CONTROL OF RECONFIGURABILITY AND NAVIGATION OF A WHEEL-LEGGED ROBOT BASED ON ACTIVE VISION

A Thesis
Presented to
The Academic Faculty

By

Douglas A. Brooks

In Partial Fulfillment
Of the Requirements for the Degree
Master's of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2008

CONTROL OF RECONFIGURABILITY AND NAVIGATION OF A WHEEL-LEGGED ROBOT BASED ON ACTIVE VISION

Approved by:

Dr. Ayanna Howard, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Patricio Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: June 2008

ACKNOWLEDMENTS

First I would like to thank God for all things possible. Next I would like to thank Dr. Howard for not only allowing me to work in the HumAnS Laboratory, but also for being a great advisor in guiding me through this process with knowledge, resources, and critique. I would like express gratitude to the members of my reading committee Dr. Magnus Egerstedt, Dr. Ayanna Howard, and Dr. Patricio Vela; I understand the rigorous schedule of ECE Faculty members, and I both commend and applaud your efforts for taking the time to give back to students. I wish to thank my fellow graduate students in the HumAnS Lab for their encouragement and input during difficult times. A special thank you to GEM for financially providing the opportunity to obtain a higher education. I wish to attribute my success to my family: my grandparents for a solid upbringing, my parents for always being there, and my aunts, uncles, and cousins for their support. Last, but certainly not least, I wish to remember those who passed and were not able to see this day but have been my inspiration for continuing my education: Aaron McLean, J.R. Price, Harold Mills Jr., Charlie Goldston, & Lamont Neil.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF SYMBOLS AND ABBREVIATIONS

p – probability density function

X – element of conditional density

x_i – element of component density

C_j – class in the training set

f_c – focal length

c_c – principle point

Z_c – height between projection centre of camera and terrain surface

F – frame rate

R – camera resolution

I_i – intensity value of a small feature window

I_2 – intensity value of a small feature window

d – changes in the x and y coordinates of a feature window

L - central location of counterpoint of small feature window

$_A$ – area of feature window

τ – timing for immobility recognition

μ_1 – percentage of pixels for horizon line determination

μ_2 – percentage of average pixel signature for object detection

α – amount of change in average ground signature

β – amount of change in landmark object

Θ_1 – angle between vertical plane and hypotenuse

Θ_2 – angle between hypotenuse and ground plane

Θ_g – downward camera tilted angle

X_g – horizontal distance from camera lens to visible center point in ground plane

H – linear distance from the camera lens to the ground plane

Y – vertical distance from camera to ground plane

A_g – average pixel signature of ground plane

s_g – standard deviation of average pixel signature of ground plane

x_0 – initial row of pixels

x_1 – current row of pixels

y_0 – initial column of pixels

y_1 – current column of pixels

$p_{i,j}$ – pixel at location (i,j)

N – # of samples from images

$\Delta\tau$ – elapsed time in milliseconds

H_o – null hypothesis

$\overline{Y}$ – sample or true mean

χ_o – standard or assumed mean

A_p – mean selected percentages

$t_{\upsilon;\, N-1}$ – upper υ critical value from the *t* distribution table

H_A – alternative hypothesis

φ – degrees of freedom for the t distribution table

σ – known standard deviation regarding percentage of a specific row

Z_a – standard critical value from the table of normal distributions

υ – significance level

SUMMARY

The ability of robotic units to navigate various terrains is critical to the advancement of robotic operation in real world environments. Next generation robots will need to adapt to their environment in order to accomplish tasks that are either too hazardous, too time consuming, or physically impossible for human-beings. Such tasks may include accurate and rapid explorations of various planets or potentially dangerous areas on planet Earth. This research investigates a navigation control methodology for a wheel-legged robot based on active vision. The method presented is designed to control the reconfigurability of the robot (i.e. control the usage of the wheels and legs), depending upon the obstacle/terrain, based on perception. Surface estimation for robot reconfigurability is implemented using a region growing method and a characterization and traversability assessment generated from camera data. As a result, a mathematical approach that directs necessary navigation behavior is implemented to control robot mobility. The hybrid wheeled-legged rover possesses a four-legged or six-legged walking system as well as a four-wheeled mobility system.

Chapter 1

INTRODUCTION

Versatility is one of the most important aspects of mobile robots used for exploration. As environments present difficulties in navigation, robotic platforms must be adaptable to ensure safe passage and continued progress. One method of versatility is to design the robot with two different types of mobility, specifically both wheels and legs. However, there lies the issue of recognizing when to utilize one type of mobility rather than the other. The goal of this project is to create a method for solving a navigation problem by employing a camera based system that will decide whether or not the robotic system is consistently changing its displacement, thus giving it the ability to decide whether to use its wheels or legs to traverse the terrain. Ideally, the wheels will be used as frequent as possible in order to explore the terrain at a faster pace. However, there may be instances where the wheels are not a feasible method of mobility such as in icy terrains or loose soil, and thus the legs will need to be utilized. Future robotic missions will need to continue the effort to increase the efficiency of exploratory robotic vehicles. Congruently, this research focuses on modularizing both hardware and software components and incorporating a vision system to create an autonomous reconfigurable robotic explorer.

1.1 Motivation and Objective

Science exploration in unknown and uncharted terrain involves operating in an unstructured and poorly modeled environment, and there are several robotic designs that are plausible in these types of environments [1]. The goal of this project is to design a control algorithm that will enable a reconfigurable robotic mobility platform to

autonomously traverse various terrains; specifically, the robotic platform is to autonomously sense its immobility and act accordingly. Surface estimation for robot reconfigurability is implemented using a region growing method and a characterization and traversability assessment generated from camera data.

Recently, there have been notable problems regarding immobility related issues such as wheel slippage in slippery terrain and wheels sinking in sand drifts. For example, NASA's Mars Exploration Rovers (MER) Opportunity and Spirit each had issues while traversing various terrains [2]. Both rovers experienced large slippage, up to 125%, on slopes, while Opportunity was stuck in a drift for several weeks due to sinkage before finally freeing itself [2]. As a result, NASA researchers felt that it may be prudent to employ a stereo vision system for detecting slippage and sinkage, ultimately determining immobility.

In order to understand the issues involved with solving the problem of immobility, the thought process must first be to define mobile or immobile. In order to do so, we must answer the question "how do we as humans know that we are moving in a particular direction?" Early psychological theories of human information processing regarded action and perception as two separate processes [3]. However, recent investigations have suggested the importance of action in perceiving objects [4]. Ideally, humans exploit the surrounding area to perceive motion. For instance, if there were a tree or a bush on your right side as you are walking, you could use that as a landmark to signify if you have moved closer to or farther away from the object. In much of the same way, our robotic design utilizes its surroundings to determine its change in displacement.

In order to induce autonomous reconfigurability, we have incorporated an active vision system that utilizes an inexpensive webcam. The camera system is first used to create a vision-based terrain characterization and traversability assessment that initially extracts those terrain characteristics that directly account for robot traversal difficulty [5]; details regarding the camera system will be discussed in Chapter 2, and the specifications can be found in Appendix D. Then, a region-growing algorithm, which is also discussed in Chapter 2, is used to assess robot displacement based on a timing scheme. Finally, a decision is made regarding which form of mobility is best suited for the specific situation, and the appropriate action is taken (i.e. the robot either continues to move with its wheels or deploys its legs and commences a defined walking gait). The experiments and results discussed in Chapter 3 will prove the algorithm used to control reconfigurability is reliable. An illustration of the robotic unit can be seen in Figure 1.

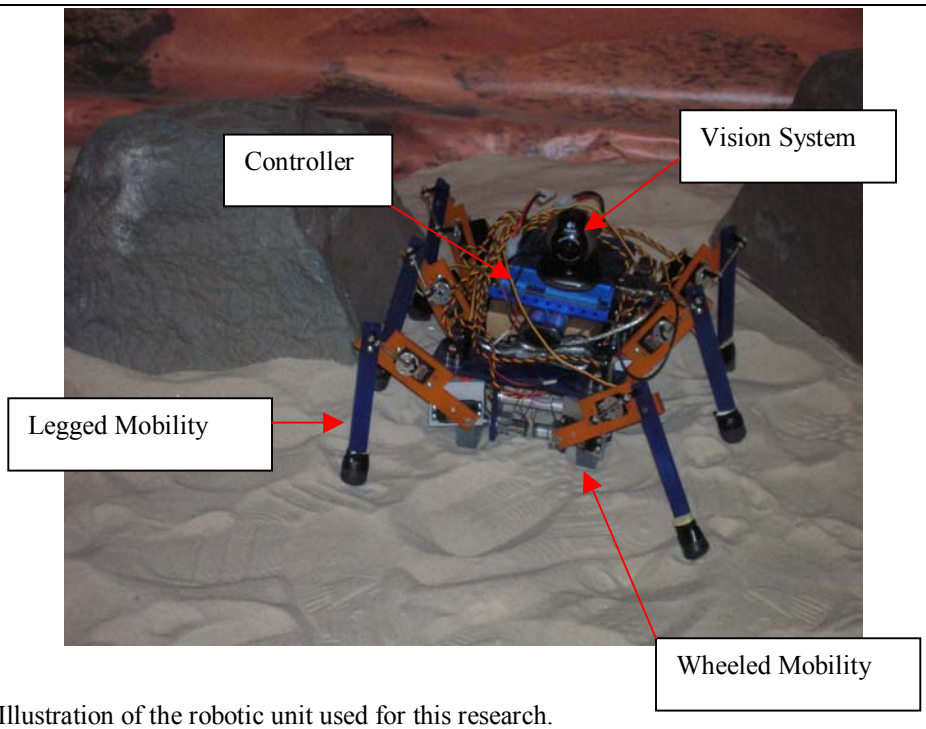


Figure 1. Illustration of the robotic unit used for this research.

1.2 Terrain Characterization Techniques

Vision-based and perception algorithms have proven to be an effective method for controlling navigation and object detection in robotic platforms [6-13]. An initial approach to characterizing various terrains is to utilize obstacles and landmarks. Talukder et al developed a 3D obstacle detector that searches for surrounding pixels in 3D space that satisfy the slope and height criteria at each valid pixel location [14]. Texture-based and color-based material classification processes are also used to characterize traversable and non-traversable terrains based on vehicle dynamics [14]. Happold and Ollis employs probability techniques coupled with boosting and mean classification [8], while Song et al, utilizes visual odometry to measure slip ratios of unmanned ground vehicles (UGVs) for autonomous navigation [10]. In the following sections, we give detail regarding techniques that have been used to create autonomous navigation for robots.

1.2.1 Probability Theory Techniques

There are a number of algorithms that implement probability theory techniques for the purpose of navigation and control. For example, some researchers use a naïve Bayesian classifier [8], which makes use of the assumption that the class conditional density $p(X|C_j)$, where X is an element of the conditional density and C_j is a class in the training set, is equal to the product of its component densities $p(x_i|C_j)$, where x_i is an element of the component density and of the class C_j, for autonomous learning. Ciftcioglu et al, who implies perception is a probabilistic concept due to the integration of photons within a certain length giving intensity of the stimulus, uses probability density

functions of random variables to observe certain aspects of visual perception such as the distance between the eye and a location on a plane [6]. Ciftcioglu et al further discuss that perception is a matter of cognition or interpretation of information, which exists within the visual scope, and can be expressed in terms of intensity, which is the integral of probability density. Although algorithms for this technique can create robustness, sheer complexity can increase the process time, thus creating a slower navigation system.

1.2.2 Optical Flow Techniques

Scientists have also utilized optical flow algorithms for a number of research projects. Terzopoulos and Rabie use an optical flow method to stabilize the visual field of an artificial fish as it locomotes [10]. Song et al use optical flow to implement a visual odometry system to measure vehicle velocity [10]. The technique presented in the research minimizes the residual error shown in Equation 1 and uses camera parameters such as focal length f_c, principal points c_c, height between the projection centre of the camera and the terrain surface Z_c, frame rate F, and camera resolution R to create a velocity estimation algorithm flow. In Equation 1, I_1 and I_2 represent the intensity values of a small feature window in two adjacent images, $\mathbf{d} = (\Delta x, \Delta y)$ represents changes in the x and y coordinates of the feature window on the image plane, $\mathbf{L} = (x, y)$ is the central location of counterpoint of the small feature window on the image plane, and $_A$ is the area of the feature window.

$$E(\mathbf{d}) = \iint_A \left[I_2\left(\mathbf{L} + \frac{\mathbf{d}}{2}\right) - I_1\left(\mathbf{L} - \frac{\mathbf{d}}{2}\right) \right]^2 d\mathbf{L} \qquad \textbf{(Eq. 1)}$$

Although optical flow has proven to be a reliable tool, there is a major drawback to its implementation. That drawback is the ideality of utilizing optical flow on a flat surface. Because distance changes between the ground plane and the camera play a significant role in calculating the optical flow parameters, surfaces that may cause the robotic platform to sink into the ground, such as sand or snow, would create undesirable variations in the optical flow data. As a result, many researchers suggest utilizing a high-powered camera for estimating real-time height information [8]. It is speculated that a stereo camera or a telecentric camera, which has a constant field of view regardless of the distance between the lens and a target, can be utilized for implementing optical flow; however, the overall cost of the project would greatly increase.

1.2.3 Region-Growing Techniques

The region-growing algorithm has proven to be an effective and inexpensive method for autonomous navigation [15]. The basic principle for the algorithm is to merge similar neighboring pixels, thus creating segmented regions that can be interpreted and utilized for various purposes such as reconfigurability. The region-growing process can be broken down into three steps:

1. Start by using any seed pixel and compare it with its neighboring pixels (up to eight in a standard matrix). A seed pixel is an initial pixel that is chosen in order to begin or continue the region-growing process.

2. Add in neighboring pixels that are similar, which will increase the size of the region.

3. When the growth of one region stops, choose another seed pixel that does not belong to a region and perform step 2 again until the region is fully grown.

The region-growing method was chosen for this project because of its robustness and low-cost implementation.

1.3 Past and Present Rovers used in Navigation

Some active vision rovers utilize tanked mobility for locomotion, such as Urbie [16] found at NASA, while others only utilize wheel mobility such as NASA's MERs [17] and Rocky 7 [18]. Yet there are other vision-guided rovers that use a leg system for locomotion such as the LEMUR [19] series also found at NASA.

Urbie, shown in Figure 2, was sponsored by the Defense Advanced Research Projects Agency (DARPA) and was a joint effort of NASA's Jet Propulsion Laboratory (JPL), iRobot Corporation, CMU's Robotic Institute, and the University of Southern California's (USC) Robotics Research Laboratory. Urbie's initial purpose was mobile military reconnaissance in city terrain, but its autonomy was suitable for a variety of tasks such as investigating urban environments contaminated with radiation or search and rescue in earthquake struck buildings. Urbie utilized the algorithm presented in [16] in order to discriminate drivable grass from obstacles during outdoor autonomous navigation tasks.

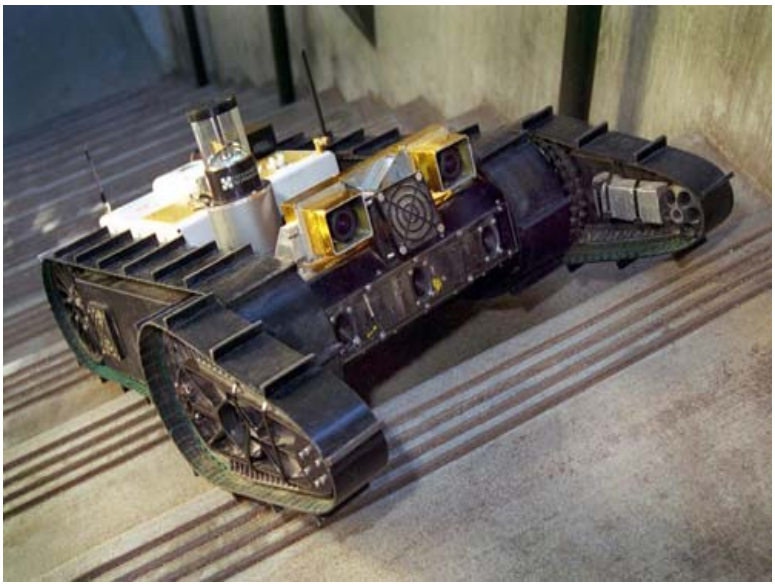


Figure 2. Illustration of Urbie during a stair climbing test procedure.

NASA's **Mars Exploration Rovers (MER)**, shown in Figure 3, navigate autonomously using stereo vision for local terrain mapping and a local, reactive planning algorithm called Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) for obstacle avoidance. Navigation is done with three sets of camera pairs: one pair of "hazcams" (hazard cameras) looking forward under the solar panel in front, another pair of hazcams looking backward under the solar panel in the back, and a pair of "navcams" (navigation cameras) on the mast. The navcams are used to employ a visual odometry system, which only converges successfully if the terrain has a sufficient number of features visible in each adjacent image pair [20]. Each camera has 1024x1024 pixel charge-coupled device (CCD) arrays that create 12 bit grayscale images. Similar to our robotic system, the baseline autonomous navigation system includes only local obstacle detection; meaning, there are no onboard global mapping, global path planning, or global localization functions. Long distance localization is done on Earth

using bundle adjustment from manually matched tie points in panoramic imagery [21]. Details of the stereo vision algorithm can be found in [22], and more details of the visual odometry algorithm can be found in [20, 23].

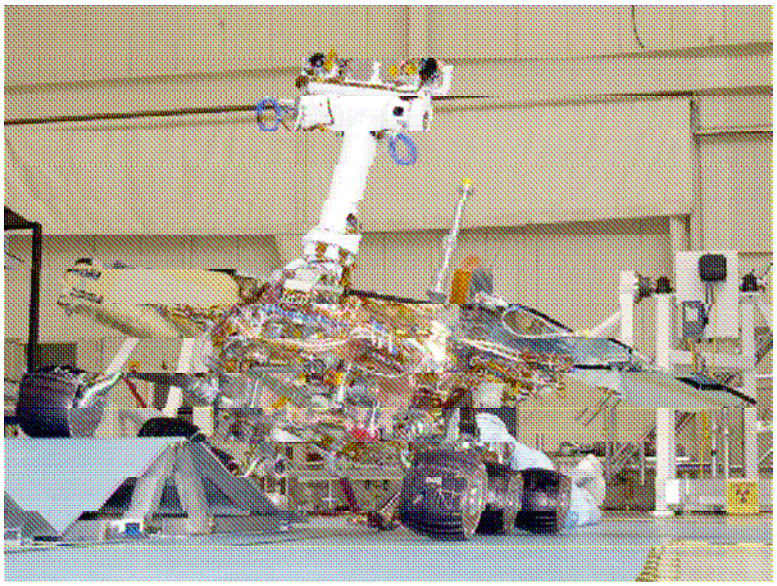


Figure 3. Illustration of a MER during testing.

Rocky 7, shown in Figure 4, was specifically designed for testing robotic units that could traverse long distances from a landing site. Rocky 7 is slightly larger and heavier than NASA's 1997 Sojourner [24], being 60x40x35cm^3 and 15.5kg. Rocky 7 employs a rocker-bogie six-wheel configuration [25] and uses black-and-white CCD cameras for hazard avoidance, navigation telemetry, and science data. Rocky 7's software architecture is based on the framework provided by Real Time Innovation's Control Shell [26], which facilitates the creation of C++ software modules that are connected into asynchronous finite-state machines and synchronous data-flow control loops.

Figure 4. Illustration of Rocky 7 during a digging process.

