SIMULATION AND ANALYSIS OF TERRAIN SENSING, POSITION DETERMINATION AND NAVIGATION BY A ROBOTIC LUNAR ROVER

By

William Russell Longhurst

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Mechanical Engineering

December, 2007

Nashville, Tennessee

Approved:

Alvin M. Strauss

George E. Cook

Kenneth R. Fernandez

ACKNOWLEDGEMENTS

I would like to thank Dr. Alvin Strauss and Dr. George Cook for their leadership and guidance not only toward this thesis research but for their overall contribution to my academic development while at Vanderbilt University.

I would also like to thank Dr. Ken Franedez at NASA's Marshall Space Flight Center for his time and contributions to this research. His willingness to provide and share incite into his work at Marshall Space Flight Center's Exploration Advanced Capabilities Office was invaluable to me.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION AND SUMMARY

Background Information

Previous surface exploration of the Earth's moon and of the planet Mars was performed with robots that were controlled remotely from Earth based command centers. Although the robots generally performed successfully, each one of their tasks had to be initiated and controlled by a human operator. This form of operation is very time consuming and labor intensive for the command center. When data is received from the robot, it has to be analyzed by an operations staff before the next command can be sent to the robot. The problems of remote control manifest themselves particularly when it comes to a mobile robot's navigation. When camera images from the robotic rover are used as the primary means of navigation, the time delay to transmit and process the image becomes a burden on the operations staff. With a time delay, the operations staff spends a lot of time waiting on imagery data in between commands. In addition, because there is no real time image, the operations staff becomes inherently limited with each motion command specifying the distance to traverse the rover across the lunar surface. If they become too aggressive and send the robotic rover to a point not clearly seen through the camera, they run the risk of damaging the robot. They even could end its mission by driving it over the edge of a cliff, or into a crater where it would remain confined. To prevent this from occurring, the operations staff with each motion command traverses the robot a small distance over the surface and then waits for an updated camera image

before commanding the rover to move again. The time delay for image transmission from the Earth's moon is small in comparison to Mars. Operating a robotic rover on Mars from an Earth based command center becomes even more difficult due to communication delays of up to forty minutes (LeMaster and Rock 2003).

To solve this problem, future robotic rovers need to depend less on human operators. A better type of control architecture for a robotic rover would be supervised control. Under supervised control there would be limited interface between the robotic rover and the human operator. The robotic rover could perform tasks under its own control without tying up valuable human resources located at the command center. In regard to navigation, the robotic rover could traverse the surface under its own control and only be assisted by a human operator in certain situations. Possible situations that might require human interface could be fault recovery, collision avoidance, instrumentation calibration and assignment of mission tasks.

A first step in developing a robotic rover that has supervised control involves addressing the position determination problem. Before a robotic rover can successfully navigate itself on a foreign surface, it must know its starting position coordinates. A method presented in this thesis calls for a laser range finder on a robotic lunar rover to measure its distance to prior mapped lunar terrain features such as craters, boulders and rocks. The laser range finder would be remotely controlled by an operator back at the command center. The operator would locate, through the use of a camera, identifiable terrain features. When three terrain features are identified by the operator, the laser range finder would be activated to measure its distance to those objects. With the position

coordinates of the terrain features and the measured distance to them, the robotic lunar rover's position can be calculated.

Research Presented

The research performed for this thesis examined the method and accuracy of determining a robotic lunar rover's position through measuring its relative distance to lunar terrain of known position. The mathematical basis behind this position determination method is called three dimensional trilateration. This same mathematical formalism is used by the United States' space based global positioning system (GPS). However, GPS uses satellites that orbit the Earth to broadcast a position and a time signal to a receiver. This broadcasted data is then used by the receiver to determine its position (LeMaster and Rock 2003). The lunar rover presented in this research measures distance with a laser range finder and an Earth based operator enters the coordinates of the ranged terrain features.

Chapter II presents a series of simulations that was created to emulate proposed lunar missions of mapping and exploration. The series began by simulating a mapping mission of the lunar surface. The modeled lunar surface was mapped by a modeled lunar reconnaissance orbiter (LRO). Each square meter of the lunar surface was mapped to an elevation accuracy of plus or minus one half meter. The data generated from the LRO created a three dimensional map of the lunar surface. The generated map from this LRO simulation was able to provide visual imagery regarding the size and location of terrain features on the lunar surface.

The modeled lunar rover measured its relative distance to the distinguishing terrain on the modeled lunar surface. Each one of the terrain features had assigned positional data. The positional data was assigned based upon the lunar map that was generated by the LRO mapping simulation. Thus inherently there was some error between the true location of each terrain feature and its measured location. In addition, there was distance measurement error associated with the limited accuracy of the laser range finder.

Chapters III and IV present the results and conclusions of these simulations. The experimentally obtained results show that the lunar rover's position was obtained on average to within 3.31 meters of its actual position. The results are even better when only the horizontal coordinates are considered. On average, the x position was determined to within 0.03 meters and the y position to within 0.80 meters.

To further evaluate this positioning error a dynamics and navigation simulation was created to graphically analyze the resulting lunar rover motion. Within this simulation the drift of an inertial measurement unit (IMU) was also modeled. The results of this simulation show the effects of the initial position error and the need to recalculate position periodically due to the accumulating measurement error of the IMU.

Survey of Literature

Within the last ten years there has been a lot of research in the field of autonomous robot navigation. This research has been inspired by the dawn of GPS and its potential applications. With the establishment of GPS, a positional reference can easily be obtained by any robot that has a receiver. Recent research has been conducted

using computer simulations to develop new alignment methods for IMU systems and techniques to overcome the errors associated with the inertia sensors (Zhang, Y. and Gao, Y., 2007). Zhang and Gao published their work in March 2007 in a paper titled "A Method to Improve the Alignment Performance of GPS-IMU System". A even more recent paper titled "Integrity of an Integrated GPS/INS System in the Presence of Slowly Growing Errors. Part I: A Critical Review" was published in July 2007 (Bhatti, U., Ochieng, W. and Feng, S., 2007). The paper presents a critical review of GPS and how it is used in conjunction with an inertia navigation system. It examines failure modes and current error monitoring methods that ensure system integrity. The authors conclude that tightly coupled GPS and inertia navigation systems are needed for accurate navigation.

Literature closely related to the research presented in this thesis can be found in the Autonomous Robot Journal. Work published by Kwon, Y. and Lee, J titled "A Stochastic Map Building Method for Mobile Robot using 2-D Laser Range Finder" presents a method for a mobile robot to use a laser range finder and build a virtual map of its surroundings (Kwon, Y. and Lee, J., 2004). The created map of the robot's environment is then used with algorithms for navigation and obstacle avoidance. A even more in depth research of this topic is presented in the article "A Set Theoretic Approach to Dynamic Robot Localization and Mapping" (Di Marco, M., Garulli, A., Giannitrapani, A. and Vicino, A., 2004).

More closely related literature to this thesis can be found in two papers. The first paper titled "Active Single Landmark Based Global Localization of Autonomous Mobile Robots", presents a method for a mobile robot to estimate its position based upon the known location of one landmark (Bais, A., Sablatnig, R., Gu, J., and Mahlknecht, S.,

2006). The proposed robot and algorithm uses a stereo vision camera mounted on a rotating head to measure the landmark's range from three different positions. A trilateration based method is then used to calculate the robots position. The second paper titled "A Local-Area GPS Pseudolite-Based Navigation System for Mars Rovers" presents a navigation system that enhances GPS (LeMaster, E. and Rock, S., 2004). The navigation system is called the Self-Calibrating Pseudolite Array (SCPA). It uses an array of GPS-based transceivers that are positioned on the ground. The array of transceivers acts as navigation beacons for mobile robots operating in a localized area. Positional accuracy of the order of a centimeter can be obtained.

Simulation Software

The two software packages used for this series of simulations were ROBOSIM and MATLAB Simulink. ROBOSIM is a robotic software simulation package developed at Vanderbilt University and used for class instruction and research (Fernandez 1988) (Springfield 1993). The software has a built in LISP interrupter. All of the modeling for the ROBOSIM simulations was performed using LISP code. A listing of LISP files created for key models and simulations is included in Appendix B. The lunar surface, lunar rover, LRO and an invisible Cartesian robot were all modeled with ROBOSIM. The LRO surface mapping and the robotic lunar rover's position determination simulations were also performed using ROBOSIM.

Simulink is a software package built into MATLAB. It is used for graphically modeling and simulating dynamic systems (The MathWorks 2004). The dynamics and navigation simulation was created using Simulink. The robotic lunar rover's proposed

propulsion, steering and navigations systems were modeled and the rover's resulting motion was graphically displayed each time the simulation was performed.

