1-1-2010

Robotic Goal-Based Semi-Autonomous Algorithms Improve Remote Operator Performance

Shawn Hunt
Wayne State University

ROBOTIC GOAL-BASED SEMI-AUTONOMOUS ALGORITHMS IMPROVE REMOTE OPERATOR PERFORMANCE

by

SHAWN HUNT

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2010

MAJOR: COMPUTER ENGINEERING

Approved by:

_____

Advisor Date

_____

_____

_____

_____

Dedication

This is dedicated to my wife for supporting me for all of the years that I have been in school.

Acknowledgments

First and foremost, I owe a debt that I will never be able to repay to Dr. Witus for giving me a chance to get started with military robots and supporting me all of these years. This dissertation would not have been possible without the thoughtful guidance from my co-advisors, Drs. Pandya and Ellis. This also would not have been possible without funding in one form or another from TARDEC and the US Army. I would also like to thank Dr. Auner for allowing me to be part of the SSIM and CARES groups.

I would like to thank Lavie Goldenberg for his selfless work on making sure the human-in-the-loop experiment was properly set up, his assistance in reviewing this dissertation, and for his friendship these past years. The contributions of my colleagues from the CARES lab and teammates from the Intelligent Ground Vehicle Competition group must also be mentioned: Wilfred Wheeler, Justin Ar-Rasheed, and Michael Jessie: thank you all very much for helping in the pilot and subject testing. Ben Wojcik, thank you for your help in getting the subject testing started and assisting me to work all of the kinks out.

Lynn DeTurk, thank you for all of your hard work and time spent reading this dissertation catching all of my spelling and grammatical errors. Sam Lee, thank you for your willingness to help me these past years. We would not have had an IGVC team had it not been for your hard work and dedication. And last but not least, Syed Ali, I doubt that I would have been able to get this done

without your help. You were my study partner for qualifiers and the best possible friend a guy could have these past years in graduate school.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Statement

Mobile robots, or Unmanned Ground Vehicles (UGVs), play an increasing role in both the defense and security of our nation and in the ability to respond to emergency situations. Robots have been used in Iraq and Afghanistan for bomb disposal. They also played a key role in searching for victims of the World Trade Center attack. They were created to keep our soldiers, or warfighters, out of harm's way.

The current method of UGV control is rate control teleoperation, is burdensome. Figure 1 depicts the current way the robots are controlled. There is a high workload that requires constant attention and limits situational awareness [1]. A dedicated operator is not able to perform multiple tasks and control of the UGV can be difficult when the terrain is rough or communications are degraded.

These robots are currently being used in countries where there is an ongoing war. Those who have attempted to view a laptop's display while out on a sunny day can attest to how difficult it can be to view the contents of the screen. Add to that scenario a stressful situation of using the laptop trying to find a bomb buried in the soil and that paints a vivid portrait of why this research is needed and important to the Army. If all the warfighter has to do is designate a

point for the robot to go to and they know that it will go to that point reliably, then their job becomes easier.

Figure 1: The current method of controlling the military robots is with the operator looking at a laptop while driving the robot using a gamepad.

Robots have been in the news in recent months due to the BP oil spill in the Gulf of Mexico [2]. The robots were remotely controlled by BP personnel to try to cap the damaged oil well. BP ran into a setback to their containment efforts when a saw blade the robot was using became stuck [3].

The motivation behind this work was to provide a level of autonomy to existing robots used in the field so that operating a robot does not require constant supervision. The costs associated with developing fully autonomous

systems may potentially outweigh the benefits [4]. The recent series of DARPA Grand Challenges prove that fully autonomous robotic systems are indeed possible but technology that creates autonomous systems has at the same time also created unwelcome “automation surprises” [5]. DARPA is the military’s research organization. It stands for Defense Advanced Research Projects Agency.

The Three Mile Island disaster in 1979 was caused by a system functioning on its own, attempting to compensate for a stuck valve. The operators of the nuclear power plant did not have sufficient time to act before the automated system transferred control to them in order to avert the disaster. The same problem occurs in the auto-pilot control in airplanes. If there are any problems with the system, they are often not communicated to their human operators in sufficient time to take proper action prior to system failure.

Situational awareness (SA) is also an important area of study, and although it isn’t studied in-depth in this research, the work developed here provides a framework to study the effects that the semi-autonomous algorithms described in this thesis have on situational awareness. In [6, 7], Endsley broke SA into three levels: 1) being able to perceive elements in the environment, 2) understanding what all of the elements mean, and 3) being able to project their status in the future.

The two semi-autonomous algorithms that are the focus of this dissertation are visual servoing and visual dead reckoning. They are both explained in more detail in Chapter 2 but in succinct terms for now, visual

servoing means using image data obtained from a camera to control a robot and visual dead reckoning uses internal properties of the camera combined with sensor data that tells the robot its current position based on its previous position.

The main research question of this work has been to determine if reliable goal-based semi-autonomous algorithms are able to improve remote operator performance. The main research questions are:

1) Can a semi-autonomous algorithm be developed that improves performance in a measurable way?
2) Which tracking algorithms for visual servoing have the best performance?
3) Can these algorithms be implemented on an existing military robot?
4) Once implemented, does the use of the algorithms improve the operator's performance and if so, by how much?

Beyond the research questions of this work, in order for this to have wide acceptance by warfighters in the field, the system has to be easy to use and easy to learn. It should require less mental workload with it than without it. It should not require constant attention. It should be able to be given a destination and the operator knows that it will go there without fail. If it is burdensome to use, it will not be an acceptable form of control.

1.2 Research Objective and Specific Aims

The research objectives in this dissertation were developed to try and determine if goal-based visual servoing improves operator performance. To

support that research question, three aims were formulated: The first aim was the development and analysis of a tracking algorithm that reliably tracks features in real-time; the second aim was the development of a testbed in order to run experiments; the third, and final, aim was the development and execution of a robust subject test. Each aim and how it relates to the research objective are described later in this dissertation. Figure 2 shows the broad overview of the aims that are a part of this research. The overall goal was to create a method of semi-autonomously controlling a robot and determining if the developed method improves operator performance or not.

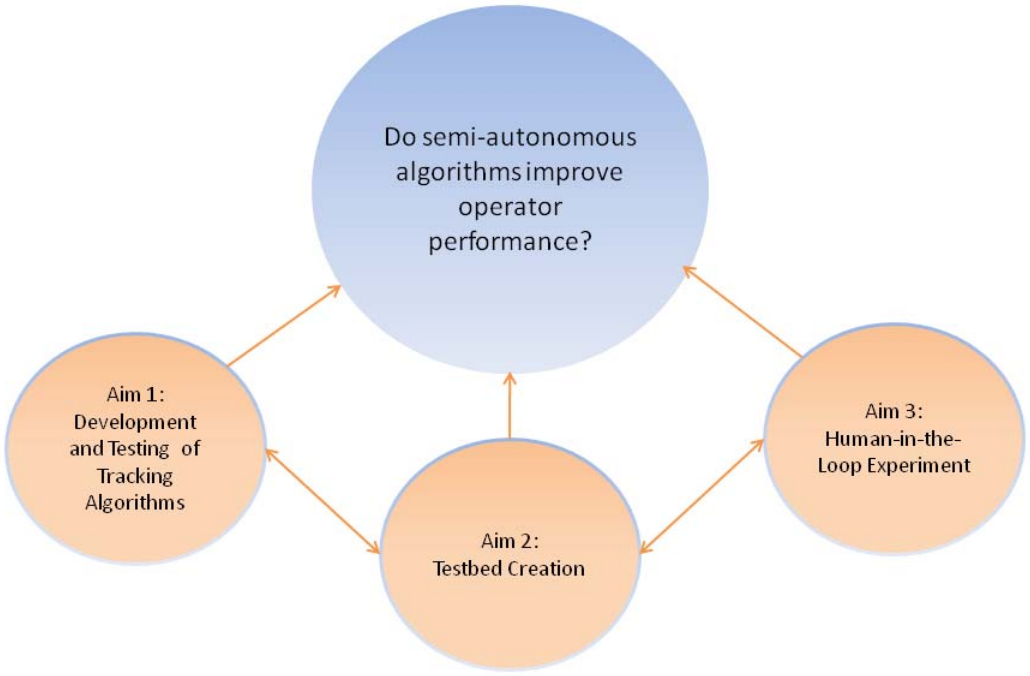

Figure 2: The main research question and the three specific aims of this dissertation.

1.3 Outline of this Dissertation

The second chapter of this dissertation provides background information on the hardware used and the software that was developed in creating a testbed system for evaluating user performance. The third chapter details the algorithms used for the visual servoing algorithm development. The fourth chapter specifies the implementation of the control algorithms that result in moving the robot based on inputs from the user. The fifth chapter describes the human-in-the-loop testing that was performed and the results of the user study that were used to determine the tradeoffs between semi-autonomous algorithms and full teleoperation perform better than teleoperation. The sixth chapter concludes with a summary of the key contributions of this work and the applications that have successfully used the algorithms developed in this research along with an analysis of future directions and extensions that would enhance this system.

CHAPTER 2

BACKGROUND

2.1 Introduction

This chapter is meant to give a brief overview and literature survey of the robots, semi-autonomous algorithms, and typical sensors that form the basis of the development of this work. Section 2.2 describes military robots in more detail. Section 2.3 describes what sensors they are typically equipped with: cameras, encoders, and Inertial Measurement Units (IMUs). Finally, Section 2.4 gives a brief overview of the algorithms that are used in this work.

2.2 Military Robots

The military has been embracing the use of robotics in recent years to help keep warfighters out of harm's way. In 2007, the Department of Defense released a roadmap [8] for the next 25 years, detailing its paradigm shift in fighting wars with robots. The roadmap report also elaborated on a series of goals that the Department of Defense wants to achieve for its unmanned systems. These goals include:

- Improving the overall effectiveness of the unmanned systems through collaboration
- Achieving greater commonality and interoperability of unmanned systems
- Developing standards that support the safe operation and integration with the manned systems

- Using rapid prototyping and deployment to get the technology out to the warfighters as quickly as possible

In 2007, iRobot was awarded a contract by the US Army to deliver up to 3,000 unmanned ground vehicles for wide-scale deployment [9]. This marked a major change in how the Army had typically purchased robots. The contract award was given the generically named “xBot” but what iRobot delivered were PackBot 510s, the same chassis that was the focus of this research.

Robots in the military are used in reconnaissance and surveillance, target identification and designation, counter-mine warfare, and detection of chemical, biological, nuclear, and explosive agents. These robots may either be Unmanned Aerial Vehicles (UAVs) or Unmanned Ground Vehicles (UGVs). The focus of this dissertation is on UGVs, specifically, a class known as Man-Transportable Robotics Systems (MTRS).

Teleoperation is the current method of control for the MTRS platform. In teleoperation, the operator controls the translation and rotation rates using a joystick. The operator remains in the control loop at all times, which requires constant attention. If there are degraded communications, due to interference, jamming, or non-line of sight, teleoperation performance may become impaired and increase the difficulty for the operator.

2.3 Robotic Sensors

This section gives a brief overview of the sensors that are typically used on UGVs. Most robots manufactured already have the sensors next described

built in. These sensors are described now because the work described in this dissertation makes use of each of the following.

2.3.1 Cameras

Robots typically are equipped with one or more cameras that transmit the video feed back to the human operator. Typically the video feed that is received at the Operator Control Unit (OCU) is compressed and may appear degraded because of blocking artifacts. A blocking, or compression artifact, is the noticeable distortion that images can take when the compression algorithm discards data to reduce the amount of space it occupies. In the case of robots, the video feed is compressed to reduce its size and the required bandwidth to transmit it as it goes over the chosen communications protocol.

The effect of blocking artifacts is Figure 3. The left image shows an image that was captured from an NTSC camera. An NTSC camera is analog and in order for it to be processed on a computer, the signal has to be converted to digital. This is usually done with a device called a “framegrabber”. The right image in Figure 3 is the same image but with 90% JPEG compression applied.

Figure 4 shows the histograms of the images in Figure 3. There is a measureable loss of detail between the uncompressed and compressed images that is able to be seen in the histograms. This is an extreme example of compression for illustration purposes but this highlights the fact that many details are lost with compression. This lack of variation between the two images is problematic when trying to track features from frame to frame.

Figure 3: The image on the left is an image obtained directly from an analog camera. The image on the right is the same image with 90% JPEG compression applied.

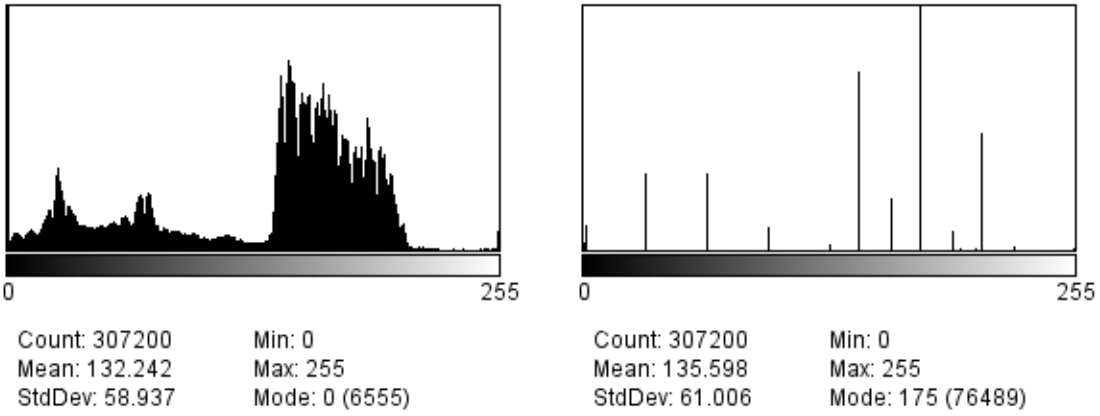

Figure 4: These are the histograms of the same two images in Figure 3.

There are two important internal properties of cameras that will become significant later on, when describing the algorithms that were implemented. The first is the Field of View (FOV). The FOV is the angular extent that the world may be seen at any given time. The FOV is calculated both in vertical and horizontal directions. The Instantaneous Field of View (IFOV) is defined as “the angle

subtended by a single detector element on the axis of the optical system" [10], or in simpler terms, it is the radians per pixel.

2.3.2 Encoders

Encoders are sensors that measure rotation. The measurement of rotation allows the calculation of displacement, velocity, and acceleration of the object they are on, such as a wheel or a motor. Typically, encoders use optical sensors along with a special reflector that provide electrical pulses to a microcontroller.

Assuming a differentially steered robot, the calculation to find the robot's location is a simple calculation [11]. First, the current distance the robot has travelled and its current heading are calculated. The equation to calculate distance travelled is shown in (2.1), where *l* is the left encoder and *r* is the right encoder. The equation to calculate θ, or the robot's heading is shown in (2.2), l is the left encoder, r is the right encoder, and w is the wheel base.

$$\frac{l+r}{2} \tag{2.1}$$

$$\frac{l+r}{w} \tag{2.2}$$

Using the distance and heading calculations, the robot's position in 2D Cartesian space may also be easily found. The robot's X position is given by (2.3) and the robot's Y position is given by (2.4). In each equation, *d* is the distance travelled.

$$x = d * \sin(\vartheta) \tag{2.3}$$

$$y = d * \cos(\vartheta) \tag{2.4}$$

As the robot moves through its environment, its calculated position using odometry drifts over time due to wheel slippage and uneven terrain. This is exactly what happens when a car is suck in the snow, for example, the wheels are turning but the vehicle is not moving. The largest error is typically with theta, or the heading of the robot. The equation to find the error in heading is shown in (2.5), where *l* and *r* are the left and right encoder values and *b* is the wheel base. Figure 5 depicts such an error in theta. If the robot's goal point was initially designated as the green dot and either slippage or rough terrain caused the odometry to be off, then it is easy for the robot to get off course and end up at the red dot's location.

$$\frac{l - r}{b} \tag{2.5}$$

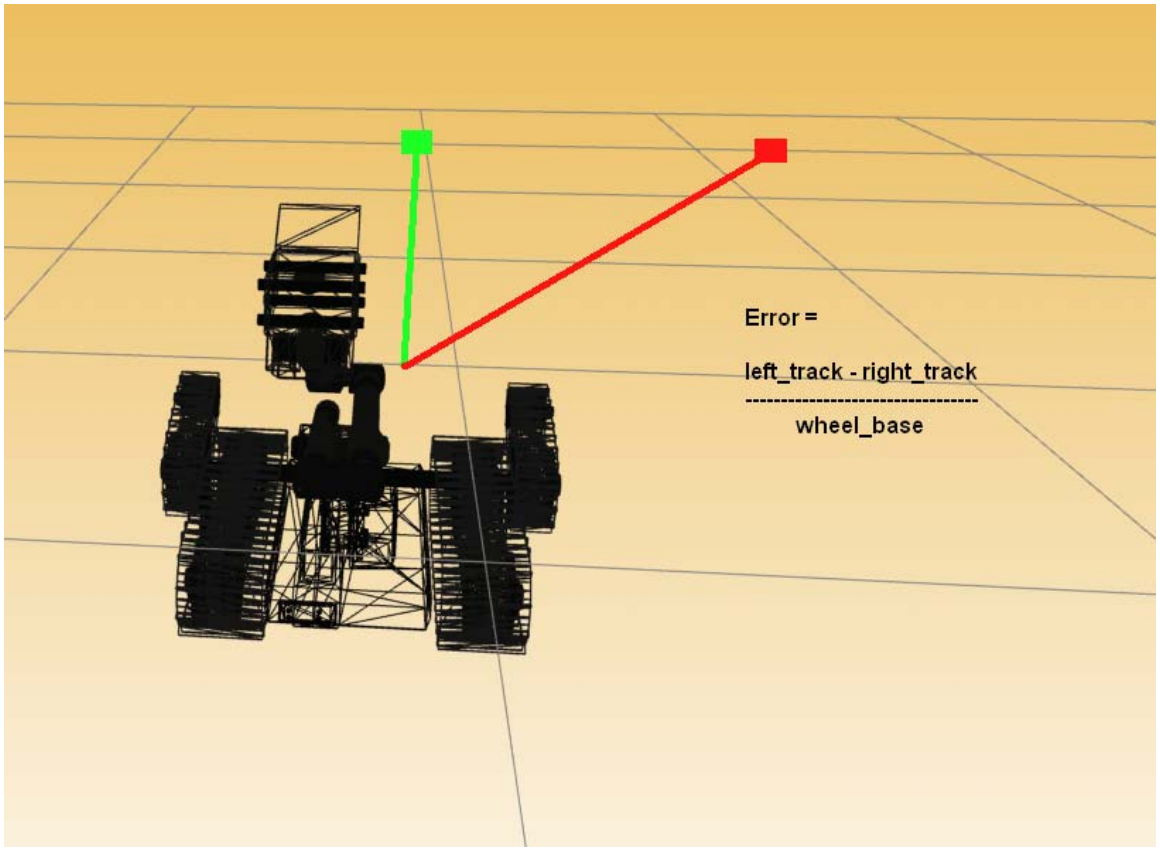

Figure 5: A visualization of small errors in theta that result in large errors in X and Y. The green dot is where the robot would go if no slippage occurs and the red dot is where it could end up if odometry is incorrect.

2.3.3 Inertial Measurement Units

An Inertial Measurement Unit (IMU) is a sensor that is able to collect and report the angular velocity and acceleration of a moving object. It is able to do this by using two separate sensors. The first sensor is a group of three accelerometers, one for each axis. The second sensor is a group of three angular rate sensors called gyroscopes. This configuration is able to report the six degrees-of-freedom of the object it is placed on. In this work, the heading from the IMU was integrated with the existing encoder feedback from the robot.

This was as simple as reading the value from the IMU, scaling it, and substituting the theta value as calculated from the encoders. This will be discussed in more detail in Chapter 4.

2.4 Semi-Autonomous Algorithms

Visual servoing [12] is simply the name given for using data captured from a camera to control the motion of a robot using computer vision techniques. The first papers published on visual servoing date back to the 1970s [13]. This has grown into a very large field of study [14] with many papers published. The papers have traditionally fallen into two broad categories: 2D, or image based (IB) [15], and 3D, or position based (PB) [16].

In position based control, image features are extracted and a model of the scene image features is used to estimate the pose of the target with respect to the camera using a geometric model of the target [17]. This approach is typically referred to as 3D visual servoing in literature. This method requires precise calibration of the camera for it to be accurate.

Camera calibration [16] is the process of finding the camera parameters that affect the imaging process. Intrinsic camera parameters do not change for a particular camera-lens combination. Intrinsic camera parameters include the exact center of the image, the focal length, the lens distortion, and the scaling factors that are used for row and column pixels. The extrinsic camera parameters describe the camera's pose, or its position and orientation, in the world coordinate system. In [18], a methodology was published for autonomously calibrating a camera. Once the intrinsic and extrinsic camera

parameters have been found, the pose of the camera in the workspace is able to be computed.

The second class of visual servoing algorithms is image-based [19]. In image-based visual servoing, the pose estimate is omitted [20] and the motion control is done solely in image space. There has also been work published on “2-1/2D” visual servoing [21] that bridges the two groups by trying to minimize the errors in the image and pose space.

Dead reckoning [22] has its roots outside the realm of robotics but it is basically estimating one’s current position based on a previously determined position and advancing that position based on known speeds over time. Dead reckoning has been shown to be used in nature [23]. Dead reckoning has also been shown to be used in marine, air, and automotive navigation and it has even been proven to be successful in predicting latency and reducing its impact on networked games [24]. Dead reckoning has been used to control the Mars rover robots [25, 26]. The implementation of “visual dead reckoning” as it pertains to this research is described in more detail in Chapter 4.

CHAPTER 3

IMPLEMENTATION AND TESTING OF TRACKING ALGORITHMS

3.1 Introduction

This chapter focuses on tracking algorithms. It starts by giving a broad overview of computer vision and also a literature survey is presented describing past work in detecting and tracking features. The approach used to track a goal point anywhere in the image using any tracking algorithm is discussed along with the approach used to determine which algorithms performed the best using defined metrics.

3.2 Literature Review

The literature on tracking a point, or multiple points, through a series of images is vast because there are as nearly as many different approaches to tracking as there are applications. The basic component of almost any tracking algorithm is feature detection and matching. There is, however, no universal definition of what constitutes a feature and beyond that, a feature that works well in one algorithm might not work well in another. Applying a filter, either in the spatial or frequency domain, to the input image may help one algorithm while not having any effect on another or may possibly render the algorithm ineffective.

A feature may be loosely defined as an “interesting” part of an image; something able to be located from frame to frame. This definition is intentionally vague because there are an abundance of feature detectors that have been published over the years. That language is also intentionally vague because the

feature depends on the algorithm and the algorithm's purpose. The most common types of features found in the literature are edges, corners, and blobs. These three types are the focus of the algorithms discussed in this chapter.

Edge detection is a method used in image processing to detect discontinuities in intensity and literature dates back to the 1970s [27]. The two main areas of study to find edges that have been apparent over this time period are template matching and the differential gradient approach. The goal of either approach is to locate where the gradient magnitude *g* is sufficiently large to indicate an edge.

Both the template matching and differential gradient approaches locate the intensity gradients using convolution masks. Convolution is a mathematical operation that is fundamental to image processing and computer vision as well as other areas of science. It is a way to multiply two arrays of numbers, which typically have different sizes but the same dimensionality, to output an array of the same dimensionality. Figure 6 shows a small example image on the left and an example convolution kernel on the right. When a convolution is used, it is typically done by sliding the convolution kernel, or mask, over the image, usually starting at the top left corner and moving it where to where it fits within the image boundaries. For example, the output of the image at pixel location I_{35} convolved by the mask would be: $O_{35} = I_{35}K_{11} + I_{36}K_{12} + I_{37}K_{13} + I_{45}K_{21} + I_{46}K_{22} + I_{47}K_{23}$.

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

Figure 6: A small example image (left) and a convolution kernel (right).

Both the differential gradient and template matching algorithms make use of convolution masks. The differential gradient algorithm uses two masks, one for the x direction and another for the y direction. The template matching algorithms can use up to 12. The Sobel operator [28] is a well-known template matching algorithm for edge detection. The convolution masks for a 3x3 Sobel operator for x and y are shown in (3.1) and (3.2). Figure 7 shows the results of the Sobel operator running on an input image.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.2}$$

Figure 7: The results of the Sobel operator running on an image.

Many trackable features tend to be corners. A commonly used corner detection was developed by Harris [29]. The basic idea used in this method is to locate points where the surrounding neighborhood shows edges in more than one direction.

In [30], Shi and Tomasi published an enhancement to the work done by Harris. They determined that a feature was good as long as the smaller of the two eigenvalues was greater than a minimum threshold [31]. OpenCV [32] is an open source computer vision library initially developed by Intel. It contains an implementation of both the Shi and Tomasi algorithm and the Harris corner detector. Figure 8 shows the results of running Shi and Tomasi algorithm on an image taken from within the lab.

Figure 8: The "good features" to track that were found running OpenCV's implementation of Shi and Tomasi's algorithm.

In [33], the authors assumed that a corner looks like a blurred wedge and then computed attributes of the wedge (the amplitude, angle, and blur). In [34], the authors generalized that work and they proposed calculating corner strength by looking at pixel values within a disc. They calculated the proportion of pixels whose intensity value is within the disc's center, or nucleus. The pixels that have a value closer to the nucleus receive a higher score. They called this measure the USAN, or Univalue Segment Assimilating Nucleus. If the USAN has a low value, then it is indicative that the USAN is a corner because it is different from its surroundings. These candidates are then run through another test to winnow out bad candidates and the resulting USANs make up the SUSAN, or Smallest USAN.

FAST [35], or Features from Accelerated Segment Test, considers the pixels inside a Bresenham circle (midpoint circle algorithm) with a radius *r*, around a candidate point. If there are *n* contiguous pixels that are all brighter than the nucleus by threshold value *t*, then the nucleus is considered to be a feature. The authors did testing to determine the optimal parameter values. If *r* has a value of 3, then the circle created using the Bresenham algorithm contains 16 pixels and when they set *n* to equal 3, they found that the algorithm did not detect lines and found only corners.

In [36], the authors took a unique approach of finding image features though the use of genetic programming. They were looking for points that had global separability, high information content, and were stable under “illumination change, rotation, scale change, and affine transformations”. They noted their future work will construct image filters that adapt to the environment, meaning that different filters would run for an indoor environment as opposed to outdoors.

Once features have been found, the next step is to track them from frame to frame. The review of literature in this field has primarily been limited to optical flow, blob, and correlation tracking because these three areas seem to be the most active areas of research. Each area is next described in detail.

The first class of tracking algorithms to be considered is optical flow. Optical flow is a method of estimating motion from frame to frame. Optical flow algorithms fall into two categories, sparse and dense. Sparse optical flow specifies a set of points to track from frame to frame, while dense optical flow looks at every pixel.

The Kanade-Lucas-Tomasi (KLT) algorithm [37] attempts to produce dense results, but their algorithm can easily be applied to a subset of points so it has become a popular sparse method. The KLT algorithm relies on local information derived from a small window surrounding each point of interest. The use of a small window size leads to problems detecting large motions. This led to the development of the KLT "pyramidal" algorithm [38]. This algorithm creates an image pyramid [39], which is a way to represent an image as a collection where the resolution changes at each level. The KLT pyramidal algorithm starts tracking from the highest level of the pyramid (the level that contains the least amount of detail) to the lowest level (the level that contains the most amount of detail). This tracking over a pyramid allows for larger motions to be caught by the larger spatial scales.

There are also several implementations of dense optical flow. A popular one from Horn and Schunck [40] was the first to use brightness constancy assumption to derive the basic brightness constancy equations. There are other methods to fall into the category of block matching, where algorithms divide the image into regions and match on those regions. More recent work published by Farnebäck [41] implemented dense optical flow by using polynomial expansion. He did this by fitting data in an image in a neighborhood to a quadratic polynomial model $I(x) = x^T Ax+b^T x+c$.

The problem with these versions of optical flow is they make certain assumptions that fail with the target application of the research proposed here. One assumption is that brightness in small regions will remain the same,

although the location will change. This may be true in data collected in a controlled environment, but in real-world use, the illumination frequently changes. It also assumes spatial and temporal persistence, but this assumption fails in this context because on a moving platform, the amount of motion is not always consistent. Optical flow also suffers from the aperture problem which is usually illustrated with a picture of a barber's pole. The actual motion of stripes on a barber's pole move horizontally but if the optical flow algorithms ran on that scene, they would all sense that the motion is vertical.

In [38], the authors used the KLT algorithm with modifications to process color images [42] along with accounting for changes in pixel brightness and contrast [43]. As features are lost from too much motion, they are replaced, keeping the number of features they track at a constant number. They also estimated the inter-frame motion to assist the tracking algorithm in a process they called "guided tracking".

Blob detection refers to algorithms to detect points or regions in an image that are either brighter or darker than their surroundings. Scale-Invariant Feature Transform (SIFT) [44], was developed to be invariant to scale. SIFT is also invariant to rotation because it detects the dominant gradient orientation at its location and records its local gradient histogram results with respect to this orientation. SIFT is also invariant to small affine transformations.

The algorithm first performs a Difference of Gaussian (DoG) operation. The DoG is an image filter that subtracts a blurred version of an original image from another, less blurred image. The blurring is done by convolving the input

image with Gaussian kernels that have differing standard deviations. This filter is capable of suppressing high-frequency spatial information. The subtraction of one image from another image preserves spatial information that is contained in the two blurred images. The second step in the SIFT algorithm performs keypoint localization where keypoints are selected based on their stability. The third step assigns orientations to each keypoint location based on local image gradient direction. The final step assigns a descriptor to each keypoint and the goal is to assign a descriptor (feature vector) that is highly distinctive. The feature vector contains a set of orientation histograms that are relative to the orientation of the keypoint. Each histogram contains 8 bins and each descriptor contains an array of 4 histograms around the keypoint, which leads to a 128 element feature vector.