The **LEMURs**, which are shown in Figure 5, are small and agile four-legged or six-legged walking and/or climbing robots that have been built at JPL to perform dexterous small-scale assembly, inspection, and maintenance. Each limb is reconfigurable to allow the integration of a variety of mechanical tools. LEMURs also utilize a stereo vision system, which consists of a set of black-and-white cameras mounted on the front of the body. The LEMURs are autonomously operated by using the combined images taken by the stereo cameras to create a 3D model of the world in view. LEMURs' other visual feedback is provided by a "palm" mounted camera, which is based upon the idea of foot sensors found in certain insects.

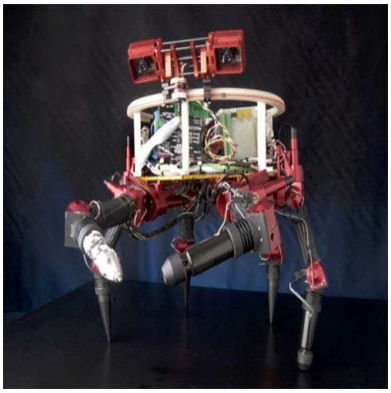
a.)  b.) 

Figure 5. Illustration of a.) LEMUR IIa during its standing algorithm and b.) LEMUR IIb during a its climbing algorithm.

These rover systems provide a brief overview of state-of-the-art robotic platforms for space exploration. While a respectable number of robotic systems have been deployed in NASA missions, none have been able to fully traverse all aspects of variable terrain found on planetary surfaces. Furthermore, to date, the issue of autonomous traversable decision making as yet to be solved. To address these issues, we have developed a robotic platform that provides the benefits of both wheel and leg locomotion without requiring user input for navigation.

Chapter 2

RESEARCH METHODOLOGY

2.1 Vision-based Terrain Characterization and Traversability Assessment

Our robotic system is made autonomous using the control loop shown in Figure 6, which uses a vision system for a terrain classification process. The first step in classifying the terrain involves extracting those terrain characteristics that directly account for robot traversal difficulty [5]. This particular algorithm determines terrain roughness by assessing ground landmarks.

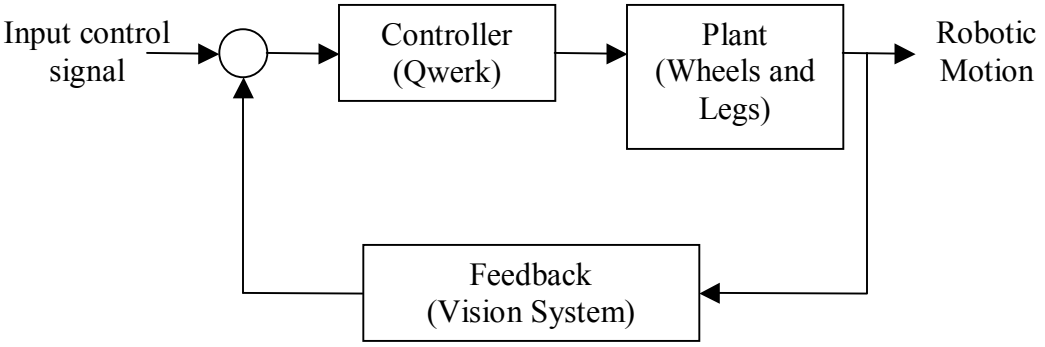


Figure 6. Control loop for robotic autonomous reconfiguration.

In order to determine the obstacles that are in the ground plane as opposed to objects in the horizon, which is not traversable by this unmanned ground vehicle, a horizon-line (i.e. boundary line) is first identified; this line segments the camera data into two regions: ground and backdrop (e.g. mountains and sky) [5]. To distinguish between the ground and the landscaped backdrop, the average pixel signature A_g and standard deviation of the ground plane s_g is first calculated [5] using the equations below:

$$\mathbf{A_g} = \frac{\sum_{j=y_0}^{y_1}\sum_{x_0}^{x_1} p_{i,j}}{(x_1 - x_0)(y_1 - y_0)} \qquad \textbf{(Eq. 2)}$$

$$\mathbf{s_g} = \sqrt{\frac{\sum_{j=y_0}^{y_1}\sum_{i=x_0}^{x_1} p_{i,j} * p_{i,j}}{(x_1 - x_0)(y_1 - y_0)} - A_g{}^2} \qquad \textbf{(Eq. 3)}$$

In order to determine the amount of pixels in a row that was most efficient for selecting a horizon line location, we used a confidence interval. By taking N samples from images obtained from the camera system and recalculating the confidence interval from each sample, we were able to determine a proportion μ_1 that would contain the desired horizon line values; details of the confidence interval process and results are discussed in Chapter 3. The point in the pixel data at which there is greater than or equal to μ_1 in a specified row that meet the criteria shown in Figure 7, where μ_2 is the percentage of A_g used to specify objects lying in the ground plane, is considered to be the horizon line. Concurrently, everything above the horizon line is shaded gray and is considered untraversable.

$$1.) p_{i,j} < A_g - s_g$$
$$2.) p_{i,j} > A_g * (1 - \mu_2)$$

Figure 7. Criteria for pixel categorization used to identify the horizon line.

Following the calculations of A_g and s_g and identification of the horizon line, a calculated pixel value assessment of the objects lying in the ground plane is evaluated. Again, we utilized a confidence interval to determine that a percentage of A_g, termed μ_2, was efficient for locating objects in the ground plane. Specifically, if a pixel value at a

point in the matrix is less than μ_2 of A_g, then the pixel is colored black and is considered to be an object other than the ground. These objects are termed landmarks. All other values in the ground plane that are not signified as being landmarks are set to a uniform value, specifically A_g. An illustration of the resulting landmark detection image along with the original image, grayscale image, and horizon line segmented image is shown in Figure 8.

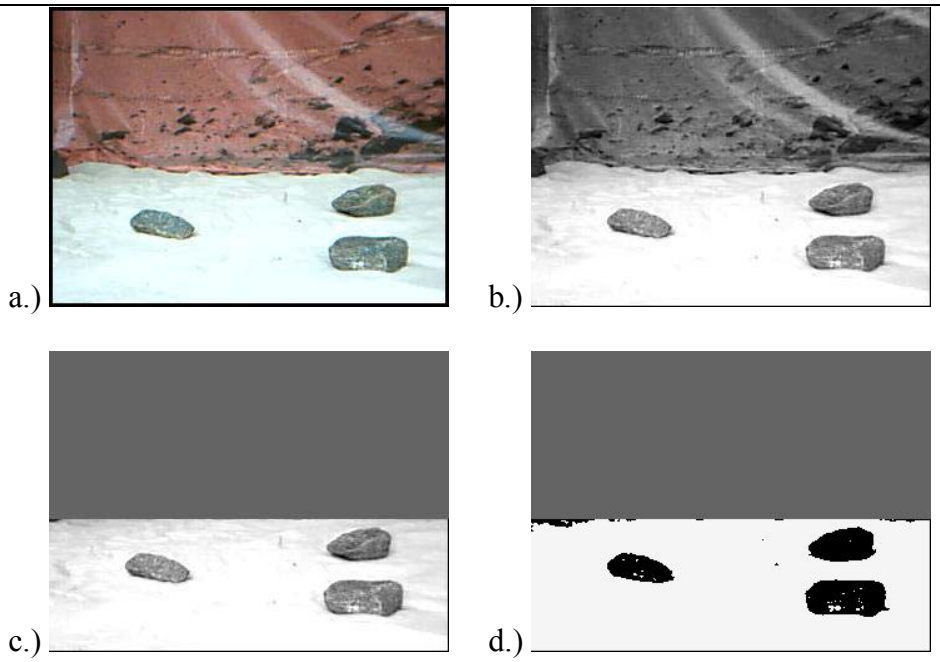


Figure 8. a.) Original image as seen by the camera on the robotic system. b.) Converted Grayscale image. c.) Horizon line segmented image. d.) Landmark detection image.

Utilizing the image in d.), our robotic system is able to use a region growing algorithm to assess whether or not its displacement has changed in the last τ seconds. As added proof of the functionality of this research and algorithm, still images of a terrain in Colorado were also used to test the system; a sample still image, along with its respective processed images is shown in Figure 9.

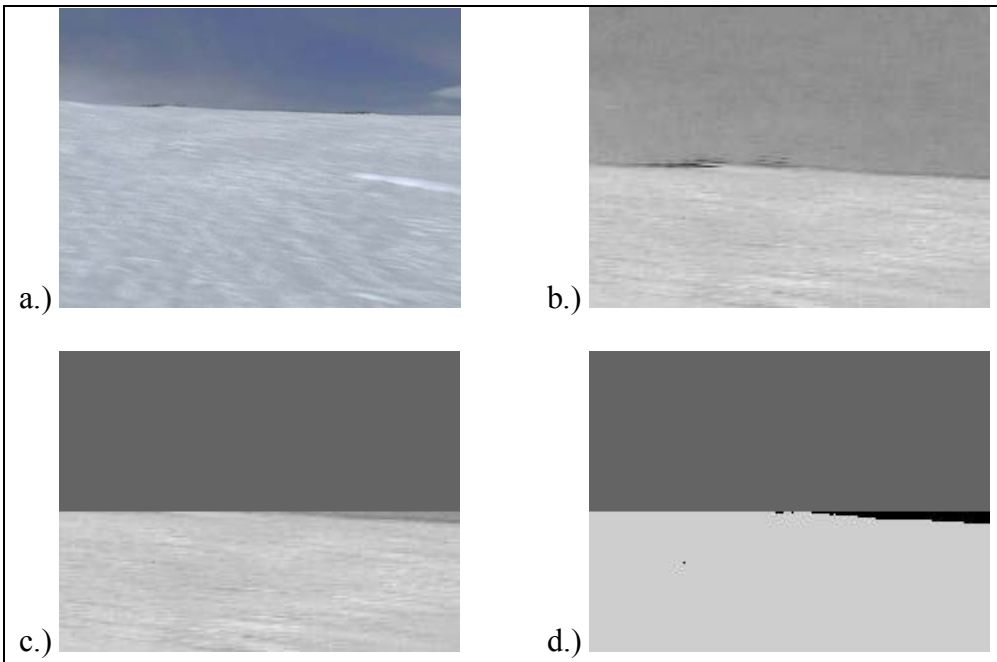


Figure 9. a.) Original image. b.) Converted Grayscale image. c.) Horizon line segmented image. d.) Landmark detection image.

It is noted that the viewable size of many of the original images area is larger than that of the processed images. The original camera data is able to output an image with a resolution of 640X480. However, due to an increase in the amount of frames per second (fps) that can be processed, which is discussed in Chapter 3, we determined that it would be more beneficial to downsample the images to a resolution of 200X150. The resulting images are basically zoomed versions of the original image.

2.2 Robotic Mobility Assessment Based on Movement Criteria and Implementing the Region-Growing Algorithm

Here we use the region-growing method to follow the logic regarding immobility as described in Chapter 1. This method checks to see if there has been any movement within the last τ seconds. The priority checking is as follows:

1. Determine if A_g has changed by more than α pixel values.

2. If A_g has not changed by more than α pixel values, then check to see if the horizon line has changed positions.

3. If the horizon line has not changed positions, then check to see if there are pixels in a specific region size (using the region-growing algorithm) that are identical and are not equal to A_g and those pixels have not changed by β pixel values.

Step three uses the following region growing process:

1. Start by using cell (i, j) as the initial seed pixel and compare it with its neighboring pixels (up to eight in a standard matrix).

2. Add in neighboring pixels that are within β pixel values of the seed pixel.

3. When the growth of one region stops, choose another seed pixel that does not belong to a region and perform step 2 again until the region is fully grown.

If all three criteria are true, then there has not been any movement in the last τ seconds. Once the above criteria have been met, the robot will consider itself to be immobile, cease its driving mechanism, and commence its walking gait. Figure 10 shows our robotic platform during its reconfiguration process.

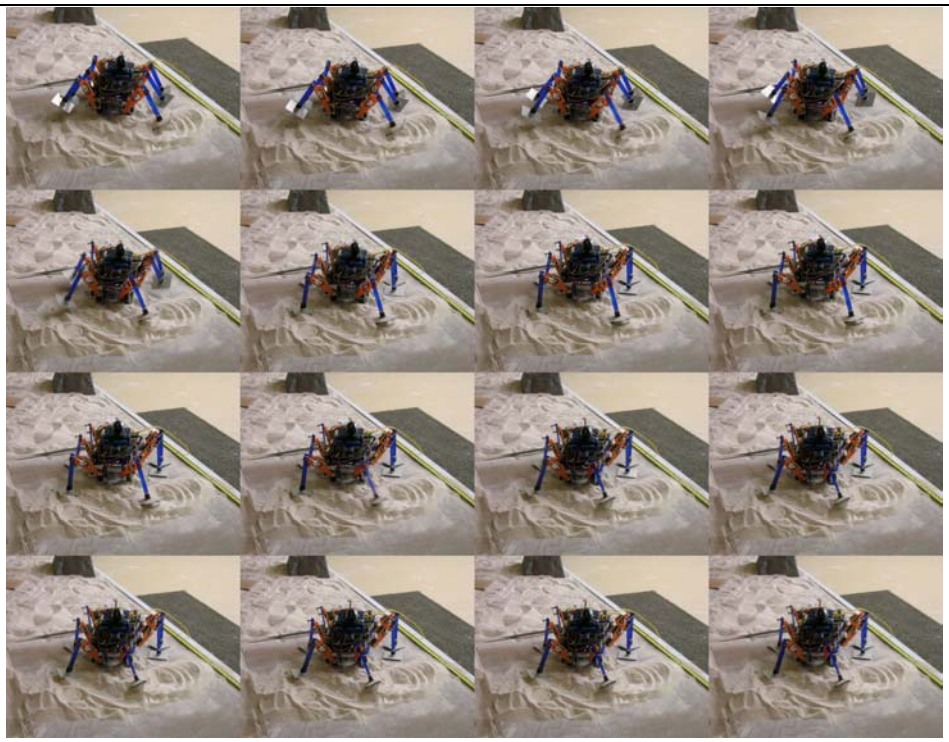


Figure 10. Robotic platform during its reconfiguration process.

The logic behind the criteria for movement recognition can be explained in the following manner. Regarding the changes of A_g, the color value of areas on the ground in one location are not perfectly identical to those in another location due to grain variations in regards to color and position and/or shadowing of landmarks. If our robotic unit has become immobile, those variables will remain constant. The significance of the horizon line positional change uses the concept that "the world is not flat". The land mass directly in front of the robot is perceived to change slightly due to its rounded contour as the robot traverses the terrain. Granted, the perception is based on a certain amount of distance traversed, which forces us to utilize a third criterion that is based on the landmark analogy described in Section 1.1.

Chapter 3

EXPERIMENTS AND RESULTS

3.1 Infrastructure

Our robotic unit, ByRobot II, was first modeled on Pro-Engineer, and then the parts were constructed in the Georgia Tech MRDC Machine Shop [1]. It is able to drive on its four wheels, as well as utilize its legs to stand and walk for operating on rough terrain. ByRobot II's core controller is the Qwerk Controller created by Charmed Labs, LLC. The Qwerk Controller powers and controls ByRobot II's various electronics. The camera system consists of the Logitech Communicate STX Webcam, and is used for processing robot displacement.

3.1.1 Qwerk Controller

Figure 11. The Qwerk Processor created by Charmed Labs [27].

The Qwerk controller is the central processing unit of ByRobot II. It powers the motors and servos for wheel and leg movement respectively, connects ByRobot II to the source code contained on the computer, and receives and sends the visual information

from the camera to the computer monitor. The specifications of the Qwerk controller can be found in Appendix C.

3.1.2 Logitech Communicate STX Camera

Figure 12. Logitech Communicate STX Webcam [28].

The Logitech Communicate STX Camera is the sole sensor system used to guide ByRobot II. It has video capture capability up to 640X480 square pixels, 1.3 megapixel still image capture capability, and a frame rate up to 30 frames per second. Details regarding the positioning of the webcam are found below and are illustrated in Figure 13.

By taking physical measurements, the length of Y, which represents the vertical distance from the camera to the ground plane and is shown in Figure 13, was 9.4" and H, which represents the linear distance from the camera lens to the ground plane (i.e. hypotenuse), was 15". Also, because the platform of ByRobot II measured 90° from the ground plane, by placing the camera parallel to the base of ByRobot II the camera was effectively placed 90° from the ground plane. Using simple trigonometry, Θ_1, Θ_2, Θ_g, and X can be calculated as shown in Equations 4, 5, and 6 respectively, where Θ_1 is the angle between the vertical plane created by the camera and the ground and H, Θ_2 is the angle between H and the ground plane, Θ_g is the downward camera tilted angle in

reference to the normal horizontal view, and X is the horizontal distance from the camera lens to the visible center point in the ground plane.

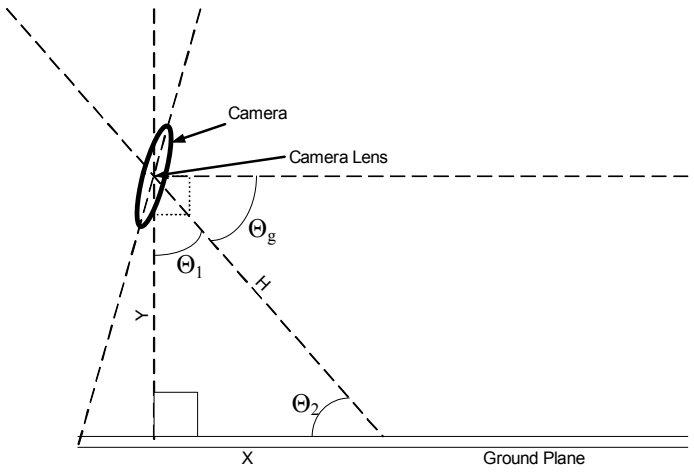


Figure 13. Illustration of the distances and angles in regards to the camera.

$$\Theta_1 = \cos^{-1}\left(\frac{9.4''}{15''}\right) = 51.19^\circ \qquad \textbf{(Eq. 4)}$$

$$\Theta_2 = \sin^{-1}\left(\frac{9.4''}{15''}\right) = 38.81^\circ = \Theta_g \, or$$
$$\Theta_g = 90^\circ - \Theta_1 = 38.81^\circ \qquad \textbf{(Eq. 5)}$$

$$X = \sin(\Theta_1) \times 15'' = 11.69'' \qquad \textbf{(Eq. 6)}$$

3.1.3 Telepresence Robot Kit (TeRK)

The software used to control ByRobot II is based upon the software provided by the researchers at Carnegie Mellon University (CMU), specifically found in CMU's TeRK project. The vision and control software was developed in the HumAnS Laboratory at the Georgia Institute of Technology and is described in the next section.

3.1.4 The Program in Detail

3.1.4.1 The Prototyping Playground GUI

The Java based software utilizes Sun Microsystems's Internet Communications Engine (ICE) in order to provide connectivity through an Ethernet cable or wirelessly. It also provides a Graphical User Interface (GUI), which is ideal for user-friendly software. The GUI, which is shown in Figure 14, displays streaming video so that the user can track what the robot is currently viewing. It also has multiple buttons that allow the user to manually control the robot, if desired. In order to begin the autonomous process the user must first begin the streaming video by clicking the "Start Video" button. After that, the user must click the button labeled "Begin Process"; this will start the robot's forward movement and enable its displacement recognition algorithm, which is described in Chapter 2.