Mathematics and Mechanics

The presented simulations utilized basic vector mechanics to position, transform and map objects. Most of the mathematics required to perform these operations are contained in ROBOSIM functions. Thus a basic understanding of the math is needed. A brief review of vectors, frame to frame mapping procedures, and three dimensional trilateration is presented in the following sections.

Position and Velocity Vectors

To mathematically describe a point or an object's position relative to a reference frame requires a position vector. The position vectors used in this research were based upon a Cartesian reference frame. A typical position vector referenced to a Cartesian coordinate frame is shown in Figure 1.1. The position vector contains three elements that relate its position to each of the three Cartesian axes. Position vectors used in the simulations positioned both points and objects. When an object was positioned in the simulations it was positioned with regards to its assigned frame. Thus every object used had its own coordinate frame. Each object's frame was positioned relative to another frame. All objects used in the simulation were eventually referenced back to the single world frame.

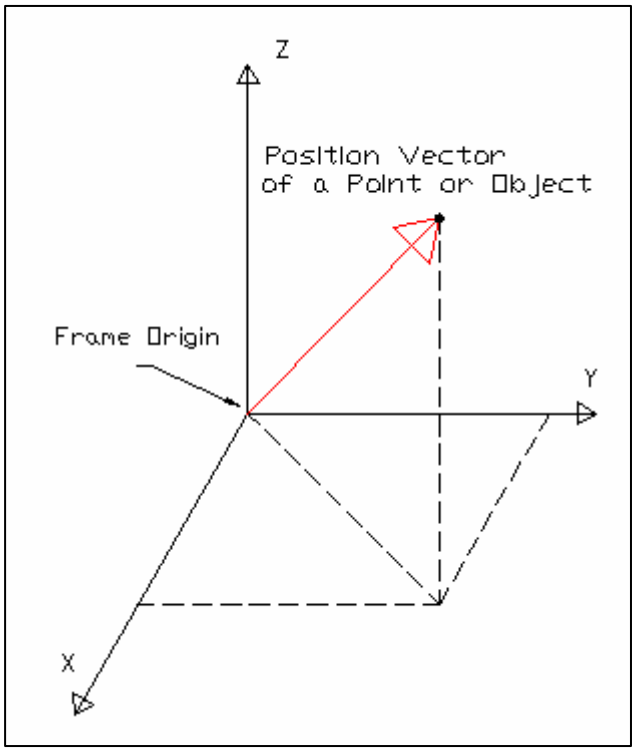

Figure 1.1: A position vector relative to a known frame.

A velocity vector is a different type of vector than a position vector (Craig, J. 2005). Although both types of vectors could have the same value of magnitude and direction, they do not possess the equivalent meaning. A velocity vector could be positioned anywhere relative to the reference frame and have the same meaning provided that it maintained its magnitude and direction. Because a position vector has dimensions of length and a line of action it can only be placed one way on the coordinate reference frame. A velocity vector was used in the dynamics and navigation simulation to describe the lunar rover's instantaneous velocity in relation to the each of the three principal axes of the world reference frame. This velocity vector was continually placed at different locations on the coordinate reference frame as the rover traversed the lunar surface but it always had the equivalent meaning shown in Figure 1.2.

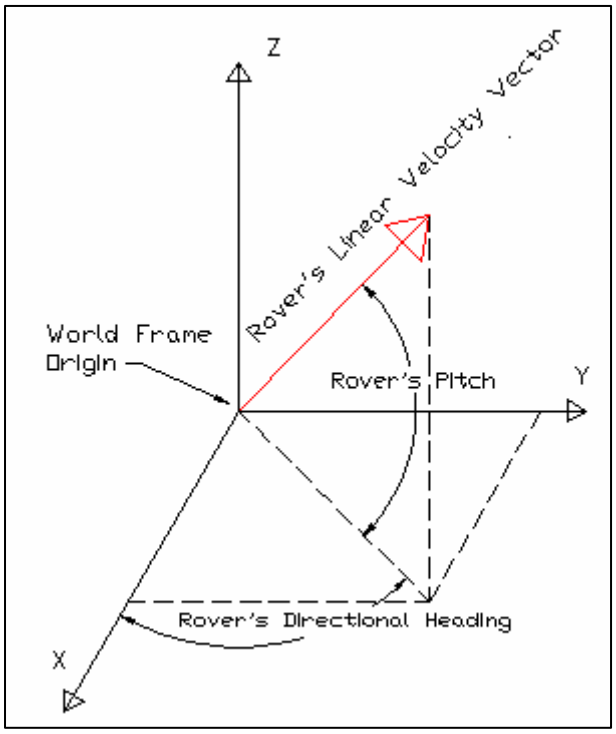

Figure 1.2: The lunar rover's velocity vector.

Orientation

As stated in the last section each object used in the simulations had a frame and its frame was positioned relative to another frame. To mathematically describe an object's frame orientation with respect to another frame, a dot product operation is used. When each of the principal axes of both the object and the reference frame is consider being a unit vector, the dot product of two axes simply becomes the value of the cosine of the angle between them. The dot product operation can be viewed as the projection of the object's frame onto the reference frame. Each of the three principal axes of the object's frame is projected onto each of the three reference frame's axes. The x axis of the object is projected on the x axis, y axis and z axis of the reference frame. The process is then

repeated for the y axis and z axis of the object. The result of this operation is a 3 x 3 matrix that is called the rotation matrix (Craig, J., 2005).

Transformation Matrix

To mathematically describe the combined position and orientation of an object with respect to a reference frame a transformation matrix is used. The transformation matrix is used because it not only allows us to mathematically describe, but also to manipulate an object's position and orientation with respect to another frame. Since it is a matrix it can be used to write an equation that strings together a series of frame relationships. The created equation simply multiplies the matrixes together to get a net relationship between the two end frames. This equation can also be used to describe or operate on an object by translating, rotating and transforming it to a certain position and orientation. Figure 1.3 shows a series of frames and illustrates and how they can be mapped together and used to either describe or operate on an object.

Frame A and B shown in Figure 1.3 are each position by a vector relative to the world frame. In addition both of their orientations are known relative to the world frame. From the mappings shown, Frame B's position and orientation is also known relative to frame A. Thus there are two ways to map frame B relative to the world frame. Frame B could first be mapped to A and the resultant can then be mapped to the world frame. This product of mappings produces the net result of frame B's position and orientation relative to the world frame. This is graphically illustrated by the series of vectors between the three frames shown in Figure 1.3.

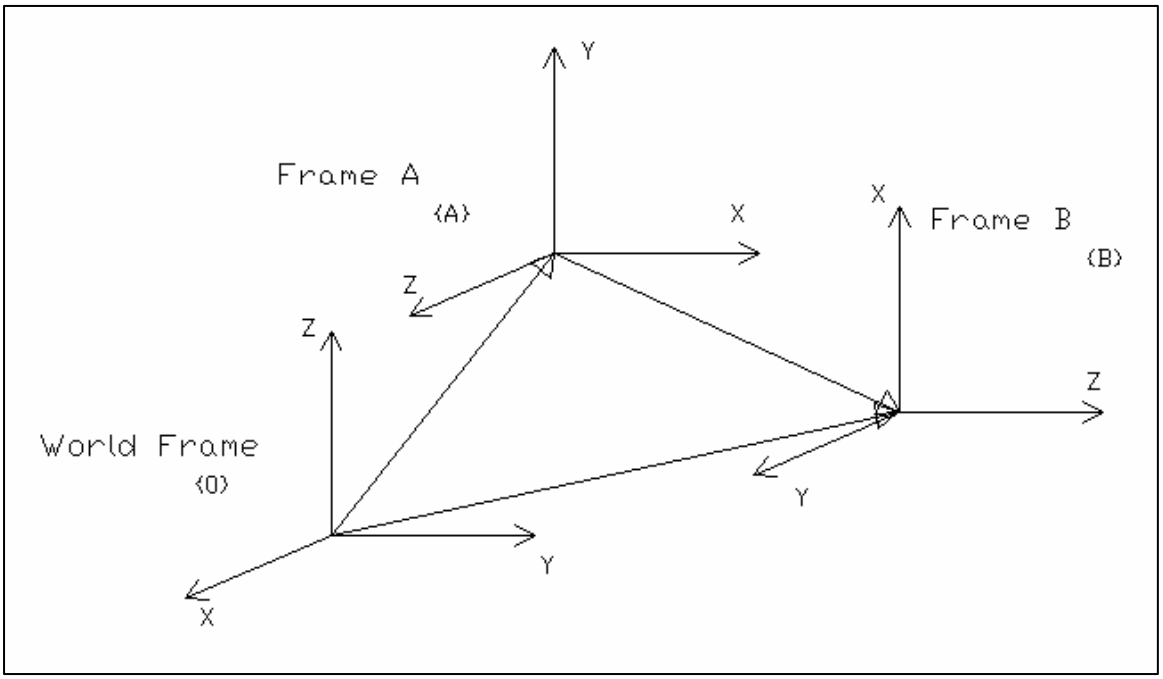

Figure 1.3: Mapped frames.