Speeded-Up Robust Features (SURF) [45], is another type of blob detection. It was developed to be faster than SIFT and more robust against different image transformations. The speed improvement in the SURF algorithm comes from its use of an “integral image” [46]. The integral image, also known as a summed area table, is calculated by the sum of the values above and to the left of a point (x,y). The integral image was first described in 1984 [47]. The SURF algorithm makes use of several rectangular regions and each region is calculated using the integral image algorithm.

Although SIFT and SURF operate differently, they both output a descriptor vector that can be matched to descriptor vectors from other images. There have been different methods of matching descriptors proposed. In [48], Lowe

proposed computing the nearest neighbor of a feature and then checking to see if the second closest neighbor is further away from a given threshold. In [49], the authors considered only the nearest neighbor or if the distance is smaller than a threshold. Another method later proposed by Beis and Lowe [48] computed only the approximate nearest neighbor using a kd-tree, which is an extension of a binary search tree.

SURF does have several descriptors types of varying length. The regular version of SURF has a length of 64 but there is also a version where they double the descriptor length to 128. U-SURF is another version where the rotation invariance is left out, which makes the calculation faster.

There are two methods that have been published using SIFT features to track points from frame to frame which could also be extended to SURF features. One is to track the SIFT feature from frame to frame, as was used in [50]. This method was tried and was not found to be reliable. The SIFT features were not able to be reliably found from one frame to the next, especially in large, open areas such as grass and sky.

Another method that has been used is to find all SIFT features from frame to frame, and then find the affine transformation (using the putative matches between the two images using either robust least squares or Random Sample Consensus (RANSAC) [51]). An affine transformation preserves collinearity and relative distancing. They allow for repositioning, scaling, skewing, and rotation. In [52], the authors used the affine transformation to obtain the angle to the target location for controlling an Unmanned Aerial Vehicle (UAV).

Correlation tracking is a well-studied method and was first published in the 1970s [53-58]. Correlation is used to measure how a given quantity changes. Correlation can be used in image processing to calculate how feature points from one frame to the next change over time. The correlation tracker that was used in this analysis will be described in more detail later.

All of the above approaches to tracking have many different parameters that can be changed. This means that an approach that works well for one algorithm dataset may not work well for another. The goal of this aim was to determine which tracking algorithms are robust and perform well in real-time with real-world conditions.

In related work, Matchmoving is a technique used in cinematography that deals with seamlessly inserting virtual objects into a real-world scene. There are several commercial products available [59-61] that all track a point through a series of input images and figure out the 3D representation. There is an open-source project [62] that has been started and currently implements the KLT optical flow and SURF algorithms.

Although the goal of matchmoving is working with one scene at a time, they do use many of the same algorithms as this work does. The reliability of the algorithms developed in this research may also be of interest in this community because this aim produced a software environment (described in more detail later) that implemented a large number of tracking algorithms.

3.3 Methods

This section describes the tracking algorithms that were developed for this research. The methodology used to track a goal point using each algorithm is detailed is discussed. Finally, a testbed that was created for this research is described.

3.3.1 Implemented Tracking Algorithms

This research used algorithms from each of the three main classes described above to see how they compare. The three classes of tracking algorithms were correlation, optical flow, and blob. This section explains each algorithm's implementation in more detail. Figure 9 depicts the overall goal of this aim. It was to analyze tracking algorithms and determine how they compare in their accuracy of tracking a goal point and how efficient they are.

.

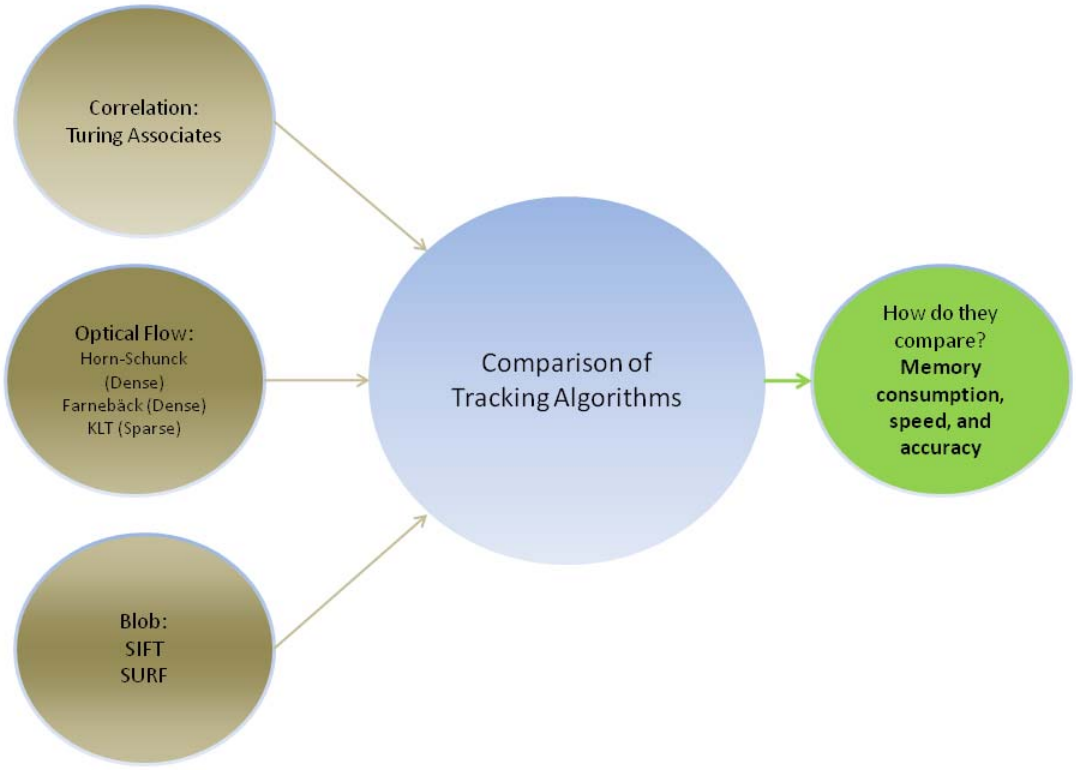

Figure 9: The overview of the three classes of tracking algorithms analyzed.

3.3.1.1 Correlation

The correlation tracker that was used in this research was developed by Turing Associates, Inc. as part of a research grant from the US Army. The algorithm finds the location of the goal point in the new frame that best correlates to the interest point in the reference frame. The novel approach that was used was a virtual large kernel "Multi-Resolution Progressive Alignment" search (MRPA). The unique attribute of this algorithm is that it is able to track a goal point without nearby features. A point on a blank white wall will obviously not track very well but the correlation tracker is the only method that can directly

track a goal point. It is able to do this because uses a foveal kernel that tracks interest points without nearby features.

An example foveal kernel is shown in Figure 10. The effect of using the MRPA is that it is sampled less densely at the periphery and more densely towards the center. It uses a large search space with a large kernel. The “progressive alignment” part of the algorithm accumulates data across resolution levels.

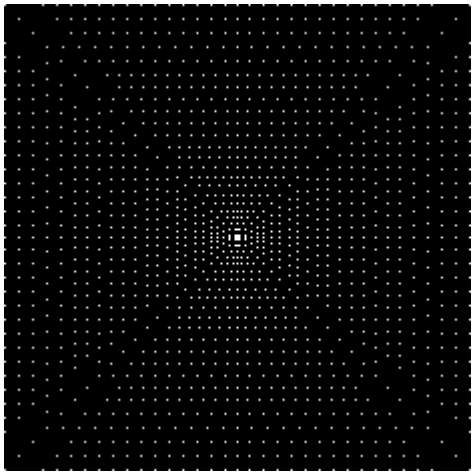

Figure 10: An example foveal kernel that the correlation tracker uses to track a point.

This tracking algorithm has undergone several revisions since the contract was awarded a number of years ago. One of the more recent modifications was done was to use a uniform kernel that had two parameters, the spacing in pixels and the number of points in the kernel. The geometry of this kernel is useful when going through a doorway, for example, because there are generally no features at the center of the doorway and the navigation is with respect to the features at the side.

The spacing in the uniform kernel allowed utilization use of Nyquist's theorem that states that a signal must be sampled at least twice as fast as the bandwidth of the signal. In the kernel spacing, this meant that the spacing could measure signals with a wavelength of 2S, where S is the spacing in pixels.

Another enhancement that was done to the correlation tracker was in making the algorithm more robust. In order to have robust tracking, there are two important considerations that apply to all of the tracking algorithms. The first is to recognize accurate tracking. The second is to be able to recognize point drift. The approach used for the correlation tracker is shown in Figure 11. This approach automatically skips corrupted frames due to motion blur or communication errors. This is also more stable because of the multiple tracks and the added median filtering.

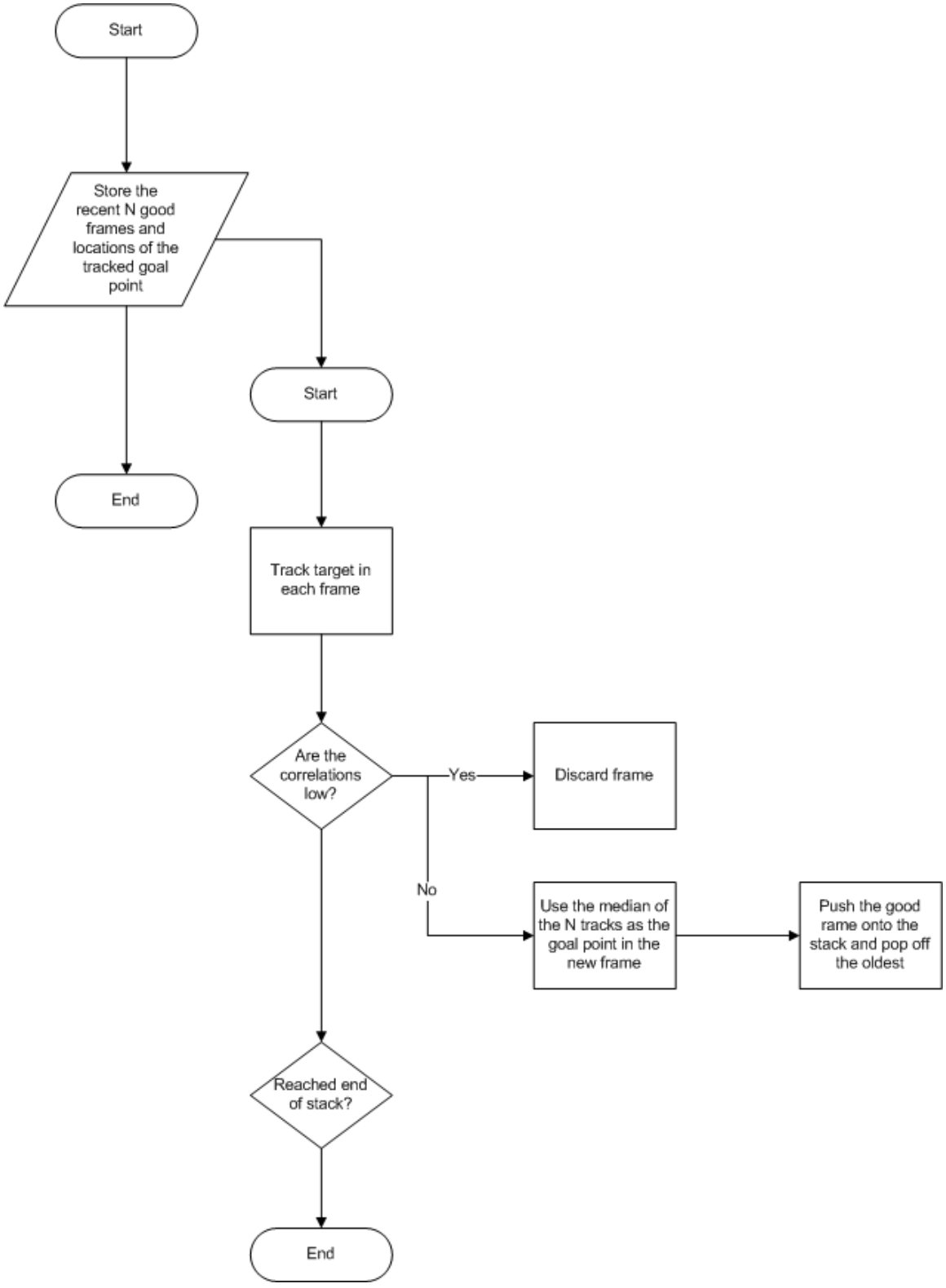

Figure 11: This is the process developed to make the correlation tracker robust.

3.3.1.2 Tracking a Goal Point

The next two classes of tracking algorithms, optical flow and blob, do not directly support tracking a goal point because the features they track depend on what each algorithm defines as a feature point. Tracking a goal point that is anywhere in the image is important for the visual servoing application because an operator will want to direct the robot to go anywhere in the camera's field of view and not be limited to only features that can easily be tracked by a particular algorithm. The image in Figure 12 shows an outdoor scene of a path with woods and vegetation on each side. The green points indicate the corners that were found in the first step of the KLT algorithm. If a robot moves straight through this scene, the points contained in the two yellow ellipses would move more than the points contained in the red ellipse.

Figure 12: An image of wooded path. The green points were found using Shi and Tomasi's corner detector. The points contained in the yellow ellipses have greater motion as the robot moves forward than the points in the red ellipse.

In the study of plane geometry, there are affine and projective transformations. An affine transformation preserves colinearity between points and the ratio of distances of collinear points. A projective transformation keeps straight lines straight but it does not preserve the angles between the lines because the warping cannot be defined as an affine transformation.

In the example of moving through the wooded scene, an affine transformation is not sufficient to describe the changes in the scene as a robot drives through it. There are actually multiple projective transformations that are occurring, one for each surface facet.

Make3D [63] is an open-source project that takes a single image still and produces a 3D model that can then be used to virtually “fly-through” the scene. The author did this by using supervised learning to predict the depth map as a function of the input image. The use of Make3D is described later in this chapter but for now, Figure 13 illustrates this point of multiple projective transformations. This is the same scene that was shown in Figure 12 but with the 3D facets, as found by Make3D, overlaid on the scene. As the camera moves through the scene, each of those facets undergoes a transformation.

Figure 13: This is the same wooded scene shown earlier but with the 3D facets found using Make3D shown.

The approach developed to track a goal point makes use of an affine transformation but it was limited to a specific region, which is described later.

Going back to the affine transformation, there are six values that specify that transformation [64]. Those six values, A, B, C, D, E, F, have to satisfy the formula in (3.3). In (3.3), the X and Y are the old coordinates and X' and Y' are the new coordinates. The equation may also be written in matrix notation as shown in (3.4), where T is a 3x3 matrix of coefficients shown in (3.3).

$$\begin{aligned} X' &= AX + BY + C \\ Y' &= DX + EY + F \end{aligned} \tag{3.3}$$

$$[x' \quad y'] = [x \quad y \quad 1]T \tag{3.4}$$

$$T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \tag{3.5}$$

A general affine transformation from 2D to 2D [65] is shown in (3.6) and requires six parameters that can come from three pairs of points.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.6}$$

Figure 14 shows a simple example of using three pairs of points (blue points) to track the goal point (red point) from frame to frame. The three input points (blue) are fed into the equation in (3.6) and the resulting affine transformation is used to calculate how the points moved and the motion is applied to the goal point (red). This approach in (3.6) is simple and if there are any errors in the input coordinates, the calculated transformation will be wrong. Those errors accumulate over time and it doesn't take long for tracking a goal point to be

significantly off. In order to reduce the amount of error, a better approach is to use many pairs of points and find a method of rejecting outliers. One method is the RANSAC algorithm described earlier. Another is similar to the least squares approach of fitting a straight line [65].

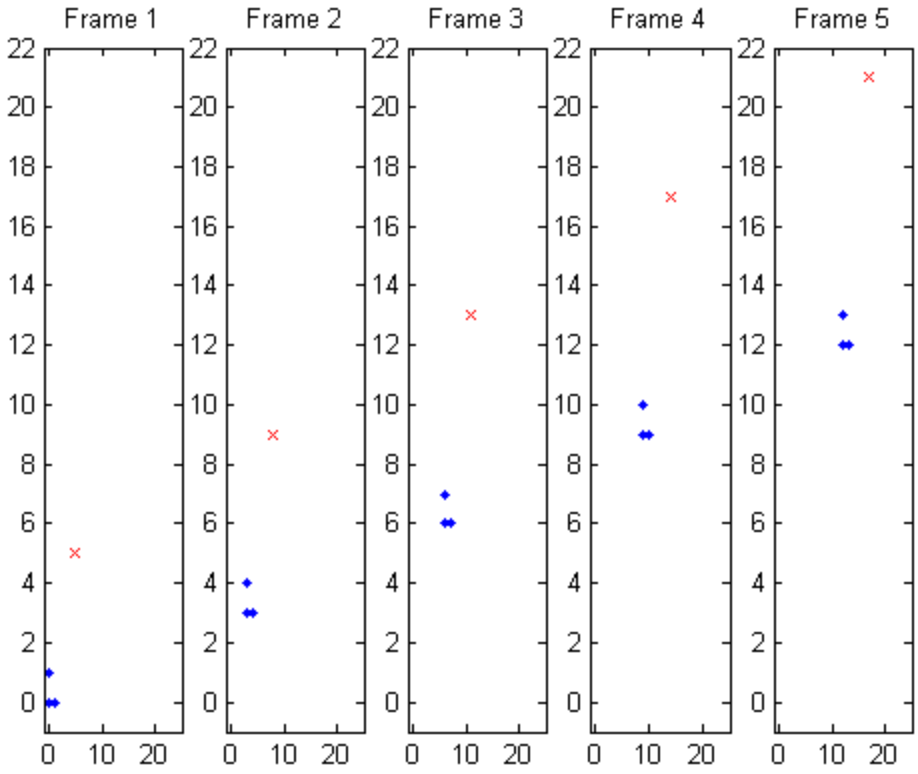

Figure 14: A simple example of tracking a goal point (the red point) from frame to frame using three non-collinear points (blue points).

The approach used in this research uses an affine transformation combined with limiting the points to a specific region. That region is found using image segmentation. Image segmentation is another popular area of computer vision research and there have been a large number of papers published on the

topic. The goal is to partition an image into a set of regions. The image segmentation algorithm used in this research is a graph-based approach [66]. This method was chosen because the authors were kind enough to release a C++ implementation of their algorithm and it was fast enough for the needs of this research and it was able to be easily implemented into the code base.

The algorithm looks for similar regions based on color and texture and attempts to group like pixels together. Figure 15 shows an example image taken from within the laboratory. The image on the left is the segmented image using the graph-based algorithm. The pseudo-color image is created by randomly drawing an RGB color for each detection region. The image on the right is the input image with the detected regions drawn in green.

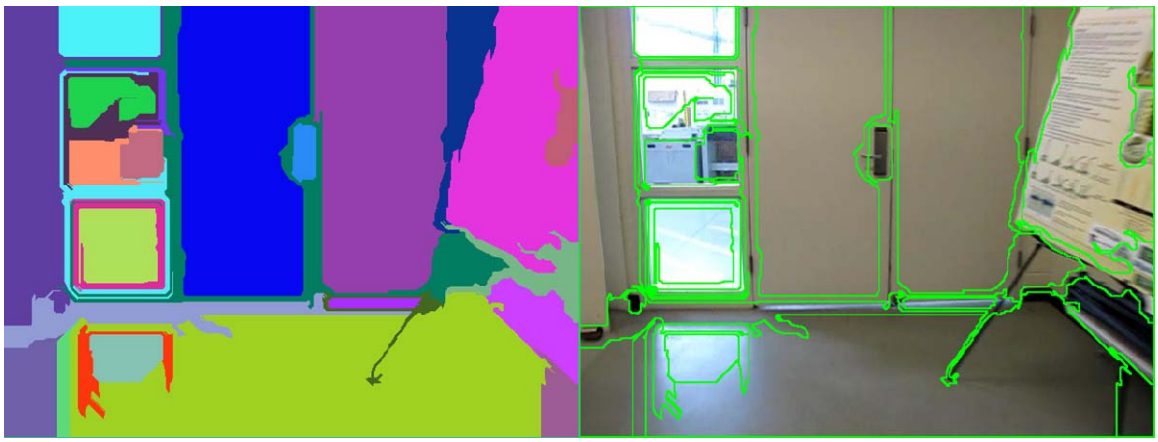

Figure 15: A scene from inside the laboratory. The image on the left is after the image was segmented using the graph-based algorithm. The image on the right is the input image with the detected regions overlaid in green.

There are several input parameters to the segmentation algorithm. Sigma controls how much smoothing to apply to the image prior to doing the

segmentation. This is important so that the algorithm doesn't detect too many regions. There is a constant, *K*, used for a thresholding function, and, c, that controls a post-processing step that will merge smaller regions together to attempt and meet minimum number of regions specified.

In this research, if the segmented image does not contain the minimum number of regions that were specified, the image is segmented again with the next set of parameters. In the first set of parameters, sigma was 0.5, K was 500, and the c, minimum number of regions, was 50. In the second set, sigma was 0.5, K was 750, and c was 100. In the third set, sigma was 0.5, K was 1000, and c was 100 These values were all empirically found and in the datasets that were worked with.

The segmented image is used in the developed tracking algorithm by making the assumption that regions belong to objects and that the operator wants to go to an object. After the region has been found, points located on that region are tracked and the affine transformation is calculated from frame to frame. There must be at least three non-collinear points for this algorithm to work.

3.3.1.3 Optical Flow

Three optical flow algorithms, Horn-Schunck, Farnebäck, and KLT, were analyzed for this class of algorithms. Horn-Schunck and Farnebäck are classified as dense because nearly every pixel's movement is calculated from frame to frame. The KLT sparse optical flow algorithm, on the other hand, tracks corners from frame to frame.

The optical flow algorithms typically operate by searching a fixed size window, from frame to frame, normally using the sum of squared differences metric to determine the position of the corner in the new frame. This method is fast but can be unreliable because a spiral search is typically used. A spiral search continues to find the next corner using Euclidean distance and once a match that meets a threshold value has been met, the search stops. This method can output many incorrect matches.

A method that has been shown in literature to successfully identify and reject the outliers, or bad matches, is the RANSAC algorithm [51]. Figure 16 shows an example of using several different methods to fit a line to noisy data [67]. RANSAC is good at rejecting outliers but because it is iterative, it can be resource-intensive to find an acceptable solution.

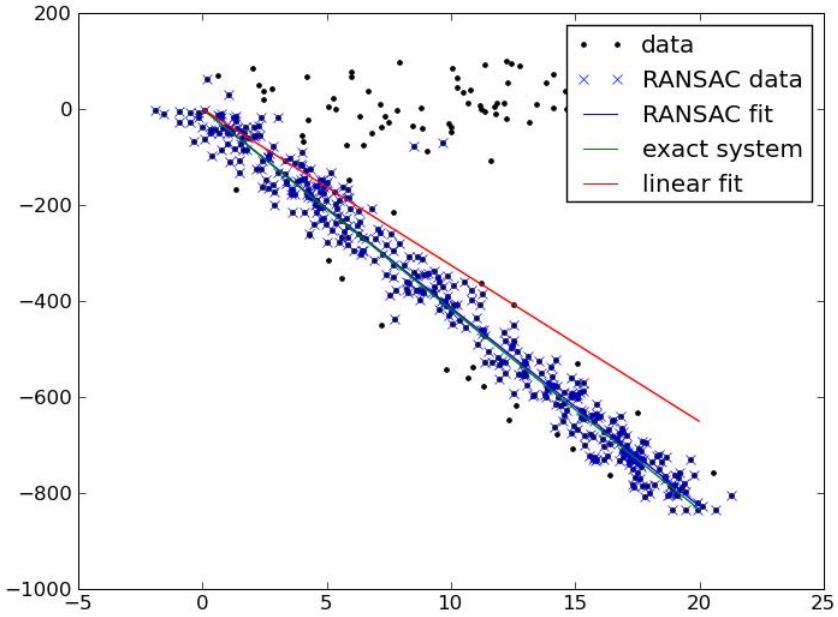

Figure 16: An example of fitting a line using several different methods, including RANSAC. The data points, containing outliers, are represented by black dots. The exact system is indicated by the green line, a linear fit is denoted by the red line, and the line that was found using the RANSAC algorithm is shown in blue.

The more accurate that the corner points can be tracked, the more accurate the calculated location of the goal point will be from frame to frame. In this research, a descriptor was added to the KLT optical flow algorithm to increase its reliability. Several different shapes, or "patches" around a corner point were implemented including a rectangle, a disc, and an annulus. The disc construction used Bresenham's circle algorithm [68]. An example of the annulus patch is shown in Figure 17. There are two parameters to the annulus, the outer and inner radii. Even though circular descriptors are computationally intensive compared to rectangular ones, they were chosen to be robust to roll.

Once corners have been found in the reference and test frames, a match can be found using several different ways. One method is to use a descriptor of some sort and then use the nearest neighbors approach. The kd-tree algorithm [69] continues to be popular in literature to match descriptors. The methodology of the kd-tree is relatively simple. At each node, the points are recursively partitioned into two sets by splitting along one dimension of the data, until some criteria to stop has been met [70]. The Approximate Nearest Neighbor [71] (ANN), is a kd-tree implementation. The author of Bundler [72], a project to build a 3D model from an unordered collection of images, converted the ANN algorithm to search a vector of unsigned bytes. The FLANN algorithm [73] uses different tree structures and automatically chooses the best one based on the data. There are also PCA trees [74], Ball trees, [75], and k-means [76]. .

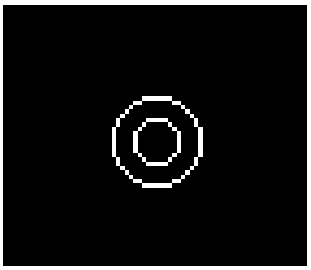

Figure 17: Example annulus descriptor used for adding a descriptor to the KLT algorithm. There are two parameters to an annulus, the inner and outer radii.

The idea behind a histogram-based approach is another way to compare images. This is highlighted in Figure 18. The image in the figure was first converted to the HSV (hue, saturation, and value) colorspace and then a patch was extracted around each interest point, and the histogram of the value, or intensity, plane is displayed.

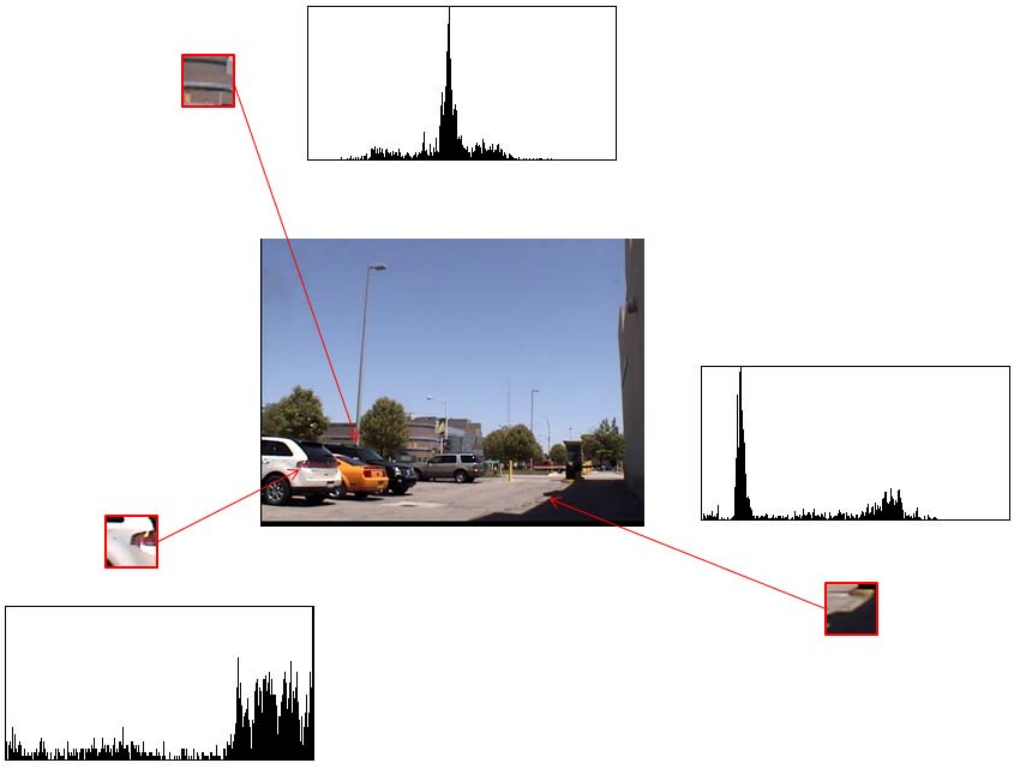

Figure 18: An image from the SUV dataset where three patches were extracted, converted to the HSV (hue, saturation, and value) colorspace, and the histogram of each patch was calculated using the value plane.

There are several algorithms already implemented in OpenCV [32] that were used to perform histogram matching. In the equations listed below, *H1* and *H2* are two histograms that are being compared. The first histogram matching algorithm is a measure of correlation, the equation for which is shown in (3.7). A perfect match is when the correlation equals 1. A total mismatch is when the correlation is 0.

$$(H1, H2) = \frac{\sum_i H_1'(i) * H_2'(i)}{\sqrt{\sum_i H'^2_1(i) * H'^2_2(i)}} \tag{3.7}$$

The Bhattacharyya matching algorithm is shown in (3.8). A perfect match occurs when the value returned is 0 and a mismatch is when the value is 1.

$$(H1, H2) = \sqrt{1 - \sum \sqrt{\frac{H1(i) - H2(i)}{\sqrt{\sum H1(i) - \sum H2(i)}}}} \tag{3.8}$$

Another method of matching two images of the same size is by calculating the 2D correlation coefficient. The equation for calculating the 2D correlation coefficient is shown in (3.9).