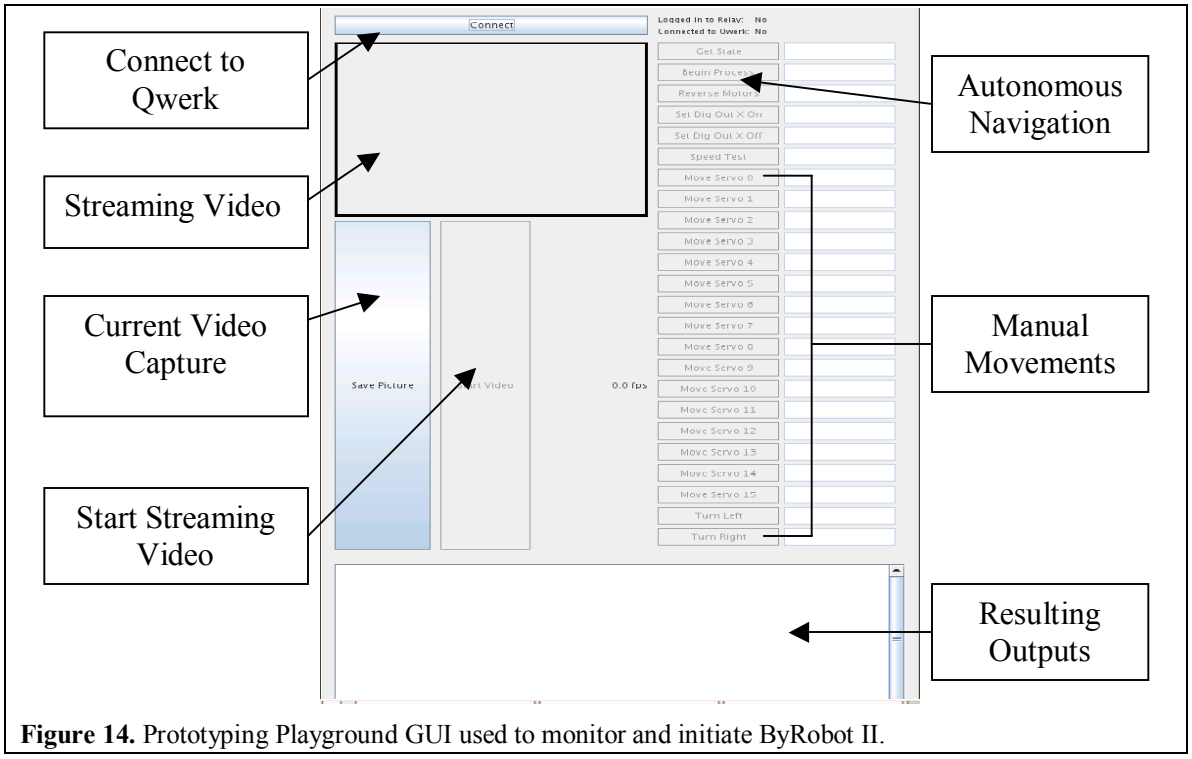


Figure 14. Prototyping Playground GUI used to monitor and initiate ByRobot II.

3.1.4.2 Changing the Input Data from an Array to a Matrix

In order to create a near real-time processing system and a format that can be utilized in the region growing algorithm, the data that is read from the camera has to be parsed into a matrix of pixels. The simplest method for converting an array into a matrix is to segment the input data by width and height. In the process of creating the data matrix, the image byte values of the array can be converted into grayscale integer values by multiplying the byte values by the hexadecimal value of 0x0FF. Converting to grayscale simplifies the immobility recognition process because there is less color data to analyze. An illustration of the conversion process is shown in Figure 15.

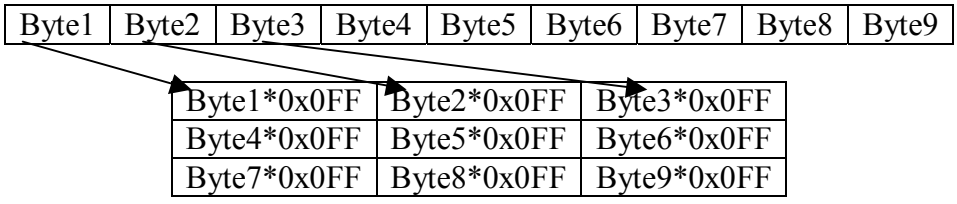


Figure 15. Illustration of converting a byte array to an integer matrix.

The newly created matrix of pixels can now be viewed as an x-y coordinate plane enabling pixel identification and location. This matrix can then be utilized to assess robot displacement changes.

3.1.4.3 Video Processing

The core of the program lies within the video processing section. Here, we are able to capture streaming video of the robot's current location and analysis regarding immobility, thus enabling us to monitor its progress. The section begins by applying the frames per second calculation described in Section 3.2.2. Following this calculation, we capture the

current image that the camera system is viewing and begin the vision based terrain characterization and region-growing processes described in Chapter 2. Our video processing also saves the current original, grayscale, horizon line segmented, and landmark detection images while processing the information for determining the movement criteria. Once the video processing has completed an iteration and determined whether or not forward progress is being achieved, the wheeled-legged mobility system will correspond based on the motion described in the following section.

3.1.4.4 Motor and Leg Movement

Currently, our mobility system utilizes its wheels as often as possible to traverse various terrains in the fastest manner. Because the goal of this research project is to recognize immobility and reconfigure to employ the leg system, various methods of exploration such as wandering and following were not implemented. Instead, we used the wheels for a continuous forward motion, and if becoming immobile, we used the legs to move forward in order to regain wheeled mobility. We did, however, try several walking gaits to determine the most effective for our purpose. We found that simply standing and pushing forward with all legs (either four or six) simultaneously generated the best results for regaining wheeled mobility.

3.2 Results

3.2.1 Horizon Line and Object Confidence Intervals

For the purposes of determining an efficient percentage of pixel values in a row that would determine the position of the horizon line, we utilized a confidence interval. A confidence interval addresses the issue of how well a sample statistic estimates an underlying population value by providing a range of values which is likely to contain the

population parameter of interest. The computational process that we use can be found in [29].

The confidence interval is one-sided, utilizing a lower-bound, because values that are greater than the lowest critical value would be considered part of the horizon. Our null hypothesis H_o was that 50% of the pixels in a row that met the horizon line criteria as described in Chapter 2 would suffice for choosing the horizon line position, while our alternative hypothesis H_A was that 50% was not sufficient. We label the assumed mean corresponding to the null hypothesis χ_o.

Using the randomly chosen percentages in Table 1 representing the amount of sufficient pixels for determining the horizon line, we calculated that the sample mean $\overline{Y}$ = 92.8 = A_p * 200 values in a row, where A_p = 46.4% is the mean in terms of percentages. Using the standard equation for calculating the standard deviation, shown in Equation 7 where Y is the current sample, we found that σ = 27.75488.

Table 1. Randomly Selected Percentages for Calculating the Sample Mean for the Horizon Line

Image 1	Image 2	Image 3	Image 4	Image 5
30	25	10	5	0
40	35	40	11	1
50	45	60	42	39
60	55	80	61	71
70	65	100	73	92

$$\sigma = \sqrt{\frac{\sum \left(Y - \overline{Y}\right)^2}{(\mathrm{N} - 1)}} \qquad \textbf{(Eq. 7)}$$

The images and their respective horizon lines based on the percentages from the table are shown in Figures 16-20.

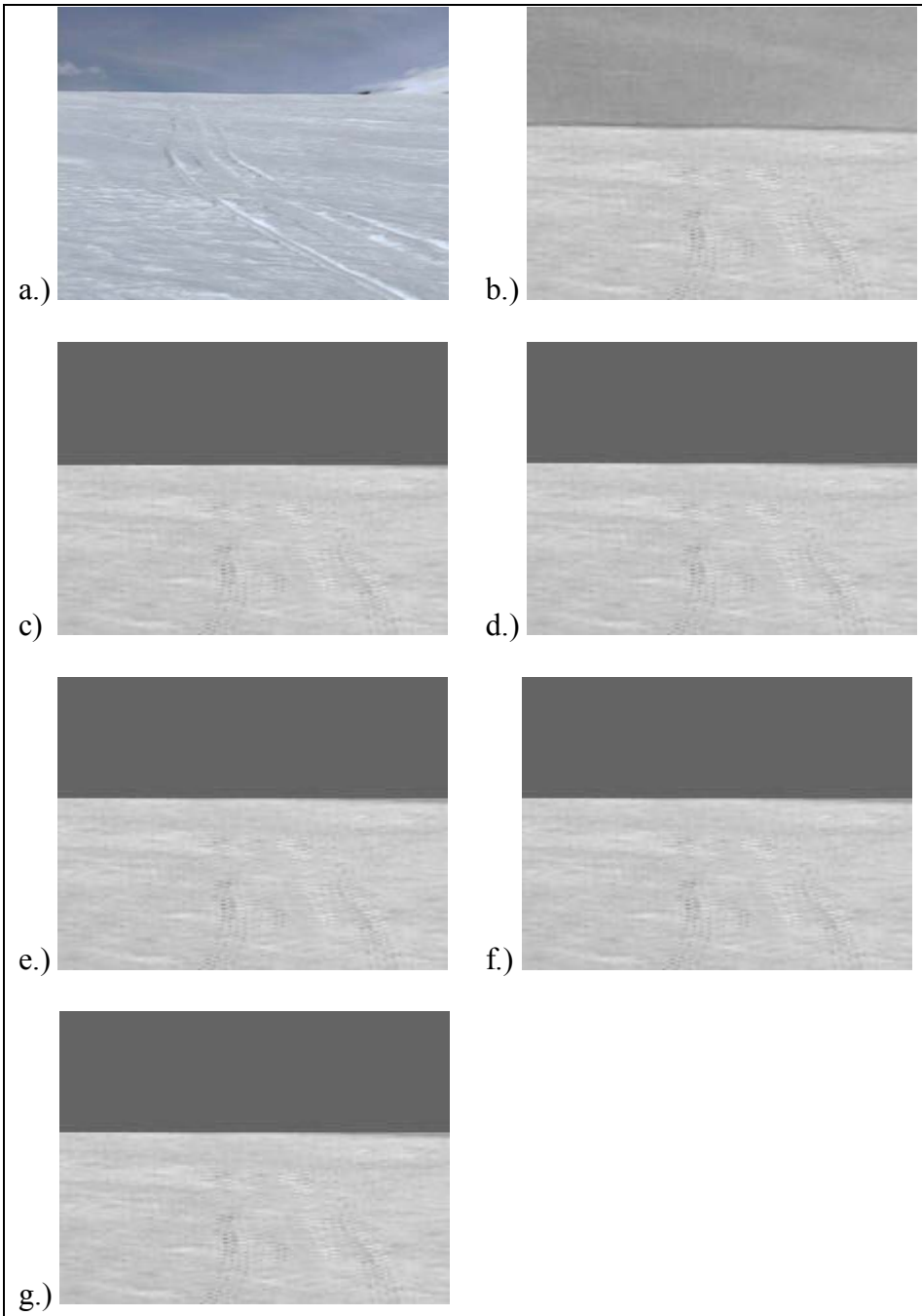


Figure 16. Image for horizon line confidence interval calculation. a.) Image 1 b.) 200x150 grayscale image c.) Horizon line at 30% d.) Horizon line at 40% e.) Horizon line at 50% f.) Horizon line at 60% g.) Horizon line at 70%

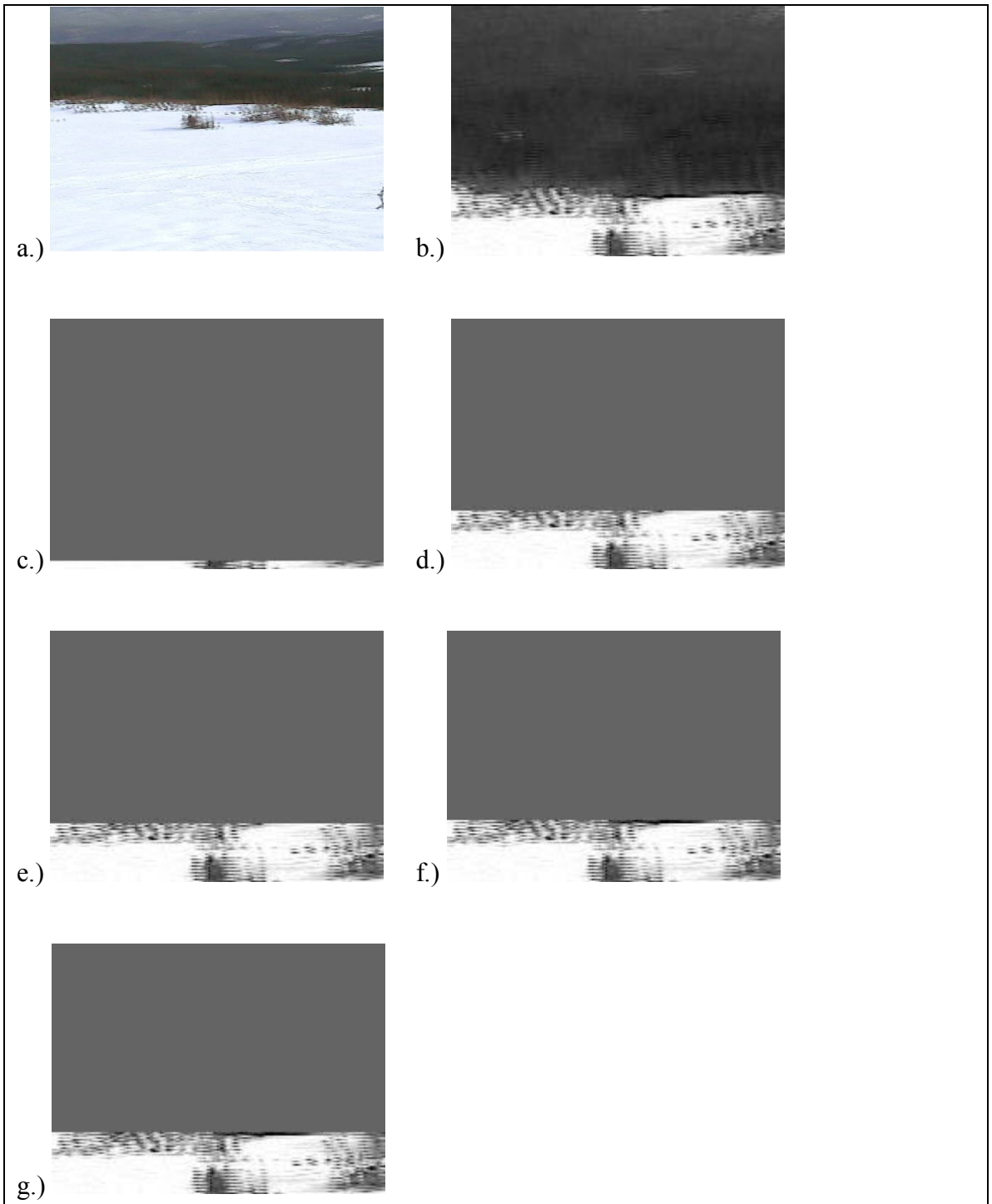


Figure 17. Image for horizon line confidence interval calculation. a.) Image 2 b.) 200x150 grayscale image c.) Horizon line at 25% d.) Horizon line at 35% e.) Horizon line at 45% f.) Horizon line at 55% g.) Horizon line at 65%

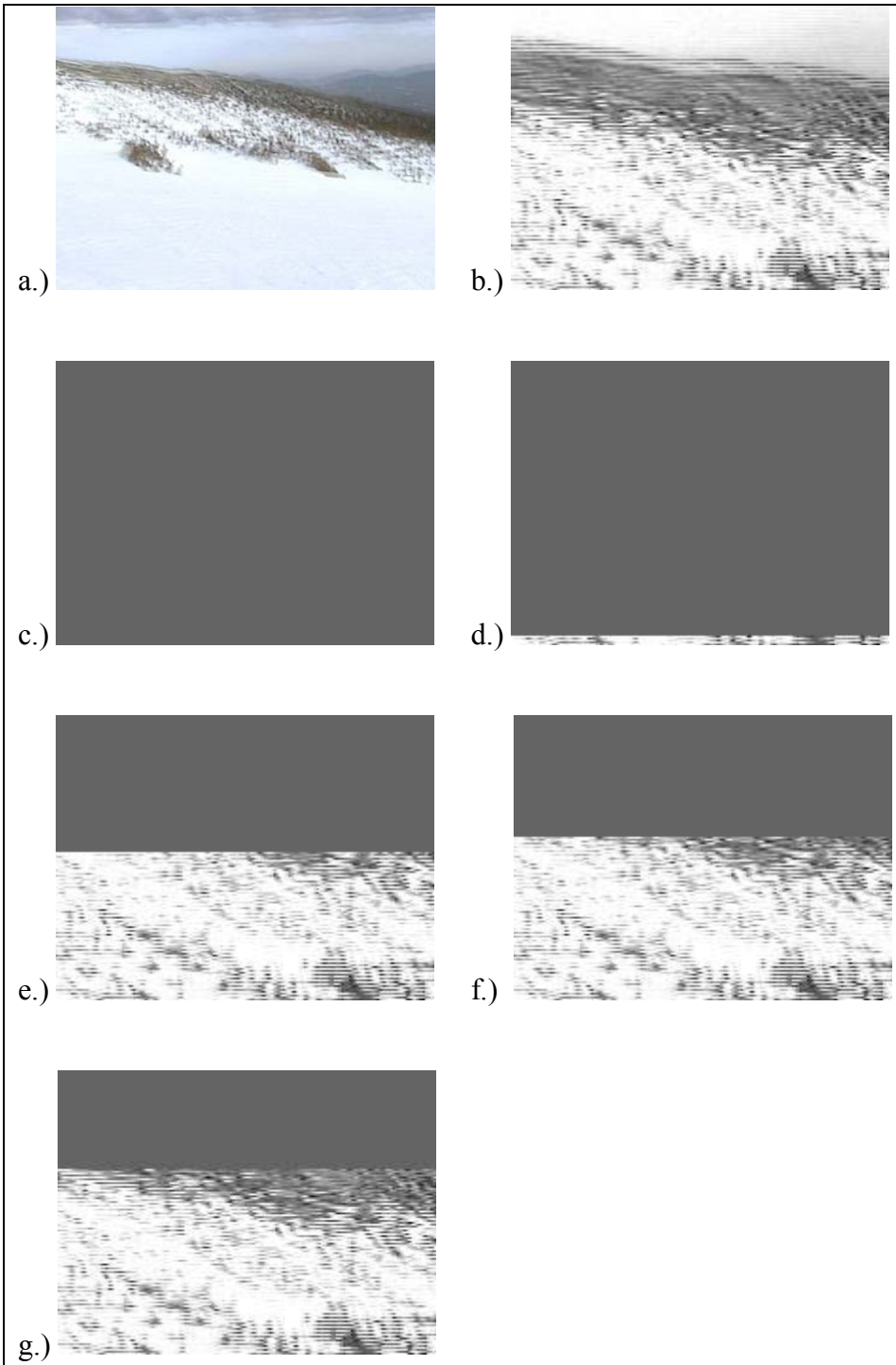


Figure 18. Image for horizon line confidence interval calculation. a.) Image 3 b.) 200x150 grayscale image c.) Horizon line at 10% d.) Horizon line at 40% e.) Horizon line at 60% f.) Horizon line at 80% g.) Horizon line at 100%