To make this mathematical operation possible a 4 x 4 homogenous transformation matrix is created (Craig, J., 2005). The homogeneous transformation matrix consists of the 3 x 3 rotation matrix and the 3 x 1 position vector. The position vector is placed in the far right column of the homogenous transformation matrix while the rotation matrix occupies the first three rows and columns. The fourth row of the homogenous transformation matrix consists of the values [0 0 0 1].

Three Dimensional Trilateration

To solve for the lunar rover's position three dimensional trilateration was used. Three dimensional trilateration uses three reference points and the distance to these points to determine a position. In association with the following text, the diagram in Figure 1.4 helps explains how three dimensional trilateration works. In addition both a geometrical

approach and numerical approach is presented in the following paragraphs to explain the position determination method.

After the lunar rover measures its distance to reference point one, its position is then known to be at a distance of r_1 from reference point one. The possible positions of the rover are now restricted to the surface of a sphere. The sphere has a radius of r_1 and is located at reference point one. As the rover completes its second distance measurement to reference point two, the rover must also be positioned somewhere on the surface of the second sphere. The second sphere has a radius of r_2 and is located at reference point two. Upon examining these two location criteria it can be determined by inspection that the rover must then lie somewhere on the circle that is created by the surface overlap of the two spheres. To further reduce the possible positions of the rover, another distance measurement is taken with respect to a third known reference point. The third distance measurement of r_3, tells us the rover must also be positioned on the surface of a sphere that has a radius of r_3 and that is located at reference point three. This third position criterion means that the rover is located at the point where the third sphere intersects the circle that was created by the surface overlap of the spheres located at reference points one and two.

From a numerical evaluation of three dimensional trilateration, the solution for position can be determined through the equations of three spheres. Each sphere can be expressed in its equation form. The position of the rover can be found by setting all three equations equal to each other and solving for the three unknown coordinates. However the math quickly becomes complicated. To simplify this problem, the spheres are together transformed into a more solvable form as shown in Figure 1.4.

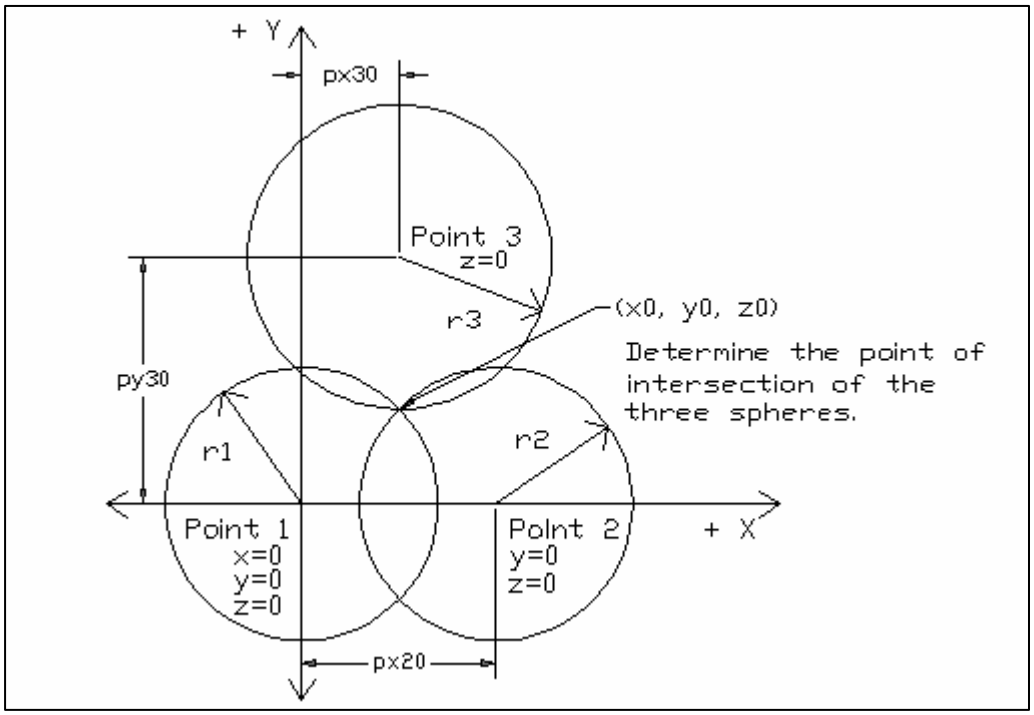

Figure 1.4: Reference points repositioned to the x-y plane and viewed as spheres.

To begin the transformation, the first reference point's location is translated to the world frame's origin. Reference points two and three are also translated to new locations but all three reference points still maintain their spatial relationship with each other. Reference point two is rotated so that it is located on the positive x axis. The first rotation is about the z axis and then the second rotation is about the y axis. To maintain the same spatial relationship, reference point three is also transformed through the same identical rotations. The last transformation involves rotating reference point three to the x-y plane. This is done by rotating it about the x axis. All of the rotation angles can easily be determined by inspecting each reference point's positional coordinates and then applying trigonometric principles to solve for the angles.

Once the transformation is complete the reference point's new locations and the corresponding measurements to them can be applied to the following three equations of the resulting spheres.

$$r_1^2 = x^2 + y^2 + z^2 \tag{1.1}$$

$$r_2^2 = (x - px20)^2 + y^2 + z^2 \tag{1.2}$$

$$r_3^2 = (x - px30)^2 + (y - py30)^2 + z^2 \tag{1.3}$$

To solve for the value of the x coordinate, equation (1.2) can be subtracted from equation (1.1) and rearranged into the form given by equation (1.4).

$$x = (r_1^2 - r_2^2 + px20^2) / (2 \cdot px20)^2 \tag{1.4}$$

Substituting equation (1.4) into equation (1.1) and then setting equation (1.1) equal to equation (1.3) yields the solution for y given in equation (1.5).

$$y = ((r_1^2 - r_3^2 + px30^2 + py30^2) / (2 \cdot py30)) - ((px30 / py30) \cdot x) \tag{1.5}$$

The calculated values of x and y can now be entered into equation (1.1). Equation (1.1) is then rearranged into equation (1.6) to solve for the value of z.

$$z = \sqrt{(r_1^2 - x^2 - y^2)} \tag{1.6}$$

With the solution of z being the square root of a number there are three possibilities that could result. If the solution to equation (1.6) is zero, it means the sphere located at reference point 3 intersects at one location, the circle created by the surface overlap of the spheres at reference point one and two. A second scenario is no real solution to equation (1.6) exists, because one can not solve for the square root of a negative number. When this occurs, it means the sphere at reference point three is outside the circle created by the surface overlap of the spheres located at reference point one and two. With noisy data this situation is very plausible. The solution when this occurs is to set z = 0 since it is the closest answer. The third scenario is when there are two real solutions to equation (1.6). This means that the sphere located at reference point three intersects at two distinct locations, the circle created by the surface overlap of the spheres located at reference point one and two.

When there are two solutions to equation (1.6), there are two options to determine the correct answer. The first is two take an additional measurement to a fourth reference point. The other is to apply logic through observing the relationship between the three reference points and the location of the rover. If the plane created by the three reference points is above the rover, the solution that leads to the rover's correct position is the negative value given in equation (1.6). On the other hand, if the plane created by the three reference points passes below the rover, the positive solution to equation (6) leads to the correct position of the rover. As long as the rover uses three reference points that are on three different sides of its location and above its current elevation, the correct solution will always be the negative value. If there is any uncertainty about whether the reference points are below or above the rover, a simple test can be performed to

determine the answer. By measuring with the laser range finder, the rover could determine the coordinates of the reference points relative to the rover. An equation of the plane created by the reference points relative to the rover could then be established. The equation could then be evaluated to determine the z coordinates' value when both x and y are set to zero. When the reference point's coordinates are established relative to the rover, the rover's position is at the origin of the reference frame. If the returned z value is negative it means the rover is above the plane created by the three reference points. If the value is positive, the rover is below the plane created by the three reference points.

Now that the x, y and z coordinates have been determined, the transformation used to place the reference points in the required criteria, must be applied in reverse to the calculated coordinates to find the true position of the rover.