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{\left(\sum_m \sum_n (A_{mn} - \bar{A})^2\right)\left(\sum_m \sum_n (B_{mn} - \bar{B})^2\right)}} \tag{3.9}$$

The Sum of Squared Distances (SSD) is another approach already used by the optical flow algorithms. The equation for the SSD metric is shown in (3.10).

$$SSD = \sum_{(i,j)} (I1(i, j) - I2(x + i, y + j))^2 \tag{3.10}$$

The importance of choosing a circular descriptor able to handle roll is shown in Figure 19. The input shape was rotated clockwise 45, 90, 135, and 180 degrees and then the 2D correlation coefficient was calculated at each rotation,

comparing the input image to the rotated image. The correlation coefficient dropped considerably with the introduced roll.

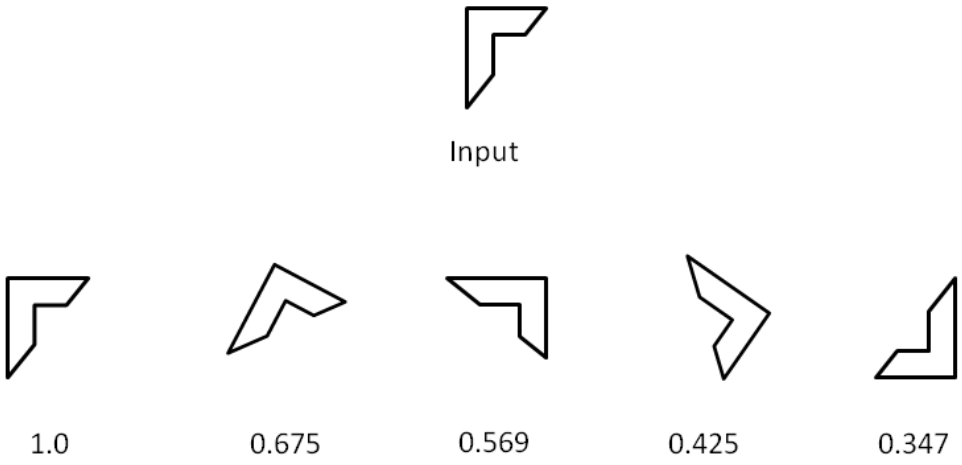

Figure 19: The computed 2D correlation coefficient of an input shape compared with the shape being rotated.

Before comparing two patches, it is important to line the two corners up. The approach taken in the experiment was to first find the SIFT features in both the reference and test frames and use that to calculate the global affine transformation. The calculated affine transformation was then applied to the test image prior to using the Harris corner finder algorithm. There are other methods to this. One approach would be to take square bounding the region and rotate the detected corner in both the reference and test frames so that the corner was vertical, as depicted in Figure 20.

Figure 20: A possible configuration (12:00 position) of aligning each corner in order to calculate the match.

An experiment was conducted to determine the accuracy of the various descriptors and the most accurate method to match them. The shapes of the descriptors around a feature point used were: a 30 pixel x30 pixel rectangle, a disc with a radius of 15 pixels, and an annulus with an outer radius of 15 and an inner radius of 5. The descriptor matching methods were the 2D correlation coefficient, the Sum of Squared Differences, a kd-tree, and the two described histogram matching algorithms. The SSD method ran an exhaustive search on all corners and did not use the spiral search described above.

In the histogram methods, the image was converted to HSV and the values were separated into 32 bins using the intensity plane. For the kd-tree algorithm, the 2D vector had to be converted to 1D. The rectangle was converted in a row-wise manner to a 1D vector. The process for the disc and annulus shapes started at the innermost edge and went in a clockwise direction, listing the values.

Table 1 summarizes the steps in the experiment. The first step was to randomly draw 50 reference images from a collection of approximately 2,000 images. This collection of images is described in more detail later in this chapter.

They are composed of four different scenes with a camera on a robot recording the video as it drove towards an object in each scene.

The test frames were the next sequential images in the dataset, meaning they were the next frame taken after the reference frame when the data were collected. Next, SIFT features were calculated in both the reference and test frames and the global affine transformation was found to align the two images. This was done to ensure the corners lined up.

Next, the Harris corner detector algorithm was run on the reference frame and found corners that had a minimum distance to 20 pixels to the next closest corner. This was done because the verification step of this algorithm was done visually and by having the features spread apart, it made it easier to check. Next, the Harris corner detector was run on the test and found all corners and no minimum distance was specified. The next step applied each of the descriptors (rectangle, disc, and annulus) to each point in both frames and matched them using one of the four metrics.

KLT Descriptor Experiment
1) Randomly sample 50 pairs of image from a collection of approximately 2,000 images.
2) Find the SIFT features, match them, and calculate the global affine transformation for the reference and test frames, warp test frame to align images.
3) Find all Harris corners in the reference frame (time *t*), a minimum of 20 pixels apart.
4) Find all Harris corners in the test frame (time *t*+1).
5) Visually inspect and determine the number of correct matches

Table 1: Methodology used to test the accuracy of various KLT descriptors.

The results were visually inspected and the number of correct matches was recorded. A total of 1,000 points were used for this test. The results of the experiment are shown in Table 2. This does pave the way for future work with this descriptor and because this functionality is within TACTCIAL, it allows for further experimentation.

	2D Corr	SSD	kd-tree	Histogram (Bhattacharyya)	Histogram (corr)
Rectangle	0.887	0.823	0.822	0.819	0.777
Disc	0.922	0.835	0.833	0.823	0.789
Annulus	0.878	0.812	0.789	0.813	0.735
Mean	**0.90**	**0.82**	**0.81**	**0.82**	**0.77**

Table 2: The results of the experiment conducted to add a descriptor to KLT optical flow.

The 2D correlation coefficient with the disc-shaped descriptor had the best performance with 92% of the corners matched correctly between the two frames. This descriptor shape and matching method was also used in determining which tracking algorithm performed the best, which is described later.

3.3.1.4 Blob Trackers

The approach to tracking a goal point for the two blob trackers is essentially the same as the optical flow algorithms. The image is first segmented to find the region that corresponds to where the operator clicked and the keypoints are matched from frame to frame. Next, the non-collinear points are used to calculate the affine transformation of the goal point from frame to frame.

This matching step in each of the blob trackers is susceptible to incorrect matches. Figure 21 shows an example of matching SIFT features from two consecutive frames in the “wood pile” dataset. The algorithm appears to match the majority of the points successfully but there are several by the tree, highlighted by the yellow ellipses, on the right side that it fails to match correctly. One method of eliminating incorrect matches is to use the RANSAC algorithm, as described earlier.

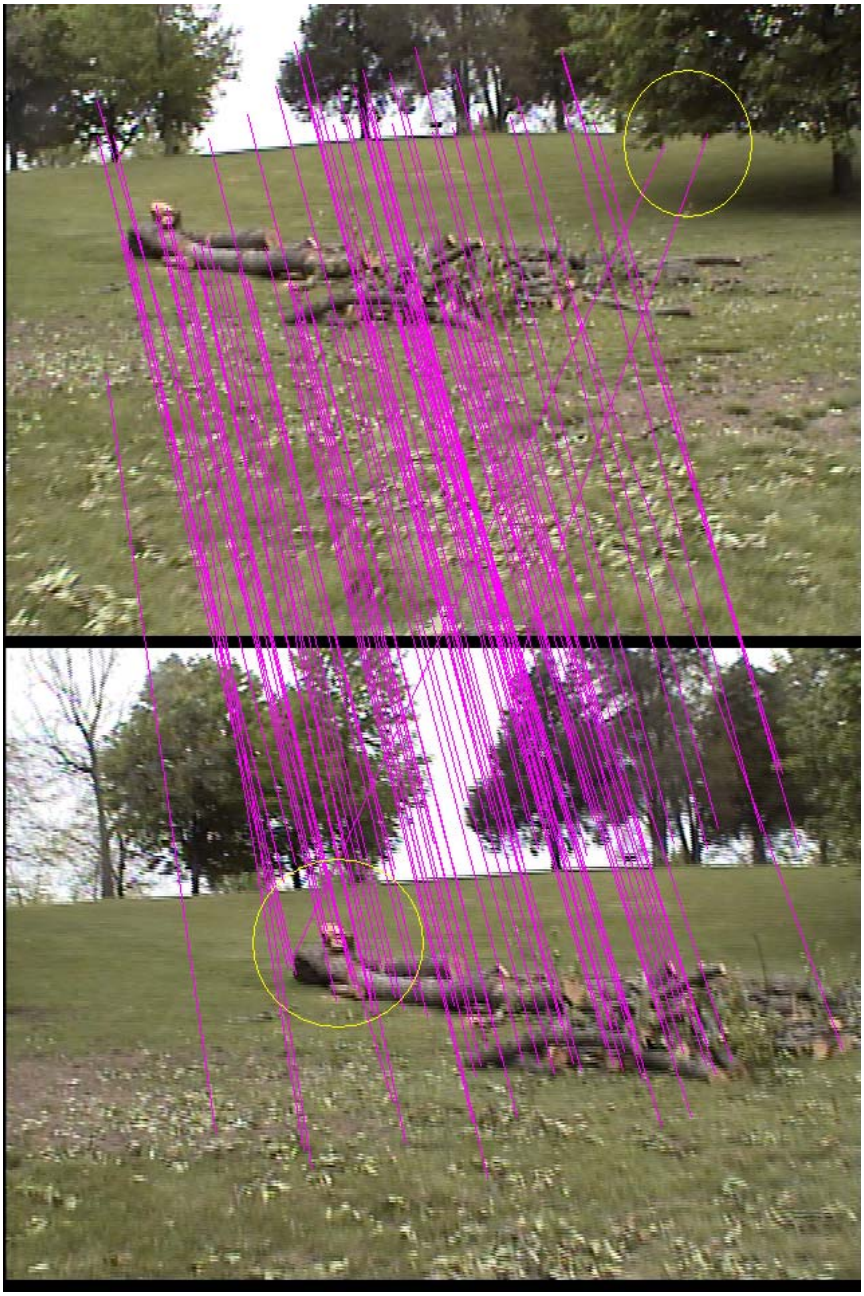

Figure 21: An example of matching SIFT features from two Images from the "wood pile" dataset. The yellow ellipses show where the SIFT algorithms made obvious mismatches.

3.3.2 Tracking in 3D

After developing a method to track a goal point using any algorithm, the next step was to see which one performed best. The first method that was tried for analyzing the tracking algorithms was to use a virtual 3D world. Make3D, described earlier, was used to create a 3D VRML model of a single image. Figure 22 shows the view from within MATLAB of tracking a point while the model is flown through in azimuth at each timestep.

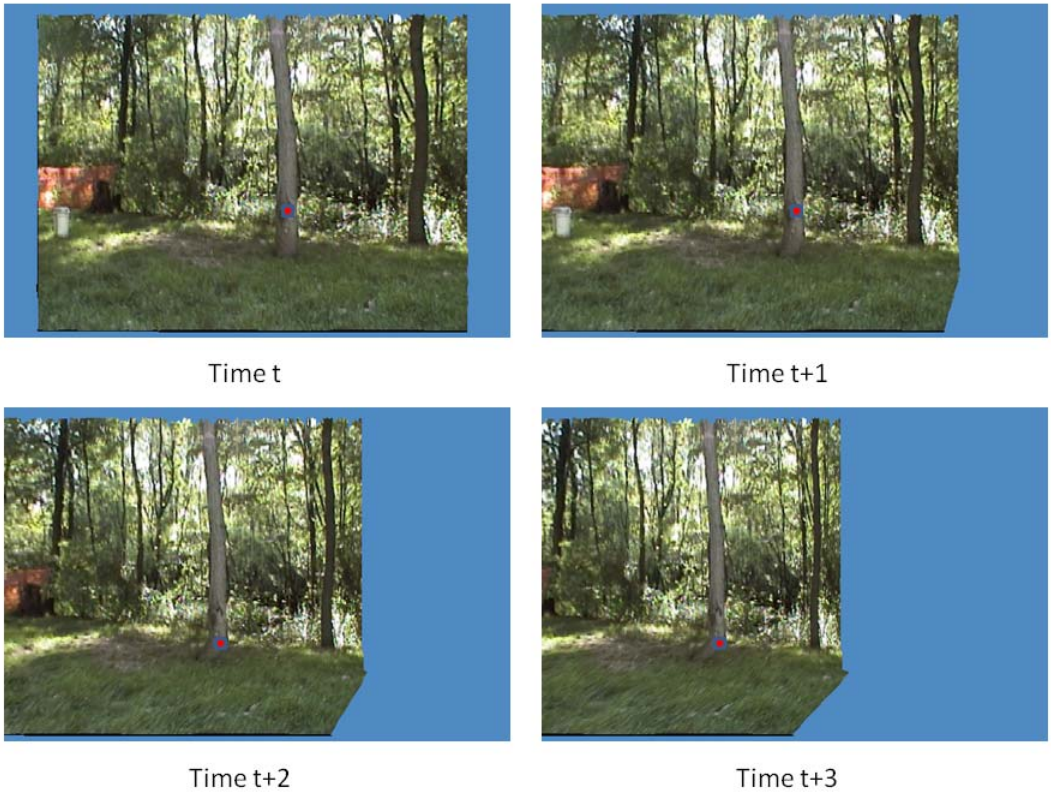

Figure 22: Tracking in 3D within MATLAB using a model generated by Make3D, an open-source project that takes an image still and produces a 3D model. The scene is rotated programmatically by a script.

MATLAB's Virtual Reality Toolbox was used to test if the idea was feasible. The scene in Figure 22 was programmatically moved a set amount by the program while the tracking algorithm tracked the goal point. Several of the tracking algorithms were already able to be called from within MATLAB as a proof of concept. Once it was established as possible, an open source project, "view3dscene" [77] was used because it is written in C++. This made it easy to implement with the existing codebase.

Figure 23 shows the same wooded scene as before, created using Make3D, but this time it is being "walked through". It is difficult to easily show but the camera's perspective is able to be change programmatically to simulate walking through the scene. The goal point is shown as a while dot, far off in the background, and the facet it is on is shown by the white triangle.

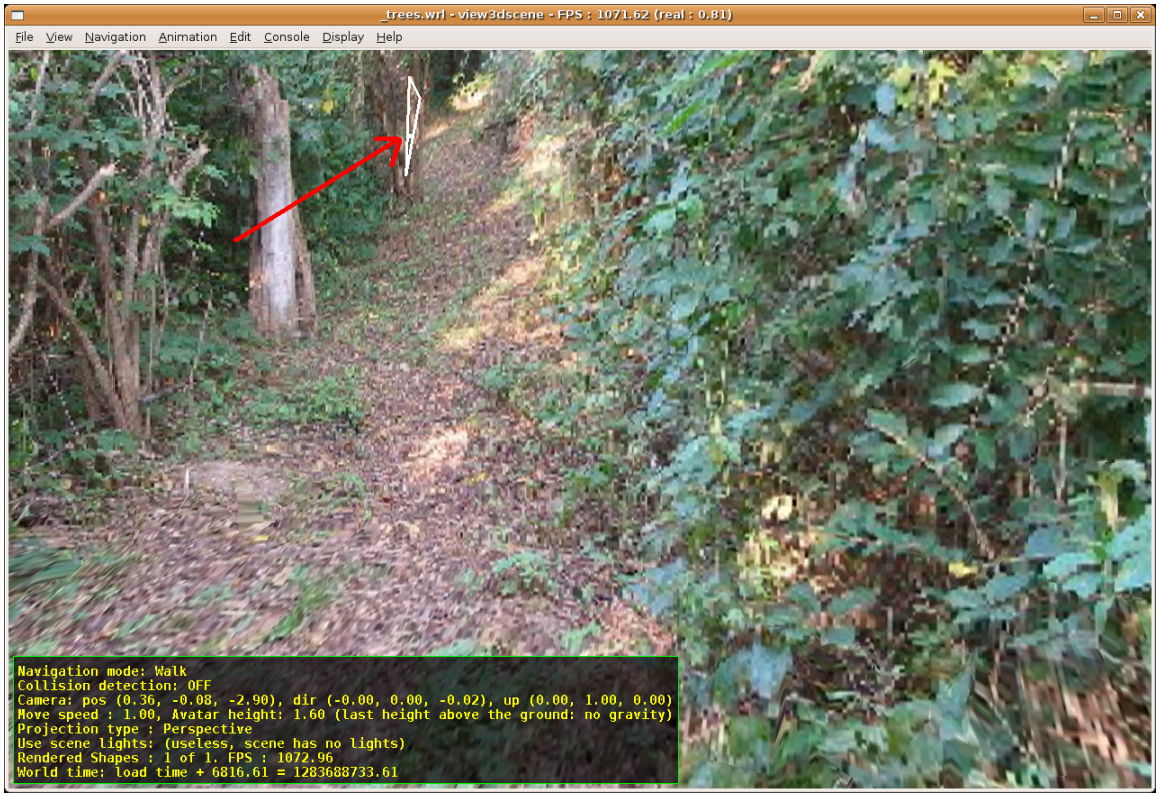

Figure 23: The same wooded scene shown before but it was converted to a 3D "fly-through" scene using Make3D. It is able to be programmatically "walked-through" using a modified open source project, "view3dscene".

Although this approach was demonstrated to be feasible, it didn't give accurate enough ground truth data. The 3D facet that the tracked point was on could be obtained programmatically but the facets were sometimes large. Being able to reliably calculate ground truth was a problem..

This approach also did not give the same motion that an actual robot driving in a real scene would give. The real-world conditions of a robot driving through a scene and the associated motion blur is difficult to simulate. As a result of the simulation environment not being accurate enough, data were

collected using a robot used that for the analysis of tracking algorithms. The next section details the data that were collected and the method used.

3.3.3 Data Collection

In order to prepare for the analysis of which tracking algorithm performed the best, data were collected using a robot to capture real-world movement along with the inherent imperfections such as motion blur. The data used for the analysis of the tracking algorithms were collected using the Intelligent Ground Vehicle Competition (IGVC) platform. It was equipped with a Sony NTSC camera, GPS, and an Inertial Measurement Unit (IMU) and is shown in Figure 24.

The platform was also equipped with encoders able to read data from the motor controller. Data were collected with an on-board computer while the robot was being driven with a joystick. The input commands from the joystick, along with the motion commands sent to the robot, were also recorded. In each dataset, the task was to drive towards a target and record the data along the way. The data collection was done around Wayne State University's campus and in a typical park.

Figure 24: The IGVC platform that was equipped with a camera, GPS, and an IMU for data collection.

Figure 25 shows the four datasets used for the analysis of tracking algorithms. The “Sign” dataset contains 241 frames. The “Silver Car” dataset contains 943 frames. The “SUV” dataset contains 562 frames and the “Woodpile” dataset contains 228 frames. These scenes were picked because the target stayed in the camera’s field of view for the entire duration and there were a substantial number of frames that could be used for the analysis. More details on the dataset are contained in Appendix B.

Figure 25: An image still from each of the four datasets used in the tracking algorithm analysis.

3.3.4 Software Development Environment (TACTICAL)

TACTICAL, Tracking Algorithm Comparison Testbed for Comprehensive Analysis and Learning, was developed as part of this research in order to have a common method of evaluating the implemented algorithms and to have a single environment for all image operations. The “Learning” in TACTICAL’s name implies the user is able to learn which filters and features work best for a given dataset and does not currently implement any machine learning algorithms. It allows the user to load a video file or a series of sequentially numbered images, change the parameters of the desired tracking algorithm, click on a point to track,

and observe how well the algorithm does at tracking the goal point. Figure 26 shows a screen capture of one of the tabs within TACTCIAL. The source code for TACTICAL, along with the datasets mentioned earlier, have been released to the open source community [78]. It was developed and released as open source in the hopes that other researchers in this field find it useful.

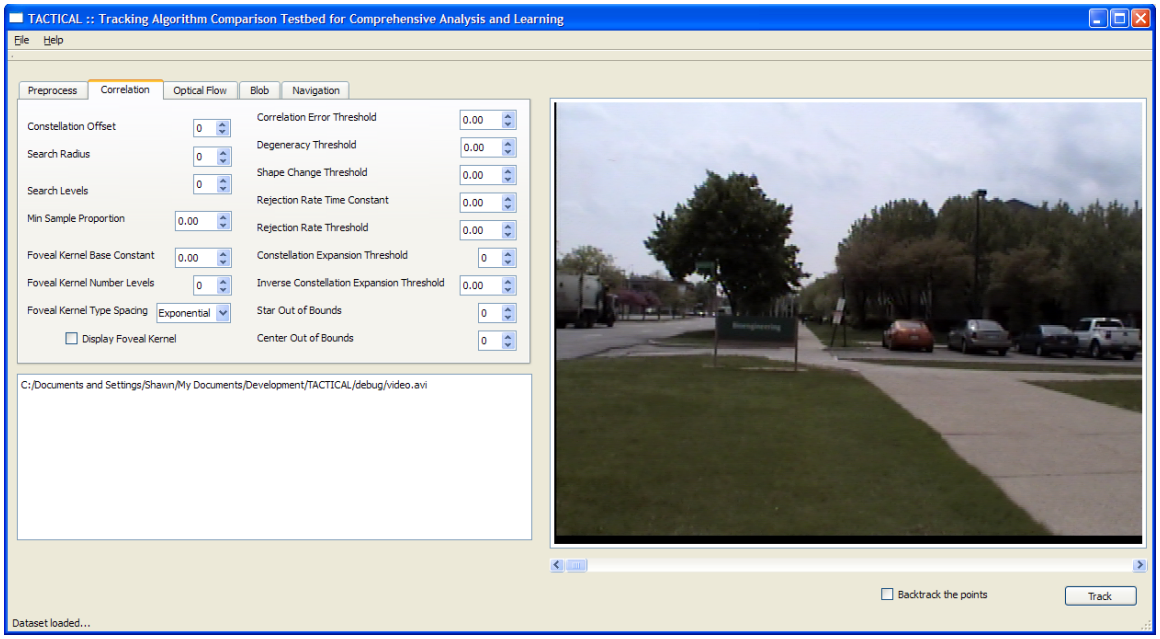

Figure 26: Screenshot of TACTICAL, the software testbed used for the tracking algorithm analysis

TACTICAL has the ability to apply various image operations on the video stream before the tracking algorithms are called. The tracking algorithms used in this research all operate on grayscale images. There are different algorithms that have been developed to convert an image to grayscale but there is a tradeoff in the quality of the converted image with the time that takes. In TACTICAL, there are several algorithms implemented. The first, and fastest, takes only the green plane and discards the red and blue. The second is the lightness method,

which takes the mean of the maximum and minimum RGB value for each pixel. The third algorithm is the average method which takes the mean of RGB pixel values. The fourth algorithm implemented is the luminosity method which applies a weighted average to each red, green, and blue pixel. The weight is adjustable within the software environment.

There is also the ability to use different color models. The Hue, Saturation, and Value (Intensity) color model (HSV) was implemented. In this color model, the intensity, or gray-level value, is decomposed into two color-carrying components, hue and saturation [79]. The Lab color model was also implemented, where L is the luminance value and a and b represent the two color channels. This is the color space that most represents human vision.

A large number of filters both in the spatial and frequency domains were developed over the course of several years. In the spatial domain, smoothing filters such as order-statistic and lowpass filters were implemented. Laplacian and gradient sharpening filters in the spatial domain were also implemented. On the frequency domain side, several lowpass and highpass filters were developed.

This collection of filters has been in development over several years. The development of TACTICAL allowed for a common place to use them all. This was useful because the effect of using a filter with a tracking algorithm isn't known until it has been tried.

A good example of this is with the SIFT algorithm. The literature does not mention it but it was found that by applying a high-pass filter to the frame prior to finding SIFT features, it improves the algorithm's performance. These

interactions are not able to be found without experimentation. The main goal of TACTICAL was to have everything easily accessible so an analysis could be quickly and easily performed.

The ability to add compression to the datasets within TACTICAL was also significant. JPEG and MPEG compression were added into order to degrade each dataset to determine how each tracking algorithm performs. This simulated the real-world conditions at the OCU.

Another important component added to TACTICAL was the ability to manually add ground truth to a dataset. This was a simple annotation tool that allowed a user to click through the video and designate the goal point in each frame. The output is simply a text file with the X and Y coordinates input by the user at each frame of the input video. There is also a method to click through, verify, and change the goal point's location, if needed.

This addition was significant because it addressed the problems experienced with tracking in 3D, which were the inability to simulate the motion of a robot moving through an environment and with the 3D facets not providing precise ground truth data. Although the location of the tracking could be obtained programmatically, the area the facet covered could be small or large. This annotation tool is not perfect because the ground truth is only as accurate as the user's ability to click on the goal point from frame to frame. It is also a very time consuming process, especially when labeling multiple points in the scene because the entire dataset has to be processed for each goal point. In hindsight, a better way to handle this would have been to program a method of

inputting how many goal points were going to be labeled for that particular dataset and allowing the user to click on all of the goal points before moving on to the next frame.

A popular method mentioned in the literature of tracking algorithm is the use of synthetic images to know ground truth [80]. While this approach is certainly valid for some problem domains, this was not the case with this research. This method captured real-world conditions so the tracking algorithms could be made to perform as robustly as possible and the algorithms could be validated as working properly before implementing on an actual system.

3.4 Results

This section details the results of the experiments run to measure which tracking algorithms performed the best. The metrics used included computational speed, overhead, accuracy, and the effect added compression had on the algorithm's ability to track the goal point. The eight algorithms used in this analysis are summarized in Table 3. The KLT with descriptor was the disc-shaped region with the 2D correlation coefficient as the metric.

Algorithm	Class
Turing's Correlation	Correlation
Horn-Schunck	Optical Flow, Dense
Farnebäck	Optical Flow, Dense
KLT	Optical Flow, Sparse
KLT with RANSAC	Optical Flow, Sparse
KLT with Disc Descriptor	Optical Flow, Sparse
SIFT	Blob
SURF	Blob

Table 3: The eight algorithms compared in this analysis.

All development of the tracking algorithms and experiments detailed below were run on an Intel Core Duo processor with 2 GB memory. The computer ran Ubuntu version 8.04. The code was all written in C++.

3.4.1 Speed

The first metric run was solely on execution time. All algorithms were run within TACTICAL and as such, were all coded in C++. Figure 27 shows the eight algorithms. The mean execution time across all four datasets is shown. The correlation and versions of KLT were orders of magnitude faster than the dense optical flow and blob algorithms. Figure 28 shows the same data as Figure 27 but with the correlation and KLT algorithms pulled out.

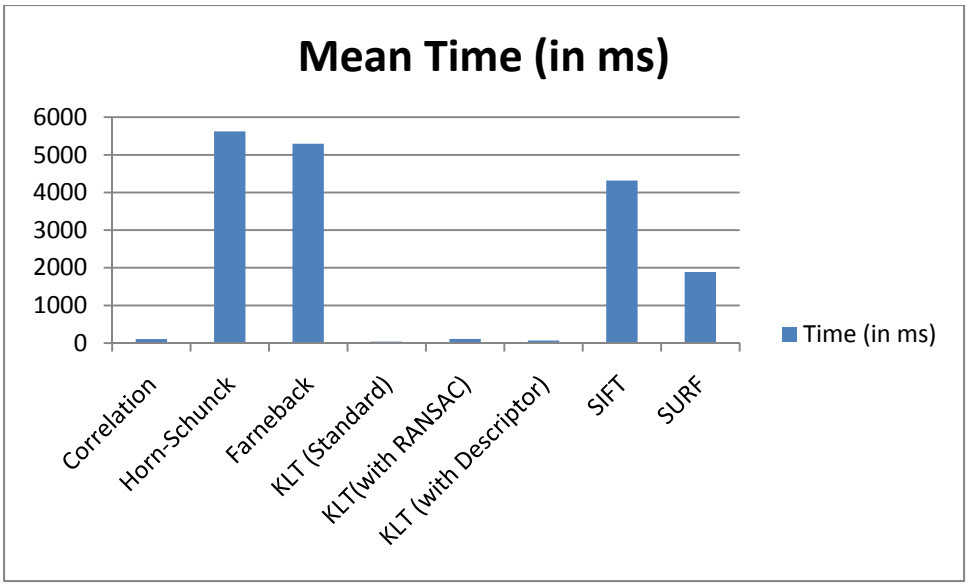

Figure 27: Execution time (in milliseconds) of the eight algorithms compared

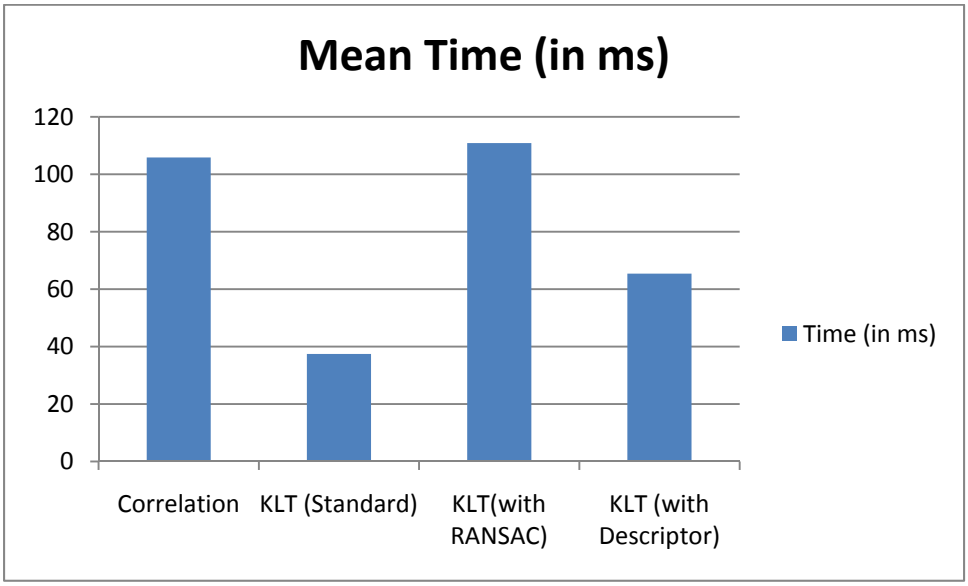

Figure 28: Execution time of the correlation tracker and KLT versions.