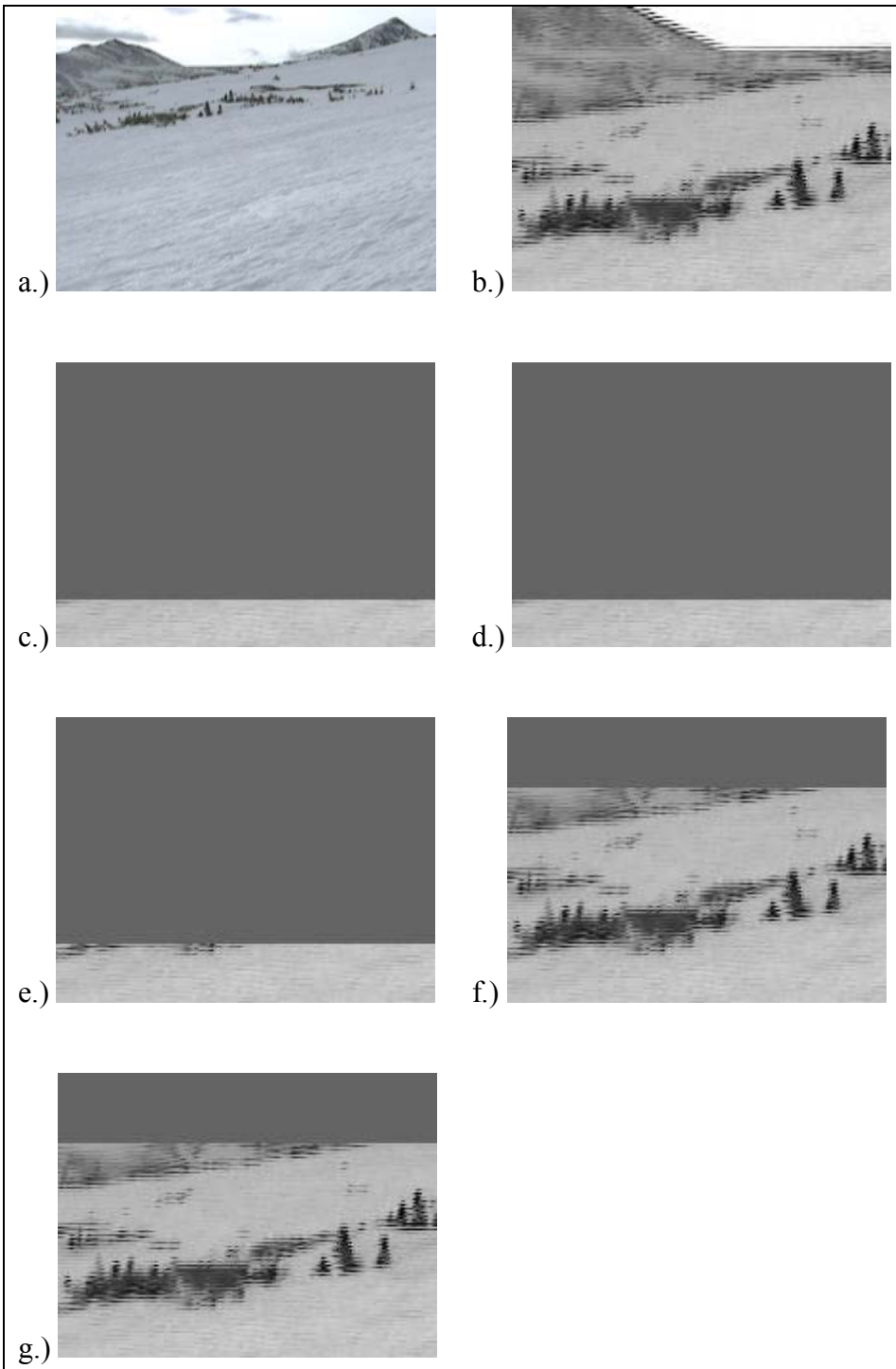


Figure 19. Image for horizon line confidence interval calculation. a.) Image 4 b.) 200x150 grayscale image c.) Horizon line at 5% d.) Horizon line at 11% e.) Horizon line at 42% f.) Horizon line at 61% g.) Horizon line at 73%

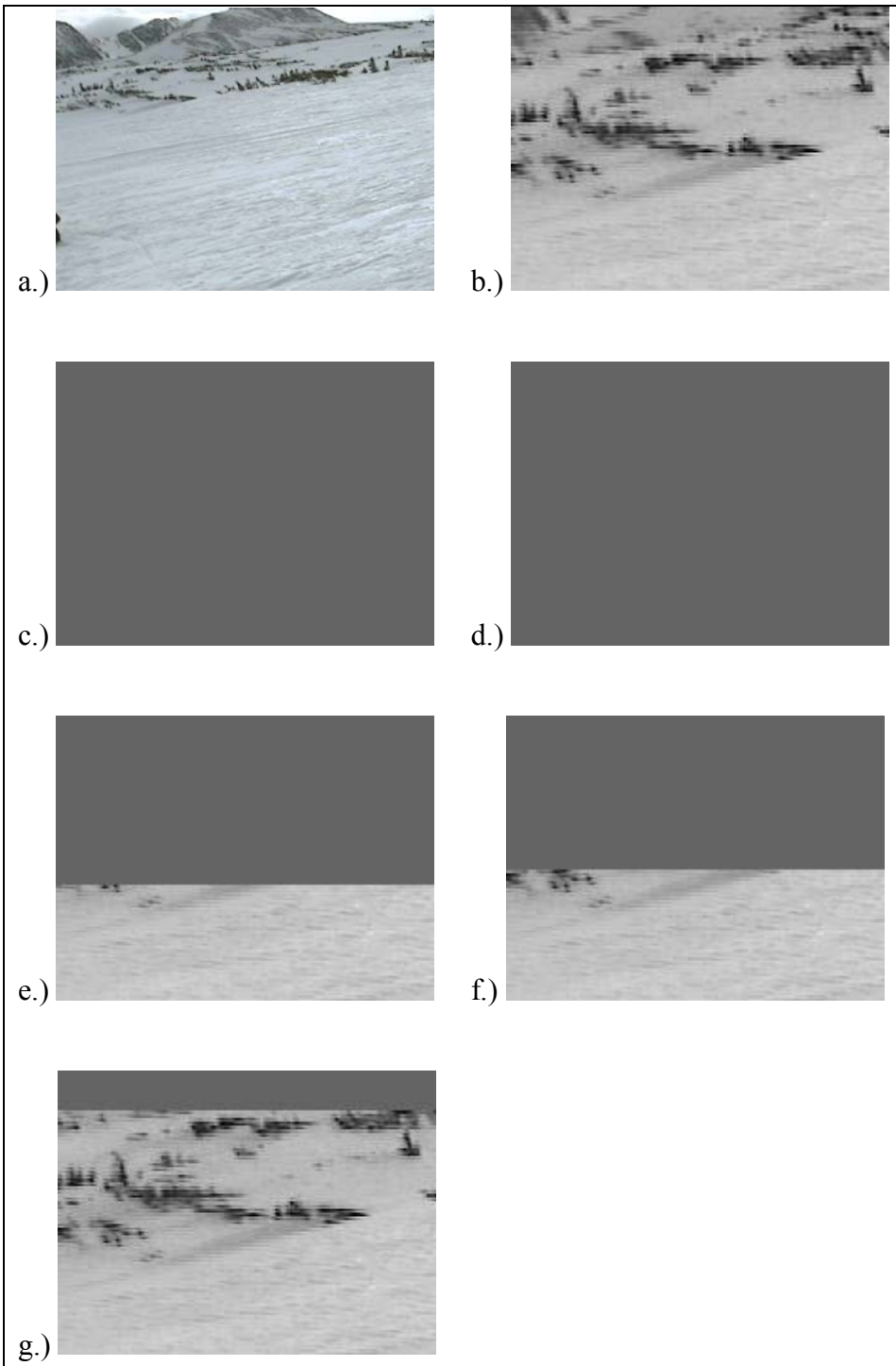


Figure 20. Image for horizon line confidence interval calculation. a.) Image 5 b.) 200x150 grayscale image c.) Horizon line at 0% d.) Horizon line at 1% e.) Horizon line at 39% f.) Horizon line at 71% g.) Horizon line at 92%

Utilizing Equation 8, where $\overline{Y}$ is the sample mean, χ_o is the assumed mean (in this case $\chi_o = 100 = 50\% * 200$ values in a row), σ is the known standard deviation, N is the number of samples, and Z_a is the standard critical value, we are able to accept or reject the sample mean.

$$Z_a = \frac{\overline{Y} - \chi_o}{\sigma/\sqrt{\mathrm{N}}} \qquad \textbf{(Eq. 8)}$$

Substituting the variables for their respective given and computed values yields:

$$Z_a = \frac{92.8 - 100}{27.75488/\sqrt{25}} = -1.297 \qquad \textbf{(Eq. 9)}$$

Using the significance level $\upsilon = 0.05$, the chances of erroneously rejecting the null hypothesis when it is true are 5% or less. In order to justify the assumption that σ is reliable, we utilize the t-statistic. Setting $Z_a = t$, we are able to use the t distribution table found in Appendix I along with the following criterion to prove the reliability of σ, where t_υ is the upper υ critical value from the *t* distribution table. Since the hypothesis would be rejected if the criterion were true, we are able to use $\geq$ 50% of the pixels in a row to determine the horizon line.

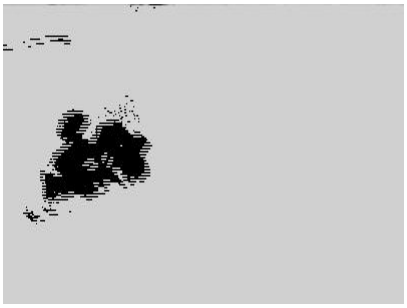
$$1.\ H_0\text{: } \overline{Y} \leq \chi_o \qquad 1.\ t \overset{?}{\geq} t_{\nu;\mathrm{N}-1} \equiv -1.297 \overset{?}{\geq} 1.711 \qquad \textbf{(Eq. 10)}$$

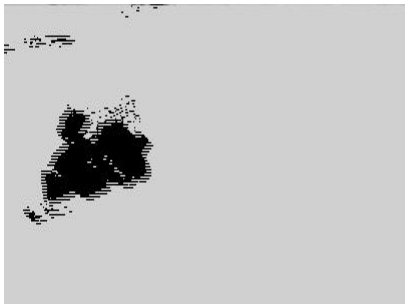
Following the same procedure above, we are able to calculate the confidence interval for determining objects lying in the ground plane. Our null hypothesis H_o is that 80% of the A_g will need to be used in order to recognize objects in the ground plane. Thus χ_o = 80%. Using the percentages, 30%, 40%, 70%, 80%, and 90%, for the image shown in Figure 21a, we calculated that $\overline{Y}$ = 62% and σ = 25.88436 using Equation 7. Given N =

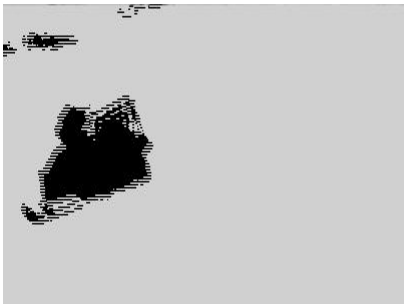
5, we can compute $Z_a = -1.5549$ using Equation 8. Setting $Z_a = t$, following the same criterion in Equation 10, using the t distribution table, and using a significance level $\upsilon = 0.05$, we can prove that 80% is 95% reliable for object recognition.

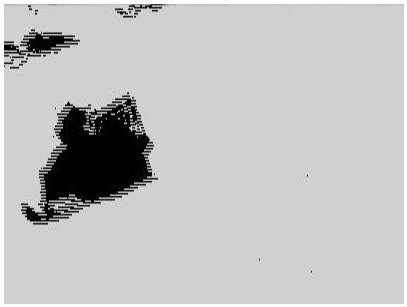
a.) 

b.) 

c.) 

d.) 

e.) 

f.) 

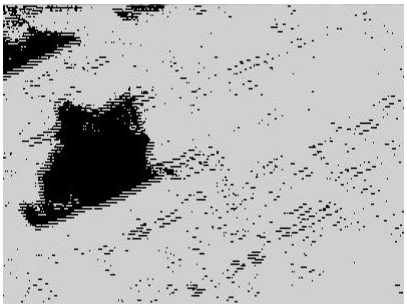
g.) 

Figure 21. Image used for object detection confidence interval calculation. a.) Original image b.) 200x150 grayscale image c.) Object recognition using 30% d.) Object recognition using 40% e.) Object recognition using 70% f.) Object recognition using 80% g.) H Object recognition using 90%

3.2.2 Resolution and Timing Results

As stated in Chapter 2, processing time is a significant factor when using imaging data. The original data from our camera system was able to utilize images with a resolution of 640X480; however, the processing time was more lengthy that when using smaller resolutions. In order to determine the maximum and minimum allowable resolutions that could be processed, we calculated the frames per second (fps) for three different camera resolutions. The fps for each resolution can be calculated using Equation 11, where F is the frame rate, η is the number of frames between each fps computation, M is the metric system computation of time (i.e. in this case M is 1000 because 1s = 1000ms), and $\Delta\tau$ is the elapsed time in milliseconds.

$$F = \frac{\eta \mathrm{M}}{\Delta \tau} \qquad \textbf{(Eq. 11)}$$

Table 2 shows the results of the frame rates calculated and tested for the three different camera resolutions. It is worth noting that at the maximum resolution of 640X480 the third criteria for a successful decision regarding immobility failed; in other words, the reconfigurability process was never commenced even during immobility.

Table 2. Frame Rates for Different Resolutions

Resolution	*F* (fps)
200X150	16.6
320X240	10.5
640X480	07.0

In regards to the amount of time allowed to determine immobility, we chose $\tau = 5$ seconds. Although this time can be varied simply in the program, it gives the camera sufficient time to update its current frame as well as allow the program to process current and past information. In comparison, the MERs communication latency time, which is governed by the speed of light, takes approximately 10 minutes each way, at the smallest distance between Mars and Earth, during landing [2]. Also, the MERs GESTALT system, described in Chapter 1, takes approximately 70s to process and compute information while their visual odometry system takes approximately 15 fps on a 1.6 GHz embedded Pentium-M board [2]. This process time slows the rovers' speed from 5cm/s to 0.6cm/s, which required researchers to incorporate a blind driving segment to speed up the navigation process [2]. In comparison, our earth-bound robotic system is capable of traversing a distance of 120cm in 5 seconds while incorporating its vision and thinking process; the resulting velocity calculates to approximately 24 cm/s.

3.2.3 Testing the Robot in Various Terrains

ByRobot II was tested in 36 experiments: three different terrain settings at three different light settings, four times each. Since locomotion behavior was assessed by monitoring the robot's forward progress [30], the first test setup was to test the displacement recognition and forward progress of ByRobot II in a static environment on solid terrain (i.e. on tile flooring). The second test setup was to test displacement recognition on loose soil (i.e. sand). During this phase of testing, the purpose was not to measure the pace that the robot could traverse the sand, rather we wanted to measure the effectiveness of the camera system and algorithm in different lighting scenarios on loose

soil. The final test setup was to test displacement recognition and forward progress on solid and loose terrain. In this final testing environment, ByRobot II was initially placed on a solid piece of cardboard and was allowed to initialize its forward movement algorithm. Along its path, different size sand dunes were placed as obstacles that would induce difficult travel via wheels; basically, the walking gait would need to commence in order to traverse the obstacles and continue forward movement. Figure 22 shows the solid/soil test setup, and Figure 23 shows our robot maneuvering over a sand dune using its legs. Each test was performed in three different light settings: 5%, 50%, and 100%. Table 3 shows the results of the experiments.

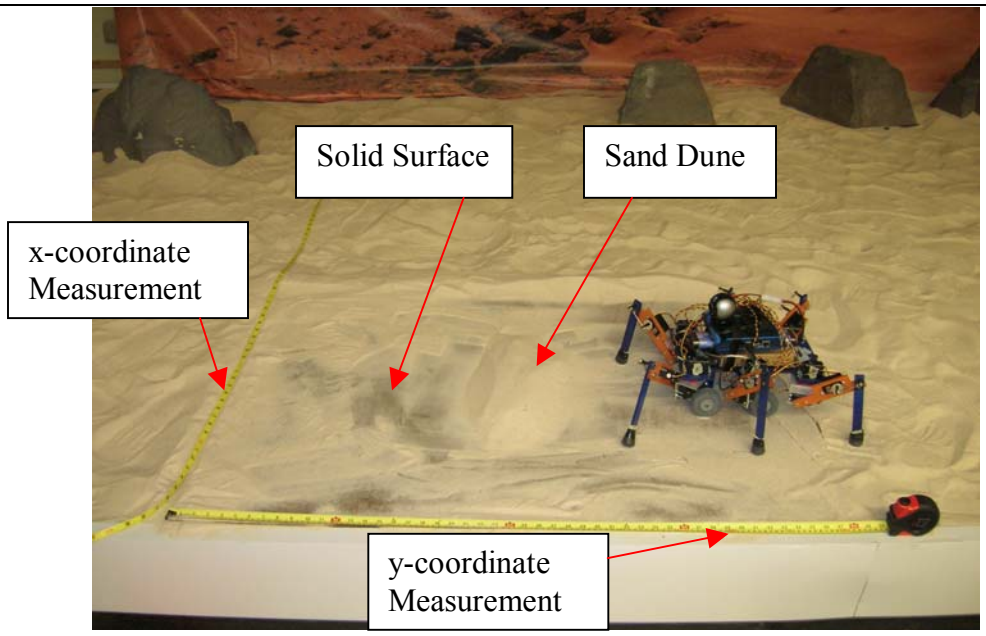


Figure 22. Solid/Soil test setup.

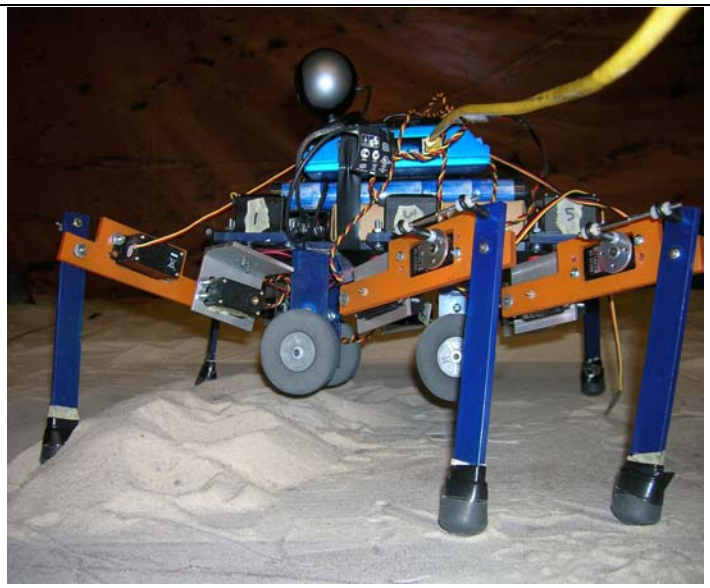


Figure 23. ByRobot II maneuvering over a sand dune.