Invisible Cartesian Robot

A key element used in the simulations was an invisible Cartesian robot. An invisible Cartesian robot is a six axis robot with no visible links. Since there are not any visible links, each axis has an infinite number of positions it can obtain. With three axes dedicated to positioning and the other three dedicated to orientation, the invisible Cartesian robot can place its sixth axis frame in any desired position and orientation. With this capability, the invisible Cartesian robot was used to grasp either the LRO or the lunar rover and move it about in the simulation environment. Through the use of the Cartesian robot and with its inverse kinematics program defined in ROBOSIM, the LRO was moved over the lunar surface to simulate its motion while in lunar orbit. In addition

through the use of the Cartesian robot and Simulink, the lunar rover was moved on the lunar surface to simulate its autonomous navigation and lunar surface exploration.

The six frames of the invisible Cartesian robot are shown in Figure 1.5. The first three axes were used for positioning along the x, y and z axes of the world frame, while the last three were used for orientation. The Denavit-Hartenberg link parameter table (Denavit, J. and Hartenberg, R., 1955) (Craig, J., 2005) shown in Table 1.1, numerically shows each of the links and how each link relates to its adjoining link.

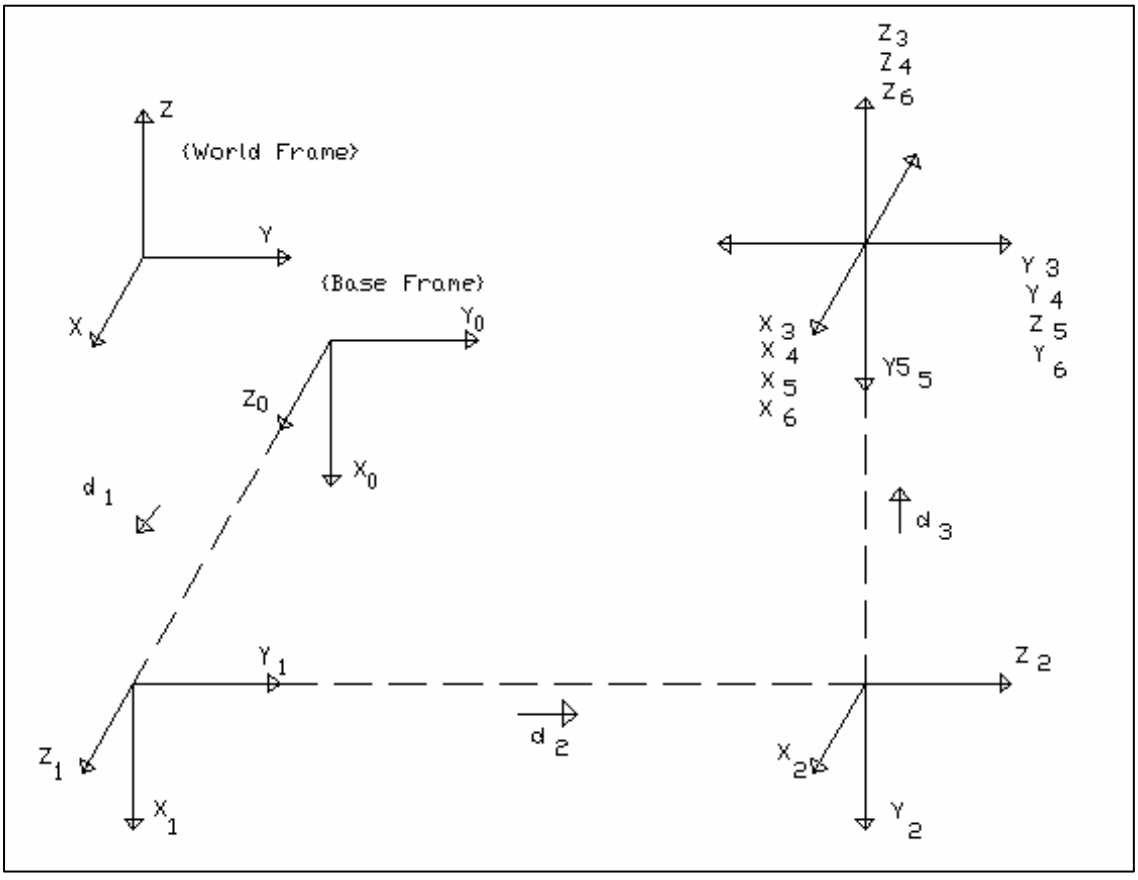

Figure 1.5: The invisible Cartesian robot frame assignments.

The information contained in the parameter table was used to construct the Cartesian robot. The notation used to construct the Denavit-Hartenberg table is defined as:

a_i = the distance from Z_i to Z_{i+1} measured along X_i;

α_i = the angle form Z_i to Z_{i+1} measure along X_i;

d_i = the distance from X_{i-1} to X_i measured along Z_i;

θ_i = the angle from X_{i-1} to X_i measured along Z_i.

The ROBOSIM LISP file for constructing this robot is listed in Appendix B. In addition the inverse kinematics equations used in positioning each axis to obtain a desired position and orientation for the LRO and the rover is derived in Appendix A.

Table 1.1: Invisible Cartesian robot link parameter table.

i	α_{i-1}	a_{i-1}	θ_i	d_i	Variable
1	0	0	0	d_1	d_1
2	-90	0	-90	d_2	d_2
3	90	0	0	d_3	d_3
4	0	0	θ_4	0	θ_4
5	-90	0	θ_5	0	θ_5
6	90	0	θ_6	0	θ_6

Control Systems

Although control systems are not a primary focus of the research presented in this thesis, some background information regarding the control methods utilized in one of the simulations is necessary. Three control systems were developed for the implementation

of the dynamics and navigation simulation. To accurately simulate the resultant motion of an autonomous rover, control systems for the propulsion, steering and navigation models were developed. Each of the three control systems used feedback to establish a closed loop system. The feedback was then used to compute the servo error.

For the propulsion and the steering system, the processing of their servo errors led to the required input torque for their motors to drive the lunar rover's velocity and steering wheels' position errors to zero. The navigation controller used the error associated with directional heading to change the steering wheels' position in order to drive the directional heading error to zero.

The propulsion and steering control systems each utilized two control laws that worked in conjunction and supplied the required torque for the propulsion and steering motors. To accomplish this each of the control systems was partitioned into two portions. The partitioned controller consisted of a model based portion and a servo based portion. The servo based portion arranged the servo error into a differential equation that had a solution that decayed to zero over time. Controller gains were selected and placed in the differential equation to control the behavioral response of the system. The model based portion was used to cancel the nonlinearity of the propulsion and steering systems and to provide enhanced control of the complex systems. This method of control modeling is known as the computed-torque method (Craig, J. 2005) (Paul, P., 1972) (Markiewicz, B., 1973) (Bejczy, A., 1974).

The navigation system was constructed in a manor that just applied proportional, integral, and derivative (PID) control to the directional heading error. The PID controller sent the conditioned error signal to the steering system as the desired steering wheels'

position. As the steering wheels turned in conjunction with the linear velocity from the propulsion system, the directional heading naturally changed. The PID controller simply drove the directional heading error to zero by commanding the steering motor to turn the wheels until the rover achieved its desired directional heading.

CHAPTER II

MODELING AND SIMULATION

Models

Lunar Surface

As previously stated the lunar surface model was created in ROBOSIM. The lunar surface model represented forty nine thousand square meters of horizontal lunar landscape. The length of each side of the square surface model was seventy meters. As seen in Figure 2.1 the surface was made of five distinct feature groups. The groups were: the two craters, the flat surface, the large boulder formation in the bottom, the large jagged rock formation on the left side and two sets of rolling hills on the top and right side of Figure 2.1. In addition to the five distinct feature groups there were several small rocks scattered throughout the flat surface and on the hillsides.

The three highest surface points on the model were the twin peaks of the large jagged rock formation and the hilltop on the far side of the lunar terrain. Both of these terrain features had a peak height of about ten meters above the flat surface. The flat surface was position at thirty meters above the x – y plane of the world frame. The two lowest elevations on the model were the two craters. Both of these craters were approximately five to seven meters wide and five meters deep.

A listing of the LISP file used to model the lunar surface is contained in Appendix B. As can be seen in the file listing, the lunar surface was modeled as a composite object

within ROBOSIM. It was necessary to establish the surface as a single simulation object for three reasons. The first was to simplify the positioning of the entire model relative to the world frame. Second, the creation and positioning of the terrain features became much simpler because they were placed relative to the flat surface instead of the world frame. The lines of code used to create and position the terrain features were clearly separated and neatly organized within the LSIP file. If changes were desired, they could have been easily accomplished due to this organization scheme. Third, fewer lines of code were used to create the scanning algorithm for the LRO scan simulation. With the surface being a composite object, only one object had to be checked for a collision with the simulated laser beam.