The dense optical flow algorithms and blob trackers could conceivably process data at 1-2 Hz. The correlation and KLT algorithms are capable of processing in the realm of 10-15 Hz on the typical OCU hardware that normally

ships with the PackBot. There have been recent advances in using the GPU for processing that could possibly increase the speed of the slower algorithms.

3.4.2 Memory Consumption

The amount of memory consumed was another metric looked at for each of the algorithms. After each algorithm executed on a frame, the system state was queried for the amount of memory being consumed. Figure 29 shows the mean memory used across all four datasets.

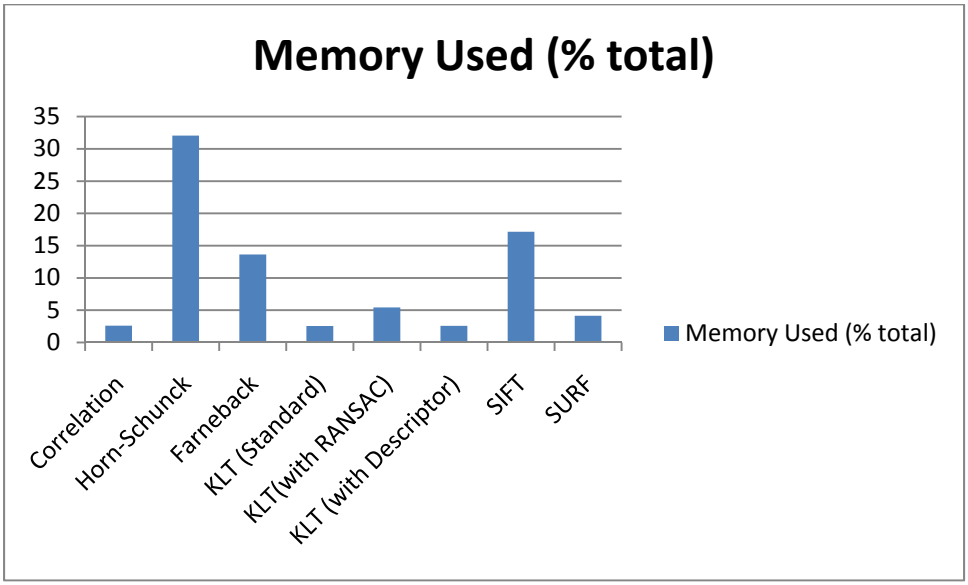

Figure 29: The mean memory consumption over all four datasets.

The results for memory consumption were consistent with what was found for the mean time. The correlation and KLT optical flow algorithms occupied the least amount of memory.

3.4.3 Accuracy

As discussed earlier, the data for the tracking algorithm analysis were collected in such a way that a specific object was designated as the goal point

and the robot was driven towards the object. Each of the four datasets had a visible target. Five different people were asked to click on the object in each of the four datasets. These goal points were then used as the goal point in each of the datasets. The ground truth data were then established for each of the five points using the addition to TACTICAL. Figure 30-31 show the aggregated mean Euclidean distance of the tracked goal point from the ground truth for each of the four datasets.

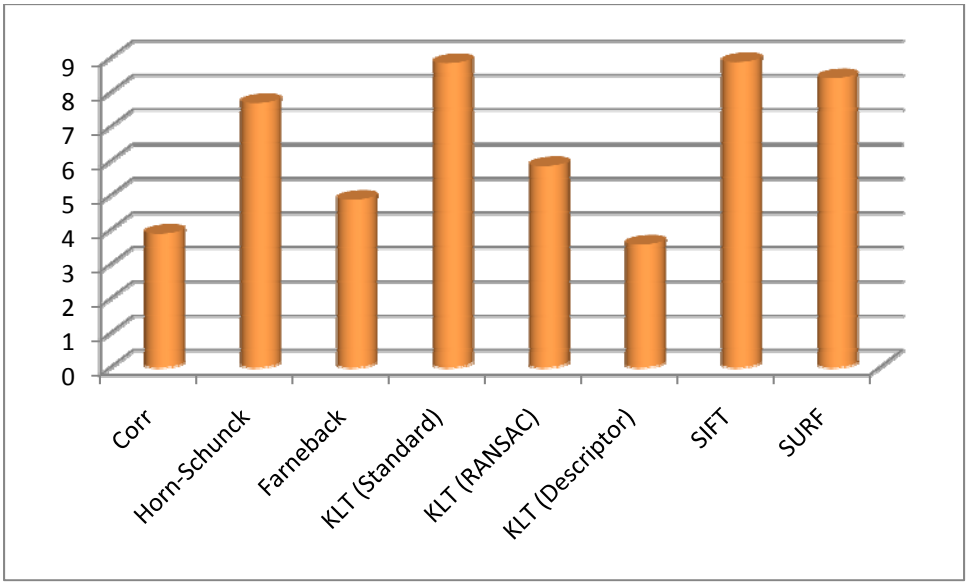

Figure 30: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the Sign dataset.

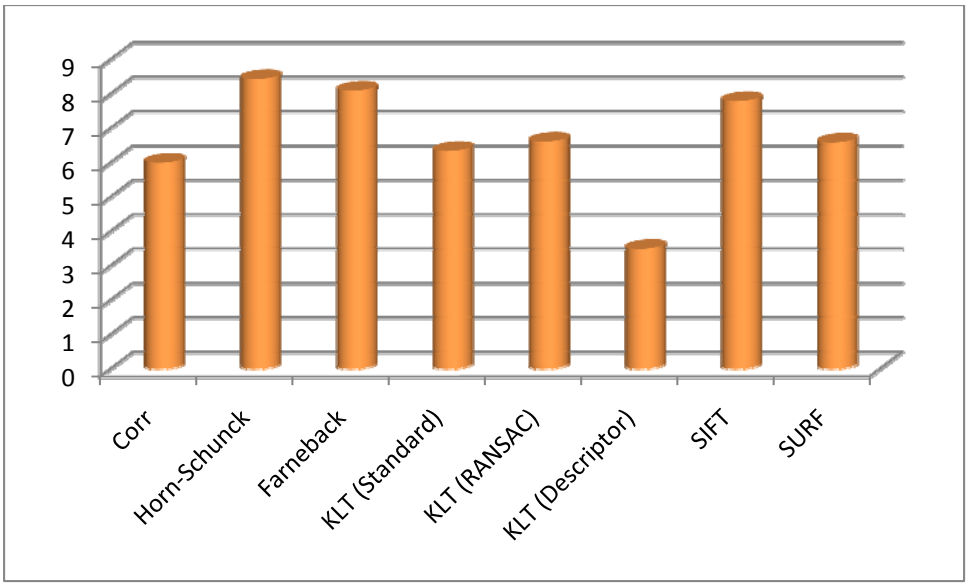

Figure 31: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the Woodpile dataset

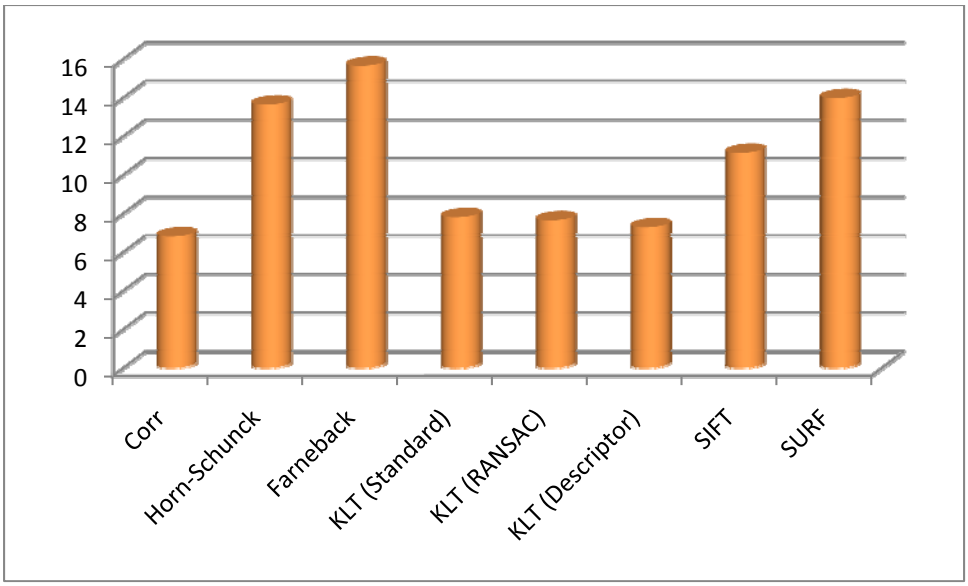

Figure 32: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the Silver Car dataset

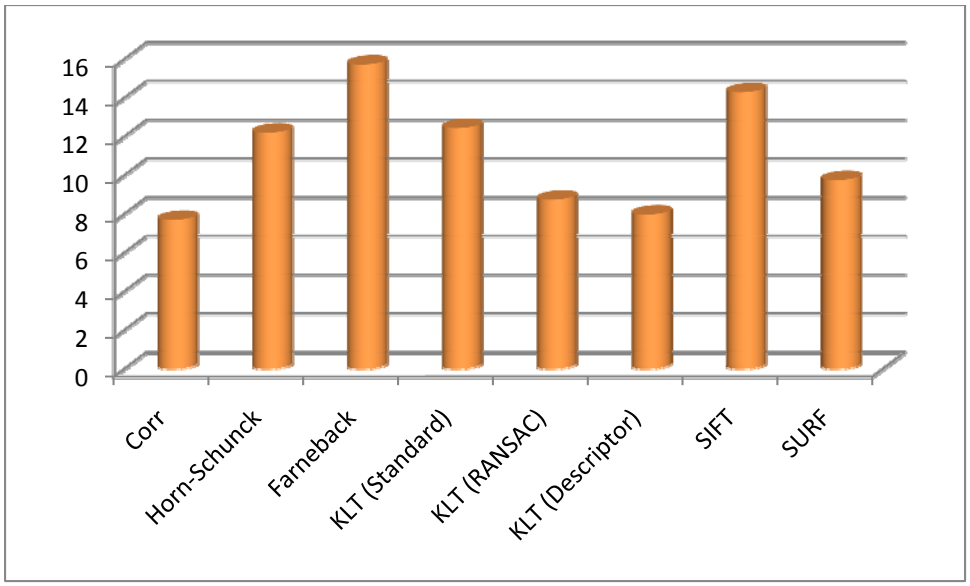

Figure 33: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the SUV dataset

Figure 34 is the aggregated mean of the Euclidean distance compared with the ground truth. The correlation and KLT (with a descriptor) performed the best.

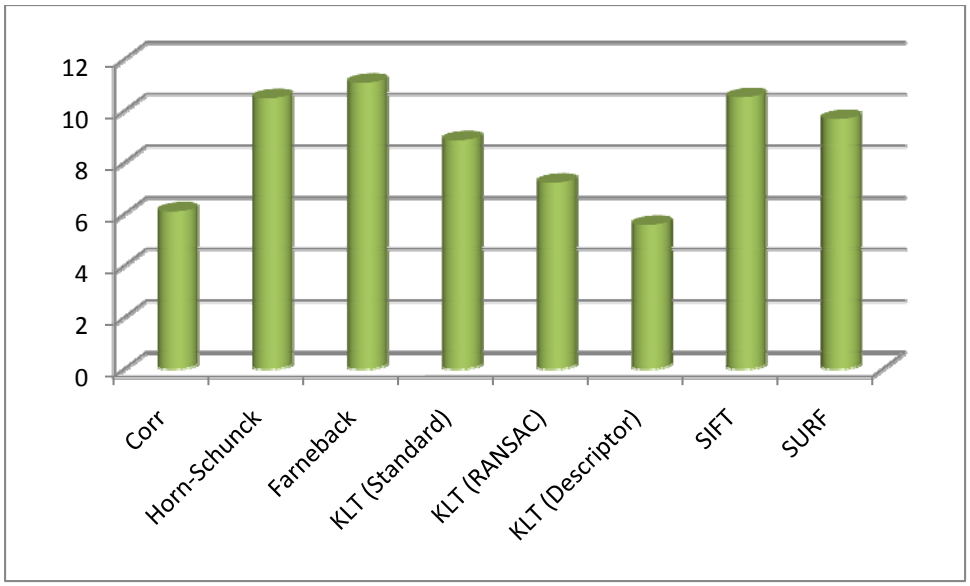

Figure 34: The aggregated mean of all four datasets compared with the ground truth data.

3.4.4 Effect of Accuracy with Image Compression Added

This metric is essentially the same as the accuracy metric described above except that each frame of the dataset was encoded with MPEG1 compression. This was chosen because it is typically the compression method used in transmitting a wireless video feed and is most like the video feed from the PackBot. Figure 35-38 shows the mean Euclidean distance from the ground truth of each of the four datasets with the added compression.

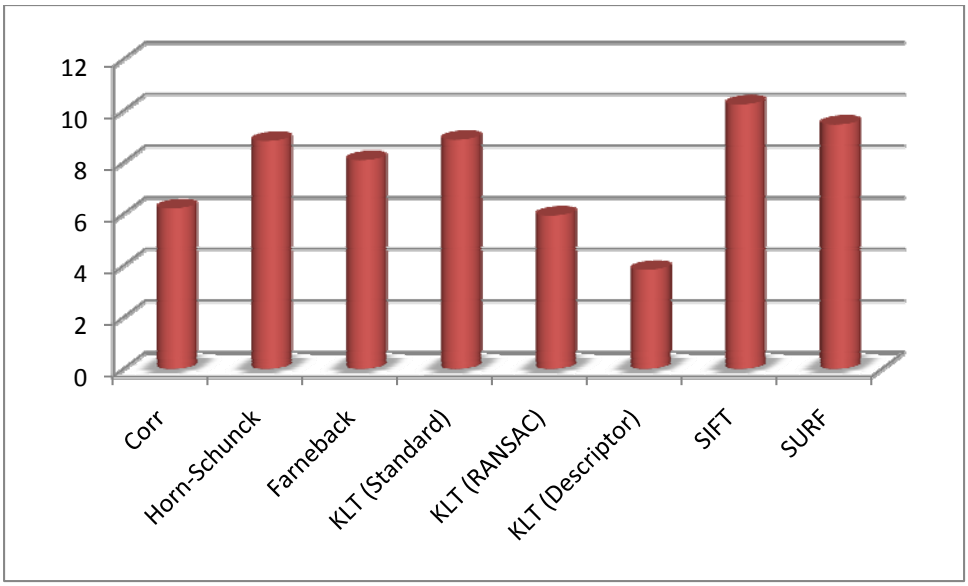

Figure 35: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the Sign dataset with added MPEG1 compression.

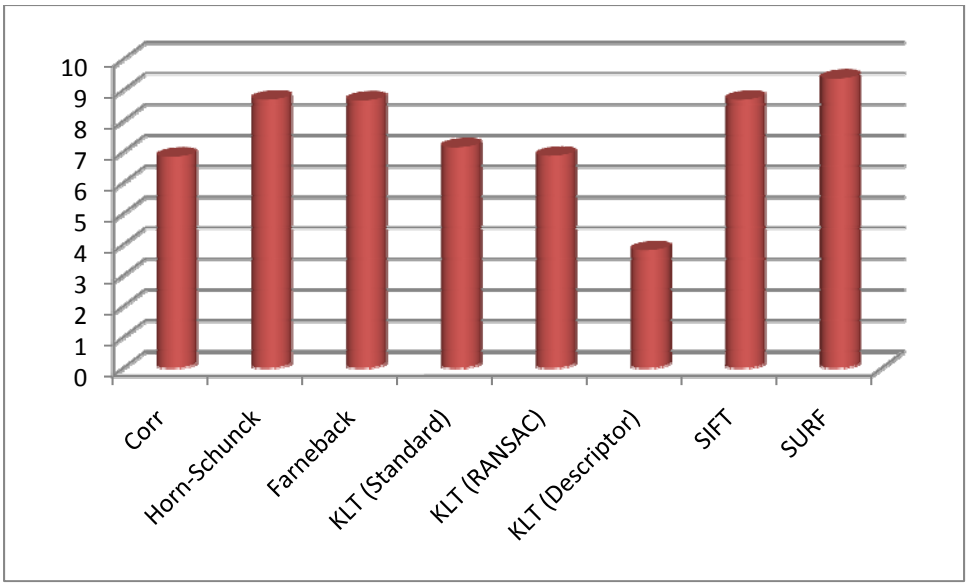

Figure 36: The mean Euclidean distance from the ground truth for each of the tracking algorithms for the Woodpile dataset with added MPEG1 compression.

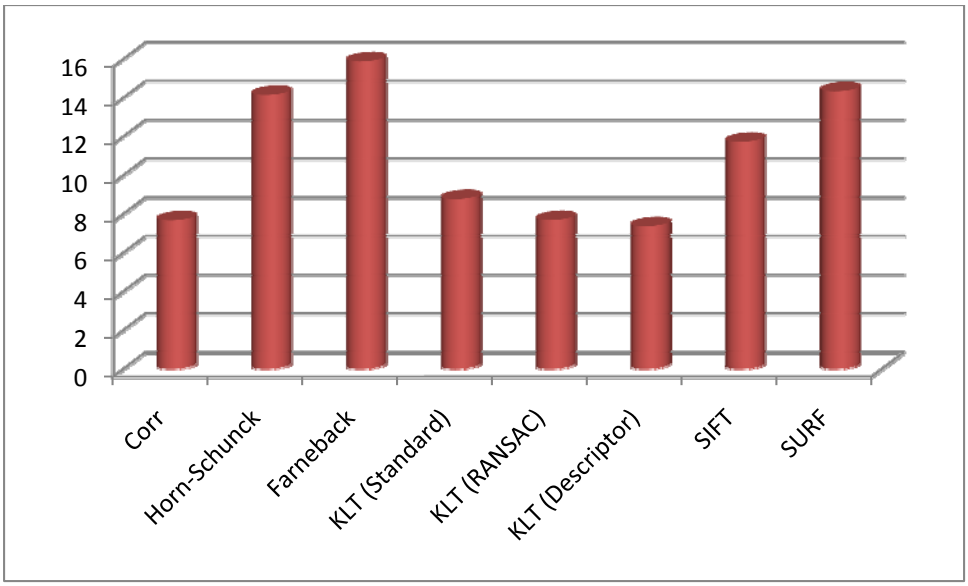

Figure 37: Ground Truth vs. the Eight Tracking Algorithms for the Silver Car dataset with Added MPEG1 Compression

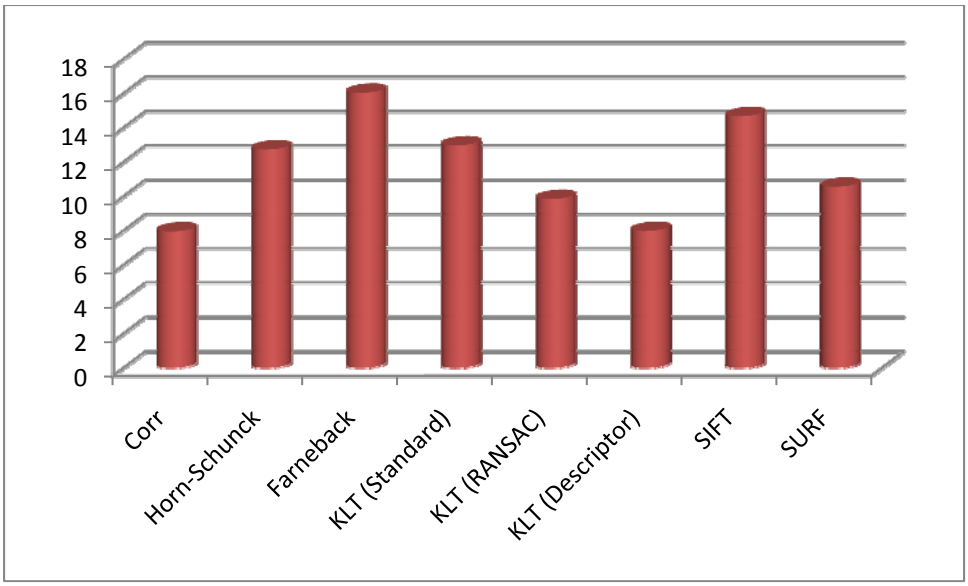

Figure 38: Ground Truth vs. the Eight Tracking Algorithms for the SUV dataset with Added MPEG1 Compression

Finally, Figure 39 shows the aggregate means of the Euclidean distance of the calculated goal point from the ground truth dataset. As can be seen, adding compression causes nearly every tracking algorithm to perform slightly worse than when using an uncompressed dataset.

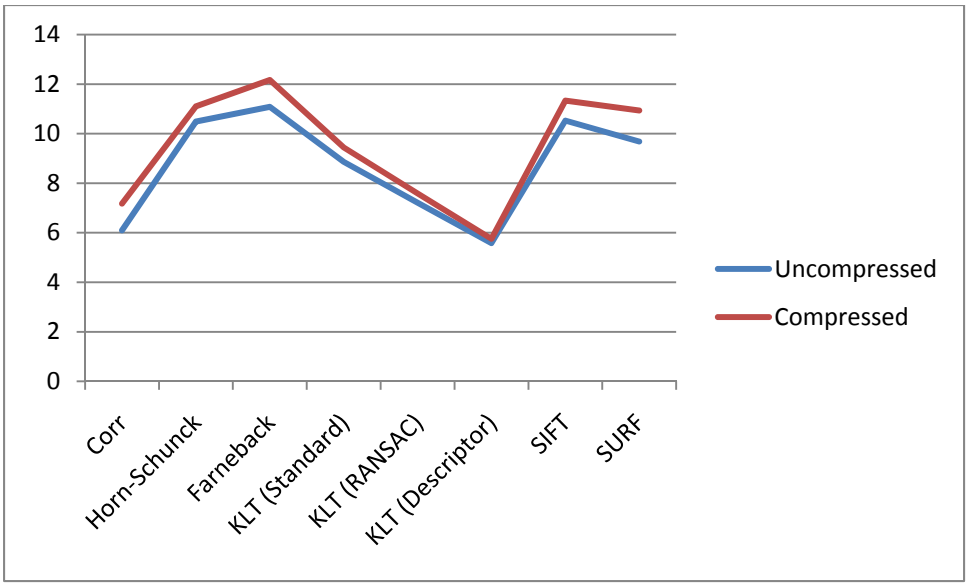

Figure 39: The aggregate mean of the Euclidean distance away from the ground truth dataset over the uncompressed and compressed datasets.

3.4.5 Determination of Winners

The algorithms were ranked according to the results in each of the above categories: speed, memory usage, accuracy, and how the accuracy of each algorithm was affected by compression. The results of which algorithm performed the best are shown in Table 4. The main concern for the visual servoing application was the ability of each algorithm to run in real-time and because of that, it was given a weight of 0.40 in the ranking calculation. Memory

usage and accuracy were each given a weight of 0.20 and how each algorithm handled compression was given a weight of 0.10;

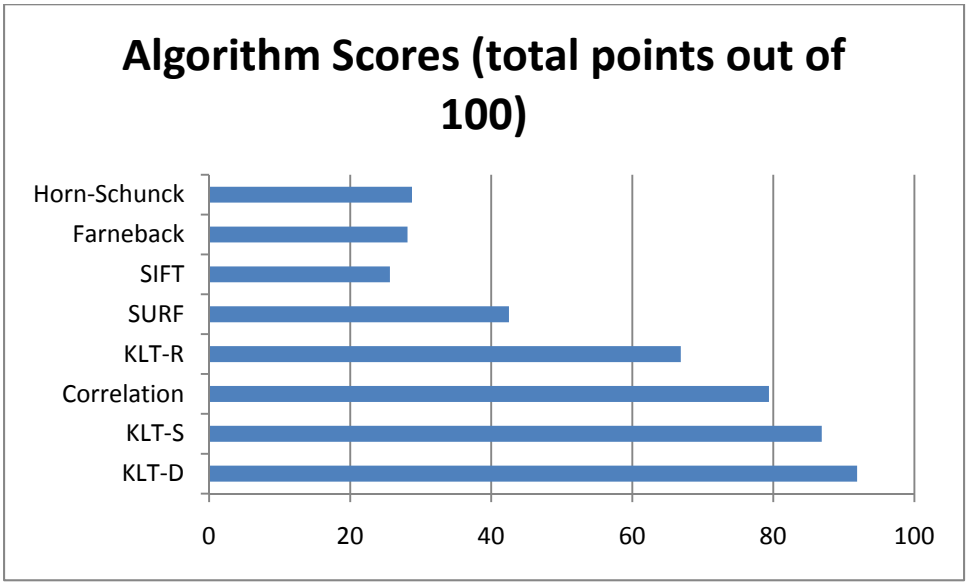

Table 4: The ranking of each algorithm on how they ranked in terms of speed, memory usage, accuracy, and the effect of compression on accuracy.

3.5 Discussion and Summary

This chapter presented the analysis of the tracking algorithms that were chosen in this study to find out which were robust enough to implement as the visual servoing method for this research. The work done by the creators of Make3D was extended by adding the ability to track feature points while virtually flying though the scene. Although this approach was a good start, it did not offer the ability to easily model real-world conditions. A large amount of data was collected by driving the robot, equipped with sensors and a camera, to create datasets used for offline analysis of the tracking algorithms. The large datasets were then carved into smaller subsets which were used in the final analysis.

This chapter also looks at a correlation tracker, optical flow, and blob tracking algorithms. Each algorithm has its own set of input parameters. A common test environment, TACTICAL, was developed that exposed each algorithm's variables in order to easily modify and see the results in real-time.

The metrics chosen included the computation time, the amount of memory used, and the accuracy of each algorithm compared to ground truth data. Compression was added to each dataset to simulate the video as if it had been received at the OCU of current military robots.

In the end, the accuracy among the tracking algorithms studied turned out to be within a few pixels of accuracy when aggregated across the datasets. The best KLT tracker had an average accuracy of 5.75 pixels compared to ground truth. The correlation had an average accuracy of 7.18 pixels compared to ground truth. The largest difference between the algorithms was their ability to execute in real time. It can be argued that with enough computational power, virtually any algorithm can be made to run faster. This is true to an extent, but in order for this technology to be adopted by the military the ability for it to run on existing hardware is an important consideration. Because of this, the two algorithms able to operate in real-time, correlation, and the KLT variants were chosen to be implemented on the PackBot. The next chapter describes the testbed creation for implementing the visual servoing and visual dead reckoning algorithms.

The most useful contribution of this aim was TACTICAL, the tracking algorithm environment. This environment consolidated many years of work in

computer vision into one environment. It was built with modularity in mind and can continue to be built upon in the years ahead.

CHAPTER 4

TESTBED CREATION

4.1 Introduction

This chapter describes implementing the tracking algorithms, along with several other algorithms, on the robot, creating the testbed to evaluate the hypothesis of this research. Section 4.2 explains the robotic platform used and the development GUI used to fine-tune the parameters of the system. Section 4.3 details the control methods that were implemented. Section 4.4 addresses the motion control algorithms used and Section 4.5 summarizes the contributions made in this work.

Figure 40 shows a broad overview of the two main components that encompass this aim. The first is the robot hardware, which in this case is an iRobot PackBot, and the second is the Operator Control Unit software developed to support the tracking algorithms.

It is important to note that although iRobot's PackBot was used in this dissertation, the implementation described in this chapter is to be regarded as fundamental research able to be applied to any robotic platform. The robot needs to be equipped with the sensors described in Chapter 2, namely a camera, an IMU, and a method of reliably calculating odometry. The exact details of how the robot operates, considered to be iRobot's proprietary information and protected under the International Traffic and Arms regulations, ITAR, and will not be discussed in this dissertation.

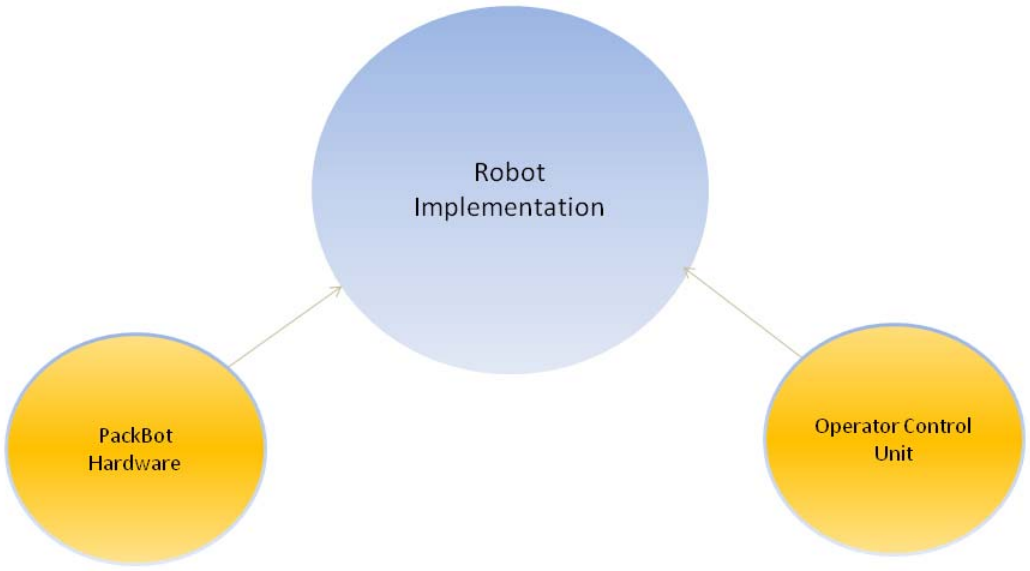

Figure 40: The two main components of the second aim.