Table 3. Results of Variable Lighting and Terrain Experiments using the Six Leg Mobility System

Trial #	Terrain	Lighting	Starting Y	Starting X	Ending Y	Ending X	Time	Wheels Stopped?	Legs Deployed?	Continued Progress?
1	Tiled Floor	Glowstick (5%)	0"	14"	23"	10"	01:54:15	yes	yes	No; Legs deployed 9 times before progress was considered discontinued; obstacle too high
2	Tiled Floor	Glowstick (5%)	0"	11"	17"	10"	02:27:86	yes	yes	No; Legs deployed 11 times before progress was considered discontinued; obstacle too high
3	Tiled Floor	Glowstick (5%)	0"	17"	22"	22"	03:09:51	yes	yes	Yes; Legs deployed 15 times in order to continue; robot eventually moved away from the light and move into darkness where criteria become obsolete

4	Tiled Floor	Glowstick (5%)	0"	17"	25"	29"	05:15:02	yes	yes	Yes; Legs deployed 21 times in order to continue; robot eventually moved away from the light and move in darkness where criteria become obsolete
1	Tiled Floor	Florescent (100%)	0"	8"	25.5"	8"	01:29:36	yes	yes	No; Legs deployed 9 times before progress was considered discontinued; obstacle too high
2	Tiled Floor	Florescent (100%)	0"	32"	25"	34"	01:24:51	no	no	No; it is speculated that battery power created an issue with the process.
3	Tiled Floor	Florescent (100%)	0"	36"	55"	60"	02:13:02	yes	yes	Yes; Legs needed to deploy 9 times in order to continue
4	Tiled Floor	Florescent (100%)	0"	37"	50"	56"	01:35:82	yes	yes	Yes: Ran into the issue where there was no obstacle in the image, but the algorithm considered itself immobile (i.e. no surroundings)
1	Tiled Floor	Lamp (50%)	0"	12"	18"	37"	04:13:02	yes	yes	Yes; Legs needed to deploy 14 times in order to continue
2	Tiled Floor	Lamp (50%)	0"	7"	22"	11"	05:02:18	yes	yes	Half of body traversed obstacle; legs deployed 21 times.
3	Tiled Floor	Lamp (50%)	0"	23.7"	25"	13"	05:00:19	yes	yes	Yes; Legs needed to deploy 21 times in order to continue
4	Tiled Floor	Lamp (50%)	0"	23.7"	20"	16"	04:02:20	yes	yes	Yes; Legs needed to deploy 15 times in order to continue
1	Sand	Glowstick (5%)	N/A	N/A	N/A	N/A	00:10:01	yes	yes	N/A

2	Sand	Glowstick (5%)	N/A	N/A	N/A	N/A	00:15:19	yes	yes	N/A
3	Sand	Glowstick (5%)	N/A	N/A	N/A	N/A	00:15:13	yes	yes	N/A
4	Sand	Glowstick (5%)	N/A	N/A	N/A	N/A	00:15:42	yes	yes	N/A
1	Sand	Florescent (100%)	N/A	N/A	N/A	N/A	00:05:59	yes	yes	N/A
2	Sand	Florescent (100%)	N/A	N/A	N/A	N/A	00:04:65	yes	yes	N/A. Low time is probably due to camera initialization or human error with clock.
3	Sand	Florescent (100%)	N/A	N/A	N/A	N/A	00:06:28	yes	yes	N/A
4	Sand	Florescent (100%)	N/A	N/A	N/A	N/A	00:06:58	yes	yes	N/A
1	Sand	Lamp (50%)	N/A	N/A	N/A	N/A	00:19:27	yes	yes	N/A
2	Sand	Lamp (50%)	N/A	N/A	N/A	N/A	00:10:46	yes	yes	N/A
3	Sand	Lamp (50%)	N/A	N/A	N/A	N/A	00:12:15	yes	yes	N/A
4	Sand	Lamp (50%)	N/A	N/A	N/A	N/A	00:12:69	yes	yes	N/A
1	Solid/Sand	Glowstick (5%)	6.5"	38"	18.5"	42"	03:56:00	yes	yes	No; Legs deployed 13 times before progress was considered discontinued
2	Solid/Sand	Glowstick (5%)	6.5"	38"	20.5"	44"	03:52:01	yes	yes	No; Legs deployed 11 times before progress was considered discontinued
3	Solid/Sand	Glowstick (5%)	6.5"	38"	18.5"	42"	03:46:02	yes	yes	No; Legs deployed 9 times before progress was considered discontinued
4	Solid/Sand	Glowstick (5%)	6.5"	38"	38"	24"	04:50:06	yes	yes	Yes; Legs needed to deploy 15 times in order to continue

1	Solid/Sand	Florescent (100%)	7"	23.7"	120.2"	9"	03:55:66	yes	yes	Yes; Legs needed to deploy 15 times in order to continue
2	Solid/Sand	Florescent (100%)	5"	23.7"	24"	25"	03:26:13	yes	yes	No; Legs deployed 8 times before progress was considered discontinued
3	Solid/Sand	Florescent (100%)	5"	23.7"	20.5"	25"	04:30:25	yes	yes	No; Legs deployed 15 times before progress was considered discontinued
4	Solid/Sand	Florescent (100%)	5"	23.7"	17.5"	15"	06:21:16	yes	yes	Yes; Legs needed to deploy 21 times in order to continue
1	Solid/Sand	Lamp (50%)	5"	23.7"	18.2"	33"	02:36:05	yes	yes	No; Legs deployed 12 times before progress was considered discontinued
2	Solid/Sand	Lamp (50%)	5"	23.7"	18.3"	30.5"	05:40:16	yes	yes	No; Legs deployed 15 times before progress was considered discontinued
3	Solid/Sand	Lamp (50%)	7.4"	23.7"	22.5"	26"	06:33:77	yes	yes	No; Legs deployed 24 times before progress was considered discontinued
4	Solid/Sand	Lamp (50%)	7.4"	36"	18.5"	18.5"	04:37:75	yes	yes	Yes; Legs needed to deploy 18 times in order to continue

The results show that our immobility recognition algorithm is a reliable method in well illuminated and poorly illuminated settings. The results also indicate that there was an issue with traversability, specifically the amount of time required to maneuver over obstacles when employing the legged locomotion. As an added measurement of our robot's ability to traverse various terrains, we tested it in the same three environments using only four legs. The results shown in Table 4 indicate that using this mobility platform enabled ByRobot II to traverse the terrain faster and with more linear movement

than our previous six legged design. Figure 24 shows our robot traversing the sand dune in using only four legs.

Table 4. Results of Variable Lighting and Terrain Experiments using the Four Leg Mobility System

Trial #	Terrain	Lighting	Starting Y	Starting X	Ending Y	Ending X	Time	Wheels Stopped?	Legs Deployed?	Continued Progress?
1	Tiled Floor	Glowstick (5%)	0.5"	14.5"	20.5"	25"	01:29:04	yes	yes	No; Legs deployed 4 times before progress was considered discontinued; obstacle too high
1	Tiled Floor	Florescent (100%)	0.5"	14.5"	24.5"	11"	02:18:83	yes	yes	Half of body traversed obstacle; legs deployed 7 times.
1	Tiled Floor	Lamp (50%)	0.5"	14.5"	24.5"	11"	01:34:46	yes	yes	Half of body traversed obstacle; legs deployed 4 times.
1	Sand	Glowstick (5%)	N/A	N/A	N/A	N/A	00:09:96	yes	yes	N/A
1	Sand	Florescent (100%)	N/A	N/A	N/A	N/A	00:10:35	yes	yes	N/A
1	Sand	Lamp (50%)	N/A	N/A	N/A	N/A	00:11:99	yes	yes	N/A
1	Solid/Sand	Glowstick (5%)	3"	42"	31"	41"	03:17:40	yes	yes	Yes; Legs needed to deploy 8 times in order to continue
1	Solid/Sand	Florescent (100%)	2"	35"	20"	45"	02:28:37	yes	yes	Yes; Legs needed to deploy 6 times in order to continue
1	Solid/Sand	Lamp (50%)	2"	35"	29"	47"	01:15:92	yes	yes	Yes; Legs needed to deploy 2 times in order to continue

Figure 24. Robotic platform traversing the sand dune using four legs instead of six.

Chapter 4

CONCLUSION & FUTURE WORK

In this research, an algorithm was introduced for determining the change in displacement for a robotic platform based on active vision. After recognizing that there had not been a change in the last few seconds, the robot acted accordingly by deploying its legs and utilizing its walking gait in order to continue its progression across the terrain. Several experimental results proved that this algorithm is effective.

There are two main limitations associated with this particular vision-based and movement method. The first issue is that of illumination. Illumination can lead to misclassification of ground texture due to shadowing. In certain situations, the terrain is obstacle-free; however, adjacent terrain features (such as mountains and hills) cast shadows on the terrain – creating what appear to be obstacles [5].

The second issue remains of how we as humans know that we are moving in an area that is completely one color. For instance, if a person were placed on a treadmill that moved consistently with the persons' speed in a completely dark room, would the person know that he or she is or is not progressing forward? In this situation, there are no surrounding landmarks to signify movement; this is a potential issue for ByRobot II in an isolated area such as Antarctica.

A future work for this research includes implementing a more robust walking gait that would incorporate a ripple gait for speed improvement and adaptable walking depending upon obstacle assessment. The calculation of specific foot placement based on terrain assessments would create more stability and an overall better walking system. However, because legged locomotion is significantly slower and requires more energy, it would

also be beneficial if the vision system provided information regarding the upcoming terrain slope so that rover could plan a path avoiding those areas of steeper slope where the wheels are more likely to be ineffective.