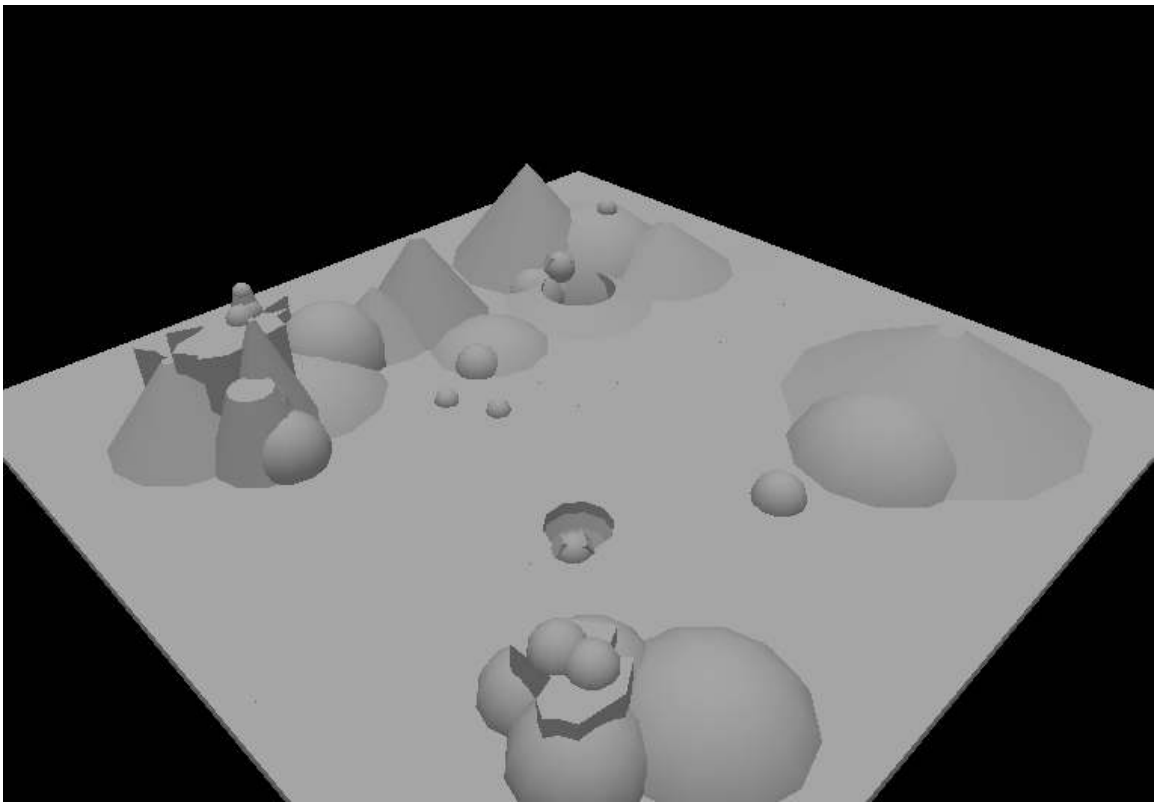

Figure 2.1: The lunar surface model.

Lunar Reconnaissance Orbiter

In the series of simulations, the lunar reconnaissance orbiter (LRO) was an artificial lunar satellite that orbited the Earth's moon and mapped the surface. The LRO moved above the lunar surface and used a laser to measure the distance between the lunar surface and its position. The LRO with its laser produced data that was used to generate within the simulation environment, a three dimensional map of the lunar surface. Each square meter of the surface was mapped to an accuracy of one half meter in elevation.

The LRO modeled in ROBOSIM consisted of its main body and two solar panels on each side of its body. The originating point of the laser used for measuring was at the base of the LRO. The surface mapping algorithm placed the laser at the LRO's base and extended the laser beam until it collided with the lunar surface. The algorithm then calculated the distance from the LRO's base to the collision point. The calculated distance was then used to generate a single pixel of the lunar surface map.

The LRO movement through its orbit was accomplished in the simulation environment with the use of the invisible Cartesian robot. The invisible Cartesian robot grasped the LRO at its base and moved it over the surface while it measured the distance to the terrain below.

Robotic Lunar Rover

The design of the lunar rover model was created based upon the existing design of a terrestrial rover called MARCbot. MARCbot stands for: multi-functional, agile, remote controlled robot. MARCbot was developed by an engineering company called Exponent. MARCbots are primarily used by the United States Army as a tool to search and inspect

potentially explosive devices. To perform their dangerous tasks, a MARCbot is remotely operated with the aid of an onboard camera. The image captured by the camera is transmitted back to the command center where it can be processed and viewed on a video screen. Based upon the image being viewed on the video screen the operator will transmit operation commands back to the MARCbot.

This type of remote operation through the use of an onboard camera is similar to how the lunar rover would operate under supervised control. The lunar rover during its position determination routine would be operated remotely from Earth. An onboard camera would allow the operator to visually inspect the local lunar terrain. While inspecting the terrain the operator would search for distinguishing terrain features that were previously identified with the LRO mapping of the lunar surface. The operator would have at his disposal the LRO generated map of the lunar surface to reference. While viewing both the image from the lunar rover's camera and the image of the LRO generated map, a comparison of the two images can be performed. The operator would try to match a terrain feature seen through the onboard camera, such as a rock or boulder with its corresponding image on the LRO generated map. Once a distinguishing terrain feature was matched to a LRO mapped terrain feature, the operator would activate the onboard laser range finder to measure the distance from the lunar rover's base to the identified terrain feature of known position.

The robotic lunar rover created in ROBOSIM and used in the simulations is shown in Figure 2.2. The rover had a turret mounted laser range finder and camera. In addition the rover had an extendable arm with a small sensor mounted on its end that could be used for additional terrain sensing and measurement. The laser range finder and

video camera had combined motion and targeting ability in both azimuth and elevation angles.

Figure 2.2: The robotic lunar rover model.

The overall physical size of the lunar rover was very small when compared to the lunar terrain. The rover's wheel base length was sixty centimeters and its wheel base width was approximately forty-eight centimeters. The height of the turret that contained the laser range finder and camera was sixty-six centimeters. The four tires with all-terrain tread were eleven centimeters wide and twenty four centimeters tall. These physical dimensions are similar to the MARCbots overall dimensions.

The rover was created with its base frame in the middle of its wheel base. When the rover was positioned within the simulation environment, it was with its base frame. In addition after the laser measured the relative distance from the laser beam's point of origin to a nearby surface terrain feature, an algorithm took that measured distance and determined the distance between the identified surface terrain feature and the rover's base frame. Thus all positioning calculations for the rover were done with respect to its base frame. The LISP file used to model the robotic lunar rover is contained in Appendix B.

The modeled lunar rover was used in the position determination simulations as well as the dynamics and navigation simulation. In the position determination simulations, the rover with its laser range finder was used in conjunction with a distance measuring algorithm and a trilateration algorithm to calculate the rover's position. A set of ten simulations was created to experimentally determine the rover's position relative to know positions of terrain. In the dynamics and navigation simulation the rover was positioned and then traversed over the lunar surface. During the simulation the lunar rover's position was continually evaluated to understand the net effect of the measurement and positioning errors.

Propulsion System

A propulsion system model for the lunar rover was created in Simulink. The propulsion model was used within the dynamics and navigation simulation to produce the desired value of linear velocity for the lunar rover. The resulting linear velocity produced by the propulsion system was then used in conjunction with the steering and navigation systems to produce the desired motion dynamics for the lunar rover.

The propulsion model shown in Figures C.5 through C.7 of Appendix C consists of an electric motor and transmission. The motor is powered by direct current (DC) while the modeled transmission has a gear ratio of 10:1 to drive the rear wheels of the lunar rover.

The input torque to the DC motor must overcome several forces in order to drive the lunar rover. These forces consist of: vehicle inertia, motor inertia, rolling friction between the surface and the wheels, viscous damping due to machine lubrication and coulomb friction in the gears and bearings. In addition, with Earth-based testing or missions to Mars, the rover would have to overcome a damping force due to atmospheric drag on the vehicle. All of these potential force components were taken into account when the motor's torque equation was developed. Equations (2.1) through (2.7) with their defined variables, give the torque (τ_p) required by the DC motor to propel the lunar rover.

$$\tau_p = I_{\text{eff. inertia}}\,\omega'_p + \nu_{\text{eff damping}}\,\omega_p + \tau_{\text{gravity}} + \tau_{\text{atmospheric drag}} + \tau_{\text{rolling friction}} + \tau_{\text{coulomb friction}} \quad (2.1)$$

$\omega'_p \rightarrow$ Rover drive wheels' angular acceleration.