4.2 Platform and Development GUI

The implementation of this system was done on an iRobot PackBot 510. A picture of it is shown in Figure 41. It was purchased under a Defense University Research Instrumentation Program (DURIP) grant. The PackBot is a fielded MTRS robot. By focusing on this platform, it ensured that the work remained relevant to the Army.

Figure 41: The iRobot PackBot 510 EOD that was used to implement this research.

Figure 42 shows a screen capture of the GUI developed to implement the supervisory control algorithms on the PackBot. Every parameter deemed important to either the tracking or motion control algorithm was placed on the GUI to expedite the implementation and allowed refinement of the parameters that worked best in the laboratory environment. The GUI displayed the current status of the computer, memory and process usage that the program was occupying. This allowed a quick visual inspection of what the current parameters did and allowed for tweaking to make tracking perform as well as possible. The developed GUI contained a large number of fields because every parameter important to either the semi-autonomous algorithms or to the motion control algorithm was added to the display. Figure 43 shows a larger view of the options for the KLT algorithm.

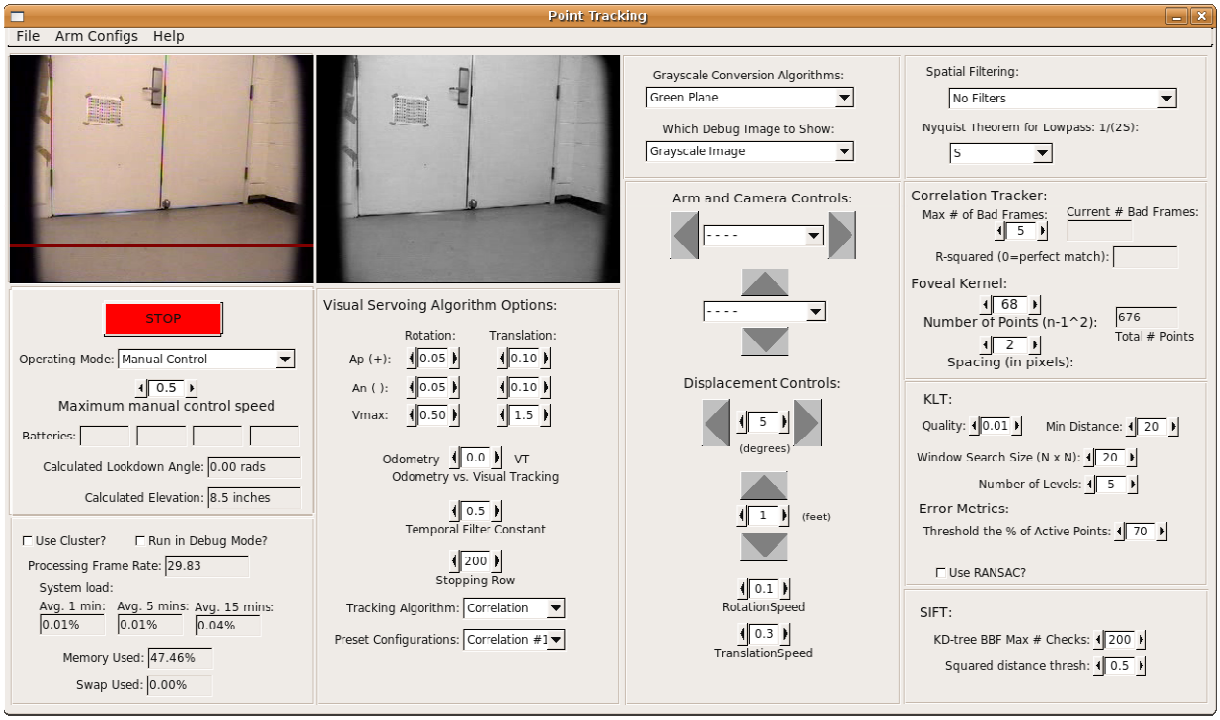

Figure 42: The development GUI that exposed all of the tracking and motion control algorithm's parameters.

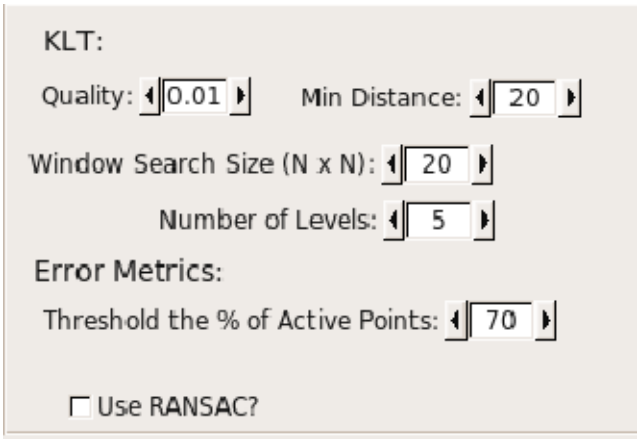

Figure 43: The options for KLT on the development GUI shown in more detail.

4.3 Control Methods

The control methods implemented and tested on the PackBot were rate-control teleoperation, displacement control, visual dead reckoning, and two

supervisory control methods, one based on the correlation tracker and the other on the KLT. In the following sections, each method is described in more detail.

4.3.1 Teleoperation

The rate-controlled teleoperation was implemented using an off-the-shelf game controller. Rotation and translation were controlled using one of the analog sticks. Rotation was controlled by the analog stick's x-axis and translation was controlled by the y-axis.

4.3.2 Displacement Control

Displacement control uses odometry to move the robot. Odometry feedback from the PackBot was used in this work. There were several implementations of displacement control developed. The first used four buttons on the gamepad. Rotation was a set amount, either two degrees to the left or right. Translation, also a set amount, and was either two feet forward or reverse. These values were determined empirically from the pilot testing, based on navigating to the target locations, which is explained in more detail in the next chapter.

The second version of displacement control used a GUI with a dial for rotation and sliders for translation control. Figure 44 shows the GUI designed for the second version. The dial enabled the subject to rotate the robot up to 180 degrees in either direction. The sliders allowed translation control in small, medium, or large increments. The left-hand slider moved the robot up to 2

meters for finer control, the middle slider moved the robot up to 8 meters, and the right-hand slider moved the robot up to 32 meters.

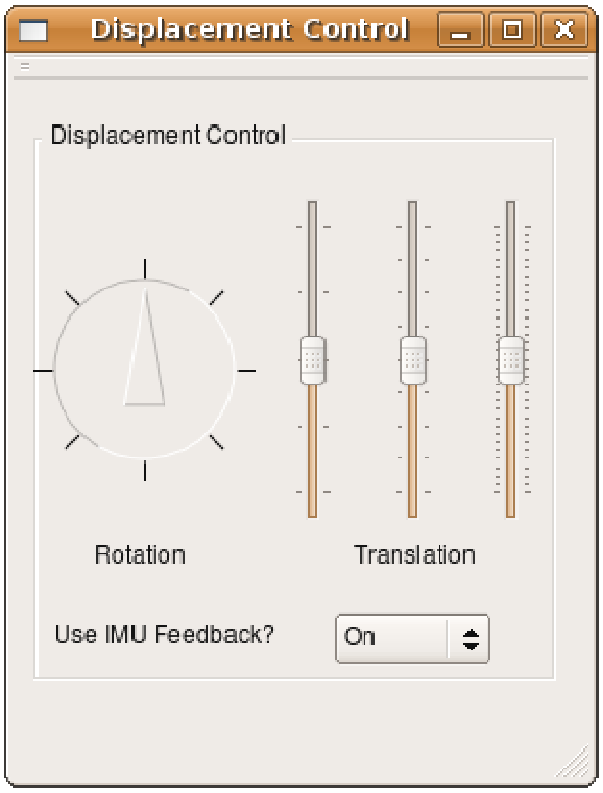

Figure 44: The GUI used for displacement control. The dial controlled rotation and the three sliders controlled translation.

4.3.3 Visual Dead Reckoning

Visual dead reckoning uses the odometry data in the PackBot, coupled with internal properties of the drive camera, along with kinematics of the arm, to go to the user specified location without the need of tracking features. Visual dead reckoning first rotates and then translates to the goal point.

First, for rotation, if the initial goal point designation is defined as C_0, and C_m denotes the middle of the image, then the pixel distance to rotate is given in

(4.1). Next, if *s* is the horizontal IFOV, then how far the robot has to rotate is given by (4.2).

$$C0 - Cm \tag{4.1}$$

$$s * (C0 - Cm) \tag{4.2}$$

Once the robot has finished rotating, it begins translating to the goal. The angle between vertical and the bottom of the image is given as *Ab* and may be calculated by (4.3), where s is the vertical IFOV. Next, if the row position of the initial goal point is given as *R0*, the stopping row is given as *Rm*, and *H* equals the height of the camera as determined by the forward kinematics, then the initial distance remaining is given by (4.4). Once this value has been found, the odometry is read from the robot and once the distance travelled exceeds this estimate, the goal point has been reached.

$$s * \frac{rows}{2} \tag{4.3}$$

$$H * \tan((Ab + s * R0) - \tan(Ab + s * Rm)) \tag{4.4}$$

4.3.3.1 IMU Correction

Chapter 2 discussed the error associated with using odometry. The error for translation is usually negligible but can be substantially off for rotation. Two versions of visual dead reckoning were implemented. One version had IMU correction and the other did not.

The first version used only odometry and the other used the IMU to correct heading. The developed algorithm uses an offset that allows the global theta to be rotated into the coordinate system of the robot instead of a coordinate system based on true north. If the IMU readings became unstable due to communications dropout, which did occur due to interference problems with other wireless networks, the algorithm went back to calculating heading from odometry.

4.3.4 Visual Servoing

The two algorithms used for testing visual servoing were correlation and KLT. These two algorithms were selected for implementation because they could run in real-time on the OCU computer. The other algorithms discussed in the previous chapter could not run in real time, which was the heaviest weighted component in the decision of which algorithms to implement for visual servoing.

Tracking was done using the video received at the OCU after the image was compressed and transmitted over the network. The blocking artifacts in the video feed were noticeable and as a result, tracking was negatively impacted. A method was developed to grab the uncompressed frame directly from the PackBot and run the visual servoing algorithm on-board. In this version, the OCU sent the x and y coordinates of the goal point the user clicked on to the on-board computer and then only status messages of the visual servoing algorithm were relayed back to the OCU. This method did not end up working because of a hardware problem with the PackBot unable to be resolved. Because of this, the visual servoing algorithms continued to run on the OCU.

The basic control loop for the visual servoing algorithms is shown in Figure 45. At a very high level, the control is very simple. The robot remains stationary until a goal point is designated by the operator. Once the goal point has been entered using the mouse, tracking is initialized and the goal point is tracked from frame to frame. If tracking is stable, motion commands are sent to the robot to control its steering and throttle. The goal point is kept in the center of the screen until the robot has advanced to a stopping point in the scene, which is shown as the blue line in Figure 46. Figure 46 also shows the simplified interface developed for subject testing. The only methods the subject had to move the robot, when using the visual servoing or visual dead reckoning algorithms, were the video feed window and the mouse. Please note from this point forward, if a figure is shown that uses the video feed from the PackBot's camera, it is degraded slightly in an abundance of caution over ITAR restrictions and iRobot's intellectual property rights.

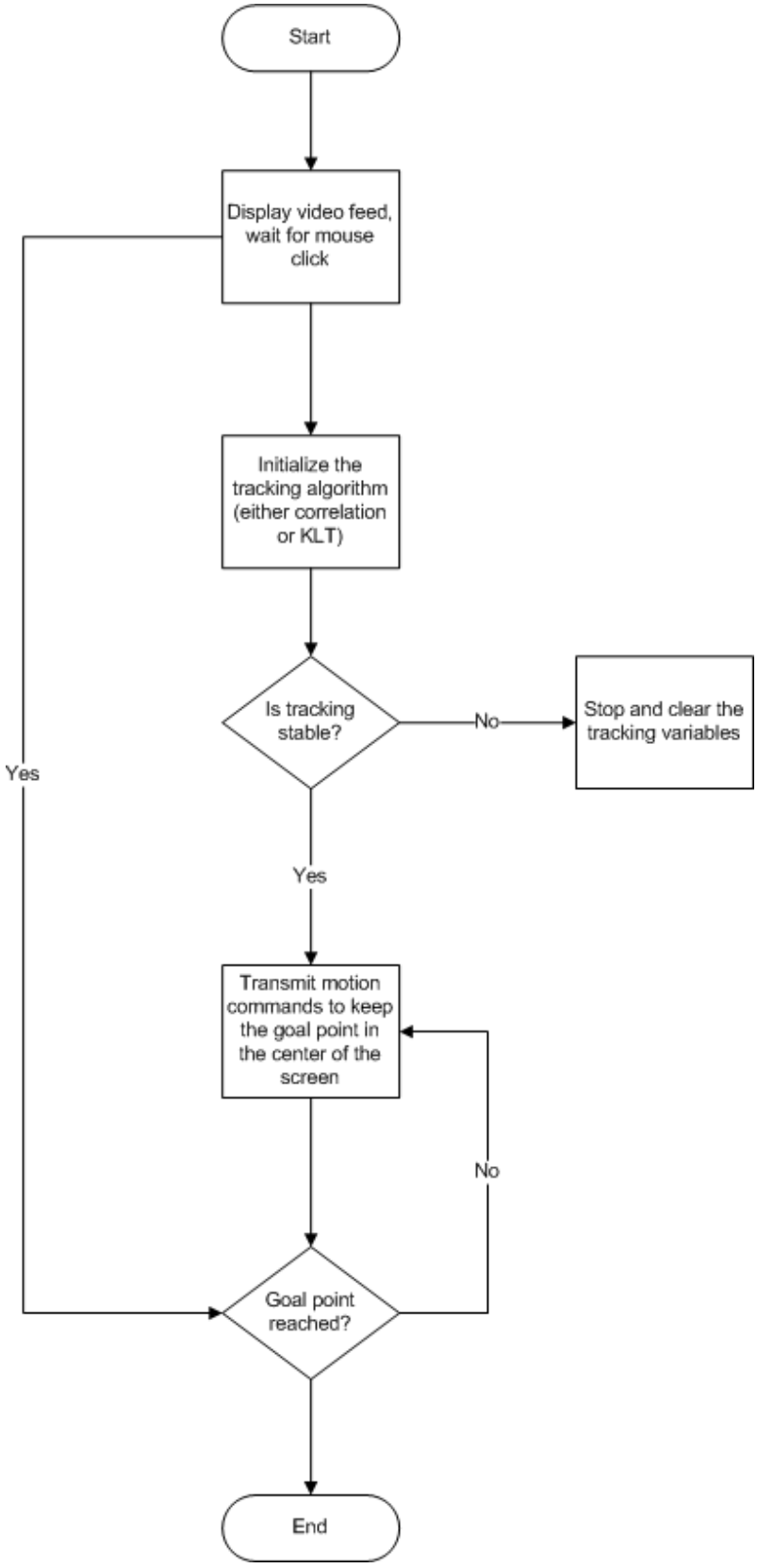

Figure 45: The basic control loop for visual servoing.

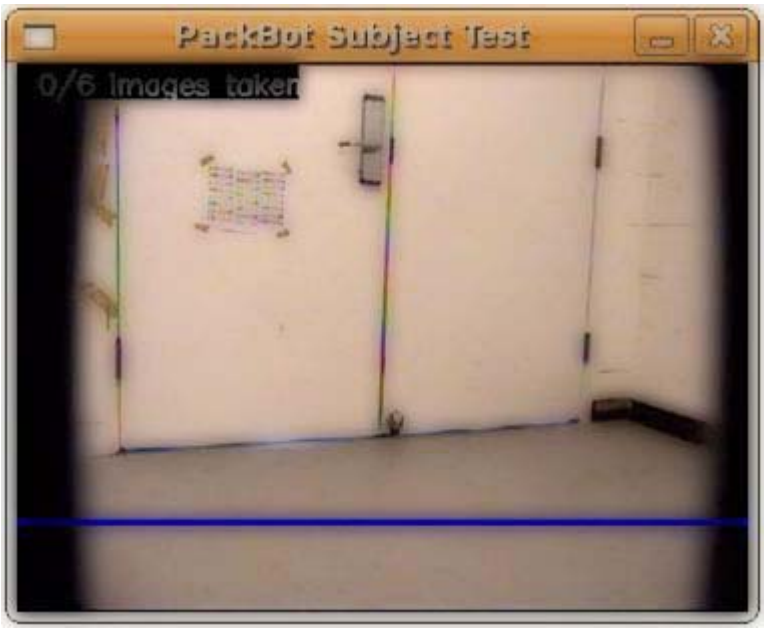

Figure 46: This is interface for visual servoing. The blue line indicates the stopping row. Once the tracked point meets the blue line, the robot stops moving.

4.4 Motion Control

The rate commands for rotation and translation were driven by the estimation of distance and speed. The conceptual profile was a constant acceleration up to either a maximum rate or a minimum distance, whichever came first, followed by a constant deceleration to the goal point.

In the implemented system, Ap was the maximum rate increment in one time cycle. The nominal rate decrement in one time cycle was given as *An* and *Vmax* was the maximum rate command. The threshold distance to start decelerating was given as *Dn*, and calculated by (4.5).

$$\frac{v\max^2}{2 * An} \tag{4.5}$$

Table 5 shows the four possible states of the motion control algorithm. In these equations, *d* is the current estimate of the distance and *r* is the current estimate of the rate. Case A is when the robot is still far from the goal but is travelling too fast. Case B is when the robot is far from goal but needs to speed up. Case C is when the robot is close but going too slow and Case D is when the robot is close but going too fast.

Case	Action
A	If d > Dn and v >= Vmax, set v = Vmax
B	If d > Dn and v < Vmax, set v = min(Vmax, v + Ap)
C	If d <= Dn and v <= d * An / Vmax, set v = min(d * An / Vmax, v + Ap)
D	If d < Dn and v > d * An / Vmax, set v = min(d * Vmax, max(0, v – An))

Table 5: The four possible cases in the implemented motion control algorithm.

The motion control algorithm was developed so the robot would accelerate when the goal point was far away and decelerate as the goal point became closer. If the goal point was far off in the scene and the robot was only capable of going one speed, it would take the robot longer to reach its destination.

4.5 Discussion and Summary

This chapter has detailed the software testbed developed to implement visual servoing and visual dead reckoning. Again, the work developed here was significant because it can be applied to any robot. In this research, the

algorithms were implemented on a PackBot. The result of this aim produced a working system. The development GUI was used to easily find the best parameters for each algorithm and allowed moving forward with the human-in-the-loop experiments, which are described in more detail in the next chapter.

CHAPTER 5

HUMAN-IN-THE-LOOP TESTING

5.1 Introduction

The goal of the human-in-the-loop test was to see how the supervisory control algorithms performed relative to teleoperation at different levels of dropout. Section 5.2 discusses the materials and methods used in the human-in-the-loop testing, Section 5.3 presents the results, and Section 5.4 ends with discussions and conclusions that can be drawn from the conducted experiment.

Figure 47 shows the goal of the third and final aim to be discussed in this chapter. This aim deals with the human-in-the-loop testing done to determine if there was any measureable difference in using the supervisory control algorithms, compared to teleoperation. Artificial degradation was added to the video stream, which is described in more detail later. This chapter also describes the independent variables used for the experiments and describes the metrics used for the presented results.

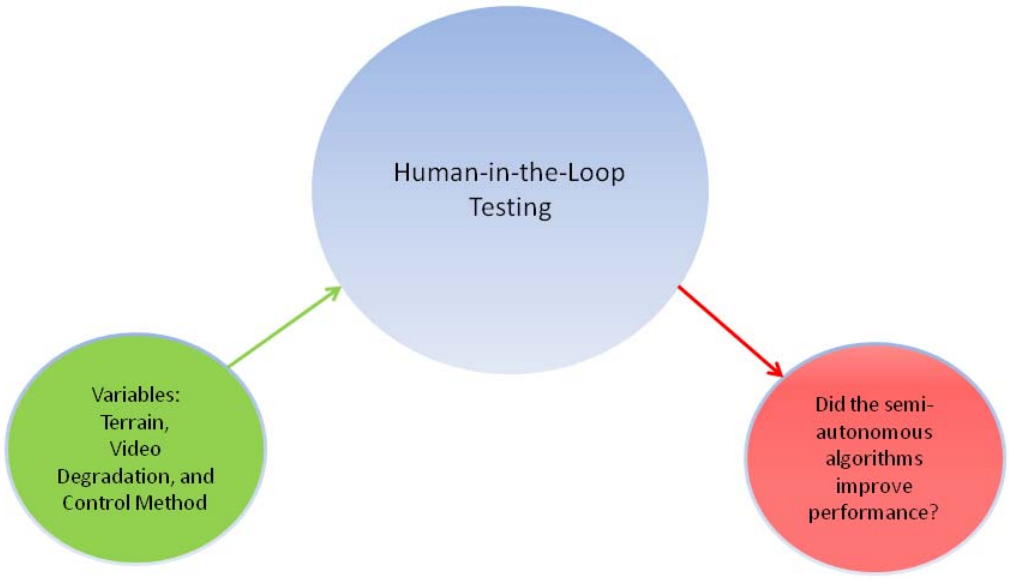

Figure 47: This aim determined if the developed semi-autonomous algorithms performed better than teleoperation.

5.2 Methods

This section describes all of the details of the experiment. It describes the participants, the design of the course, and what the assigned task was. The design of the experiment is described along with the dependent and independent variables. The trial procedures and how the data were verified and finally, the results are presented and discussed.

5.2.1 Participants

There were six participants, all students at Wayne State. All of the participants had normal/corrected vision. No subject had any cognitive impairment. All subjects had prior experience using a computer and playing video games.

5.2.2 Course Design

Three courses were constructed that looked similar to what is shown in Figure 48. The first course was made out of masking tape applied to the floor. The second course was designed to simulate small bumps and was made out of 1x2s as the bumps, with 2x4 as the rails. The third course simulated large bumps and was made out of 2x4s as the bumps and 2x4s as the rails. The total length of the course was forty-five feet.

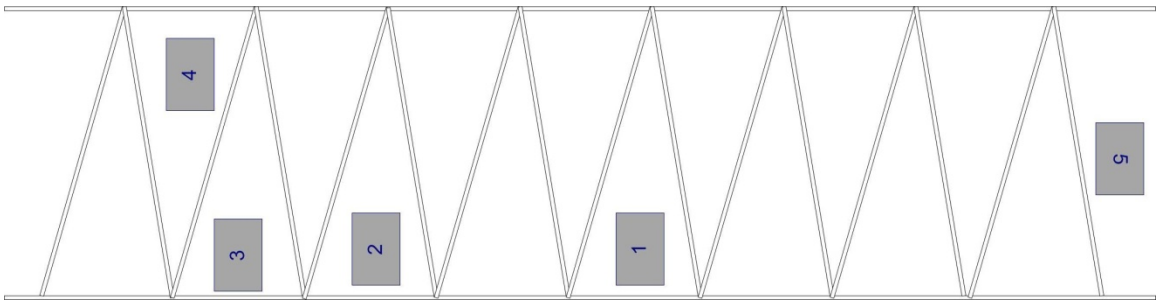

Figure 48: The layout of the courses with the five inspection targets.

The course was designed to have the robot traverse and come back. There were four stops on the down portion of the track. The four stops on the down portion of the track were approximately 22, 11, 5, and 2 feet apart. The fifth stop, going back to the starting position, had a distance of 45 feet. Figure 49

shows an overhead view of the three courses that were created in the laboratory. Figure 50 is another view of the PackBot going over the 2x4 course.

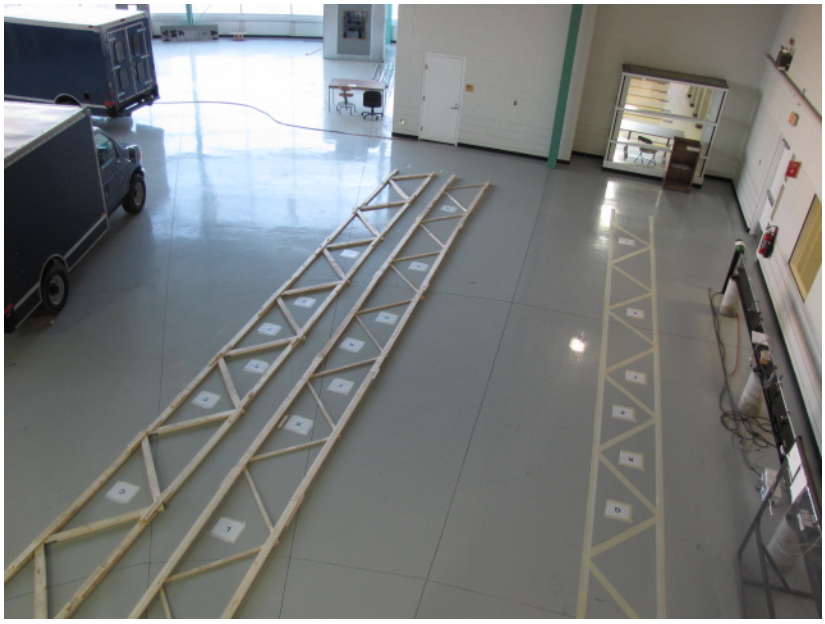

Figure 49: The overhead view of the three courses: flat (with tape), 1x2s, and 2x4s.

Figure 50: The PackBot going over the 2x4 course during subject testing.

5.2.3 Inspection Tasks

Each participant was asked to move the robot to each of the five targets, using each of the different control methods, while keeping the robot inside the rails. Once the participant felt the robot was positioned correctly, they were instructed to take a snapshot. A running count of how many snapshots had been taken was shown in the upper-left corner of the screen, as shown in Figure 51. This was done in order to provide an easy method of analyzing the data after the experiments were done. The post-processing process looked at the timestamp of when the subject took the picture to break each run up into the different segments. There were several occasions in the data files where this did not happen and the subject took too many snapshots by accident. In these cases,

the odometry data were used to determine when the robot started to move, indicating a new goal point was issued.

For the supervisory control algorithms, the participant was told that they could issue an emergency stop to the robot at any time by pressing the space bar on the keyboard, but the goal was not to rely on it because ideally, both visual servoing and visual dead reckoning should go exactly where the subject clicked. If the subject had to press the emergency stop many times, it indicated that the semi-autonomous algorithms were not performing well. This is a metric that was used later on and will be discussed in more detail.

5.2.4 Dropout Rates

Simulated degraded communications were introduced by corrupting data packets. A corrupted data packet is one that cannot be decoded. The data containing control messages, from the OCU to the PackBot, and data packets containing the video feed, from the PackBot to the OCU were both artificially corrupted. In the current fielded system, when a corrupted video packet is received, a black frame is shown. In this implementation, the last good frame was displayed. The data corruption was modeled as a Bernoulli process, i.e. all of the packets had an equal probability of being corrupted. There were four levels of communication degradation implemented: 0, 3/8, 9/16, and 3/4 seconds.

5.2.5 Pilot Testing

Several rounds of pilot testing were conducted to find the best settings to use. The objective of the pilot test was to make sure all of the control methods were working properly along with the display dropout function. The data logging software was verified to be working and the data captured were scrutinized to ensure they would be sufficient for processing and analysis later. Also during the pilot testing, the procedures were double checked to make sure they were clear. The development GUI was used to find the proper acceleration and deceleration rates to use for each control method. This made sure that the visual servoing algorithms performed with the optimal parameters. The other important outcome of the pilot studies were to verify dropout rates picked were difficult enough to the subject that there could definitively be a point where the semi-autonomous algorithms performed better than teleoperation.

The GUI used in the first round of pilot testing was a plain video feed that required the user to designate two points. The first click designated the row and column of the goal point and the second click designated the stopping row. It was determined this confused the subjects and the interface was changed to accept a single click for the row and column of the goal point and the stopping row was shown with a blue line, as shown in Figure 51. The stopping row was adjustable by using the up and down arrows on the OCU's keyboard. A visual cue was added to indicate how many images the subjects had taken.

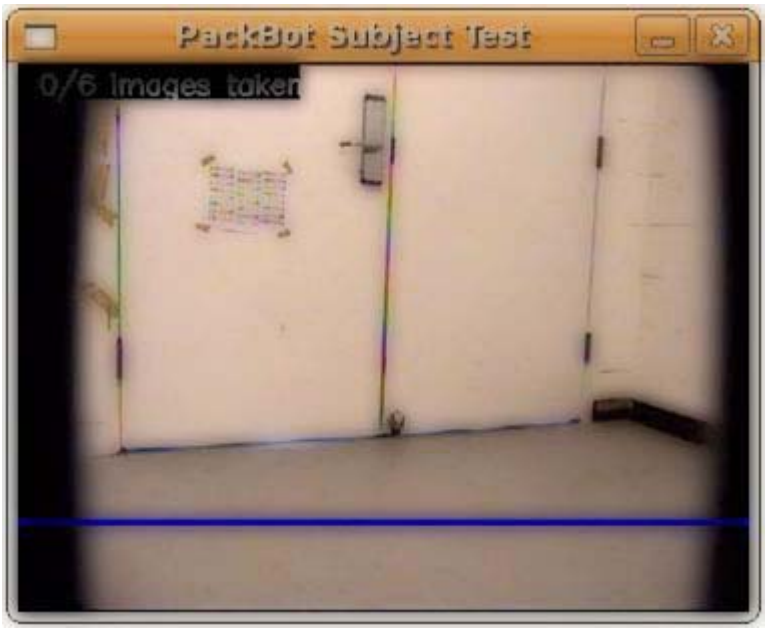

Figure 51: The simple GUI used in pilot and subject testing.