APPENDIX A

RECONFIGURABILITY AND NAVIGATION CONTROL CODE

```
//package com.sun.image.code.jpeg;
import java.awt.*;
import java.awt.event.*;
import java.awt.color.*;
import java.awt.image.*;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.swing.*;
import javax.imageio.*;
import java.io.*;
//import javax.media.jai.*;  Media package won't work due to lack of a video card
import java.lang.Object;
import java.lang.Math;
import edu.cmu.ri.mrpl.TeRK.QwerkState;
import edu.cmu.ri.mrpl.TeRK.client.components.framework.BaseGUIClient;
import edu.cmu.ri.mrpl.TeRK.client.components.framework.GUIClientConstants;
import edu.cmu.ri.mrpl.TeRK.client.components.framework.GUIClientHelper;
import edu.cmu.ri.mrpl.TeRK.client.components.framework.GUIClientHelperEventHandlerAdapter;
import edu.cmu.ri.mrpl.TeRK.client.components.services.QwerkController;
import edu.cmu.ri.mrpl.TeRK.client.components.userinterface.video.VideoStreamEventListener;
import edu.cmu.ri.mrpl.swing.AbstractTimeConsumingAction;
import edu.cmu.ri.mrpl.swing.ImageFormat;
import edu.cmu.ri.mrpl.swing.SavePictureActionListener;
import edu.cmu.ri.mrpl.swing.SpringLayoutUtilities;
import edu.cmu.ri.mrpl.util.ArrayUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;



/**
 * @author Chris Bartley (bartley@cmu.edu)
 */
public final class PrototypingPlayground extends BaseGUIClient
   {
   private static final Log LOG = LogFactory.getLog(PrototypingPlayground.class);

   /** The application name (appears in the title bar) */
   private static final String APPLICATION_NAME = "Prototyping Playground";

   /** Properties file used to setup Ice for this application */
   private static final String ICE_DIRECT_CONNECT_PROPERTIES_FILE =
"/PrototypingPlayground.direct-connect.ice.properties";
   private static final String ICE_RELAY_PROPERTIES_FILE =
"/PrototypingPlayground.relay.ice.properties";

   /** Dimensions used for spacing out GUI elements */
   private static final Dimension SPACER_DIMENSIONS = new Dimension(5, 5);
```

```
/** Line separator, used for appending messages to the message area */
private static final String LINE_SEPARATOR = System.getProperty("line.separator");

/** Number of programmable buttons and text fields */
private static final int NUM_BUTTONS_AND_TEXT_FIELDS = 24;

/** Number of columns to display in the text fields */
private static final int TEXT_FIELD_COLUMNS = 10;

/** Number of frames between each fps computation */
private static final int FPS_FRAME_COUNT = 10;
private static final int FPS_FRAME_COUNT_TIMES_1000 = 1000 * FPS_FRAME_COUNT;


public static void main(final String[] args)
  {
  //Schedule a job for the event-dispatching thread: creating and showing this application's GUI.
  SwingUtilities.invokeLater(
      new Runnable()
      {
      public void run()
        {
        new PrototypingPlayground();
        }
      });
  }

/**Used for Video capture to begin autonomous movement*/
//private final Executor executorPool = Executors.newCachedThreadPool();

/** Programmable buttons */
private final JButton[] buttons = new JButton[NUM_BUTTONS_AND_TEXT_FIELDS];
private final JTextField[] textFields = new JTextField[NUM_BUTTONS_AND_TEXT_FIELDS];

/** buttons */
private final JButton savePictureButton = GUIClientHelper.createButton("Save Picture", true);
private final JButton pauseResumeVideoButton = GUIClientHelper.createButton("Start Video");
private final JButton clearMessageAreaButton = GUIClientHelper.createButton("Clear", true);

private final JLabel framesPerSecondLabel = GUIClientHelper.createLabel("0 fps");

private boolean isVideoStreamPaused = true;

//Condition to contstantly move forward
private boolean MoveForward = true;
//Condition to signify no movement and need to move back and forth and stand to free robot
private boolean BackandForth = false;
private boolean Stand = false;

private long fpsTimestamp = 0;
private int fpsCount = 0;
private float fps = 0;
//running total of elapsed time up to 10 seconds
private long SummedElapsedSec = 0;
//keep track of the previous ground signature
```

```
        private int PrevAvgGndSig = 0;
        //keep track of the previous horizon line position
        private int PrevHorLine = 0;
        private final FPSDisplayRunnable fpsDisplayRunnable = new FPSDisplayRunnable();

        /** Date formatter, used for time-stamping messages in the message area */
        private final SimpleDateFormat dateFormatter = new SimpleDateFormat("HH:mm:ss,SSS: ");

        // text area for messages
        private final JTextArea messageTextArea = new JTextArea(16, 50);

       //used for the Mask width in the Weigted Median Filter
       private int MaskNumber = 805;
       //Masked integer matrix
       private int[][] Mask;
       //width and height of the picture matrix
       private int width,height;
       // A matrix for the pixel values, one for the selected labels, one which indicates
       // whether a pixel already has a label.
       private int[][] pixels;
       private int[][] labels;
       //Weight Median Filtered Image
       private int[][] WeightedImage;
       //In order to recover grayscale image.
       private byte[] grayImage;
       // The position, i.e. number of estimated algorithm steps we've already done.
       private long position;
       // The number of regions on the (finished) task.
       private int numberOfRegions;
       // Counters for pixels in each region
       private Map<Integer,Integer> count;



        private PrototypingPlayground()
          {
          super(APPLICATION_NAME, ICE_RELAY_PROPERTIES_FILE,
ICE_DIRECT_CONNECT_PROPERTIES_FILE);
          setGUIClientHelperEventHandler(
              new GUIClientHelperEventHandlerAdapter()
              {
              public void executeAfterRelayLogin()
                {
                appendMessage("Logged in to relay.");
                }

              public void executeAfterRelayLogout()
                {
                appendMessage("Logged out from relay.");
                }

              public void executeBeforeDisconnectingFromQwerk()
                {
                MoveForward = false;
                getQwerkController().getMotorService().stopMotors(0, 1); //stop the motors while getting the
state
```

```
                appendMessage("Disconnecting from qwerk...");
                super.executeBeforeDisconnectingFromQwerk();
                }

              public void executeAfterEstablishingConnectionToQwerk(final String qwerkUserId)
                {
                appendMessage("Connected to qwerk " + qwerkUserId);
                isVideoStreamPaused = true;

                    //Initialize legs
                    boolean[] servoMask = { true, true, true, true, true, true, true, true, true, true, true, true,
                                                                                                                true,
true, true, true };
                    int[] positions = { 70, 70, 120, 70, 70, 70, 70, 120, 90, 100, 120, 90, 100, 90, 100, 120};
                    int[] velocities = { 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20};
                    getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
                                                                        positions, velocities);
                }

              public void executeAfterDisconnectingFromQwerk(final String qwerkUserId)
                {
                appendMessage("Disconnected from qwerk " + qwerkUserId);
                fps = 0;
                updateFramesPerSecondLabel();
                }

              public void toggleGUIElementState(final boolean isConnectedToQwerk)
                {
                for (int i = 0; i < NUM_BUTTONS_AND_TEXT_FIELDS; i++)
                  {
                  buttons[i].setEnabled(isConnectedToQwerk);
                  textFields[i].setEnabled(isConnectedToQwerk);
                  }
                messageTextArea.setEnabled(isConnectedToQwerk);
                pauseResumeVideoButton.setEnabled(isConnectedToQwerk);
                pauseResumeVideoButton.setText("Start Video");
                }
              });

        // CONFIGURE GUI ELEMENTS
==========================================================================
============

        // create and configure the buttons and text fields
        for (int i = 0; i < NUM_BUTTONS_AND_TEXT_FIELDS; i++)
          {
          final JButton button = new JButton();
          button.setFont(GUIClientConstants.FONT_SMALL);
          button.setEnabled(false);
          button.setOpaque(false);// required for Macintosh

          buttons[i] = button;

          textFields[i] = new JTextField(TEXT_FIELD_COLUMNS);
          final Dimension fieldSize = new Dimension(textFields[i].getWidth(), textFields[i].getHeight());
          textFields[i].setMaximumSize(fieldSize);
```

```
  textFields[i].setEnabled(false);
  }

// set up the message text area
messageTextArea.setFont(new Font("Monospaced", 0, 10));
messageTextArea.setLineWrap(true);
messageTextArea.setWrapStyleWord(true);
messageTextArea.setEditable(false);
messageTextArea.setEnabled(false);
final JScrollPane messageTextAreaScrollPane = new JScrollPane(messageTextArea,
                                        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                                        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

// add action listeners to the buttons
buttons[0].setText("Get State");
buttons[0].addActionListener(new GetQwerkStateAction());

buttons[1].setText("Begin Process");
buttons[1].addActionListener(new MoveMotorAction(1));

buttons[2].setText("Reverse Motors");
buttons[2].addActionListener(new MoveMotorAction(2));

buttons[3].setText("Set Dig Out X On");
buttons[3].addActionListener(new SingleDigitalOutAction(3, true));

buttons[4].setText("Set Dig Out X Off");
buttons[4].addActionListener(new SingleDigitalOutAction(4, false));

buttons[5].setText("Speed Test");
buttons[5].addActionListener(new SpeedTestAction());

buttons[6].setText("Move Servo 0");
buttons[6].addActionListener(new MoveServoAction(6, 0));

buttons[7].setText("Move Servo 1");
buttons[7].addActionListener(new MoveServoAction(7, 1));

buttons[8].setText("Move Servo 2");
buttons[8].addActionListener(new MoveServoAction(8, 2));

buttons[9].setText("Move Servo 3");
buttons[9].addActionListener(new MoveServoAction(9, 3));

buttons[10].setText("Move Servo 4");
buttons[10].addActionListener(new MoveServoAction(10, 4));

buttons[11].setText("Move Servo 5");
buttons[11].addActionListener(new MoveServoAction(11, 5));

buttons[12].setText("Move Servo 6");
buttons[12].addActionListener(new MoveServoAction(12, 6));

buttons[13].setText("Move Servo 7");
buttons[13].addActionListener(new MoveServoAction(13, 7));
```

```
        buttons[14].setText("Move Servo 8");
        buttons[14].addActionListener(new MoveServoAction(14, 8));

        buttons[15].setText("Move Servo 9");
        buttons[15].addActionListener(new MoveServoAction(15, 9));

        buttons[16].setText("Move Servo 10");
        buttons[16].addActionListener(new MoveServoAction(16, 10));

        buttons[17].setText("Move Servo 11");
        buttons[17].addActionListener(new MoveServoAction(17, 11));

        buttons[18].setText("Move Servo 12");
        buttons[18].addActionListener(new MoveServoAction(18, 12));

        buttons[19].setText("Move Servo 13");
        buttons[19].addActionListener(new MoveServoAction(19, 13));

        buttons[20].setText("Move Servo 14");
        buttons[20].addActionListener(new MoveServoAction(20, 14));

        buttons[21].setText("Move Servo 15");
        buttons[21].addActionListener(new MoveServoAction(21, 15));

        buttons[22].setText("Turn Left");
        buttons[22].addActionListener(new MoveMotorAction(22));

        buttons[23].setText("Turn Right");
        buttons[23].addActionListener(new MoveMotorAction(23));


        savePictureButton.addActionListener(new SavePictureActionListener(this,
getVideoStreamViewport().getComponent(), ImageFormat.JPEG));
        pauseResumeVideoButton.addActionListener(new PauseResumeVideoAction());
        clearMessageAreaButton.addActionListener(
            new ActionListener()
            {
            public void actionPerformed(final ActionEvent actionEvent)
              {
               messageTextArea.setText("");
              }
            });

        // compute and display frames per second and implement rpg calculation
        getVideoStreamPlayer().addVideoStreamEventListener(
            new VideoStreamEventListener()
            {


         public void handleFrame(final byte[] frameData)
               {



              fpsCount++;
```

```
            if (fpsCount == FPS_FRAME_COUNT)
              {
              final long elapsedMilliseconds = System.currentTimeMillis() - fpsTimestamp;

                  SummedElapsedSec += elapsedMilliseconds; //used to count approx. 5 seconds of
elapsed time.
                  LOG.info("Time =" +SummedElapsedSec);
                  fps = FPS_FRAME_COUNT_TIMES_1000 / (float)elapsedMilliseconds;

              updateFramesPerSecondLabel();

              fpsTimestamp = System.currentTimeMillis();
              fpsCount = 0;
              }//End If statement


                  //create image from input data
                  final Image picture = Toolkit.getDefaultToolkit().createImage(frameData);
                  width = 320;
                  height = 240;
                  WeightedImage = new int[width][height];
                  labels = new int[width][height];
                  pixels = new int[width][height];
              final int[] pixels2 = new int[width * height];
                  Mask = new int [MaskNumber][MaskNumber];
                  //change width/2 and height/2 to 1 each for more area
              final PixelGrabber pg = new PixelGrabber(picture,
                                                       1,
                                                       1,
                                                       width,
                                                       height,
                                                       pixels2,
                                                       0,
                                                       width);
              try
                {
                pg.grabPixels();

                    }
              catch (InterruptedException e)
                {
                LOG.error("InterruptedException while grabbing pixels", e);
                }


             /**
                  * Turn pixels2 into a matrix rather than an "array"
                  * and change the byte info into 0 to 255 data
                  **/
                  grayImage = new byte[width * height];
                  int A = 0;
                  for (int i = 0; i < height; i++){
                            for (int j = 0; j < width; j++){
                                      pixels[j][i] = (int)pixels2[A] & 0x0ff;//change data from byte to 0 to
255
```

```
                                labels[j][i] = -1;
                                //Used for viewing the grayscale picture
                                grayImage[A] = (byte) pixels[j][i];//place gray pixel data in previous
array
                                A++;
                        }
                    }

                //create buffered image for displaying
                BufferedImage image = toImage(grayImage, width, height);

                /**************Begin Vision Based Terrain**************************/
                /**
                 * This method performs a vision based terrain characterization.  It processes
                 * information using the following steps:
                 * 1 - Calculate the average pixel signature and standard deviation of the ground plane
                 *     using only an initial ground region of pixels.
                 * 2 - Calculate the horizontal line creating the separation between ground and
                 *     horizon by doing the following:
                 *   3 - Calculate point at which percentage of backdrop pixels becomes larger than
                 *           ground pixels for a given area.
                 * 4 - Identify target objects on the ground using the region growing algorithm.
                 *   5 - Calculate small versus large objects.
                 **/
                  int AvgGndSig = 0;
                  double GndStanDev = 0;
                  int SumOfPixels = 0;
                  int SumOfPixels2 = 0;
                  double Calc = 0;
                  //Figure out the average ground signature and ground standard deviation
                  for(int h=height-1;h>=height-40;h--)
                          for(int w=width-1;w>=0;w--){
                              SumOfPixels += pixels[w][h];
                              AvgGndSig = SumOfPixels / ((width)*((height-1) - (height-40)));
                              SumOfPixels2 += (pixels[w][h])*(pixels[w][h]);
                              Calc = Math.abs((SumOfPixels2/((width)*((height-1) - (height-40)))) -
(AvgGndSig*AvgGndSig));
                              GndStanDev = Math.sqrt(Calc);
                          }//end for loop

                  //LOG.info("AvgGndSig = " + AvgGndSig);
                 // LOG.info("GndStanDev = " + GndStanDev);

                  //Calculate when there are more pixels representing the horizon than there are
representing the ground.
                  int HorLine = 0; //the horizon line extraction
                  for(int h=height-1;h>=0;h--){
                          int NumOfHoz = 0;
                          for(int w=width-1;w>=0;w--){
                             if (pixels[w][h] < (AvgGndSig-GndStanDev) && pixels[w][h] >
(AvgGndSig*0.20))
                                  NumOfHoz++;
                             if (NumOfHoz >= (width*0.60)){
                                  //LOG.info("Place = " + h);
                                  for (int h2=h;h2>=0;h2--)
                                      for (int w2=width-1;w2>=0;w2--)
```

```
                                        pixels[w2][h2] = 100;
                                HorLine = h;
                                w = -1;
                                h = -1;
                            }//end if statement
                        }//end second for loop
                    }//end first for loop

                    //Create new byte array for displaying newly create image
                    byte[] HorizonImage = new byte[width*height];
                    int position5=0;
                    for(int h=0;h<height;h++)
                            for(int w=0;w<width;w++)
                            HorizonImage[position5++] = (byte)pixels[w][h];

                    BufferedImage RegionImage2 = toImage(HorizonImage, width, height);
                   //Identify target regions by creating averages and standard deviations of the specific
region and
                   //comparing those values to the values for the ground using a given threshold.
                    int AvgRegSig = 0;
                    double RegStanDev = 0;
                    int SumOfRegPixels = 0;
                    int SumOfRegPixels2 = 0;
                    double RegCalc = 0;
                    double ratio = 0; //ratio of ground standard deviation to region standard deviation
                    double minratio = 0; //miniumum ratio of ground standard deviation to region standard
deviation
                    int keeptrack = 0; //keeps track of the number of iterations up to 3
                    int AvgPixThresh = 150; //threshold value for average signature differences
                    int AvgRatioThresh =  5;//threshold value for minimum ratio
                    for(int h=height-1;h>HorLine;h--){
                            for(int w=width-1;w>=0;w--){
                                if (pixels[w][h] <= (AvgGndSig *0.80))
                                        pixels[w][h] = 0;
                                else
                                        pixels[w][h] = AvgGndSig;
                            }//end second for loop
                    }//end first for loop

                    //Create new byte array for displaying newly create image
                    byte[] VisionBasedImage = new byte[width*height];
                    int position=0;
                    for(int h=0;h<height;h++)
                            for(int w=0;w<width;w++)
                            VisionBasedImage[position++] = (byte)pixels[w][h];

                    BufferedImage VisionImage = toImage(VisionBasedImage, width, height);

                  /********Begin Movement Criteria and Region Growing**********************/

                    /**
                     * This method checks to see if there has been any movement within approx the last ten
seconds.  The priority
                     * checking is as follows:
                     * 1 - See if the Average Ground Pixel Signature has changed by > +/-2 pixel values.
                     * 2 - If not, then check to see if horizon line (place) has changed positions.
```

```
                    Note: if the horizon line is zero (i.e. the ground is the only thing in view), then
skip this step.
                * 3 - If not, then check to see if there are pixels in a user-defined region size that are
identical
                    and are not equal to the average ground pixel signature and those pixels have
not changed by +/-5
                    (or some other user-defined) pixel values, then there has not been any
movement in the last
                    ten seconds.
                **/
                 int counter = 0;
                 if (SummedElapsedSec >= 2000  && SummedElapsedSec <= 3000)
                        PrevAvgGndSig = AvgGndSig;
                        PrevHorLine = HorLine;
                 if (SummedElapsedSec >= 5000){
                        SummedElapsedSec = 0; //reinitialize elapsed time
                        if (Math.abs(PrevAvgGndSig - AvgGndSig) <= 2){
                                LOG.info("First criteria true; proceed to next.");
                                if (Math.abs(PrevHorLine - HorLine) <= 2){
                                        LOG.info("Second criteria true; procceed to next");
                                        position = 0;
                                        count = new TreeMap<Integer, Integer>();
                                        numberOfRegions = 0;
                                        Stack<Point> mustDo = new Stack<Point>();

                                        for(int h=height-1;h>HorLine;h--)
                                                for(int w=height-1;w>HorLine;w--){

                                                        // Is this pixel unlabeled?
                                                        if (labels[w][h] < 0){
                                                        position++;
                                                                numberOfRegions++;
                                                                mustDo.add(new Point(w,h));
                                                                labels[w][h] = numberOfRegions;
// label it as one on a new region

                                                                count.put(numberOfRegions,1);
                                                                counter=0; //reinitialize counter for
new label

                                                        }//end if statement
                                                        // Check all the pixels on the stack. There
may be more than one!

                                                        while(mustDo.size() > 0){
                                                                Point thisPoint = mustDo.get(0);
mustDo.remove(0);

                                                                // Check 8-neighborhood
                                                                for(int th=-1;th<=1;th++)
                                                                for(int tw=-1;tw<=1;tw++){
                                                                                int rx =
thisPoint.x+tw;
                                                                        int ry = thisPoint.y+th;
                                                                                // Skip pixels
outside of the image.

                                                                        if ((rx < 0) || (ry < 0) ||
(ry>=height) || (rx>=width)) continue;

                                                                        if (labels[rx][ry] < 0)
```

```
                                                                    if
(((pixels[thisPoint.x][thisPoint.y])-5<=pixels[rx][ry]) &&
        (pixels[rx][ry]<=(pixels[thisPoint.x][thisPoint.y])+5)
                                                                                &&
pixels[rx][ry] != AvgGndSig){
        mustDo.add(new Point(rx,ry));
        labels[rx][ry] = numberOfRegions;
        count.put(numberOfRegions, count.get(numberOfRegions)+1);
                                                                                //if
(pixels[rx][ry] != AvgGndSig){
        counter++;
                                                                                //}//end
if statement
                                                                    }//end data comparison for
region growing
                                                        } // ended neighbors checking
                                                } // ended stack scan

                                                        if (counter >= 8){
                                                                //LOG.info("counter =" +counter);
                                                                LOG.info("Third criteria true; you
are stuck");
                                                                //MoveForward = false;
                                                                BackandForth = true;
                                                                Stand = true;
                                                                counter = 0;
                                                        }//end if statement

                                                } // ended image scan
                                                position = width*height;
                                        }//end if statement checking criteria three
                                }// end if statement checking criteria two
                        } //end if statement checking criteria one

                       /**
                        * This part of the code gives the output image.
                        **/
                        // Create a new image based on the labels array.
                        byte[] imageDataSingleArray = new byte[width*height];
                        int count=0;
                        for(int h=0;h<height;h++)
                                for(int w=0;w<width;w++)
                                imageDataSingleArray[count++] = (byte)labels[w][h];

                     /********End Movement Criteria and Region Growing Algorithm**********/

                      /****************End Vision Based Terrrain Characterization***********/

                        // Create a Data Buffer from the values on the single image array.
                        DataBuffer dbuffer = new DataBufferByte(imageDataSingleArray, width*height);
```

```
                    WritableRaster raster = Raster.createBandedRaster(dbuffer, width, height, 1, new
int[]{0},
                                                                        new int[]{0}, new
Point(0,0));
                    ColorSpace cs = ColorSpace.getInstance(ColorSpace.CS_GRAY);
                    ColorModel cm = new ComponentColorModel(cs, false, false,
                                                                Transparency.OPAQUE,
DataBuffer.TYPE_BYTE);
                     BufferedImage RegionImage = new BufferedImage(cm, raster, false, null);
                     //BufferedImage RegionImage3 = toImage(imageDataSingleArray2, width, height);
                     //display(RegionImage);
                    try {
                            ImageIO.write(image, "jpeg", new File("grayimage.jpeg"));
                            ImageIO.write(VisionImage, "jpeg", new File("VisionImage.jpeg"));
                            ImageIO.write(RegionImage2, "jpeg", new File("RegionImage.jpeg"));
                            //ImageIO.write(WeightedMedianImage, "jpeg", new
File("WeightedMedianImage.jpeg"));
                    }
                    catch (IOException e){
                            LOG.error("InterruptedException while grabbing pixels", e);
                    }

                }//End handleFrame

                /**
                 * This method returns a grayscale BufferedImage.
                 */
                public BufferedImage toImage(byte[] pixels, int w, int h) {

                DataBuffer db = new DataBufferByte(pixels, w*h);
                            WritableRaster raster = Raster.createInterleavedRaster(db, w, h, w, 1, new
int[]{0}, null);
                            ColorSpace cs = ColorSpace.getInstance(ColorSpace.CS_GRAY);
                            ColorModel cm = new ComponentColorModel(cs, false, false,
                                                                Transparency.OPAQUE,
DataBuffer.TYPE_BYTE);
                            return new BufferedImage(cm, raster, false, null);

                }

                /**
                 * This method displays the newly created BufferedImage in a jframe
                 */
                public void display(BufferedImage image) {

                            final JFrame f = new JFrame("");
                            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                            f.getContentPane().add(new JLabel(new ImageIcon(image)));
                            f.pack();
                            SwingUtilities.invokeLater(new Runnable(){
                                    public void run() {
                                    f.setLocationRelativeTo(null);
                                    f.setVisible(true);
                                    }
                            });
                }
```

```
                /**
                 * This method returns the number of regions on the segmetation task. This
                 * number may be partial if the task has not finished yet.
                 */
                public int getNumberOfRegions()
                {
                        return numberOfRegions;
                }

                /**
                 * This method returns the pixel count for a particular region or -1 if the
                 * region index is outside of the range.
                 */
                public int getPixelCount(int region)
                  {
                  Integer c = count.get(region);
                  if (c == null) return -1; else return c;
                  }

                /**
                 * This method returns the estimated size (steps) for this task. We estimate it as
                 * being the size of the image.
                 */
                public long getSize()
                  {
                  return width*height;
                  }

                /**
                 * This method returns the position on the image processing task.
                 */
                public long getPosition()
                  {
                  return position;
                  }

                /**
                 * This method returns true if the image processing task has finished.
                 */
                public boolean isFinished()
                  {
                  return (position == width*height);
                  }

        });//End getvideo function
        updateFramesPerSecondLabel();

        // LAYOUT GUI ELEMENTS
==========================================================================
==============

        // create a JPanel to hold the buttons and text fields
        final JPanel buttonsAndTextFieldsPanel = new JPanel(new SpringLayout());
        for (int i = 0; i < NUM_BUTTONS_AND_TEXT_FIELDS; i++)
          {
```

```
  buttonsAndTextFieldsPanel.add(buttons[i]);
  buttonsAndTextFieldsPanel.add(Box.createRigidArea(SPACER_DIMENSIONS));
  buttonsAndTextFieldsPanel.add(textFields[i]);
  }
SpringLayoutUtilities.makeCompactGrid(buttonsAndTextFieldsPanel,
                        NUM_BUTTONS_AND_TEXT_FIELDS, 3, // rows, cols
                        0, 0, // initX, initY
                        0, 5);// xPad, yPad

// create a JPanel to hold the buttons, text fields, and color panel
final JPanel videoButtonsPanel = new JPanel(new SpringLayout());
videoButtonsPanel.add(savePictureButton);
videoButtonsPanel.add(Box.createRigidArea(new Dimension(10, 10)));
videoButtonsPanel.add(pauseResumeVideoButton);
videoButtonsPanel.add(Box.createGlue());
videoButtonsPanel.add(framesPerSecondLabel);
SpringLayoutUtilities.makeCompactGrid(videoButtonsPanel,
                        1, 5, // rows, cols
                        0, 0, // initX, initY
                        0, 0);// xPad, yPad

// create a JPanel to hold the buttons, text fields, and color panel
final JPanel videoPanel = new JPanel(new SpringLayout());
videoPanel.add(getVideoStreamViewportComponent());
videoPanel.add(Box.createRigidArea(SPACER_DIMENSIONS));
videoPanel.add(videoButtonsPanel);
SpringLayoutUtilities.makeCompactGrid(videoPanel,
                        3, 1, // rows, cols
                        0, 0, // initX, initY
                        0, 0);// xPad, yPad

// create a JPanel to hold video port and the buttons
final JPanel controlsPanel = new JPanel(new SpringLayout());
controlsPanel.add(getConnectDisconnectButton());
controlsPanel.add(getConnectionStatePanel());
controlsPanel.add(videoPanel);
controlsPanel.add(buttonsAndTextFieldsPanel);
SpringLayoutUtilities.makeCompactGrid(controlsPanel,
                        2, 2, // rows, cols
                        0, 0, // initX, initY
                        10, 10);// xPad, yPad

// create a JPanel to hold video port and the buttons
final JPanel messageAreaButtonsPanel = new JPanel(new SpringLayout());
messageAreaButtonsPanel.add(Box.createGlue());
messageAreaButtonsPanel.add(clearMessageAreaButton);
SpringLayoutUtilities.makeCompactGrid(messageAreaButtonsPanel,
                        1, 2, // rows, cols
                        0, 0, // initX, initY
                        0, 0);// xPad, yPad

// create a JPanel to hold video port and the buttons
final JPanel messageArea = new JPanel(new SpringLayout());
messageArea.add(messageTextAreaScrollPane);
messageArea.add(messageAreaButtonsPanel);
SpringLayoutUtilities.makeCompactGrid(messageArea,
```

```
                                    2, 1, // rows, cols
                                    0, 0, // initX, initY
                                    0, 5);// xPad, yPad

        // Layout the main content pane using SpringLayout
        getMainContentPane().setLayout(new SpringLayout());
        getMainContentPane().add(controlsPanel);
        getMainContentPane().add(messageArea);
        SpringLayoutUtilities.makeCompactGrid(getMainContentPane(),
                                    2, 1, // rows, cols
                                    10, 10, // initX, initY
                                    10, 10);// xPad, yPad

        // ADDITIONAL GUI ELEMENT CONFIGURATION
================================================================================

        // pack the window so the GUI elements are properly sized
        pack();

        // limit the text area's size (must do this AFTER the call to pack())
        final Dimension messageTextAreaScrollPaneDimensions = new
Dimension(messageTextArea.getWidth(), messageTextArea.getHeight());
        messageTextAreaScrollPane.setPreferredSize(messageTextAreaScrollPaneDimensions);
        messageTextAreaScrollPane.setMinimumSize(messageTextAreaScrollPaneDimensions);
        messageTextAreaScrollPane.setMaximumSize(new Dimension(10000,
messageTextArea.getHeight()));

        pack();

        setLocationRelativeTo(null);// center the window on the screen

        setVisible(true);
        }

      private void updateFramesPerSecondLabel()
        {
        SwingUtilities.invokeLater(fpsDisplayRunnable);
        }

      /** Appends the given <code>message</code> to the message text area */
      private void appendMessage(final String message)
        {
        SwingUtilities.invokeLater(
            new Runnable()
            {
            public void run()
              {
              messageTextArea.append(dateFormatter.format(new Date()) + message +
LINE_SEPARATOR);
              messageTextArea.setCaretPosition(messageTextArea.getDocument().getLength());
              }
            });
        }

      /** Retrieves the value from the specified text field as an <code>int</code>. */
      @SuppressWarnings({"UnusedCatchParameter"})
```

```
        private int getTextFieldValueAsInt(final int textFieldIndex)
          {
          final int i;
          final String str = getTextFieldValueAsString(textFieldIndex);
          try
            {
            i = Integer.parseInt(str);
            }
          catch (NumberFormatException e)
            {
            appendMessage("NumberFormatException while trying to convert [" + str + "] into an int.
Returning 0 instead.");
            return 0;
            }
          return i;
          }

        /** Retrieves the value from the specified text field as a {@link String}. */
        @SuppressWarnings({"UnusedCatchParameter"})
        private String getTextFieldValueAsString(final int textFieldIndex)
          {
          if (SwingUtilities.isEventDispatchThread())
            {
            final String textFieldValue;
            try
              {
              final String text1 = textFields[textFieldIndex].getText();
              textFieldValue = (text1 != null) ? text1.trim() : null;
              }
            catch (Exception e)
              {
              appendMessage("Exception while getting the value from text field " + textFieldIndex + ".
Returning null instead.");
              return null;
              }
            return textFieldValue;
            }
          else
            {
            final String[] textFieldValue = new String[1];
            try
              {
              SwingUtilities.invokeAndWait(
                  new Runnable()
                  {
                  public void run()
                    {
                    textFieldValue[0] = textFields[textFieldIndex].getText();
                    }
                  });
              }
            catch (Exception e)
              {
              LOG.error("Exception while getting the value from text field " + textFieldIndex, e);
              appendMessage("Exception while getting the value from text field " + textFieldIndex + ".
Returning null instead.");
```

```
            return null;
            }

          return textFieldValue[0];
          }
        }

      private class PauseResumeVideoAction extends AbstractTimeConsumingAction
        {
        private PauseResumeVideoAction()
          {
          super(PrototypingPlayground.this);
          }

        protected void executeGUIActionBefore()
          {
          PrototypingPlayground.this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
          if (isVideoStreamPaused)
            {
            pauseResumeVideoButton.setText("Pause Video");
            }
          else
            {
            pauseResumeVideoButton.setText("Resume Video");
            }
          }

        protected Object executeTimeConsumingAction()
          {
          if (isVideoStreamPaused)
            {
            getVideoStreamPlayer().resumeVideoStream();
            }
          else
            {
            getVideoStreamPlayer().pauseVideoStream();
            }
          return null;
          }

        protected void executeGUIActionAfter(final Object resultOfTimeConsumingAction)
          {
          isVideoStreamPaused = !isVideoStreamPaused;

PrototypingPlayground.this.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
          fpsTimestamp = System.currentTimeMillis();
          fpsCount = 0;
          fps = 0;
          updateFramesPerSecondLabel();
          }
        }

      private final class GetQwerkStateAction extends AbstractTimeConsumingAction
        {
        private GetQwerkStateAction()
          {
```

```
        super(PrototypingPlayground.this);
        }

      protected Object executeTimeConsumingAction()
        {
       return getQwerkController().getQwerkState();
        }

      protected void executeGUIActionAfter(final Object resultOfTimeConsumingAction)
        {
        // read the values from QwerkState and format it nicely for display in the message area
        final QwerkState state = (QwerkState)resultOfTimeConsumingAction;
        if (state != null)
          {
          final StringBuffer s = new StringBuffer("Qwerk State:" + LINE_SEPARATOR);
          if (state.analogIn != null)
            {
            s.append("   Analog Inputs:
").append(ArrayUtils.arrayToString(state.analogIn.analogInValues)).append(LINE_SEPARATOR);
            }
          if (state.button != null)
            {
            s.append("   Button State:
").append(state.button.buttonStates[0]).append(LINE_SEPARATOR);
            }
          if (state.digitalIn != null)
            {
            s.append("   Digital Inputs:
").append(ArrayUtils.arrayToString(state.digitalIn.digitalInStates)).append(LINE_SEPARATOR);
            }
          if (state.motor != null)
            {
            s.append("   Motor Currents:
").append(ArrayUtils.arrayToString(state.motor.motorCurrents)).append(LINE_SEPARATOR);
            s.append("   Motor Positions:
").append(ArrayUtils.arrayToString(state.motor.motorPositions)).append(LINE_SEPARATOR);
            s.append("   Motor Velocities:
").append(ArrayUtils.arrayToString(state.motor.motorVelocities)).append(LINE_SEPARATOR);
            s.append("   Motor Done:
").append(ArrayUtils.arrayToString(state.motor.motorDone)).append(LINE_SEPARATOR);
            }
          if (state.servo != null)
            {
            s.append("   Servo Positions:
").append(ArrayUtils.arrayToString(state.servo.servoPositions)).append(LINE_SEPARATOR);
            }
          if (state.battery != null)
            {
            s.append("   Battery Voltage:
").append(state.battery.batteryVoltage).append(LINE_SEPARATOR);
            }

          // display the state in the message area
          appendMessage(s.toString());
          }
        else
```

```
            {
            appendMessage("QwerkState is null!");
            }
          }
        }

      private final class MoveMotorAction extends AbstractTimeConsumingAction
        {
        private final int buttonIndex;

        private MoveMotorAction(final int buttonIndex)
          {
          super(PrototypingPlayground.this);
          this.buttonIndex = buttonIndex;
          }

        protected Object executeTimeConsumingAction()
          {
          // get the motor index
          final int motorIndex = getTextFieldValueAsInt(buttonIndex);

         if (buttonIndex == 1) //move forward
         {


                  while (MoveForward == true){
                        while(BackandForth == false){
                        getQwerkController().getMotorService().setMotorVelocitiesByIds(0, 5000, 1, -
5000);
                        }//end while for forward movement
                        if(BackandForth == true){
                          for(int a = 0; a<3; a++){
                                // stop the motor
                                getQwerkController().getMotorService().stopMotors(0, 1);
                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }
                                //for (int i = 0; i<2; i++){

                                /*********Begin Walking Gait**********************/
                                //Move legs to position for standing
                                boolean[] servoMask = { true, true, true, true, true, true, true, true, true,
true, true, true,

          true, true, true, true };
                                int[] positions = { 70, 140, 128, 70, 0, 70, 0, 128, 90, 200, 128, 90, 200,
90, 40, 128 };
                                int[] velocities = { 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40};

          getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
```

```
        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }
                                        //Stand
                                        positions = new int[] { 0, 140, 128, 0, 0, 0, 0, 128, 255, 200, 128, 255,
200, 255, 40, 128 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }

                                        /*****Begin Left Side Movement******************/
                                        //Pull front left leg up and move forward
                                        positions = new int[] { 75, 200, 120, 0, 0, 0, 0, 128, 255, 200, 128, 255,
200, 255, 40, 128 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }

                                        //Put front left leg down in forward position
                                        positions = new int[] { 0, 200, 120, 0, 0, 0, 0, 128, 255, 200, 128, 255,
200, 255, 40, 128 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
```

```
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }

                                //Pull middle left leg up and move forward
                                positions = new int[] { 0, 200, 10, 70, 100, 0, 0, 120, 255, 200, 120,
255, 200, 255, 30, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }

                                //Put middle left leg down in forward postion
                                positions = new int[] { 0, 200, 120, 0, 100, 0, 0, 120, 255, 200, 120,
255, 200, 255, 30, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }

                                //Pull back left leg up and move forward
                                positions = new int[] { 0, 200, 120, 0, 100, 70, 100, 120, 255, 200, 120,
255, 200, 255, 30, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }

                                //Put back left leg down in forward postion
```

```
                                positions = new int[] { 0, 200, 120, 0, 100, 0, 100, 120, 255, 200, 120,
255, 200, 255, 30, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }
                                /***************End Left Side Movement**************/

                                /**************Begin Right Side Movement************/
                                //Pull front right leg up and move forward
                                positions = new int[] { 0, 200, 120, 0, 0, 0, 0, 128, 255, 200, 128, 255,
200, 90, 0, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }


                                //Put front right leg down in forward position
                                positions = new int[] { 0, 200, 120, 0, 0, 0, 0, 128, 255, 200, 128, 255,
200, 255, 0, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);

                                try
                                {
                                Thread.sleep(2000);
                                }
                                catch (InterruptedException e1)
                                {
                                LOG.error("InterruptedException while sleeping", e1);
                                }

                                //Pull middle right leg up and move forward
                                positions = new int[] { 0, 200, 120, 0, 100, 0, 100, 120, 255, 200, 120,
90, 100, 255, 0, 120 };
```

```
        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }

                                        //Put middle right leg down in forward position
                                        positions = new int[] { 0, 200, 120, 0, 100, 0, 100, 120, 255, 200, 120,
255, 100, 255, 0, 120  };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
        positions, velocities);

                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }

                                        //Pull back right leg up and move forward
                                        positions = new int[] { 0, 200, 120, 0, 100, 0, 100, 120, 90, 100, 120,
255, 100, 255, 0, 120 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
                                        }
                                        catch (InterruptedException e1)
                                        {
                                        LOG.error("InterruptedException while sleeping", e1);
                                        }

                                        //Put back right leg down in forward position
                                        positions = new int[] { 0, 200, 120, 0, 100, 0, 100, 120, 255, 100, 120,
255, 100, 255, 0, 120  };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,
        positions, velocities);
                                        try
                                        {
                                        Thread.sleep(2000);
```

```
                                    }
                                    catch (InterruptedException e1)
                                    {
                                    LOG.error("InterruptedException while sleeping", e1);
                                    }
                                    /**************End Back Right Leg Movement**************/

                                    /****************Push Forward with front Legs**********/
                                    positions = new int[] { 0, 0, 128, 0, 0, 0, 0, 128, 255, 255, 128, 255,
255, 255, 255, 128  };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                    try
                                    {
                                    Thread.sleep(5000);
                                    }
                                    catch (InterruptedException e1)
                                    {
                                    LOG.error("InterruptedException while sleeping", e1);
                                    }
                                    /************End Push Forward with all Legs*****/



                                    //Return legs to original position (right side first)
                                    positions = new int[] { 0, 0, 128, 0, 0, 0, 0, 128, 90, 100, 128, 90, 100,
90, 100, 128 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                    try
                                    {
                                    Thread.sleep(2000);
                                    }
                                    catch (InterruptedException e1)
                                    {
                                    LOG.error("InterruptedException while sleeping", e1);
                                    }

                                    //Return legs to original position (now the left side too)
                                    positions = new int[] { 70, 70, 128, 70, 70, 70, 70, 128, 90, 100, 128,
90, 100, 90, 100, 128 };

        getQwerkController().getServoService().setPositionsWithVelocities(servoMask,

        positions, velocities);
                                    try
                                    {
                                    Thread.sleep(5000);
                                    }
                                    catch (InterruptedException e1)
                                    {
                                    LOG.error("InterruptedException while sleeping", e1);
```