$\omega_p \rightarrow$ Rover drive wheels' angular velocity.

$$I_{\text{eff. inertia}} = \tfrac{1}{2}\, m\, r^2 + n^2 I_m \quad (2.2)$$

$m \rightarrow$ Mass of the rover.

$r \rightarrow$ Radius of the rover's wheels.

$n \rightarrow$ Transmission's gear ratio.

$I_m \rightarrow$ Motor's inertia.

$$\nu_{\text{eff damping}} = \nu_{\text{wheels}} + n^2 \nu_{\text{motor}} \tag{2.3}$$

ν_{wheels} → Viscous damping constant for the rover's wheels.

ν_{motor} → Viscous damping constant for the motor.

n → Transmission's gear ratio.

$$\tau_{\text{gravity}} = m\, g\, r \sin(\alpha) \tag{2.4}$$

m → Mass of the rover.

g → Gravity.

r → Radius of the rover's wheels.

α → Pitch angle of the rover.

$$\tau_{\text{atmospheric drag}} = \tfrac{1}{2}\, C\, A\, \rho\, V^2\, r \tag{2.5}$$

C → Coefficient of drag for the rover.

A → Front surface area of rover.

ρ → Atmosphere's density.

V → Rover's linear velocity.

r → Radius of the rover's wheels.

$$\tau_{\text{rolling friction}} = \mu\, m\, g\, r \cos(\alpha) \tag{2.6}$$

μ → Coefficient of rolling friction between the wheels and the surface.

m → Mass of the rover.

g → Gravity

r → Radius of the rover's wheels.

$\alpha \rightarrow$ Pitch angle of the rover.

$$\tau_{\text{coulomb friction}} = \beta \, \text{sign}(\omega_p) \tag{2.7}$$

$\beta \rightarrow$ Coulomb fiction constant in gears and bearings.

To control the propulsion system and ultimately the linear velocity of the lunar rover, a closed loop partitioned control scheme was selected. The controller was partitioned into a model portion and a servo portion. The model portion was further divided into an alpha (A_p) part and a beta (B_p) part. The model portion of the controller was utilized to provide more precise control of the complex propulsion system. In addition the model portion's control law along with its interaction with the servo portion's control law provided a method to process the nonlinear dynamics of the propulsion system. With the use of the alpha and beta sections of the model portion, the nonlinear components of the system were nullified. The servo control law was setup in the form of an error equation. The mathematical nature of the servo portion's control law in conjunction with the model portion's control law provided an algorithm that forced the motor's output speed to the value of the desired speed.

The control law for the model portion of the controller is given by equation (2.8). The control law equates the required torque from equation (2.1) to the model portions of the partitioned controller plus an input (τ'_p) from the servo portion.

$$\tau_p = A_p \, \tau'_p + B_p \tag{2.8}$$

Setting the variable τ'_p equal to ω'_p and then equating equations (2.1) and (2.8) leads to the mathematical definition of A_p and B_p. The values for the model portions of the controller are given by equations (2.9) and (2.10).

$$A_p = I_{\text{eff. inertia}} \tag{2.9}$$

$$B_p = \nu_{\text{eff damping}}\,\omega_p + \tau_{\text{gravity}} + \tau_{\text{atmospheric drag}} + \tau_{\text{rolling friction}} + \tau_{\text{coulomb friction}} \tag{2.10}$$

The servo portion of the controller used feedback to establish an error (e_p) pertaining to the difference between the desired input angular wheel velocity and the resultant output angular wheel velocity. The calculated angular wheel velocity error, the desired angular wheel acceleration (ω'_{pd}), the value of τ'_p, and the selected proportional and integral control gains ($K_p p$ & $K_p i$) were used to establish the servo control law given in equation (2.11).

The angular velocity error signal was parallel processed in two ways by the servo control law. In one circuit branch it was multiplied by the proportional control gain. In another branch the error signal was integrated and then multiplied by the integral control gain. The results of these two operations were added together along with the desired angular acceleration of the wheels and then fed into the model portion of the controller as the signal value for τ'_p.

$$\tau'_p = (K_p p)\, e_p + (K_p i) \int e_p + \omega'_{pd} \tag{2.11}$$

By recalling from the model control law that τ'_p equaled ω'_p and then subtracting ω'_p from both sides of equation (2.11), equation (2.12) is produced. This differential equation descries how the velocity error was conditioned and processed in the servo portion of the partitioned controller. In addition this differential equation describing the error signal dynamics has a solution that decays to zero over time. Thus the servo control drove the difference between the inputted desired velocity and the actual rover velocity to zero. Since the differential equation has a solution in which the error decays to zero, the response of the servo control system was controlled by the selection of the gain values. For this servo control system values of twenty five and five were selected for $K_p p$ and $K_p i$.

$$e'_p + (K_p p)\, e_p + (K_p i) \int e_p = 0 \qquad (2.12)$$

With the closed loop feedback of angular wheel velocity, the partitioned control system calculated and supplied the required current needed for the DC motor. The control system increased speed of the rover until it reached a velocity of one meter per second. This was accomplished by setting the desired speed profile in the form of a straight line with a positive slope. After five seconds of operation the desired speed reached its maximum linear velocity of one meter per second. Once the desired speed achieved this velocity, the desired speed was maintained at one meter per second. The controller was tuned so that the lunar rover responded accordingly to the desired speed profile. The lunar rover continually increased its speed for five seconds up to a value of

one meter per second. Once the maximum velocity of one meter per second was reached, the lunar rover maintained that value of velocity.

The modeled controller did not convert the physical system values into electrical process signals. Input and output variables of velocity and acceleration were mathematically processed through the system without any conversions of their forms. In addition since torque is proportional to current, the simulation model ignored operational amplifiers and thus furthered simplified the model by not converting torque to an equivalent value of current.

Steering System

The steering system modeled in Simulink consists of a DC motor, a rack gear, a pinion gear, linkage mechanisms, the two front wheels of the rover and a controller. The Simulink model is shown in Figures C.3 and C.4 of Appendix C. All of these modeled components worked in series together to turn the lunar rover's front wheels. In the previously presented propulsion model, the control system controlled the rear wheel's angular velocity about their horizontal axes to cause the wheels to move with a rolling motion. In contrast the steering control system controlled the two front wheel's angular position about their vertical axes in order to control the horizontal direction they rolled.

The steering system was modeled as a rack and pinion type of steering device. Torque from a DC motor turned the pinion gear that was attached to the motor's shaft. The rotating pinion gear was engaged with a rack gear. The rotating pinion gear caused the gear rack to translate in a linear motion. The rack with its linear motion was attached to mechanical linkages. The linear motion of the rack caused the linkages to turn the

front wheels about their vertical axes. The front wheels were limited to thirty degrees of rotation in either direction.

To control the angular positioning of the front wheels, a partitioned controller was utilized. The steering system's controller was designed almost identical to the propulsion system's partitioned controller. The steering system's partitioned controller also consisted of a model and a servo portion. The model portion was further divided into two additional portions. The model and servo portions each had their own control laws that worked together to supply the required torque to the DC motor used to steer the lunar rover.

The input torque to the DC steering motor must overcome several forces in order to turn the lunar rover's front wheels. These forces consist of: steering mechanism inertia, motor inertia, spring force in the steering mechanism, friction between the surface and the wheels, viscous damping due to machine lubrication and coulomb friction in the gears and bearings. All of these potential force components were taken into account when the steering motor's torque equation was developed. Equations (2.13) through (2.16) with their defined variables, give the torque (τ_s) required by the DC motor to turn the lunar rover's wheels.

$$\tau_s = I_{\text{eff. inertia s}}\,\Theta''_s + \nu_{\text{eff damping s}}\,\Theta'_s + k_{\text{eff spring}}\Theta_s + f_{\text{coulomb friction}}\,\text{sign}(\Theta'_s) \qquad (2.13)$$

$\Theta''_s \rightarrow$ Rover front wheels' angular acceleration.

$\Theta'_s \rightarrow$ Rover front wheels' angular velocity.

$\Theta_s \rightarrow$ Rover front wheels' angular position.

$k_{\text{eff spring}} \rightarrow$ Effective spring torque constant for the steering system.

$$I_{\text{eff. inertia s}} = \tfrac{1}{2}\, m_{st}\, r_{pg}^{2} + I_{st\ m} \tag{2.14}$$

m_{st} → Mass of the steering linkages, gears and wheels.

r_{pg} → Radius of the rover's pinion gear.

$I_{st\ m}$ → Motor's inertia.

$$\nu_{\text{eff damping s}} = \nu_{st\ \text{wheels}} + \nu_{st\ \ \text{motor}} \tag{2.15}$$

$\nu_{st\ \text{wheels}}$ → Viscous damping constant for the rover's steering mechanism.