5.2.6 Trial Procedure

Prior to beginning, each subject was given the same presentation detailing the objectives of the study. A graphic of the course was first used to explain where the target locations were and the course was also walked with the subject to show where each target was located. The subject was given ample time to use each control method before the actual test and indicated to the test proctor when they were comfortable enough to proceed.

The subject was positioned in an area having no direct line of sight to the robot, as shown in Figure 52. Each subject was instructed not to turn around and look at the PackBot while they were controlling it. At the end of a course run, each subject was asked to enter a difficulty rating on a scale from one to ten, where one meant easy and ten meant difficult. This provided the examiner with a difficulty rating for each control method, dropout rate, and course roughness.

There were times when the PackBot would become unresponsive due to communication interference or it would stop because of discharged batteries. If this occurred during a trial run, the trial was repeated.

Figure 52: Another view of the experiment in the highbay. Each participant was positioned in such a way that the PackBot could not be seen.

The subjects were asked to complete a task, namely to drive to a target on the ground and stop the PackBot when the target is still visible in the display and is within reach of the PackBot's arm. An example of a "good" stopping point is shown in Figure 53. This position was chosen because the target was in reach of the grippers on the PackBot's arm. In a realistic setting, this would be similar to driving up to something buried in the ground that a warfighter wants to examine with the robot.

Figure 53: The "good" stopping distance from a target that each subject was trained to stop at.

The courses constructed from 1x2s and 2x4s introduced motion blur to the PackBot's camera. Figure 54 shows an example image taken from the PackBot's camera as it traversed the 2x4 course. The image containing the motion blur, combined with the compression artifacts, were input into the visual servoing algorithms that relied on tracking features from frame to frame.

Figure 54: The view from going over the 2x4 course.

5.2.7 Experimental Design

The experiment was run with six subjects. For each subject, the test was blocked by control method: teleoperation, visual dead reckoning, visual servoing using the correlation tracker, and visual servoing using the KLT tracker. Each of these blocks was then subdivided into four blocks by the dropout rate. Each of these blocks consisted of runs on each of the three courses. Each subject ran a total of 48 courses for a total of 288 course runs over each of the six subjects. Each subject took between 4 and 6 hours to complete all runs and each subject completed the test in a single block of time, i.e. no one came back at a later date to complete the test.

5.2.8 Data Validation

The first pass of the data occurred before the subject left. This test made sure that the data files had all been properly recorded. The data parsing program used the timestamps from when the operator took the picture based on when they felt the robot was positioned correctly. The operator would sometimes accidentally press the button too many times. If the data reduction program ran into this scenario, it would automatically try to combine the timestamps based on movement of the robot. There was a field in the reduced file that indicated when this happened so the result could be manually verified to make sure nothing was lost.

The difficulty scores from of each of the runs were stored separately from data collected from the robot. These had to be combined at data reduction time. An inspection was made of each record to make sure the difficulty ratings from the database were brought over correctly in the final file. The reduced file was also visually inspected to make sure all of the fields were within the normal range, i.e. the angles from the IMU readings were all between 0 and 360 degrees.

5.2.9 Aggregation and Analysis

This section presents the aggregation and analysis of the data collected from the subject testing. The items examined include the difficulty rating that each participant gave, the number of seconds it to took to reach each target, and the number of emergency stops. This is a within-subject design with subjects used as replicates. The rest of section 5.2.9 contains two-way ANOVAs. The

values for the course roughness in the supplemental figures and tables are: 0 = flat, 1 = 1x2 course, and 2 = 2x4 course. The values for control method are: 0 = teleoperation, 4 = visual dead reckoning, 5 = correlation, and 6 = KLT. The values for the dropout rate are: 0 = no delay, 1 = 3/8, 2 = 9/16, and 3 = ¾ seconds.

5.2.9.1 Difficulty Rating

After each run, the participant was asked to rank the difficulty on a scale from 1 to 10, where 1 meant easy and 10 meant difficult. Table 6 shows the results of a two-way ANOVA of difficulty rating as a function of dropout and course roughness ($F_{2,1716}$ = 24.73, p=0.0000). The difficulty rating increased as the roughness of the terrain increased. The difficulty rating increased as the dropout rate increased as well. There was no significance in the interaction between dropout rate and course roughness. Figure 55 shows the box plot of the two-way ANOVA results. Figure 56 shows the mean values of the difficulty ratings by dropout rate and course roughness. The 3/4 dropout rate had the highest average difficulty rating at 5.229 and the 2x4 course had the highest average difficulty rating at 5.339.

Two-way ANOVA: difficulty_rating_1easy_ versus course_number, dropout_rate

Source	DF	SS	MS	F	P
course_number	2	295.1	147.536	24.73	0.000
dropout_rate	3	173.6	57.875	9.70	0.000
Interaction	6	15.0	2.502	0.42	0.867
Error	1716	10238.2	5.966		
Total	1727	10721.9			

S = 2.443 R-Sq = 4.51% R-Sq(adj) = 3.90%

Table 6: Two-way ANOVA results of difficulty rating as a function of course roughness and dropout rate.

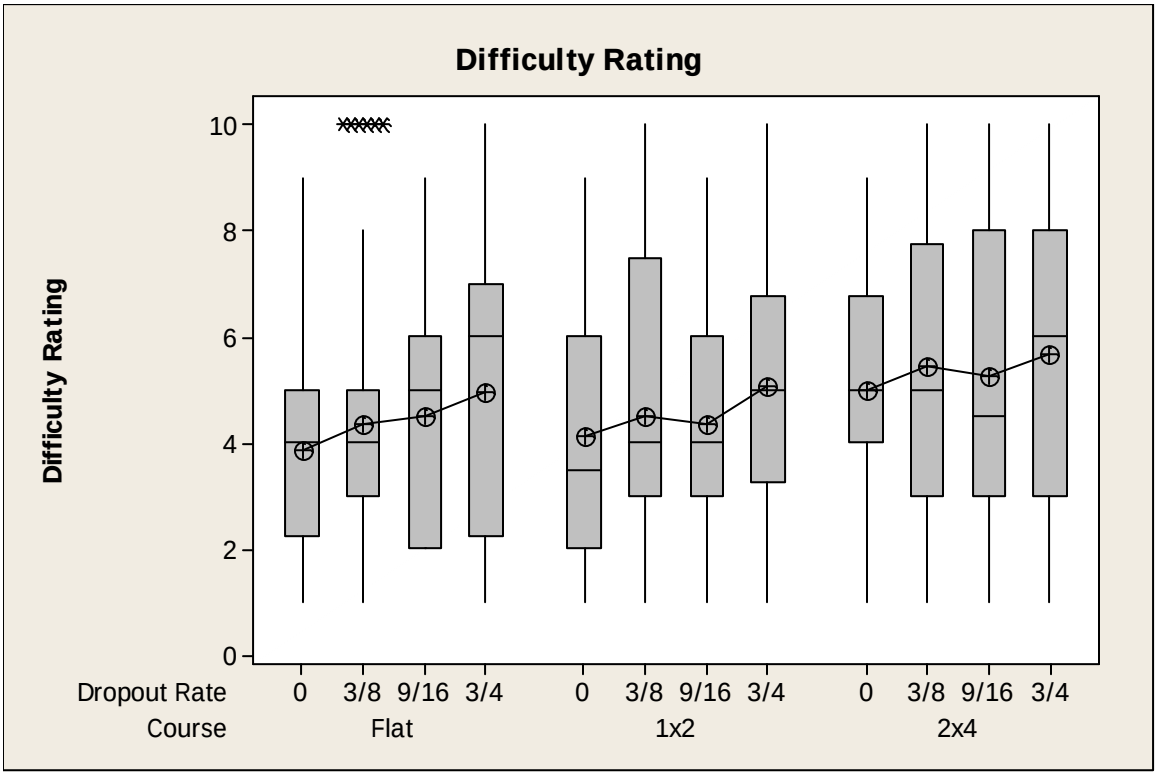

Figure 55: Box plot of difficulty rating as a function of dropout rate and course roughness.

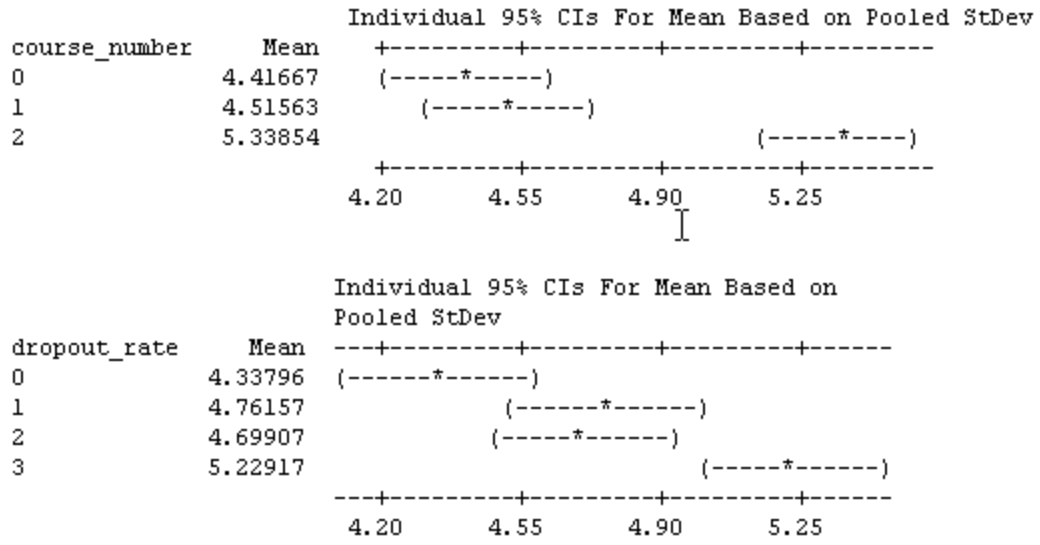

Figure 56: The mean difficulty ratings for course roughness and dropout rates.

Table 7 shows the results of a two-way ANOVA of difficulty rating as a function of course roughness and control method ($F_{2,1716}$ = 26.46, p=0.0000). There was significance in the interaction between control method and course roughness. The mean difficulty rating for visual dead reckoning was the lowest of all control methods at 4.03. Figure 57 shows the box plot of the two-way ANOVA results. Figure 58 shows the mean values of the difficulty ratings by course roughness and control method. The mean value of the difficulty rating increased as the course became rougher.

Two-way ANOVA: difficulty_rating_1easy_ versus course_number, control_method

Source	DF	SS	MS	F	P
course_number	2	295.1	147.536	26.46	0.000
control_method	3	664.4	221.472	39.72	0.000
Interaction	6	193.6	32.259	5.78	0.000
Error	1716	9568.9	5.576		
Total	1727	10721.9			

S = 2.361 R-Sq = 10.75% R-Sq(adj) = 10.18%

Table 7: Two-way ANOVA results of difficulty rating as a function of course roughness and control method.

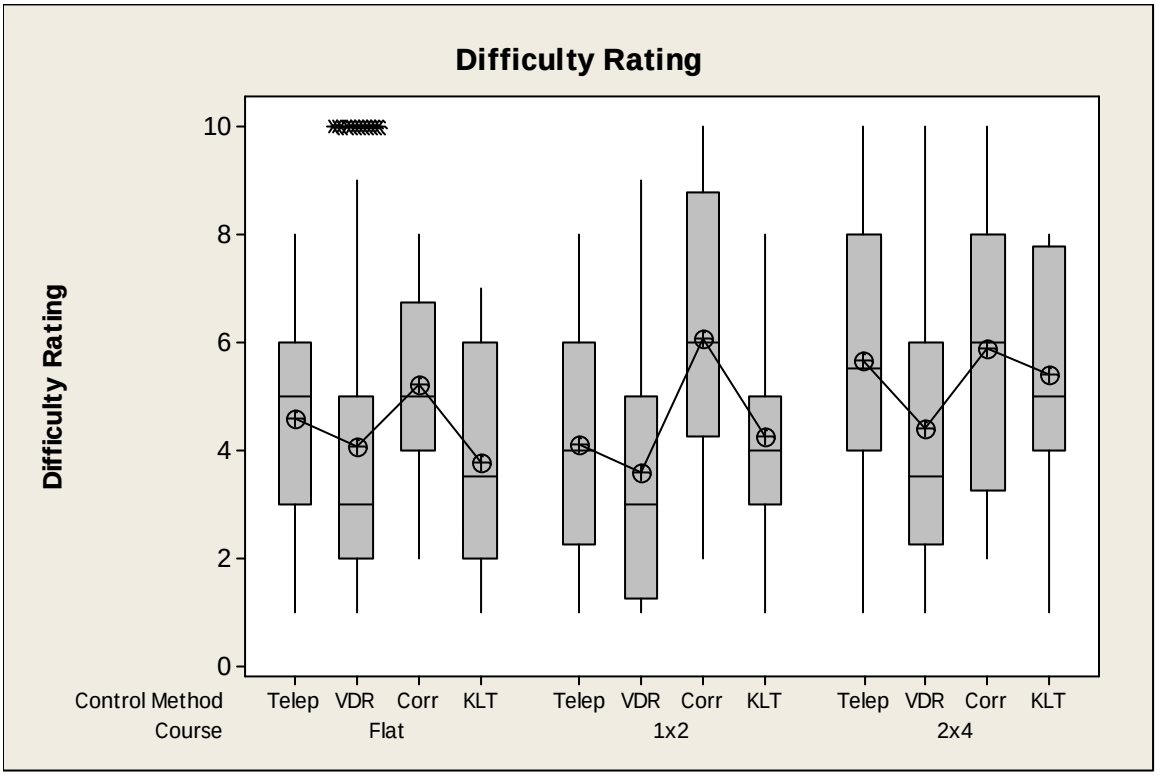

Figure 57: Box plot of difficulty rating as a function of control method and course roughness.

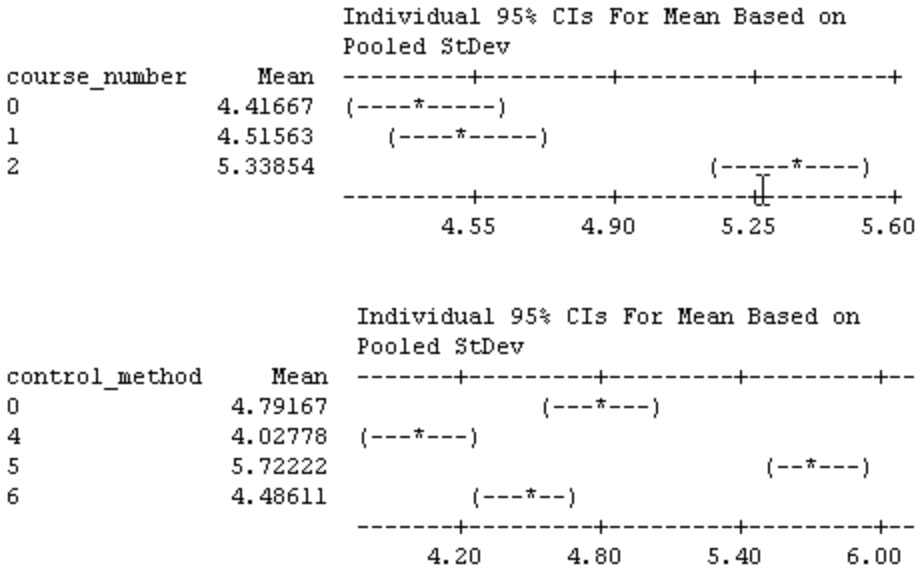

Figure 58: The mean difficulty ratings for course roughness and control method.

Table 8 shows the results of a two-way ANOVA of difficulty rating as a function of control method and dropout rate ($F_{3,1712}$ = 39.04, p=0.0000). The difficulty rating for visual dead reckoning was the lowest across all courses and control methods. There was significance in the interaction between control method and dropout rate. Figure 59 shows the box plot of the two-way ANOVA results. .Figure 60 shows the mean values of the difficulty ratings by control method and dropout rate.

Two-way ANOVA: difficulty_rating_1easy_ versus control_method, dropout_rate

Source	DF	SS	MS	F	P
control_method	3	664.4	221.472	39.04	0.000
dropout_rate	3	173.6	57.875	10.20	0.000
Interaction	9	171.1	19.011	3.35	0.000
Error	1712	9712.8	5.673		
Total	1727	10721.9			

S = 2.382 R-Sq = 9.41% R-Sq(adj) = 8.62%

Table 8: Two-way ANOVA results of difficulty rating as a function of dropout rate and control method.

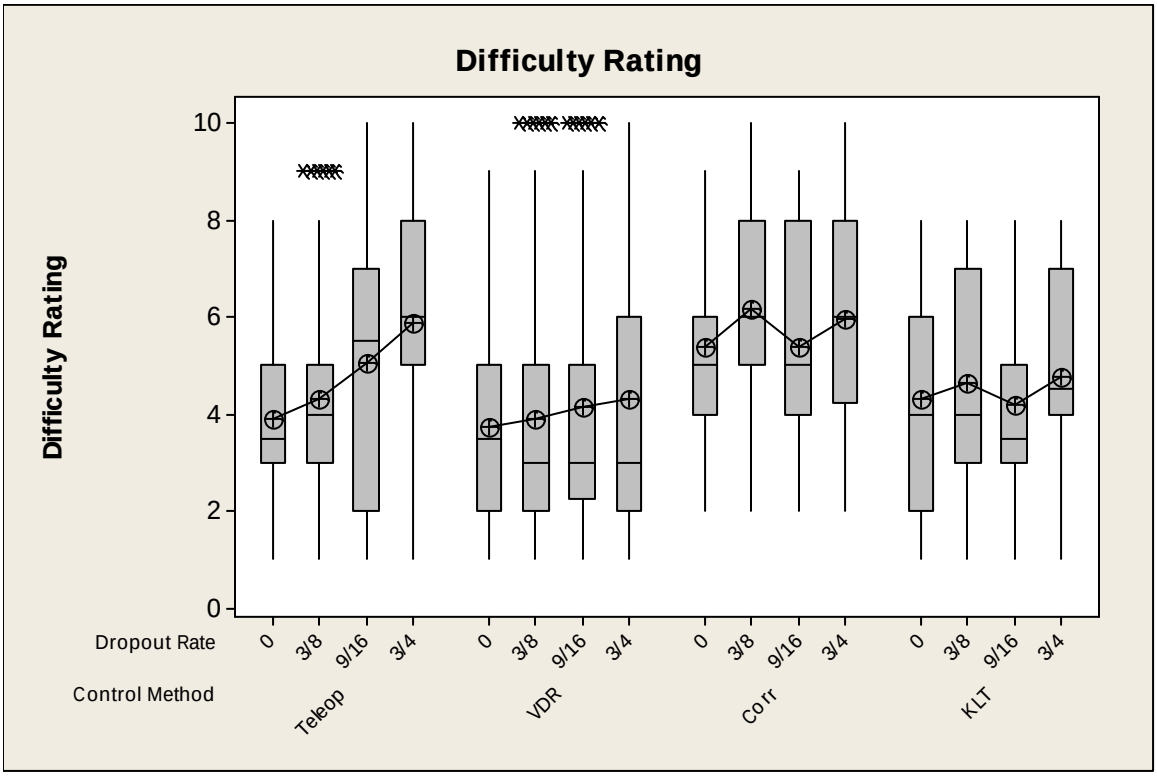

Figure 59: Box plot of difficulty rating as a function of dropout rate and control method.

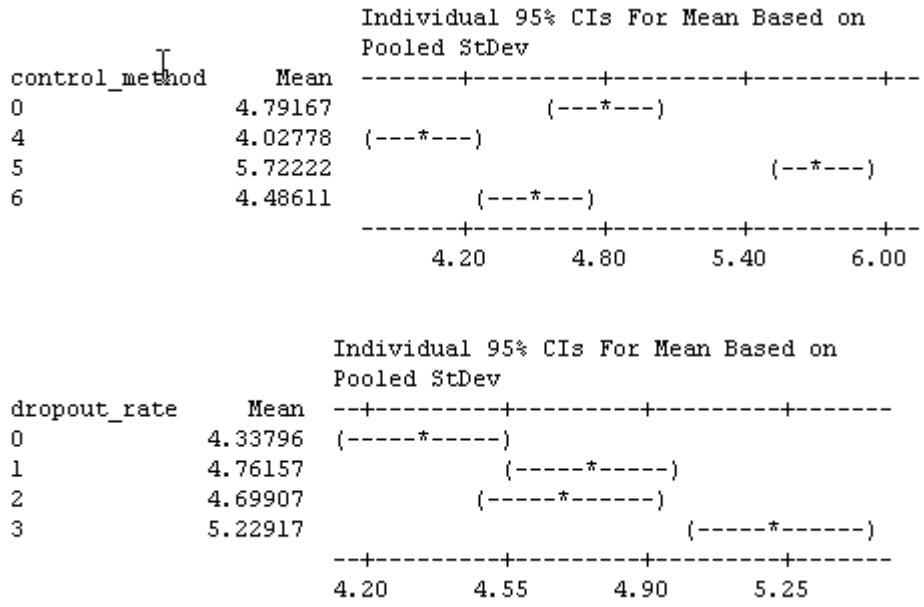

Figure 60: The mean difficulty ratings for control method and dropout rate.

5.2.9.2 Inspection Time

The inspection time was the time it took the operator to navigate the robot to the target position. All times are in seconds. As noted before, sometimes the robot would become stuck on the wooden courses. If that happened, the particular segment was redone.

Table 9 shows the results of a two-way ANOVA of inspection time as a function of course roughness and dropout rate ($F_{2,1716} = 23.48$, p=0.0000). There was significance in the interaction between course roughness and dropout rate. The mean inspection time increased as the level of course roughness increased. The dropout rate did not affect the mean inspection time as much as the course roughness did. Figure 61 shows the box plot of the two-way ANOVA results. Figure 62 shows the mean values of the inspection time by course roughness and dropout rate.

Two-way ANOVA: inspection_time_secs versus course_number, dropout_rate

Source	DF	SS	MS	F	P
course_number	2	40918	20459.1	23.48	0.000
dropout_rate	3	2869	956.4	1.10	0.349
Interaction	6	1609	268.1	0.31	0.933
Error	1716	1495442	871.5		
Total	1727	1540838			

S = 29.52 R-Sq = 2.95% R-Sq(adj) = 2.32%

Table 9: Two-way ANOVA results of inspection time as a function of course roughness and dropout rate.

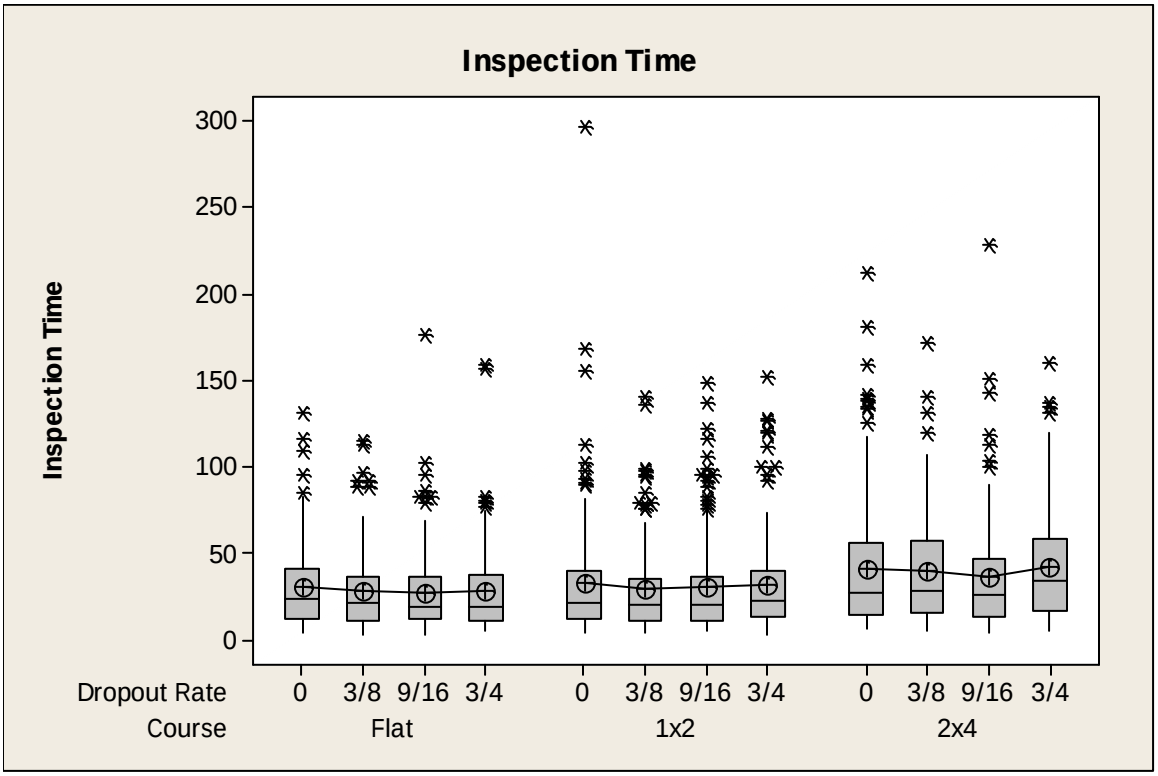

Figure 61: Two-way ANOVA of inspection time as a function of dropout rate and course roughness.

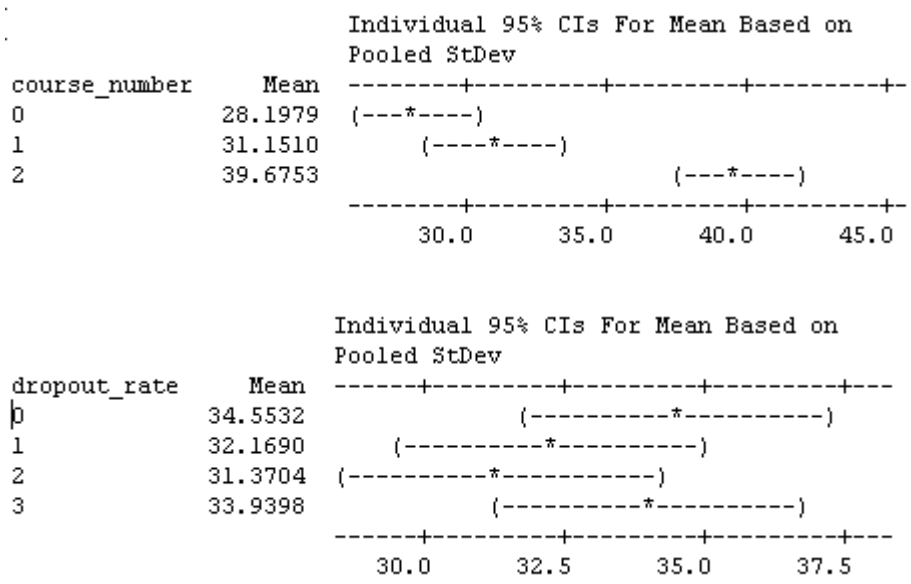

Figure 62: The mean inspection time for course roughness and dropout rate.

Table 10 shows the results of a two-way ANOVA of inspection time as a function of course roughness and control method ($F_{2,1716}$ = 25.22, p=0.0000). There was no significance in the interaction between course roughness and control method. Teleoperation had the lowest average inspection time of the control methods at 22.89 seconds with visual dead reckoning having the second lowest at 30.05 seconds. Figure 63 shows the box plot of the two-way ANOVA results. Figure 64 shows the mean values of the inspection time by course roughness and control method.

Two-way ANOVA: inspection_time_secs versus course_number, control_method

Source	DF	SS	MS	F	P
course_number	2	40918	20459.1	25.22	0.000
control_method	3	98839	32946.3	40.61	0.000
Interaction	6	8856	1476.0	1.82	0.092
Error	1716	1392224	811.3		
Total	1727	1540838			

S = 28.48 R-Sq = 9.64% R-Sq(adj) = 9.07%

Table 10: Two-way ANOVA results of inspection time as a function of course roughness and control method.