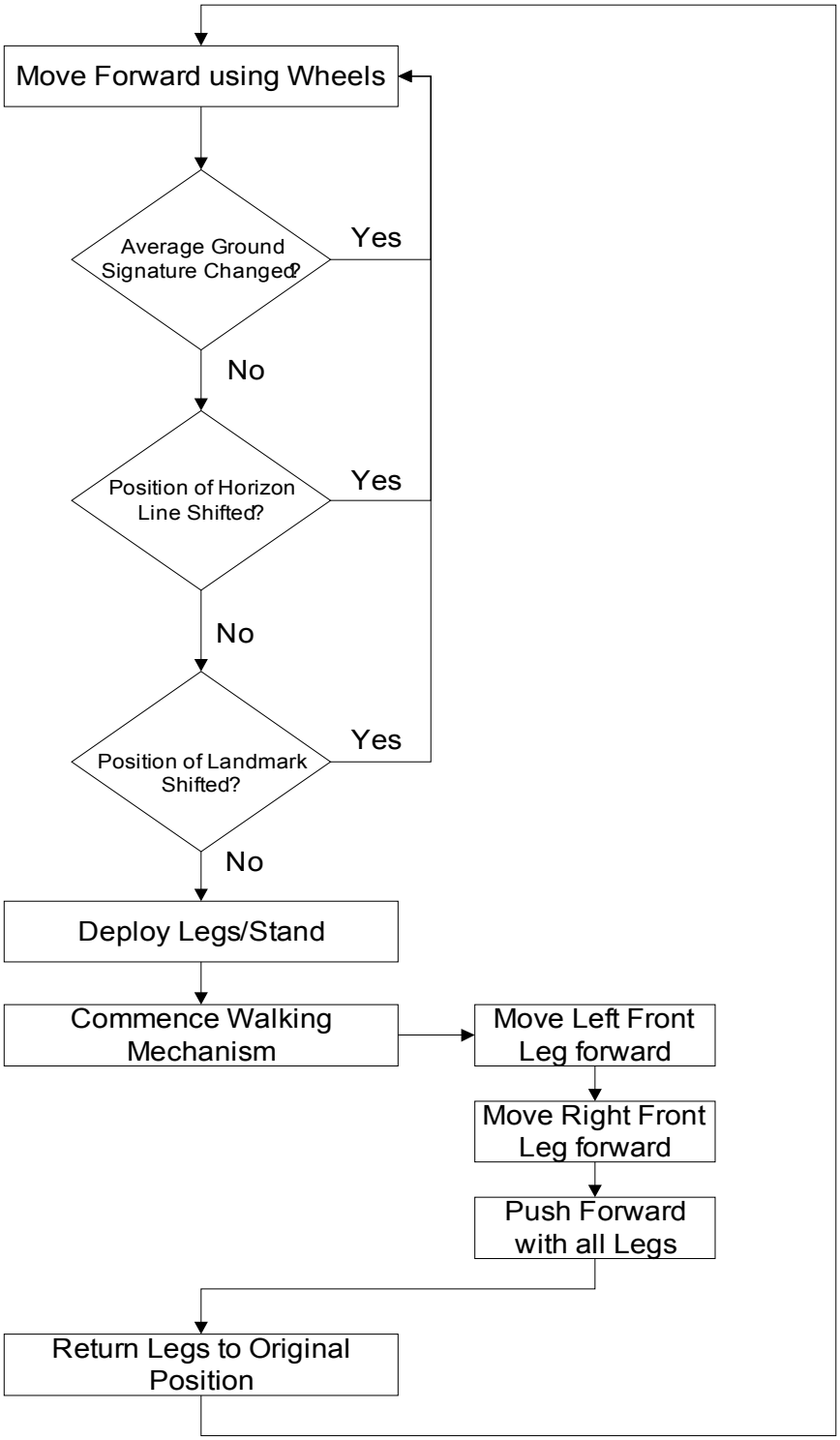
```
                                    }
                                    /****************End Walking Gait*****************/


                                }//end for loop
                                 BackandForth = false;
                                }//end backandforth if statement
                        }//end wheel movement while loop
    }//end buttonIndex == 1
    else if (buttonIndex == 2) //move in reverse
    {
    getQwerkController().getMotorService().setMotorVelocitiesByIds(0, -5000, 1, 5000);
    }
    else if (buttonIndex == 22) //turn left
    {
    getQwerkController().getMotorService().setMotorVelocitiesByIds(0, 5000, 1, 2500);
    }
    else //turn right
    {
    getQwerkController().getMotorService().setMotorVelocitiesByIds(0, -2500, 1, -5000);
    }
      // sleep for 20 seconds
      try
        {
        Thread.sleep(20000);
        }
      catch (InterruptedException e1)
        {
        LOG.error("InterruptedException while sleeping", e1);
        }

      // stop the motor
      getQwerkController().getMotorService().stopMotors(0, 1);

      return null;
      }
    }

private final class MoveServoAction extends AbstractTimeConsumingAction
    {
    private final int buttonIndex;
    private final int servoIndex;

    private MoveServoAction(final int buttonIndex, final int servoIndex)
      {
      super(PrototypingPlayground.this);
      this.buttonIndex = buttonIndex;
      this.servoIndex = servoIndex;
      }

    protected Object executeTimeConsumingAction()
      {
      // get the desired servo position
      final int servoPosition = getTextFieldValueAsInt(buttonIndex);

      // set the servo position
```

```
      getQwerkController().getServoService().setPosition(servoPosition, servoIndex);

     return null;
      }
    }

private final class SingleDigitalOutAction extends AbstractTimeConsumingAction
  {
  private final boolean digitalOutState;
  private final int buttonIndex;

  private SingleDigitalOutAction(final int buttonIndex, final boolean digitalOutState)
    {
    super(PrototypingPlayground.this);
    this.buttonIndex = buttonIndex;
    this.digitalOutState = digitalOutState;
    }

  protected Object executeTimeConsumingAction()
    {
    // get the desired digital out port
    final int digitalOutPort = getTextFieldValueAsInt(buttonIndex);

    appendMessage("Sending [" + digitalOutState + "] to digital out port [" + digitalOutPort + "]...");

    // set the digital out
    getQwerkController().getDigitalIOService().setOutputs(digitalOutState, digitalOutPort);

    appendMessage("Done sending digital out command!");

    return null;
    }
  }

private class SpeedTestAction extends AbstractTimeConsumingAction
  {
  private static final int NUM_CALLS = 50;

  private SpeedTestAction()
    {
    super(PrototypingPlayground.this);
    }

  protected Object executeTimeConsumingAction()
    {
    final long[] millisecondsPerCall = new long[NUM_CALLS];

    final QwerkController qwerkController = getQwerkController();

    for (int i = 0; i < NUM_CALLS; i++)
      {
      final long startTime = System.currentTimeMillis();
      qwerkController.getQwerkState();
      final long endTime = System.currentTimeMillis();
      millisecondsPerCall[i] = endTime - startTime;
      }
```

```
            return millisecondsPerCall;
            }

          protected void executeGUIActionAfter(final Object resultOfTimeConsumingAction)
            {
            final long[] millisecondsPerCall = (long[])resultOfTimeConsumingAction;
            final StringBuffer str = new StringBuffer("Milliseconds per call: " + LINE_SEPARATOR);
            int sum = 0;
            for (int i = 0; i < NUM_CALLS; i++)
              {
              str.append("   ").append(millisecondsPerCall[i]).append(LINE_SEPARATOR);
              sum += millisecondsPerCall[i];
              }
            str.append("   Total:   ").append(sum).append(" ms");
            str.append("   Average: ").append(sum / (double)NUM_CALLS).append(" ms");
            appendMessage(str.toString());
            }
          }

        private final class FPSDisplayRunnable implements Runnable
          {
          private final NumberFormat decimalFormatter = new DecimalFormat("#,##0.0");

          public void run()
            {
            PrototypingPlayground.this.framesPerSecondLabel.setText(decimalFormatter.format(fps) + "
fps");
            }
          }
        }//End PrototypingPlayground
```

APPENDIX B

HIGH-LEVEL PROGRAM FLOW CHART



APPENDIX C

THE CHARMED LABS QWERK CONTROLLER

Hardware [27]

- 200 MHz ARM9 RISC processor with MMU and hardware floating point unit
- 32 Mbytes SDRAM, 16 Mbytes flash memory
- Xilinx Spartan 3E FPGA for custom I/O peripherals
- Linux 2.6 installed
- WiFi wireless networking support
- WebCam video input support
- 4 Amp switching power supply, 90% efficient, 7 to 30 Volt input range
- 5.1” x 5.8” x 1.3”, 11.8 Ozs

I/O

- 4 closed-loop 2.0 Amp motor controllers (supports both quadrature encoder and back-EMF “sensorless” feedback)
- 16 RC-servo controllers
- 16 programmable digital I/Os
- 8 12-bit analog inputs
- 2 RS-232 ports
- USB 2.0 host ports for connecting standard USB PC peripherals
- 10/100BT Ethernet port
- Built-in audio amp for playing MP3 and WAV files

APPENDIX D

THE LOGITECH QUICKCAM® COMMUNICATE STX™

System Requirements [28]

- Windows® 2000, Windows® XP
  Pentium® 4 1.4 GHz or AMD Athlon™ 1GHz processor (Pentium® 4 2.4 GHz recommended)
  128MB RAM (256 MB recommended)

- Windows Vista™
  Pentium® 4 2.4 GHz (2.8 GHz recommended)
- 512MB RAM (1GB recommended)
- 200 MB hard drive
- CD-ROM drive
- 16-bit color display adapter
- OS compatible sound card and speakers
- 1.1 or 2.0 USB port