$\nu_{st\ \text{motor}}$ → Viscous damping constant for the steering motor.

$$f_{\text{coulomb friction}} = f_{\text{suf}} + f_{\text{mech}} \tag{2.16}$$

f_{suf} → Sliding friction constant between wheels and surface.

f_{mech} → Sliding friction constant with steering mechanism.

Like the propulsion controller, the steering controller also used a closed loop partitioned control scheme. The steering controller was partitioned into a model portion and a servo portion. The model portion consisted of an alpha (A_s) part and a beta (B_s) part. The model portion control law was utilized to provide more precise control and a means to nullify the nonlinear dynamics of the steering system. The servo control law was established using the error from the wheel's angular position and angular velocity. The mathematical nature of the servo portion's control law in conjunction with the model portion's control law provided an algorithm that turned the front wheels to the desired angular position.

The control law for the model portion of the steering controller is given by equation (2.17). The control law equates the required torque form equation (2.13) to the model portions of the partitioned controller plus an input (τ'_s) from the servo portion.

$$\tau_s = A_s \tau'_s + B_s \tag{2.17}$$

Setting the variable τ'_s equal to Θ''_s and then equating equations (2.13) and (2.17), leads to the mathematical definition of A_s and B_s. The values for the model portions of the steering controller are given by equations (2.18) and (2.19).

$$A_s = I_{\text{eff. inertia s}} \tag{2.18}$$

$$B_s = \nu_{\text{eff damping s}} \Theta'_s + k_{\text{eff spring}} \Theta_s + f_{\text{coulomb friction}} \operatorname{sign}(\Theta'_s) \tag{2.19}$$

The servo portion of the steering controller used feedback of the wheel's angular position and velocity to establish error signals (e_s & e'_s). The calculated angular position error, angular velocity error, a desired angular acceleration (Θ''_{sd}), the value of τ'_s, and the selected proportional and derivative control gains (K_sp & K_sv) were used to establish the servo control law given in equation (2.20).

The angular position error signal was multiplied by the proportional control gain while the velocity error signal was multiplied by the derivative control gain. The results of these two operations were added together along with the desired angular acceleration (Θ''_{sd}) and then fed into the model portion of the steering controller as the signal value for τ'_s.

$$\tau'_s = (K_sp)\, e_s + (K_sv)\, e'_s + \Theta''_{sd} \tag{2.20}$$

By recalling from the model control law that τ'_s equaled Θ''_s and then subtracting Θ''_s from both sides of equation (2.20), equation (2.21) is produced. This differential equation descries how the position and velocity error signals were conditioned and processed in the servo portion of the partitioned controller. In addition this differential equation describing the error signal dynamics has a solution that decays to zero over time. Thus the servo control drove the difference between the desired input position of the front wheels and the actual front wheel's position to zero. Since the differential equation has a solution in which the error signal decays to zero, the response of the servo control system was controlled by the selection of the proportional and derivative gain values. For this servo control system values of nine and six were selected for K_sp and K_sv.

$$e''_s + (K_sv)\, e'_s + (K_sp)\, e_s = 0 \tag{2.21}$$

With the closed loop feedback of angular position and velocity, the partitioned control system supplied the required current needed for the DC motor to turn the lunar rover's front wheels to the desired position.

Dynamics System

The dynamics system created in Simulink is a kinematics model of the lunar rover. The system received input signals form the lunar rover's propulsion and steering

systems and then calculated the resulting instantaneous directional heading and position vector of the rover as it traversed the lunar surface. The model can be seen in Figure C.8 through Figure C.10 of Appendix C.

There are two major subsystems of the model. One subsystem calculated the rover's directional heading with respect to the world frame while the other continually integrated the rover's velocity vector to determine the position of the rover. The rover's physical characteristics of wheel base length (WB_{length}) and width (WB_{width}), along with the position angle of the rover's front wheels and the linear velocity of the rover were needed to determine the rover's directional heading. The wheel base length and width were of constant values but the rover's steering wheels' position and linear velocity values were dynamic. The every changing position angle of the wheels was fed into the dynamics system as the output signal of the steering system. Likewise the linear input velocity was the output signal of the propulsion system.

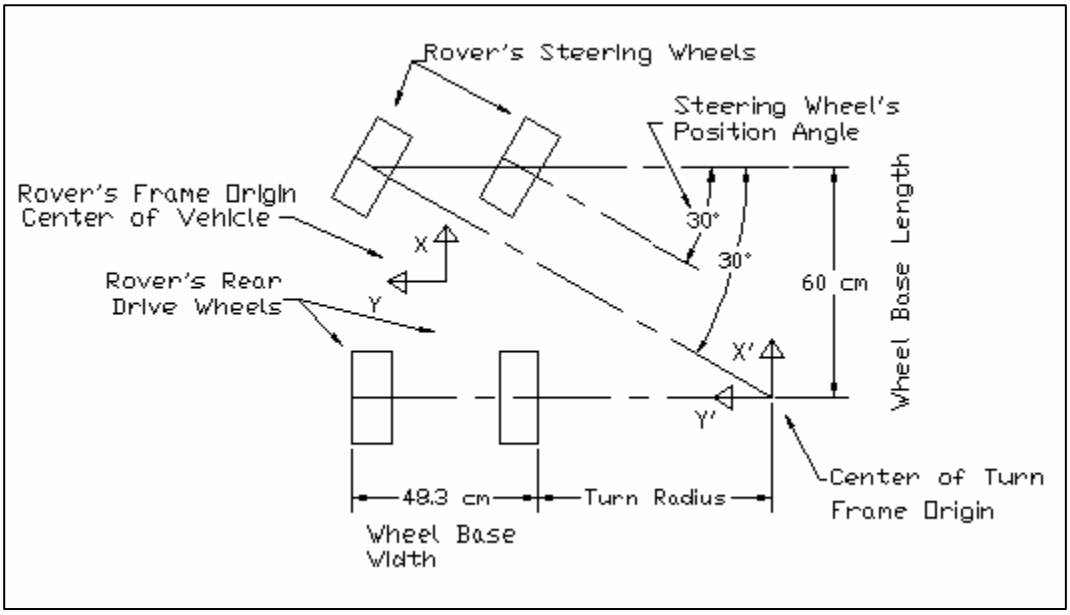

Figure 2.3: Turn radius as determined by the steering wheels' position.

Before the directional heading could be determined the turn radius of the rover had to be established. As can be seen in Figure 2.3, the turn radius was a function of the wheel base length and width as well as the front wheels' angular position. Equation (2.22) expresses the front wheel's position angle as a function of rover parameters.

$$\Theta_s = \tan^{-1}(WB_{length}) / (TR + WB_{width}) \tag{2.22}$$

Equation (2.22) can be rearranged into the more useful form given by Equation (2.23). Equation (2.23) calculated the needed value of the rover's instantaneous turn radius (TR).

$$TR = (WB_{length} - (\tan(\Theta_s) \times WB_{width})) / (\tan(\Theta_s)) \tag{2.23}$$

Next the rover's turn radius, liner velocity and wheel base width were used to define the rover's angular rotation rate (φ'). As can be seen in Figure 2.3, when the rover turned about a center point of turn, the center point's frame (X'-Y') orientation was the same as the rover's frame (X-Y). Thus the rover's orientation about both frames was always identical. When the angular rotation rate was calculated about the turn point's origin, it was mapped to the rover's frame. The mapping occurred by multiplying the angular rotation equation by a rotation matrix (R). Since the frames had the same orientation, the rotation matrix's value was unity. The rover's angular rotation rate was calculated using equation (2.24). To determine the direction heading (φ) of the rover, the angular rotation rate signal was simply integrated as given by equation (2.25).

$$\varphi' = R\,(V_{rover} / (TR + \tfrac{1}{2}\,WB_{width})) \tag{2.24}$$

$$\varphi = \int R\,(V_{rover} / (TR + \tfrac{1}{2}\,WB_{width})) \tag{2.25}$$

The instantaneous position vector of the rover was calculated by simply integrating the known velocity vector. Recall from Chapter I that the velocity vector could be placed anywhere within the world frame as long as the magnitude and orientation with respect to the world reference frame did not change. Since the dynamics model continually determined the rover' directional heading and linear velocity with respect to the world frame, the velocity vector signal was continually integrated to generate the rover's instantaneous position vector. With the velocity vector graphically defined in Figure 1.2 the velocity vector's component (X, Y, and Z) were calculated using equations (2.26 – 2.28).

$$X_{rover} = \int V_{rover} \cos(\alpha) \cos(\varphi) \tag{2.26}$$

$$Y_{rover} = \int V_{rover} \cos(\alpha) \sin(\varphi) \tag{2.27}$$

$$Z_{rover} = \int V_{rover} \sin(\alpha) \tag{2.28}$$

Navigation System

The proposed navigation model created in Simulink controlled the horizontal position and directional heading of the rover. The model shown in Figures C.1 and C.2 of Appendix C consisted of two subsystems. One subsystem served as the controller while the other served as the central processing unit.