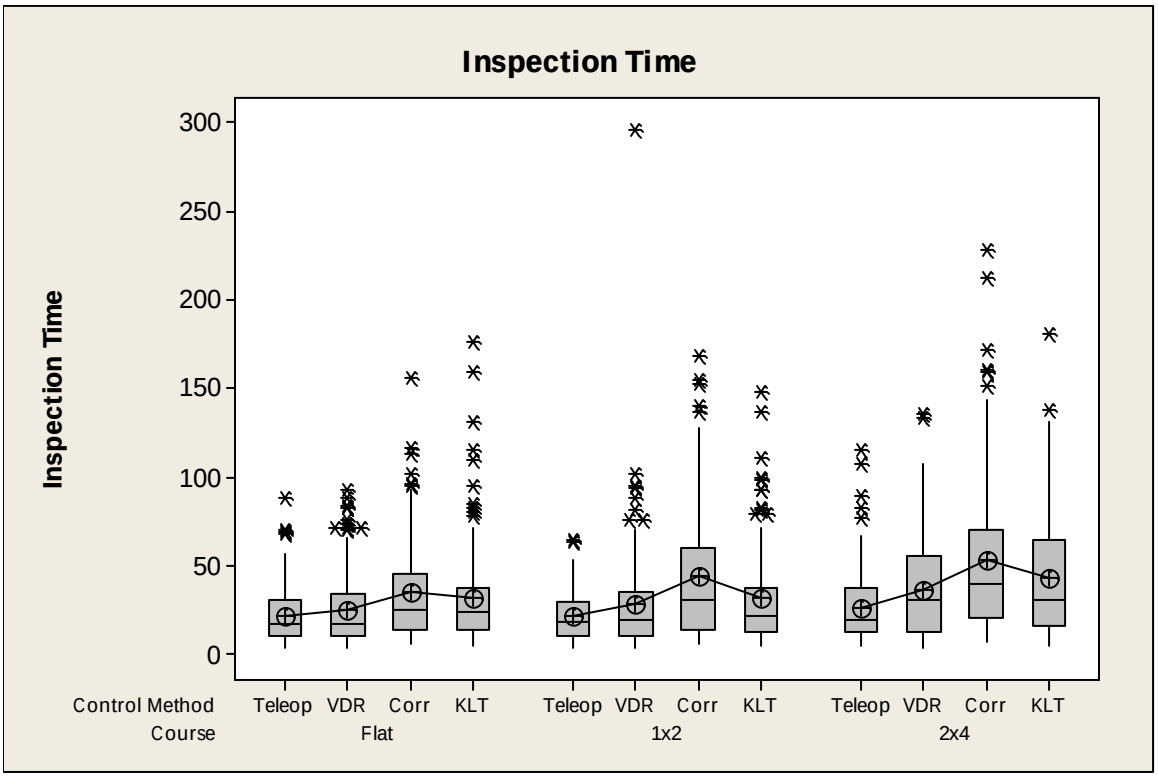

Figure 63: Inspection time as a function of control method and course roughness.

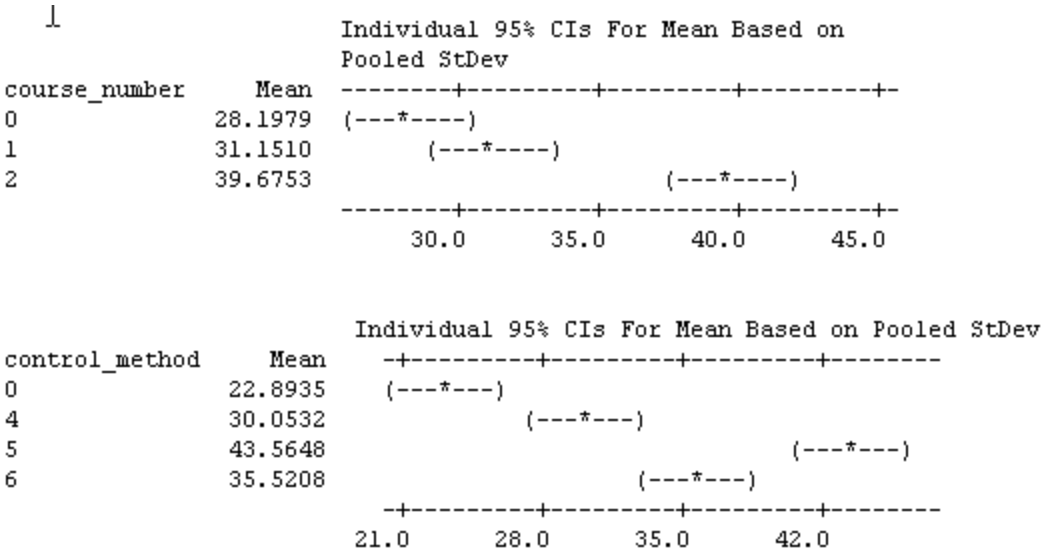

Figure 64: The mean inspection time for course roughness and control method.

Table 11 shows the results of a two-way ANOVA of inspection time as a function of control method and dropout rate ($F3_{,1712}$ = 39.40, p=0.0000). There was significance in the interaction between control method and dropout rate. Teleoperation had the lowest average inspection time of the control methods, with visual dead reckoning having the second lowest, followed by KLT. Figure 65 shows the box plot of the two-way ANOVA results. Figure 66 shows the mean values of the inspection time by control method and dropout rate.

Two-way ANOVA: inspection_time_secs versus control_method, dropout_rate

Source	DF	SS	MS	F	P
control_method	3	98839	32946.3	39.40	0.000
dropout_rate	3	2869	956.4	1.14	0.330
Interaction	9	7396	821.8	0.98	0.452
Error	1712	1431734	836.3		
Total	1727	1540838			

S = 28.92 R-Sq = 7.08% R-Sq(adj) = 6.27%

Table 11: Two-way ANOVA results of inspection time as a function of control method and dropout rate.

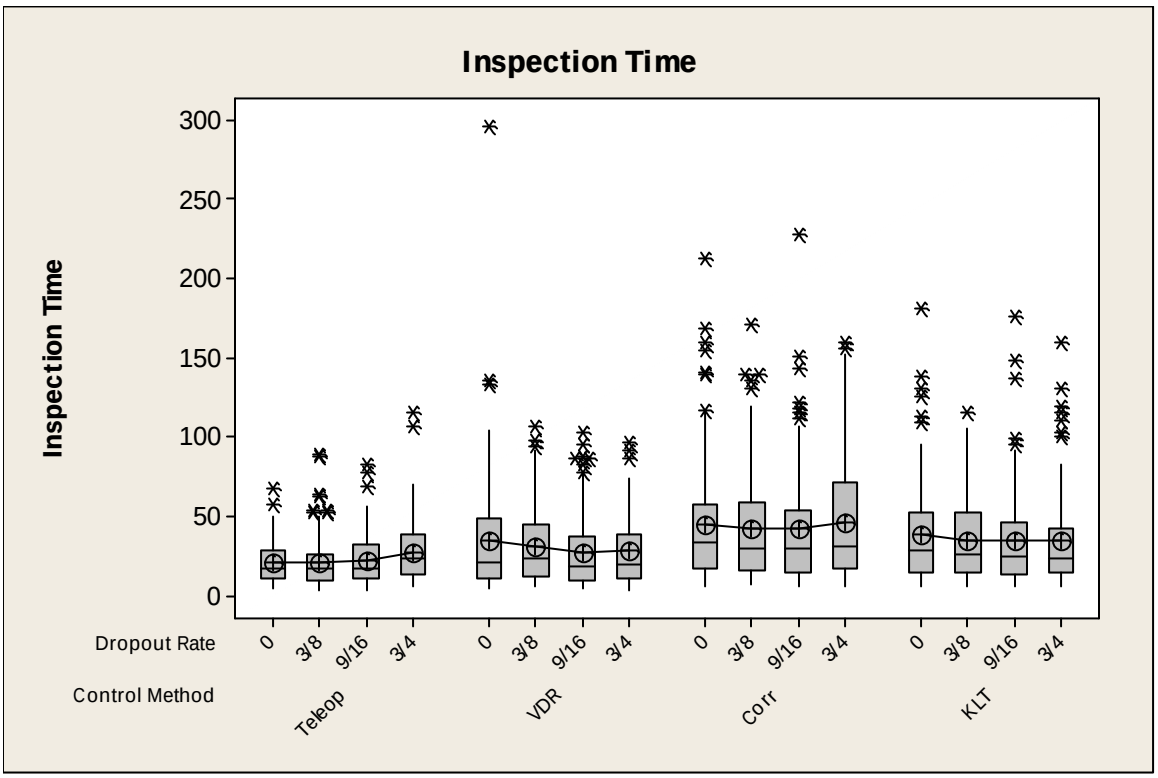

Figure 65: Inspection time as a function of dropout rate and control method.

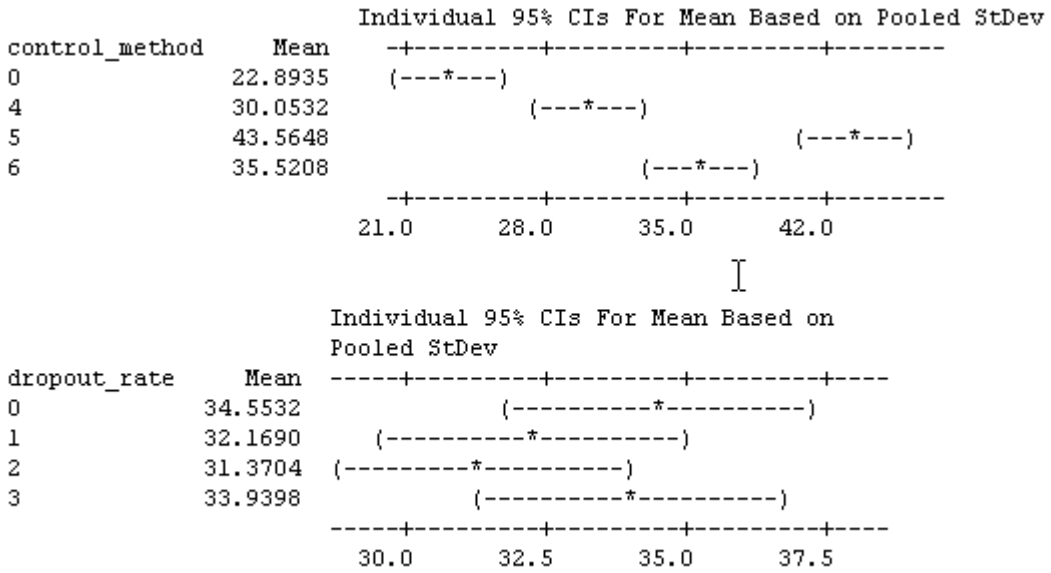

Figure 66: The mean inspection time for control method and dropout rate.

5.2.9.3 Number of Times Stopped

The final metric analyzed was the number of times stopped. In teleoperation, if the operator allowed the x and y axes on the gamepad to both go to zero, it was counted as a stop. In the semi-autonomous algorithms, the operator designated a stop by pressing the space bar on the laptop's keyboard.

Table 12 shows the results of a two-way ANOVA of number of times stopped as a function of course roughness and dropout rate ($F_{2,1716}$ = 152.80, p=0.0000). There was significance in the interaction between course roughness and dropout rate. The mean number of times stopped increased as both the level of course roughness and dropout rate increased. Figure 67 shows the box plot of the two-way ANOVA results. Figure 68 shows the mean values of number of times stopped by course roughness and dropout rate.

Two-way ANOVA: number_of_times_stopped versus course_number, dropout_rate

Source	DF	SS	MS	F	P
course_number	2	13495.8	6747.88	152.80	0.000
dropout_rate	3	1482.6	494.21	11.19	0.000
Interaction	6	1412.0	235.34	5.33	0.000
Error	1716	75779.5	44.16		
Total	1727	92169.9			

S = 6.645 R-Sq = 17.78% R-Sq(adj) = 17.26%

Table 12: Two-way ANOVA results of the number of times stopped as a function of course roughness and dropout rate.

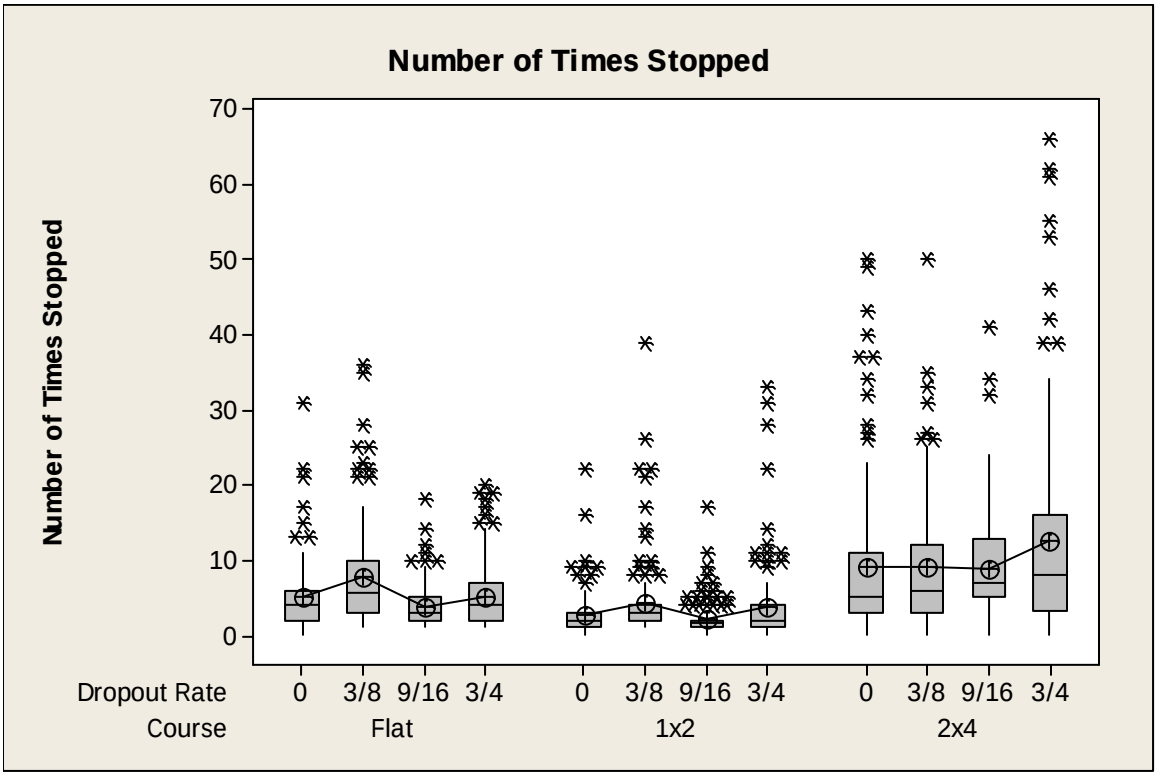

Figure 67: The number of times stopped as a function of dropout rate and course roughness.

```
                          Individual 95% CIs For Mean Based on
                          Pooled StDev
course_number     Mean    -------+---------+---------+---------+--
0              5.41667            (--*--)
1              3.16667  (--*--)
2              9.89063                                    (-*--)
                          -------+---------+---------+---------+--
                               4.0       6.0       8.0      10.0


                          Individual 95% CIs For Mean Based on
                          Pooled StDev
dropout_rate      Mean    -------+---------+---------+---------+--
0              5.62731         (-----*------)
1              6.94676                   (-----*------)
2              4.91204  (-----*-----)
3              7.14583                     (-----*------)
                          -------+---------+---------+---------+--
                               5.0       6.0       7.0       8.0
```

Figure 68: The mean number of times stopped for course roughness and dropout rate.

Table 13 shows the results of a two-way ANOVA of number of times stopped as a function of course roughness and control method ($F_{2,1716}$ = 164.62, p=0.0000). There was significance in the interaction between course roughness and dropout rate. The control method with the least number of stops was KLT, followed by correlation, teleoperation. Visual dead reckoning had the most. Figure 69 shows the box plot of the two-way ANOVA results. Figure 70 shows the mean values of the inspection time by course roughness and control method.

Two-way ANOVA: number_of_times_stopped versus course_number, control_method

Source	DF	SS	MS	F	P
course_number	2	13495.8	6747.88	164.62	0.000
control_method	3	4634.8	1544.93	37.69	0.000
Interaction	6	3699.9	616.65	15.04	0.000
Error	1716	70339.4	40.99		
Total	1727	92169.9			

S = 6.402 R-Sq = 23.69% R-Sq(adj) = 23.20%

Table 13: Two-way ANOVA results of the number of times stopped as a function of course roughness and control method.

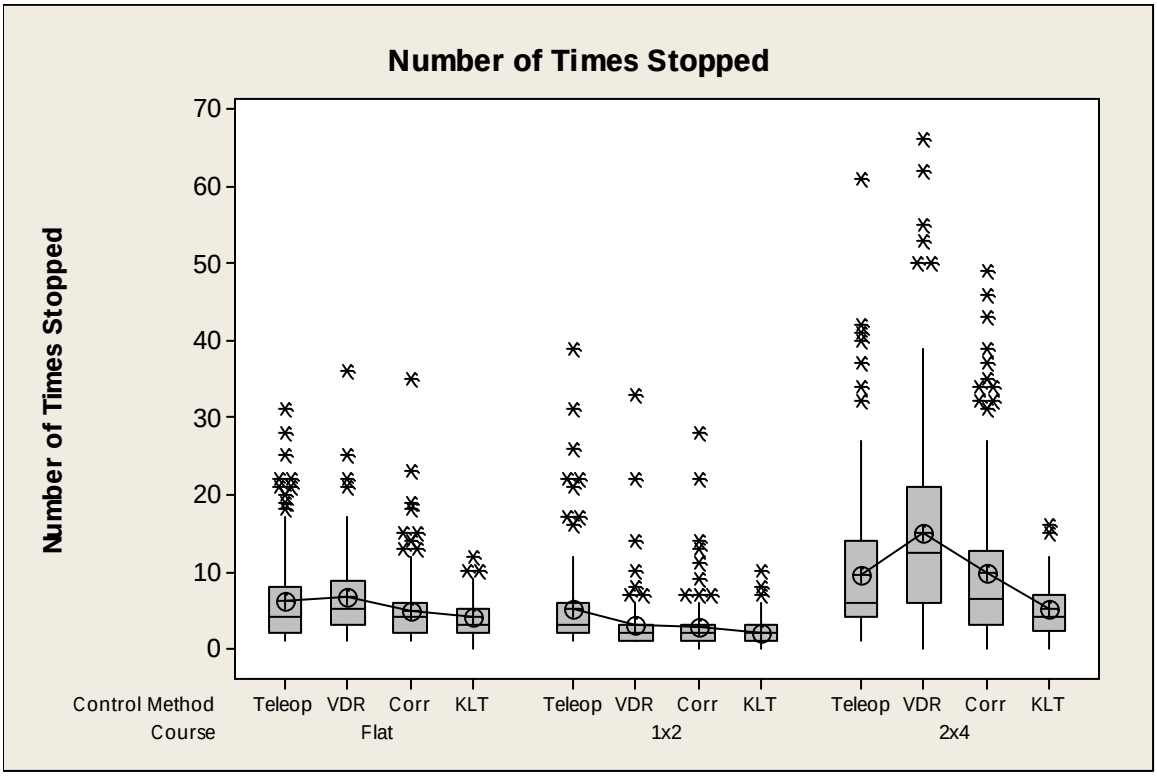

Figure 69: The number of times stopped as a function of control method and course roughness.

Individual 95% CIs For Mean Based on Pooled StDev

course_number	Mean
0	5.41667
1	3.16667
2	9.89063

4.0 6.0 8.0 10.0

Individual 95% CIs For Mean Based on Pooled StDev

control_method	Mean
0	6.94213
4	8.18056
5	5.77546
6	3.73380

4.5 6.0 7.5 9.0

Figure 70: The mean number of times stopped for course roughness and control method.

Table 14 shows the results of a two-way ANOVA of number of times stopped as a function of control method and dropout rate ($F_{3,1712}$ = 31.47, p=0.0000). There was significance in the interaction between control method and dropout rate. Figure 71 shows the box plot of the two-way ANOVA results. Figure 72 shows the mean values of the inspection time by control method and dropout rate.

Two-way ANOVA: number_of_times_stopped versus control_method, dropout_rate

Source	DF	SS	MS	F	P
control_method	3	4634.8	1544.93	31.47	0.000
dropout_rate	3	1482.6	494.21	10.07	0.000
Interaction	9	1999.3	222.15	4.52	0.000
Error	1712	84053.1	49.10		
Total	1727	92169.9			

S = 7.007 R-Sq = 8.81% R-Sq(adj) = 8.01%

Table 14 : Two-way ANOVA results of the number of times stopped as a function of control method and dropout rate.

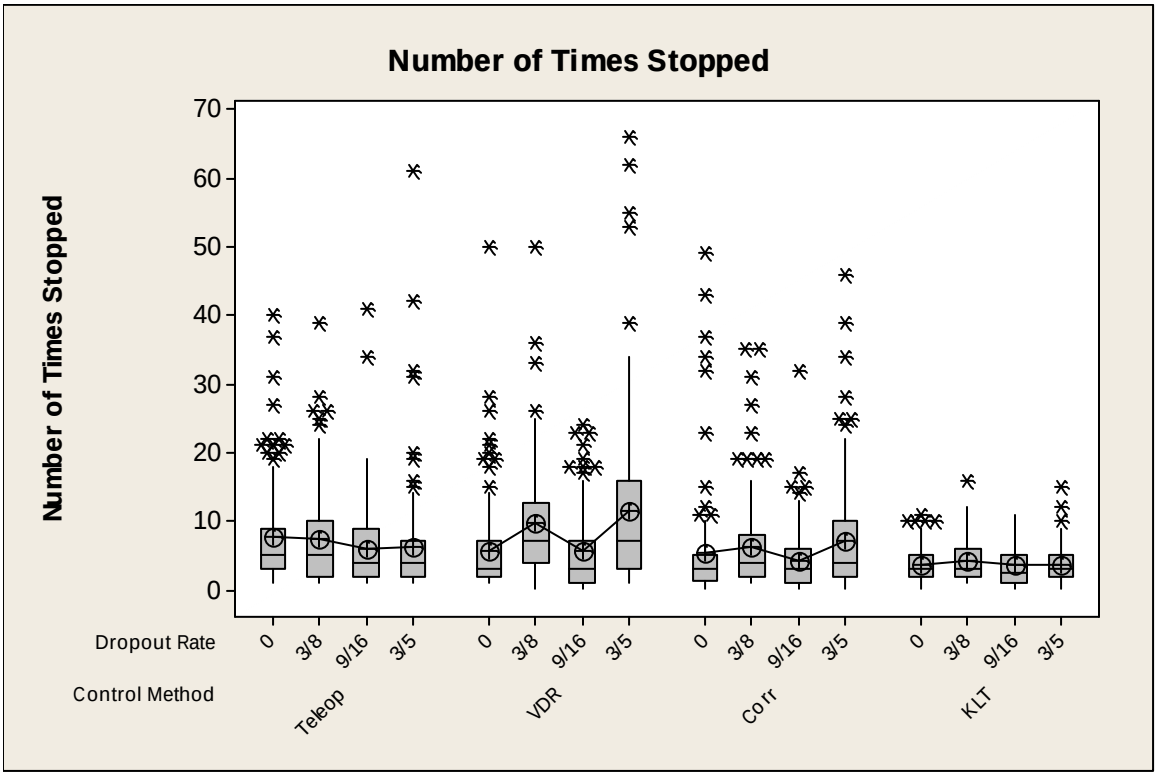

Figure 71: The number of times stopped as a function of dropout rate and control method.

Individual 95% CIs For Mean Based on Pooled StDev

control_method	Mean
0	6.94213
4	8.18056
5	5.77546
6	3.73380

3.0 4.5 6.0 7.5

Individual 95% CIs For Mean Based on Pooled StDev

dropout_rate	Mean
0	5.62731
1	6.94676
2	4.91204
3	7.14583

5.0 6.0 7.0 8.0

Figure 72: The mean number of times stopped for control method and dropout rate.

5.3 Results

The most significant factor found was the difficulty rating. The mean difficulty rating for visual dead reckoning was less than every other control method, which indicates that the subjects found it to be easier than teleoperation. The difficulty rating of the visual servoing methods were close and teleoperation was last, especially as the difficulty of terrain and dropout rates increased.

The completion time and number of emergency stops were similar enough so a clear winner was not able to be detected. Overall, teleoperation proved to take the least amount of time and visual dead reckoning was second. The two supervisory control methods also had the lowest number of emergency stops. It was interesting to find that visual dead reckoning had the highest average of stops. This may be due to the fact it did accelerate faster than the visual servoing algorithms because it did not have to track features.

5.4 Discussion and Summary

The subject testing took an average of four to five hours to complete. All participants were able to complete the test in one block of time. Generally speaking, the r^2 values were all low. This indicated there was noise that was not accounted for. This may be due to the fact that participants served as replicates. This could also be due to the operators not feeling comfortable with the control methods. The lighting in the laboratory could not be controlled and it may have caused the visual servoing algorithms to not perform as well as they could.

The sound of the PackBot when it is operating was very loud in the laboratory setup. Although the subject was positioned in such a way that the

robot was not visible at any time during the test, it would be a different experience if the subject was operating the robot in another room where the robot could not be easily heard.

This did prove that visual dead reckoning was the preferred and most robust of the semi-autonomous algorithms. This also proved that visual servoing algorithms, as implemented in this research, may not be robust enough for adoption by the Army. The laboratory setting was a benign environment compared to the missions these robots are required to operate in. If they do not perform well in this setting, it is logical to conclude they won't perform well in Iraq and Afghanistan.

Displacement control, described in the previous chapter, did not end up making it to the final subject testing. At the time, it was felt that visual dead reckoning was superior to displacement control because it could do both rotation and translation with one mouse click. In hindsight, displacement control should have been included because it allows the operator to rotate larger amounts much easier than visual dead reckoning. This is because visual dead reckoning is constrained by the field of the view of the camera. If an operator wishes to rotate more than thirty degrees at any given time, it takes several mouse clicks.

It had also been considered to add one more trial to the subject testing, allowing the operator to dynamically select which control method they wanted to use at any given time. This would have created another dataset to analyze to determine which control method was preferred as a function of dropout rates. However, as the experiment stands, there is a wealth of information that has

been collected that can be used to extract how people drive, and more importantly, a testbed was established along with a method of collecting data that has since been used for other work that focused on the effect of latency on operator performance.

CHAPTER 6

SUMMARY, CONTRIBUTIONS, AND FUTURE WORK

6.1 Summary

In very broad terms, the research covered in this dissertation implemented a novel method of tracking points in a video feed, used the developed tracking algorithm to issue rotation and translation commands for semi-autonomous operations of a military robot, and finally performed an experiment to determine if the new method of control enhanced operator performance or not. The developed algorithms also have broader impact and application that are outside of being used to control a robot. This chapter will first summarize the key contributions of this work and will then discuss the additional applications of this research.

It is also important to note the Department of Defense maintains a Technology Readiness Level (TRL) to describe how ready hardware and software is for transition to the field, where 1 is the lowest level and 9 means that the system has been proven to successfully work in a mission. The work performed in this test would possibly be classified at TRL 4 or 5. All of the testing so far has been performed only in the lab setting. It would be interesting to perform the same experiment outside on real terrain and see if there is a measureable change in operator performance.

6.2 Key Novel Contributions

There were several contributions of this research. First and foremost, a novel method to control any unmanned ground vehicle was developed. The rest of the contributions were all in support of the new control method. Key contributions are highlighted in Table 15.

One contribution was the extension to Make3D, allowing tracking of a goal point in a virtual environment by flying through the scene. Although this method was not ultimately used for the analysis, the capability has been developed and this is a viable method to use for testing tracking algorithms. Another key contribution was the large dataset collected using the IGVC platform that was made freely available. This dataset contains video, GPS, IMU, encoder, and joystick command data.

Another contribution was the ability to track a goal point using any algorithm that is able to track features from frame to frame. As described earlier, there have been many algorithms developed over the years that track features. This research provides a methodology to track a goal point using any algorithm for purposes of visual servoing.

TACTICAL was another important contribution. This software provided the method to perform a comprehensive analysis of the three classes of tracking algorithms described in this work. It has also been released as open source and is freely available. The ground truth annotation tool that was added allowed for a precise measurement of accuracy of each tracking algorithm. The ability to add compression was also a key development. This allowed compressing each frame of the dataset using several different algorithms including JPEG and

MPEG. This allowed the dataset used to be degraded in a real-world way to test the resiliency of the tracking algorithms.

Another contribution was adding a descriptor to the KLT optical flow algorithm, improving the tracking algorithm. Several shapes around the corner feature were experimented with: a rectangle, a disc, and an annulus. Multiple matching metrics were implemented included histogram matching and 2d correlation. In the experiments conducted, the disc shape with a 2D correlation metric proved to be the best method.

This work has also produced a dataset of how people operating a robot drive to an inspection point. It could be used for further research in the human factors arena. The work developed in this research can be used in future experiments.

Key Novel Contributions
Open source tracking environment
Extension to Make3D allowing to tracking a point virtually
Large dataset collected for analysis
Ability to tracking a goal point anywhere using any algorithm
Comprehensive analysis of tracking algorithms
Ability to add compression to each dataset
Descriptors added to optical flow
Implementation on a fielded military robot
Method to capture how people drive to a target
Generalized OCU concept to handle ITAR restrictions

Table 15: Summary of the major contributions of this research.

The improved corner tracker has applications to other areas such as image stabilization and stereo matching. Image stabilization using features is done by finding the overall motion from frame to frame. If features can be reliably and quickly matched from frame to frame, then stabilization becomes easier. This has already been used in another project being developed in the lab. It is a “persistent stare” application where a stationary camera is watching a scene looking for any changes.

Figure 73 shows two images from a dataset taken with a camera on a tripod overlooking a parking lot and street from a parking garage. The images on the left are with the change detection algorithm run on the data as it was. The images on the right are the same, but image stabilization, using the KLT algorithm with histogram corner matching, was run first. As can be seen, the algorithm was able to ignore the small camera motion and detect only items of interest in the scene. This technology may, at some point, be employed on a MTRS robot with a camera on a mast, watching scenes of interest and reporting back items of interest.

