between A81 and G27
DA3B1 – Difference between A3 and B1
DA9B1 – Difference between A9 and B1
DA27B1 – Difference between A27 and B1
DA81B1 – Difference between A81 and B1
DA9B3 – Difference between A9 and B3
DA27B3 – Difference between A27 and B3
DA81B3 – Difference between A81 and B3
DA27B9 – Difference between A27 and B9
DA81B9 – Difference between A81 and B9
DA81B27 – Difference between A81 and B27
DR1I3 – Difference between R1 and I3
DR1I9 – Difference between R1 and I9
DR1I27 – Difference between R1 and I27
DR1I81 – Difference between R1 and I81
DR3I9 – Difference between R3 and I9

DR3I27 – Difference between R3 and I27
DR3I81 – Difference between R3 and I81
DR9I27 – Difference between R9 and I27
DR9I81 – Difference between R9 and I81
DR27I81 – Difference between R27 and I81
DG1I3 – Difference between G1 and I3
DG1I9 – Difference between G1 and I9
DG1I27 – Difference between G1 and I27
DG1I81 – Difference between G1 and I81
DG3I9 – Difference between G3 and I9
DG3I27 – Difference between G3 and I27
DG3I81 – Difference between G3 and I81
DG9I27 – Difference between G9 and I27
DG9I81 – Difference between G9 and I81
DG27I81 – Difference between G27 and I81
DB1I3 – Difference between B1 and I3
DB1I9 – Difference between B1 and I9
DB1I27 – Difference between B1 and I27
DB1I81 – Difference between B1 and I81
DB3I9 – Difference between B3 and I9
DB3I27 – Difference between B3 and I27
DB3I81 – Difference between B3 and I81
DB9I27 – Difference between B9 and I27
DB9I81 – Difference between B9 and I81
DB27I81 – Difference between R1G1 and I81

Gradient Features (Vertical and Horizontal)

V-*C*-*S* – Sum of *C* (red, green or blue) values from $x - (S / 2)$ to x and $y - (S / 2)$ and $y + (S / 2)$ minus the sum of *C* (red, green or blue) values from x to $x + (S / 2)$ and $y - (S / 2)$ and $y + (S / 2)$
H-*C*-*S* – Sum of *C* (red, green or blue) values from $x - (S / 2)$ to $x + (S / 2)$ and $y - (S / 2)$ and y minus the sum of *C* (red, green or blue) values from $x - (S / 2)$ to $x + (S / 2)$ and y and $y + (S / 2)$
VA-*S* – Max of VR-*S*, VG-*S* and VB-*S*
HA-*S* – Max of HR-*S*, HG-*S* and HB-*S*
Valid values of *S* are 2, 4, 8, 16, 32 and 64 for a total of 48 features

Differenced Averages

DRA27 – Difference between RAVE27 and average of the red values in a 9x9 area centered on $(x, y - 10)$
DRA81 – Difference between RAVE81 and average of the red values in a 27x27 area centered on $(x, y - 40)$
DRA100 – Difference between RAVE100 and average of the red values in a 27x27 area centered on $(x, y - 50)$

DRA144 – Difference between RAVE144 and average of the red values in a 27x27 area centered on *(x, y – 72)*
DRA225 – Difference between RAVE225 and average of the red values in a 27x27 area centered on *(x, y – 112)*
DGA27 – Difference between GAVE27 and average of the red values in a 9x9 area centered on *(x, y – 10)*
DGA81 – Difference between GAVE81 and average of the red values in a 27x27 area centered on *(x, y – 40)*
DGA100 – Difference between GAVE100 and average of the red values in a 27x27 area centered on *(x, y – 50)*
DGA144 – Difference between GAVE144 and average of the red values in a 27x27 area centered on *(x, y – 72)*
DGA225 – Difference between GAVE225 and average of the red values in a 27x27 area centered on *(x, y – 112)*
DBA27 – Difference between BAVE27 and average of the red values in a 9x9 area centered on *(x, y – 10)*
DBA81 – Difference between BAVE81 and average of the red values in a 27x27 area centered on *(x, y – 40)*
DBA100 – Difference between BAVE100 and average of the red values in a 27x27 area centered on *(x, y – 50)*
DBA144 – Difference between BAVE144 and average of the red values in a 27x27 area centered on *(x, y – 72)*
DBA225 – Difference between BAVE225 and average of the red values in a 27x27 area centered on *(x, y – 112)*
DAVER225G225 – Difference between RAVE225 and GAVE225
DAVER225B225 – Difference between RAVE225 and BAVE225
DAVEG225B225 – Difference between GAVE225 and BAVE225
DAVER144G144 – Difference between RAVE144 and GAVE144
DAVER144B144 – Difference between RAVE144 and BAVE144
DAVEG144B144 – Difference between GAVE144 and BAVE144
DAVER100G100 – Difference between RAVE100 and GAVE100
DAVER100B100 – Difference between RAVE100 and BAVE100
DAVEG100B100 – Difference between GAVE100 and BAVE100
DAVER81G81 – Difference between RAVE81 and GAVE81
DAVER81B81 – Difference between RAVE81 and BAVE81
DAVEG81B81 – Difference between GAVE81 and BAVE81
DAVER27G27 – Difference between RAVE27 and GAVE27
DAVER27B27 – Difference between RAVE27 and BAVE27
DAVEG27B27 – Difference between GAVE27 and BAVE27