Technical Specifications

Hardware

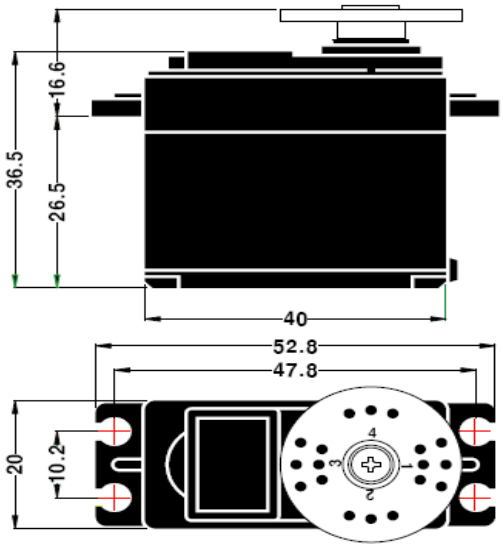
- High quality VGA sensor with RightLight™ Technology
- Video capture: Up to 640 x 480
- Still image capture: Up to 1.3 megapixel with software enhancement
- Built-in microphone with RightSound™ Technology
- Frame rate: Up to 30 frames per second with recommended system
- Adjustable base fits any monitor or notebook
- USB 2.0 certified
- Optics: Fixed Focus

APPENDIX E

HS-322HD STANDARD DELUXE SERVO

1.TECHNICAL VALUE

CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
TEST VOLTAGE	:AT 4.8V	AT 6.0V
OPERATING SPEED	:0.19sec/60? AT NO LOAD	0.15sec/60? AT NO LOAD
STALL TORQUE	:3kg.cm(41.66oz.in)	3.7kg.cm(51.38oz.in)
IDLE CURRENT	:7.4mA AT STOPPED	7.7mA AT STOPPED
RUNNING CURRENT	:160mA/60? AT NO LOAD	180mA/60? AT NO LOAD
STALL CURRENT	:700mA	800mA
DEAD BAND WIDTH	:5usec	5usec
OPERATING TRAVEL	:40? /ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
MOTOR TYPE	:CORED METAL BRUSH	
POTENTIOMETER TYPE	:4 SLIDER/DIRECT DRIVE	
AMPLIFIER TYPE	:ANALOG CONTROLLER & TRANSISTOR DRIVER	
DIMENSIONS	:40x20x36.5mm(1.57x0.78x1.43in)	
WEIGHT	:43g(1.51oz)	
BALL BEARING	:TOP/RESIN BUSHING	
GEAR MATERIAL	:2 HEAVY DUTY RESIN	
HORN GEAR SPLINE	:24 SEGMENTS/?5.76	
SPLINED HORNS	:RESULAR/R-C,R-D,R-I,R-O,R-X,SUPER/R-XA	
CONNECTOR WIRE LENGTH	:300mm(11.81in)	
CONNECTOR WIRE STRAND COUNTER	:40EA	
CONNECTOR WIRE GAUGE	:25AWG	



2.FEATURES

LONG LIFE POTENTIOMETER, TOP RESIN BUSHING, 2 HEAVY DUTY RESIN GEARS

3.APPLICATIONS

AIRCRAFT 20-40 SIZE,STEERING AND THROTTLE SERVO FOR CARS, TRUCK AND BOATS

4.ACCESSORY & OPTION

CASE SET/
- HS322T:1EA
- HS322M:1EA
- HS322L:1EA
- PH/T-2 2x30 NI:4EA

GEAR SET/
- HS322G1:1EA
- HS325G2:1EA
- HS325G3:1EA
- HS322G4:1EA
- HS300RB:1EA

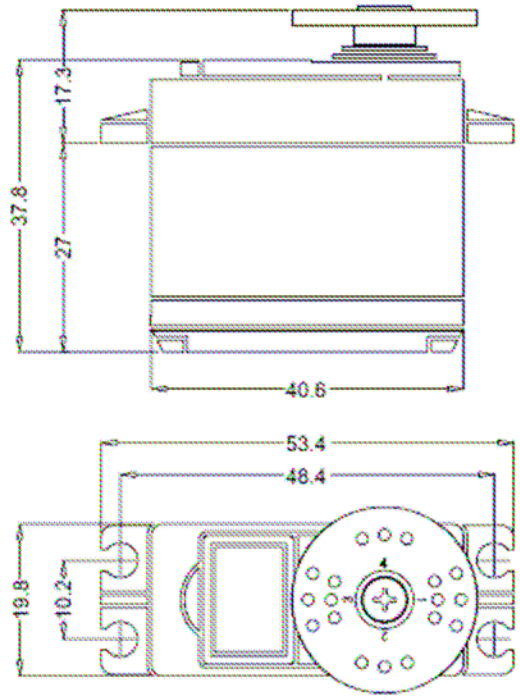
HORN SET/
- R-C:1EA
- R-D:1EA
- R-I:1EA
- R-O:1EA
- R-X:1EA
- R-XA:1EA
- WH/W 2.1x15 NI:4EA
- BST 3x5.5:4EA
- NBR 9x6.5x6:4EA

APPENDIX F

HS-645MG STANDARD DELUXE HIGH TORQUE SERVO

1.TECHNICAL VALUES

CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	:-20 TO +60°C	
TEST VOLTAGE	:AT 4.8V	:AT 6.0V
OPERATING SPEED	:0.24sec/60°AT NO LOAD	:0.2sec/60°AT NO LOAD
STALL TORQUE	:7.7kg.cm(106.93oz.in)	:9.6kg.cm(133.31oz.in)
OPERATING ANGLE	:45°ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
IDLE CURRENT	:8.8mA	:9.1mA
RUNNING CURRENT	:350mA	:450mA
DEAD BAND WIDTH	:8usec	
CONNECTOR WIRE LENGTH	:300mm(11.81in)	
DIMENSIONS	:40.6x19.8x37.8mm(1.59x0.77x1.48in)	
WEIGHT	:55.2g(1.94oz)	



2.FEATURES

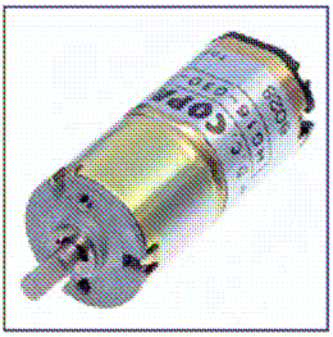
3-POLE FERRITE MOTOR
DUAL BALL BEARING
LONG LIFE POTENTIOMETER
3-METAL GEARS & 1-RESIN METAL GEAR
HYBRID I.C

3.APPLICATIONS

AIRCRAFT TO 1/4 SCALE
30 TO 60 SIZE HELICOPTERS
STEERING AND THROTTLE FOR 1/10TH & 1/8TH ON-ROAD AND OFF-ROAD VEHICLES

APPENDIX G

HG16-060-AA DC MOTORS



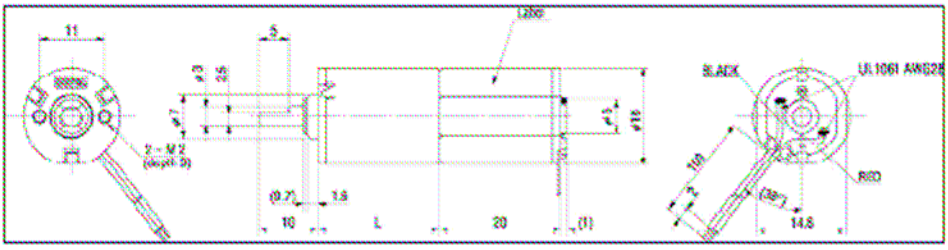
Case	:Metal
Bearing	:Sleeve Bearing
Operating Temp	:0℃~50℃
Rotation	:CW, CCW
Net Weight	:25g

- Resin gears for quiet operation and long life
- Planetary gearbox
- Special configurations available upon request

Performance (Reference)

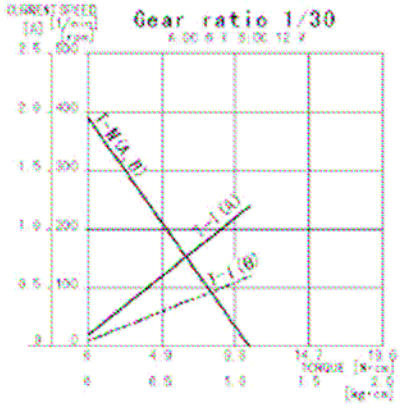
Model	Rated Voltage (DC)	Operating Voltage (DC)	Rated Current (mA)	Stall Current (mA)	Rated Torque oz-in	Rated Torque g-cm	Stall Torque oz-in	Stall Torque g-cm	rated speed (r.p.m.)	No Load Speed (r.p.m.)
HG16-030-AA	6	5.4~6.6	450	1800	3.47	250	11	800	310	390
HG16-030-AB	12	10.8~13.2	250	800						

Outline Dimensions



L= 16mm

NIDEC COPAL GmbH
Charlotton Str. 51, 40210 Duesseldorf, Germany
Phone: 49-211-550-469-0
Fax: 49-211-550-469-29
URL http://www.nidec-copal-de.com/



(Provided by COPAL- producer of gear motors)

APPENDIX H

ROBOT PRICE LIST

Item	Company	Price per Unit	# of Units	Total Price
Qwerk Processor	Charmed Labs	$349.00	1	$349.00
Logitech Communicate STX Camera	Amazon	$40.59	1	$40.59
AirLink AWLL3026 Ultra Slim USB 2.0 Adapter	Amazon	$35.00	1	$35.00
12V 4200mAh Flat NiMH Battery Pack for DC Power	Total Power Solution - All Battery.com	$48.99	1	$48.99
	Total for Current Parts			***$473.58***
Parts from original design				
Servo Controller SSC-32	Lynxmotion	$39.95	1	$39.95
Lite Flite" Foam Tires (pair)	Robot Marketplace	$4.79	2	$9.58
Wheel Hub/Prop Adapter, for 1/8" & 3mm shaft	Robot Marketplace	$4.95	4	$19.80
Motor Bracket Mounts	Robot Marketplace	$8.99	4	$35.96
COPAL 60:1 Gearmotor	Robot Marketplace	$21.99	4	$87.96
Hitec Standard Servo - Ball Bearing	Robot Marketplace	$7.99	16	$127.84
Hitec HS-5645MG High-Torque 2BB MG Servo	Lynxmotion	Package of 6	6	$239.94
Dubro Full Threaded Rod 4-40 12"	Tower Hobbies	$14.39	1	$14.39
Dubro Threaded Coupler 4-40 (2)	Tower Hobbies	$1.49	3	$4.47
Dubro Heavy Duty Ball Links 4-40 (12)	Tower Hobbies	$19.99	1	$19.99
	Total for Robot Parts to Purchase			***$1,013.52***

APPENDIX I

UPPER CRITICAL VALUES OF t DISTRIBUTION WITH φ DEGRESS OF FREEDOM [29]

	Probability of exceeding the critical value					
φ	0.10	0.05	0.025	0.01	0.005	0.001
1.	3.078	6.314	12.706	31.821	63.657	318.313
2.	1.886	2.920	4.303	6.965	9.925	22.327
3.	1.638	2.353	3.182	4.541	5.841	10.215
4.	1.533	2.132	2.776	3.747	4.604	7.173
5.	1.476	2.015	2.571	3.365	4.032	5.893
6.	1.440	1.943	2.447	3.143	3.707	5.208
7.	1.415	1.895	2.365	2.998	3.499	4.782
8.	1.397	1.860	2.306	2.896	3.355	4.499
9.	1.383	1.833	2.262	2.821	3.250	4.296
10.	1.372	1.812	2.228	2.764	3.169	4.143
11.	1.363	1.796	2.201	2.718	3.106	4.024
12.	1.356	1.782	2.179	2.681	3.055	3.929
13.	1.350	1.771	2.160	2.650	3.012	3.852
14.	1.345	1.761	2.145	2.624	2.977	3.787
15.	1.341	1.753	2.131	2.602	2.947	3.733
16.	1.337	1.746	2.120	2.583	2.921	3.686
17.	1.333	1.740	2.110	2.567	2.898	3.646
18.	1.330	1.734	2.101	2.552	2.878	3.610
19.	1.328	1.729	2.093	2.539	2.861	3.579
20.	1.325	1.725	2.086	2.528	2.845	3.552
21.	1.323	1.721	2.080	2.518	2.831	3.527
22.	1.321	1.717	2.074	2.508	2.819	3.505
23.	1.319	1.714	2.069	2.500	2.807	3.485
24.	1.318	1.711	2.064	2.492	2.797	3.467
25.	1.316	1.708	2.060	2.485	2.787	3.450
26.	1.315	1.706	2.056	2.479	2.779	3.435
27.	1.314	1.703	2.052	2.473	2.771	3.421
28.	1.313	1.701	2.048	2.467	2.763	3.408
29.	1.311	1.699	2.045	2.462	2.756	3.396
30.	1.310	1.697	2.042	2.457	2.750	3.385
31.	1.309	1.696	2.040	2.453	2.744	3.375
32.	1.309	1.694	2.037	2.449	2.738	3.365
33.	1.308	1.692	2.035	2.445	2.733	3.356
34.	1.307	1.691	2.032	2.441	2.728	3.348
35.	1.306	1.690	2.030	2.438	2.724	3.340
36.	1.306	1.688	2.028	2.434	2.719	3.333
37.	1.305	1.687	2.026	2.431	2.715	3.326
38.	1.304	1.686	2.024	2.429	2.712	3.319
39.	1.304	1.685	2.023	2.426	2.708	3.313
40.	1.303	1.684	2.021	2.423	2.704	3.307
41.	1.303	1.683	2.020	2.421	2.701	3.301
42.	1.302	1.682	2.018	2.418	2.698	3.296
43.	1.302	1.681	2.017	2.416	2.695	3.291
44.	1.301	1.680	2.015	2.414	2.692	3.286
45.	1.301	1.679	2.014	2.412	2.690	3.281

REFERENCES

1. Johns, B., *Design and Control of a New Reconfigurable Robotic Mobility Platform*, in *School of Mechanical Engineering*. 2007, Georgia Institute of Technology: Atlanta, GA. p. 112.
2. Matthies, L., et al., *Computer Vision in the Mars Exploration Rover (MER) Mission*, in *Computational Vision in Neural and Machine Systems*. 2007, Cambridge University Press. p. 71-84.
3. Milner, A.D. and M.A. Goodale, *The Visual Brain in Action*. 1995: Oxford University Press.
4. Chao, L.L. and A. Martin, *Representation of manipulable man-made objects in dorsal stream.* NeuroImage, 2000(12): p. 478-484.
5. Howard, A. and H. Seraji, *"Vision-Based Terrain Characterization and Traversability Assessment". Journal of Robotic Systems*, 2001. **18**(10): p. 10.
6. Ciftcioglu, O., M.S. Bittermann, and I.S. Sariyildiz, *"Towards Computer-Based Perception by Modeling Visual Perception: A Probabilistic Theory"*, in *IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC '06*. 2006: Taipei. p. 5152-5159.
7. Ferreira, F., V. Santos, and J. Dias. *"Landmark Detection for Vision-based Navigation using Multi-Scale Image Techniques"*. in *Proceedings of the World Automation Congress, 2004*. 2004. Seville.
8. Happold, M. and M. Ollis, *"Autonomous Learning of Terrain Classification within Imagery for Robot Navigation"*, in *IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC '06* 2006: Taipei. p. 260-266.
9. Kim, D., et al., *"Tracking Control of a Moving Object for Robokers with Stereo Visual Feedback"*, in *IEEE International Conference on Integration Technology, 2007. ICIT '07*. 2007: Shenzhen. p. 52-57.
10. Song, X., et al., *"Visual Odometry for Velocity Estimation of UGVs"*, in *International Conference on Mechatronics and Automation, 2007. ICMA 2007*. 2007: Harbin. p. 1611-1616.
11. Sun, T.-Y., et al. *"The study on intelligent vehicle collision-avoidance system with vision perception and fuzzy decision making"*. in *IEEE Proceedings on Intelligent Vehicles Symposium, 2005*. 2005.
12. Terzopoulos, D. and T.F. Rabie. *"Animat vision: Active vision in artificial animals"*. in *Proceedings of the Fifth International Conference on Computer Vision, 1995*. 1995. Cambridge, MA.
13. Balch, T. and R. Arkin. *Avoiding the past: a simple but effective strategy for reactive navigation*. in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. 1993. Atlanta, GA.
14. Talukder, A., et al. *"Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles"*. in *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on* 2002. Pasadena, CA.
15. Xiangjun, G., et al., *"The Path Planning of Virtual Endoscopy Based on Image Segmentation"*, in *Control Conference, 2007. CCC 2007. Chinese*. 2007: Hunan.

16. Castano, A. and L. Matthies, *"Foliage Discrimination using a Rotating Ladar"*, in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*. 2003: Taipei, Taiwan.
17. Maimone, M., et al. (2004) *"Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission"*. NASA Jet Propulsion Laboratory **Volume**,
18. Volpe, R., *Navigation Results from Desert Field Tests of the Rocky 7 Mars Rover Prototype.* The International Journal of Robotics Research, 1999. **18**(7): p. 14.
19. Kennedy, B., et al., *"LEMUR: Legged Excursion Mechanical Utility Rover"*, NASA Jet Propulsion Laboratory, California Institute of Technology: Pasadena, CA. p. 11.
20. Biesiadecki, J.J. and M.W. Maimone, *The Mars Exploration Rover surface mobility flight software driving ambition*, in *2006 IEEE Aerospace Conference*. 2006: Pasadena, CA, USA.
21. Li, R., et al., *"Rover Localization and Landing Site Mapping Technology for the 2003 Mars Exploration Rover Mission"*. Journal of Photogrammetric Engineering and Remote Sensing, 2004.
22. Goldber, S., M. Maimone, and L. Matthies, *"Stereo Vision Rover Navigation Software for Planetary Exploration"*, in *IEEE Aerospace Conference*. 2002: Big Sky, Montana.
23. Olson, C., L. Matthies, and M. Schoppers, *"Rover Navigation using Stereo Ego-motion"*. Robotics and Autonomous Systems, 2003. **43**(4).
24. Wilcox, B. and T. Nguyen, *"Sojourner on Mars and Lessons Learned for Future Planetary Rovers"*, in *28th International Conference on Environmental Systems*. 1998: Danvers, MA.
25. Bickler, D., *"A New Family of JPL Planetary Surface Vehicles"*, in *Missions, Technologies, and Design of Planetary Mobile Vehicles*. 1992: Toulouse, France. p. 301-306.
26. Schneider, S., V. Chen, and G. Pardo-Castellote, *"The ControlShell Component-Based Real-Time Programming System"*, in *IEEE International Conference on Robotics and Automation*. 1995. p. 2381 - 2388.
27. Charmed Labs, L. *CharmedLabs - Qwerk*. 2006 [cited 2008 March 19]; Available from: http://www.charmedlabs.com/index.php?option=com_content&task=view&id=29 .
28. Corp., L. *Logitech > Webcams + Communications > Webcams > QuickCam®*. 2008 [cited 2008 March 18]; Available from: http://www.logitech.com/index.cfm/webcam_communications/webcams/devices/352&cl=us,en.
29. Pham, H., ed. *Springer Handbook of Engineering Statistics*. 2006, Springer-Verlag London Limited.
30. Howard, A. and L. Parker, *"A hierarchical strategy for learning of robot walking strategies in natural terrain environments"*, in *2007 IEEE International Conference on Systems, Man and Cybernetics*. 2007: Montreal, QC, Canada. p. 2336-2341.