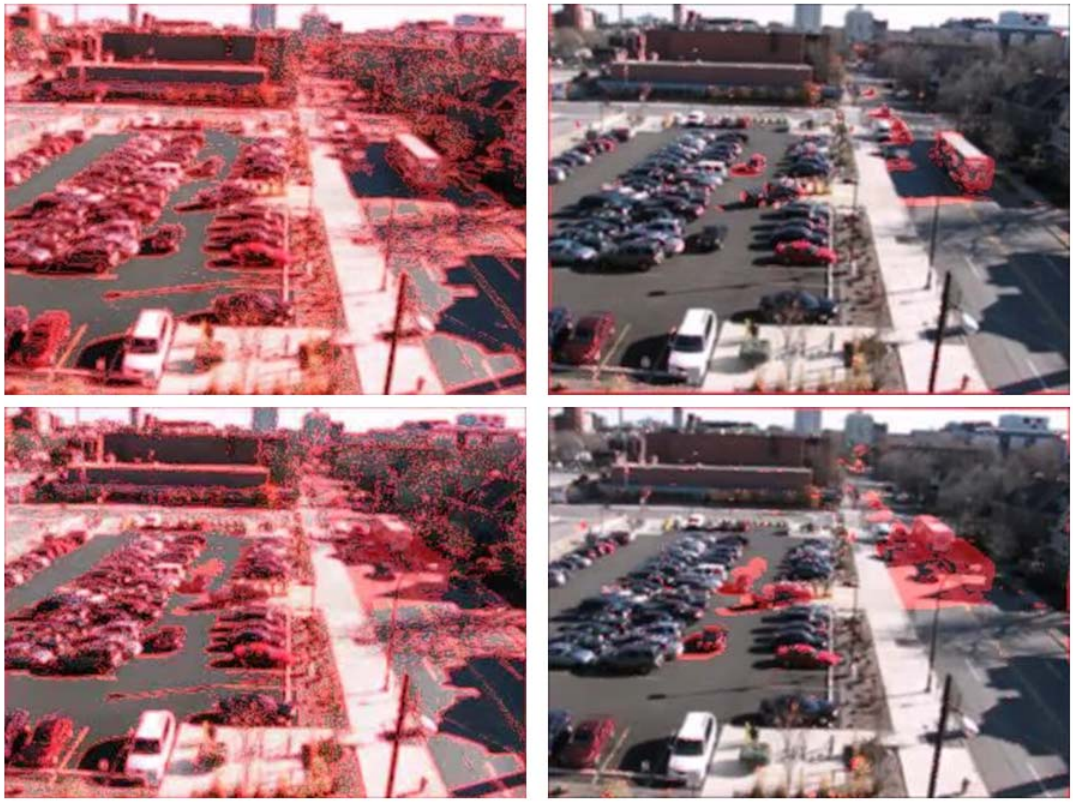

Figure 73: The "persistent stare" application that uses a stationary camera to look at a scene. The images on the left are the output of the algorithm without image stabilization and the images on the right are the output of the algorithm with image stabilization using the KLT tracker with the descriptor to match features from frame to frame.

Another important contribution to the lab where this work was performed was the development of a workaround to the ITAR (International Traffic and Arms Regulations) restrictions on the PackBot. Every detail about the PackBot is protected both by iRobot's Intellectual Property rights and also by ITAR. This means only approved United States citizens are able to operate, and more

importantly, develop applications for it. This can be a problematic a large number of the graduate student population is comprised of foreign nationals.

The developed OCU is not only able to operate any robot that has the sensors discussed, but it is also able to act as a server communicating with the PackBot in its native format. The OCU shares the communication cloud with the PackBot but it is capable of accepting and relaying messages from any computer using a non-ITAR restricted protocol.

Figure 74 depicts what the network topology looks like. The PackBot and its ITAR OCU are connected over a wireless network. The non-ITAR OCU is connected to the ITAR OCU using Ethernet and there is a firewall between the two exposing one port that passes only data from the developed protocol. This is done entirely in software and requires no special hardware. Although there have not been any projects that have made use of this yet, it will be an important piece of software that could allow the lab to follow ITAR restrictions while allowing the students interested in robotics, a chance to work on a production robot.

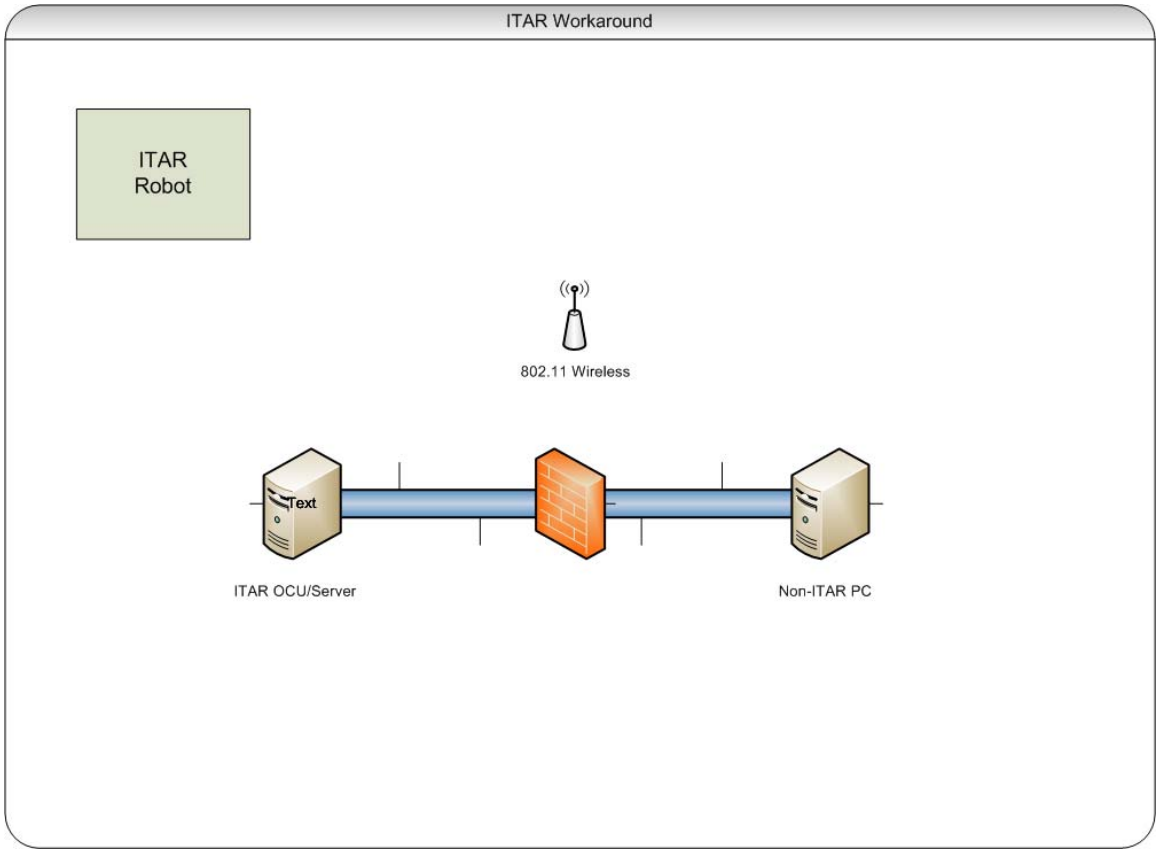

Figure 74: The developed OCU is able to function as a proxy to the any ITAR-restricted robot.

6.3 Future Work

The work performed so far does not include any path planning or obstacle avoidance. There are multiple forward-facing cameras on the PackBot. One approach to obstacle avoidance might be to use those cameras in an uncalibrated stereo application that could be used for obstacle detection. Another possibility might be to build up a 3D model of the environment from the video frames as the robot drives to the goal point. The path planning algorithm could then take an approach similar to Make3D and allow the operator to virtually zoom into the scene to plan the path out for the robot.

One interesting addition to this work might be to add a hands-free interface to the GUI instead of using the mouse. A simple approach would be to add a course grid, such as shown in Figure 75. The OCU would have to be able to recognize a small set of verbal commands, such as “Go to F6”, and “Stop”.

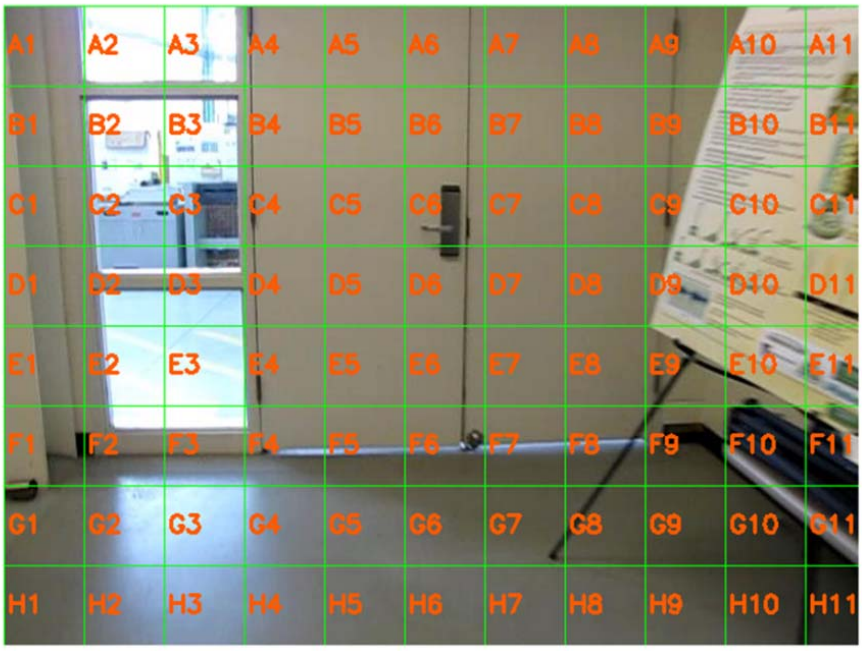

Figure 75: An example of a grid overlay that the supervisory control algorithms could extend to.

Another approach may be to utilize content-based image retrieval algorithms trained to recognize objects in the environment. If the algorithm could preprocess the scene and indicate it recognizes a door or a window, the interface could display to the operator what it recognizes in the current scene and the operator could instruct the robot to go to a location instead of a grid location.

The subject testing for the visual servoing algorithms were limited to the two methods that ran in real-time, correlation and KLT. This work could also be

extended to dynamically change the parameters of each algorithm depending upon how well the tracking is performing. It could look at whether the operator is frequently pressing the emergency stop button or designating a new goal point and use that feedback to either change the parameters of the tracking algorithm or try a different algorithm.

The work developed in this dissertation has been used in a contract awarded to Signature Research, Inc. in early 2010. The research conducted in accordance to that contract was to study the effect latency has on operator performance. During successful completion of this contract, a fixed amount of latency was simulated and studied. The visual dead reckoning algorithm was transferred to this work and it was enhanced slightly. The new method of selecting a point is shown in Figure 76. This was a slight modification to show the blue ray as the operator moves the mouse. Once the user releases the mouse, the display changes to what is shown in Figure 77 with the goal point showing as a green dot. The robot stops when the green dot reaches the red line.

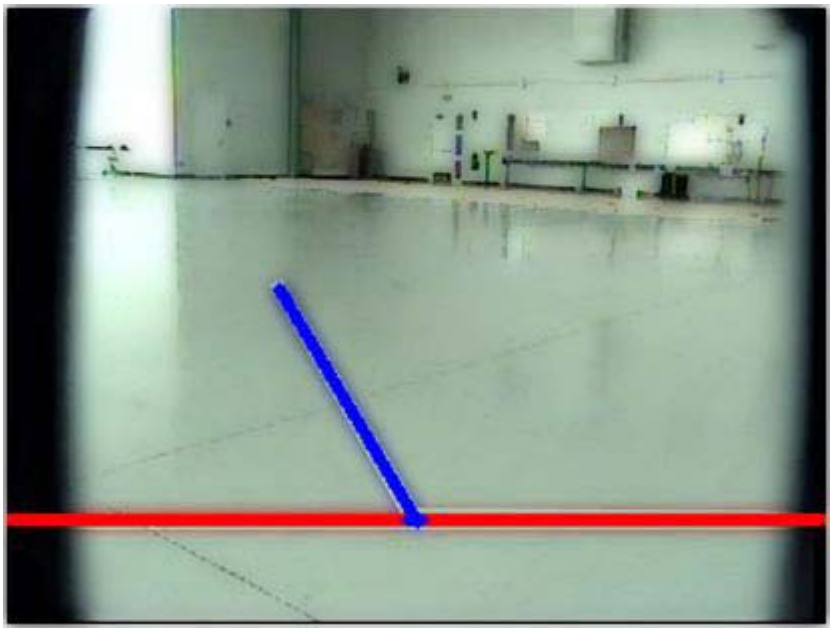

Figure 76: The revised interface for visual dead reckoning. The blue ray traces the mouse as the operator moves.

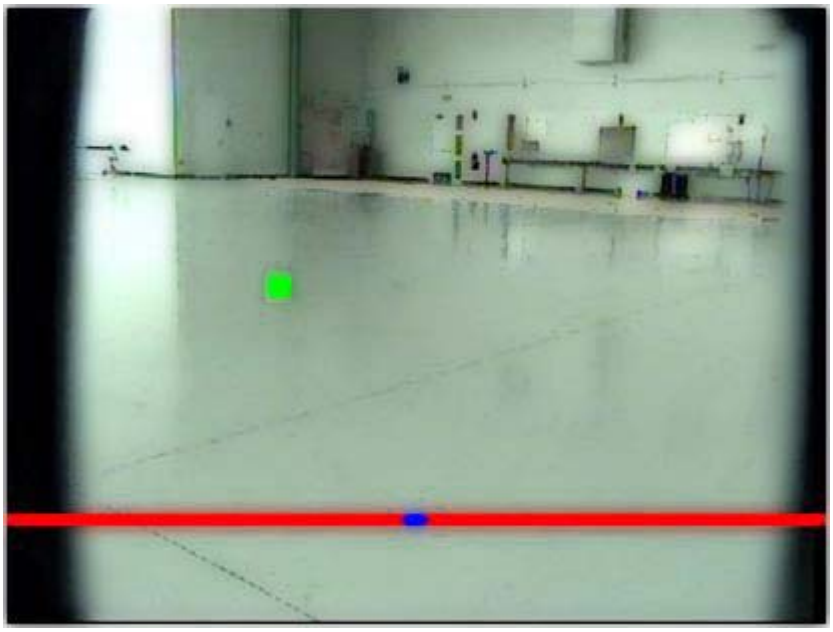

Figure 77: Once the mouse is released in visual dead reckoning, the greet dot shows the goal point.

Fulfilling the terms of the contract required developing augmented and virtual reality displays that took the queue of latent commands and showed the calculated position and orientation of the robot after the queue of commands had been processed. Figure 78 shows the augmented reality display and Figure 79 show the virtual reality display.

Figure 78: Augmented reality predictive display (ARPD).

Figure 79: Virtual reality predictive display (VRPD).

This work may also be extended to more autonomous behaviors and could be done by having an algorithm designate new goal points to the

supervisory control algorithm. The autonomous algorithm could determine path planning and determine if the path is clear or not.

There is no reason why the “point-and-go” algorithm has to apply only to the chassis. The same idea can also apply to controlling the arm. In this work, the forward kinematics of the arm were calculated to obtain the angle of the camera but arm control would require the inverse kinematics. There are open source libraries available that are solve the inverse kinematics that could be used for this purpose.

The research described in this dissertation proved that the concept of a “point-and-go” controlled robot works and that the operators expressed a preference for point-and-go over teleoperation. This is fundamental research that can be easily ported to run on any ground vehicle.

Appendix A

HIC Approval

WAYNE STATE UNIVERSITY

HUMAN INVESTIGATION COMMITTEE
101 East Alexandrine Building
Detroit, Michigan 48201
Phone: (313) 577-1628
FAX: (313) 993-7122
http://hic.wayne.edu

NOTICE OF EXPEDITED AMENDMENT APPROVAL

To: Abhilash Pandya
Electrical & Computer Engineering
5050 Anthony Wayne D, Rm 3129

From: Ellen Barton, Ph.D. _____
Chairperson, Behavioral Institutional Review Board (B3)

Date: September 03, 2009

RE: HIC #: 082105B3E(R)

Protocol Title: Human Factors Analysis for Robotic Teleoperation

Sponsor:
- U.S. ARMY TANK-AUTOMOTIVE & ARMAMENTS COMMAND (TACOM)
- NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
- CHILDREN'S HOSPITAL OF MICHIGAN

Protocol #: 0705004861

Expiration Date: April 29, 2010

Risk Level / Category: Research not involving greater than minimal risk

---

The above-referenced protocol amendment, as itemized below, was reviewed by the Chairperson/designee of the Wayne State University Institutional Review Board (B3) and is APPROVED effective immediately.

- Addition of Shawn Hunt as key personnel.

Appendix B

Dataset Details

Although a complete dataset was collected, the only data used for analyzing the performance of the tracking algorithms were the images recorded from the camera. The dataset contains:

- Video frames
- GPS
- IMU data
- Encoder feedback
- Motor controller feedback
- Joystick commands

The dataset is available at: http://gbvs.sourceforge.net/

References

1. Chen, J., E. Haas, and M. Barnes, *Human performance issues and user interface design for teleoperated robots.* IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2007. **37**(6): p. 1231-1245.
2. *BP robots still trying to contain oil spill*. 2010-09-11]; Available from: http://www.cbc.ca/world/story/2010/06/02/gulf-of-mexico-oil-spill.html.
3. *BP oil spill containment stalled as robotic saw becomes stuck*. [cited 2010-09-11; Available from: http://www.examiner.com/world-news-in-national/bp-oil-spill-containment-stalled-as-robotic-saw-becomes-stuck.
4. Gunderson, J. and L. Gunderson, *Autonomy (What's it Good for?).*
5. Sarter, N., D. Woods, and C. Billings, *Automation surprises.* Handbook of human factors and ergonomics, 1997. **2**: p. 1926–1943.
6. Endsley, M., *Toward a theory of situation awareness in dynamic systems.* Human Factors: The Journal of the Human Factors and Ergonomics Society, 1995. **37**(1): p. 32-64.
7. Endsley, M., *Situation awareness in aviation systems.* Handbook of aviation human factors, 1999: p. 257-276.
8. Defense, S.o., *Unmanned Systems Roadmap 2007-2032*, OSD, Editor. 2007, US Department of Defense: Washington D.C.
9. iRobot. *iRobot Wins $286 Million U.S. Army Contract*. 2007 [cited 2010; Available from: http://www.irobot.com/sp.cfm?pageid=86&id=377.

10. *IFOV (Instantaneous Field of View) Definition*. 11-Sep-2010]; Available from: http://www.ssec.wisc.edu/sose/tutor/ifov/define.html.

11. Anderson, D. *IMU Odometry*. 31-July-2010]; Available from: http://www.geology.smu.edu/~dpa-www/robo/Encoder/imu_odo/#sec2.

12. Hutchinson, S., G. Hager, and P. Corke, *A tutorial on visual servo control.* IEEE transactions on robotics and automation, 1996. **12**(5): p. 651-670.

13. Perkins, W. and T. Binford, *A corner finder for visual feedback.* Computer Graphics and Image Processing, 1973. **2**(3-4): p. 355-376.

14. Kragic, D. and H. Christensen, *Survey on visual servoing for manipulation.* Computational Vision and Active Perception Laboratory, Fiskartorpsv. **15**.

15. Wilson, W., C. Hulls, and G. Bell, *Relative end-effector control using Cartesian position based visual servoing: Special section on vision-based control of robot manipulators.* IEEE Transactions on Robotics and Automation, 1996. **12**(5): p. 684-696.

16. Espiau, B., *Effect of camera calibration errors on visual servoing in robotics.* Experimental Robotics III, 1994: p. 182-192.

17. Corke, P., *Visual control of robot manipulators {a review.* Visual Serving: Real Time Control of Robot Manipulators Based on Visual Sensory Feedback, 1993: p. 1.

18. Tsai, R., *A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses.* IEEE Journal of robotics and Automation, 1987. **3**(4): p. 323-344.

19. Espiau, B., F. Chaumette, and P. Rives, *A new approach to visual servoing in robotics.* Geometric Reasoning for Perception and Action, 1993: p. 106-136.

20. Martinet, P. and J. Gallice. *Position based visual servoing using a nonlinear approach*.

21. Malis, E., F. Chaumette, and S. Boudet, *2 1/2 D visual servoing.* IEEE Transactions on Robotics and Automation, 1999. **15**(2): p. 238-250.

22. Bowditch, N., *Dead Reckoning.* The American Practical Navigator, an Epitome of Navigation, 1802.

23. Gallistel, C., *The organization of learning*. 1990: MIT press Cambridge, MA.

24. Pantel, L. and L. Wolf. *On the suitability of dead reckoning schemes for games*. 2002: ACM.

25. Prewitt, J., *Object enhancement and extraction.* Picture processing and Psychopictorics, 1970: p. 75-149.

26. Sobel, I. and G. Feldman, *A 3x3 isotropic gradient operator for image processing.* Presentation for Stanford Artificial Project, 1968.

27. Harris, C. and M. Stephens. *A combined corner and edge detector*. 1988: Manchester, UK.

28. Shi, J. and C. Tomasi. *Good features to track*. 1994.

29. Kerr, D., S. Coleman, and B. Scotney. *Comparing Cornerness Measures for Interest Point Detection*. 2008.

30. Bradski, G. and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. 2008: O'Reilly Media, Inc.

31. Guiducci, A., *Corner characterization by differential geometry techniques.* Pattern Recognition Letters, 1988. **8**(5): p. 311-318.

32. Smith, S. and J. Brady, *SUSAN—A new approach to low level image processing.* International Journal of Computer Vision, 1997. **23**(1): p. 45-78.

33. Rosten, E. and T. Drummond, *Machine learning for high-speed corner detection.* Computer Vision–ECCV 2006, 2006: p. 430-443.

34. Trujillo, L. and G. Olague. *Synthesis of interest point detectors through genetic programming*. 2006: ACM New York, NY, USA.

35. Lucas, B. and T. Kanade. *An iterative image registration technique with an application to stereo vision*. 1981: Citeseer.

36. Bouguet, J., *Pyramidal implementation of the lucas kanade feature tracker description of the algorithm.* Intel Corporation, Microprocessor Research Labs, 2000.

37. Adelson, E., et al., *Pyramid methods in image processing.* 2004.

38. Horn, B. and B. Schunck, *Determining optical flow.* Computer vision, 1981. **17**: p. 185-203.

39. Farnebäck, G., *Polynomial expansion for orientation and motion estimation*. 2002: Univ.

40. Heigl, B., D. Paulus, and H. Niemann. *Tracking points in sequences of color images*. 1998: Citeseer.

41. Jin, H., P. Favaro, and S. Soatto. *Real-time feature tracking and outlier rejection with changes in illumination*. 2001.

42. Lowe, D. *Object recognition from local scale-invariant features*. 1999: Corfu, Greece.

43. Bay, H., T. Tuytelaars, and L. Van Gool, *Surf: Speeded up robust features.* Lecture Notes in Computer Science, 2006. **3951**: p. 404.

44. Viola, P. and M. Jones. *Rapid Object Detection using a Boosted Cascade of Simple*.

45. Crow, F., *Summed-area tables for texture mapping.* ACM SIGGRAPH Computer Graphics, 1984. **18**(3): p. 212.

46. Beis, J. and D. Lowe. *Shape indexing using approximate nearest-neighbour search in high-dimensional spaces*. 1997: INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE).

47. Zhang, W. and J. Kosecka. *Image based localization in urban environments*. 2006.

48. Battiato, S., et al. *SIFT features tracking for video stabilization*. 2007.

49. Fischler, M. and R. Bolles, *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.* 1981.

50. Mondragon, I., et al. *Visual model feature tracking for UAV control*. 2007: Citeseer.

51. Connor, D. and J. Limb, *Properties of frame-difference signals generated by moving images.* Communications, IEEE Transactions on [legacy, pre-1988], 1974. **22**(10): p. 1564-1575.

52. Cafforio, C. and F. Rocca, *Methods for measuring small displacements of television images.* IEEE transactions on Information Theory, 1976. **22**(5): p. 573-579.

53. Ryan, T., R. Gray, and B. Hunt, *Prediction of correlation errors in stereo-pair images.* Optical Engineering, 1980. **19**(3): p. 312-322.

54. Forstner, W. and A. Pertl. *Photogrammetric standard methods and digital image matching techniques for high precision surface measurements*. 1986.

55. Wood, G., *Realities of automatic correlation problem.* Photogrammetric Engineering and Remote Sensing, 1983. **49**(4): p. 537-538.

56. Tian, Q. and M. Huhns, *Algorithms for subpixel registration.* Computer Vision, Graphics, and Image Processing, 1986. **35**(2): p. 220-233.

57. Systems, S. *Scenespector - VooCAT*. [cited 2009 13-July-2009]; Available from: http://www.scenespector.com/index.php?option=com_content&task=view&id=10&Itemid=25.

58. Andersson Technologies, L. *SynthEyes Camera Tracker*. [cited 2009 13-July-2009]; Available from: http://www.ssontech.com/synovu.htm.

59. Farm, T.P. *PFTrack by The Pixel Farm*. [cited 2009 13-July-2009]; Available from: http://www.thepixelfarm.co.uk/products/products.aspx?PID=3.

60. *libmv (a structure from motion library)*. Available from: http://code.google.com/p/libmv/.

61. Saxena, A., M. Sun, and A. Ng. *Make3D: Depth Perception from a Single Still Image*. 2008: AAAI.

62. Bloomberg. *Affine transformations*. 30-July-2009]; Available from: http://www.leptonica.com/affine.html.

63. Shapiro, L. and G. Stockman, *Computer Vision. 2001*. 2001, Prentice Hall.

64. Felzenszwalb, P. and D. Huttenlocher, *Efficient graph-based image segmentation.* International Journal of Computer Vision, 2004. **59**(2): p. 167-181.

65. *Cookbook/RANSAC*. [cited 2010 20-Aug-2010]; Available from: http://www.scipy.org/Cookbook/RANSAC.

66. Kuzmin, Y. *An efficient circle-drawing algorithm*. 1990: John Wiley & Sons.

67. Bentley, J., *Multidimensional binary search trees used for associative searching.* Communications of the ACM, 1975. **18**(9): p. 517.

68. Kumar, N., L. Zhang, and S. Nayar, *What is a good nearest neighbors algorithm for finding similar patches in images?* Computer Vision–ECCV 2008, 2008: p. 364-378.

69. Mount, D. and S. Arya. *ANN: A library for approximate nearest neighbor searching*. 1997: Citeseer.

70. Snavely, N., S. Seitz, and R. Szeliski. *Photo tourism: exploring photo collections in 3D*. 2006: ACM New York, NY, USA.

71. Muja, M. and D. Lowe. *Fast approximate nearest neighbors with automatic algorithm configuration*. 2009: Citeseer.

72. Sproull, R., *Refinements to nearest-neighbor searching in k-dimensional trees.* Algorithmica, 1991. **6**(1): p. 579-589.

73. Omohundro, S., *Five balltree construction algorithms.* International Computer Science Institute Technical Report, 1989: p. 89-063.

74. MacQueen, J. *Some methods for classification and analysis of multivariate observations*. 1967: California, USA.

75. *view3dscene*. 10-July-2009]; Available from: http://vrmlengine.sourceforge.net/view3dscene.php.

76. Hunt, S. *Goal-Based Visual Servoing*. 2010 4-August-2010]; Available from: http://sourceforge.net/projects/gbvs/.

77. Gonzalez, R. and E. Richard, *Woods, digital image processing.* Beijlng: Publishing House of Electronics Industry, 2005: p. 420-450.

78. Roth, S. *Syntehtic Optical Flow Database*. 21-Aug-2010]; Available from: http://www.gris.tu-darmstadt.de/~sroth/research/flow/downloads.html.

Abstract

ROBOTIC GOAL-BASED SEMI-AUTONOMOUS ALGORITHMS IMPROVE REMOTE OPERATOR PERFORMANCE

by

SHAWN HUNT

December 2010

Advisors: Dr. Abhilash Pandya & Dr. R. Darin Ellis

Major: Computer Engineering

Degree: Doctor of Philosophy

The focus of this research was to determine if reliable goal-based semi-autonomous algorithms are able to improve remote operator performance or not. Two semi-autonomous algorithms were examined: visual servoing and visual dead reckoning. Visual servoing uses computer vision techniques to generate movement commands while using internal properties of the camera combined with sensor data that tell the robot its current position based on its previous position. This research shows that the semi-autonomous algorithms developed increased performance in a measurable way. An analysis of tracking algorithms for visual servoing was conducted and tracking algorithms were enhanced to make them as robust as possible. The developed algorithms were implemented on a currently fielded military robot and a human-in-the-loop experiment was conducted to measure performance.

Autobiographical Statement

SHAWN HUNT

Mr. Hunt received his BS degree in Computer Science from Franklin University in 2004 and his MS in Electrical and Computer Engineering from Wayne State University in 2008. He is currently transitioning into a position with TARDEC in Warren, MI. Previous positions include a Systems Analyst for several automotive suppliers and as an engineer for Turing Associates, Inc. in Ann Arbor, MI.

He has had the privilege of leading Wayne State's entry to the annual Intelligent Ground Vehicle Competition for the past two years and looks forward to assisting as a mentor in future years. He has also published several conference papers, has submitted his first peer-reviewed journal paper, and is currently working on several more journal papers based on this research.

Publications in process:

S. Hunt, G. Witus, R. D. Ellis A. Pandya (2010), "Remove Operators Prefer Goal-based Semi-Autonomous Algorithms", Journal of Intelligent and Robotic Systems. Submitted September 2010.

Selected publications:

S. Hunt, G. Witus, R. Karlsen (2010), "Comparison of Teleoperation and Supervisory Control for Navigation and Driving under Reduced Bandwidth", SPIE 2010.

G. Witus, R. Karlsen, S. Hunt (2009), "Sequential Learning for Robot Vision Terrain Classification", Proceedings of the 2009 SPIE Conference on Defense and Security.

G. Witus, S. Hunt, R.D. Ellis (2008), "Monocular Visual Ranging", Proceedings of the 2008 SPIE Conference on Defense and Security, Vol. 6962.