The controller subsystem used proportional, integral and derivative (PID) control to condition and process the error signals in order to manipulate the directional heading of the rover. The error signals (e'_d and e_d) were determined by the difference between the rover's desired and actual values of angular velocity and directional heading. The actual change to the directional heading occurred after the navigation controller fed the conditioned and processed error signal to the steering system. The conditioned and processed error signal described the needed change to the directional heading. Since the directional heading is a function of the angular position of the rover's front wheels, the directional heading was changed simply by turning the front wheels. As the rover moved with a linear velocity, the front wheels were turned based upon the value of the conditioned and processed error signal. The conditioned and processed error signal was fed into the steering system as the desired front wheels' angular position (Θ_{sd}). The change in angular position of the front wheels in conjunction with the linear velocity induced an angular velocity to the rover. The rover's front wheels were continually turned by the navigational controller until the rover's directional heading maintained the desired value.

The PID controller conditioned and processed the error signals of the rover's directional heading and angular rotation rate as shown in equation (2.29). As can be seen in equation (2.29), the rover's angular velocity error (e'_d) was multiplied by a gain value (K_nv) while the directional heading error (e_d) was multiplied by a gain value (K_np). In addition and separately, the directional heading error was integrated and multiplied by another gain (K_ni). The results of these operations were added together to define conditioned and processed error signal.

$$e'_d K_n v + e_d K_n p + K_n i \int e_d = \Theta_{sd} \quad (2.29)$$

The central processing unit provided the desired directional heading to the navigational controller. The desired directional heading was determined from the x and y world frame coordinates entered into the navigation plan. It was calculated by taking the inverse tangent of the dividend produced when the y coordinate was divided by the x coordinate. The resulting angle value from the trigonometric function was sent to the navigational controller as the desired directional heading.

The navigation plan consisted of four coordinate sets. Each coordinate set was entered and stored in the model as a sequential order of planned navigation points. During the simulation the rover would drive to each set of coordinates in the predefined order. When the rover's position was within ten centimeters of the desired navigation point the central processing unit would switch the desired directional heading to the next navigation point. Once the forth navigation point was reached the simulation would automatically stop.

Inertial Measurement Unit

An inertial measurement unit (IMU) is a device that is used to measure angular and linear acceleration through the use of accelerometers. An IMU has the ability to measure linear acceleration along its three major axes and angular acceleration about those axes. The IMU is also able to integrate the measured acceleration signals twice to establish its position relative to a reference frame. When the IMU is first activated an initial position of the unit must be entered. Once the initial position data is entered into

the unit, the unit will continually determine its location as it is moved about the reference frame. Inevitably there are measurement errors associated with the accelerometers. The measurement errors lead to an induced drift velocity which in turn provides an ever increasing positional error.

A simplified model of an IMU was created in Simulink to simulate the measurement of the rover's acceleration and to calculate its position. The three principal axes as well as the directional heading were taken into account. After the initial position was entered, the IMU model received the rover's linear and angular velocity signals from the dynamics system. Once the values were received a drift velocity was added to each signal. The model added a total linear drift velocity of three tenths of a nautical mile per hour to the rover's velocity. The total drift velocity was distributed across all three of its axes. In addition the model added one tenth of a degree per hour of angular drift to the rover's angular velocity. Once the drift was added to the actual velocity values, the signals were integrated to determine the rover's position. The position signals were fed into the navigation system as feedback of the rover's determined position. The IMU model can be viewed in Figure C.11 of Appendix C.

Simulations

Lunar Reconnaissance Orbiter Scan

The LRO scan simulation emulated a lunar surface mapping mission. During the simulation the LRO flew above the modeled lunar surface and scanned the terrain below. While emulating the mission, the simulated LRO generated a three dimensional surface

map of the modeled lunar surface. The generated surface map was comprised of an array of three dimensional pixels. Each pixel represented one square meter of the lunar surface. Each pixel's height dimension represented that square meter's elevation. The generated model was accurate to within one half meter for each square meter of horizontal surface.

The LRO scan simulation had two purposes. The first purpose was to generate realistic data in order for future mission controllers to become familiar with the nature of the data a real LRO would generate. The second purpose was to produce a surface map that would be used to identify distinguishing terrain features and assign coordinates to those terrain features. Once the LRO generated lunar surface map had coordinates assigned to its terrain features, it was then used in the next simulation to determine the lunar rover's position. The LISP file used to create the ROBOSIM simulation of the LRO scan is listed in Appendix B.

To start the simulation the lunar surface model, the LRO model and the invisible Cartesian robot were loaded into the simulation environment. The invisible Cartesian robot was used to grasp and move the LRO about the simulation environment. With the use of the robot's axes the LRO flight was graphically simulated. In addition the joint value of each of the robot's axes was used to determine the position of the LRO and the laser beam.

As the LRO moved above each square meter of lunar surface, it emitted a simulated laser beam. The laser beam was used to measure the distance to the lunar surface below. The simulated laser beam was modeled as a single photon of light. The modeled photon was in the form of a cube with its frame origin at the center of the cube.

The photon's two surfaces that were parallel to the lunar surface were each one square meter in area while the height of the four surfaces perpendicular to the lunar surface was one meter in length. The laser beam's photon originated at the base frame of the LRO. As the simulation progressed the photon was indexed further away from the LRO and toward the lunar surface. The positive z axis of the LRO's frame always remained perpendicular to the horizontal lunar surface and pointing down toward it. Each time the photon was repositioned it was along the LRO's positive z axis. Once the photon was repositioned it was checked to see if it collided with the lunar surface. ROBOSIM has a collision detection heuristic built into its software. The heuristic compared the two objects in the simulation environment to see if their surfaces collided. If the photon collided with the lunar surface, the distance from the LRO's frame to the photon's frame was stored and used to generate a map pixel. The one half meter of elevation accuracy for each square meter of horizontal lunar surface was achieved since the photon's frame was at the center of the cube. As previously mentioned the two surfaces of the photon that were parallel to the lunar surface were one square meter in size. This along with the fact that the photon's frame origin was one half meter from either of the two surfaces produced the desired accuracy when the photon's position was indexed one meter, each time along the LRO's positive z axis.

To perform the scanning of the lunar surface by the means described above an algorithm was set up. The algorithm utilized three nested loops to perform the mapping. The first loop positioned the LRO with respect to the x axis of the world frame. The second loop positioned the LRO with respect to the y axis of the world frame. Since the

LRO was grasped by the invisible Cartesian robot, the algorithm was actually controlling the first two axes of the robot instead of directly positioning the LRO.

Once the LRO's position had been established above the surface, the third loop within the algorithm activated the laser beam. The laser beam's photon was indexed toward the lunar surface along the LRO's positive z axis. Each time the photon was indexed it was checked for a collision. If no collision was detected the photon was indexed one meter further away from the LRO. If no collision occurred after one hundred indexes the algorithm would exit the third loop and return to the second loop.

In the second loop the algorithm would index the position of the LRO one meter along the y coordinate. After the LRO's position along the y axis was indexed, the algorithm would once again enter the third loop and activate the laser beam. The algorithm then once again began to index the photon and check for a collision. If a collision between the laser beam's photon and the lunar surface was detected the algorithm would store the joint axis values of the Cartesian robot and the indexed position of the photon. Through the multiplication of transformation matrices that spatially described each axis of the Cartesian robot and the photon's position relative to the LRO, a generated pixel was placed in the simulation environment.

The pixel was assigned a color based upon its surface elevation. The pixel colors were comprised of various shades of red, green and blue. If the surface was detected to be four and half meters below the nominal flat surface the pixel was assigned the color red. If the surface was detected to be thirteen meters above the nominal flat surface the pixel was assign the color blue. All distance in between were proportionally assigned based upon this range.

After the pixel had been generated the algorithm would then exit the third loop and return to the second loop to index the LRO along the y axis of the world frame. The algorithm would then repeat the laser beam index and the collision detection process again. The algorithm continued to index along the y axis until it reached a value of seventy meters. At that point the algorithm would exit the second loop and return to the first loop. The first loop would index the LRO along the x axis of the world frame. After the LRO was positioned in the new location the algorithm would then return and process through the second and third loops to continue the mapping. Once the first loop had moved the LRO seventy meters along the x axis of the world frame and the algorithm completed its processing through the interior loops, the LRO generated lunar surface model was complete.

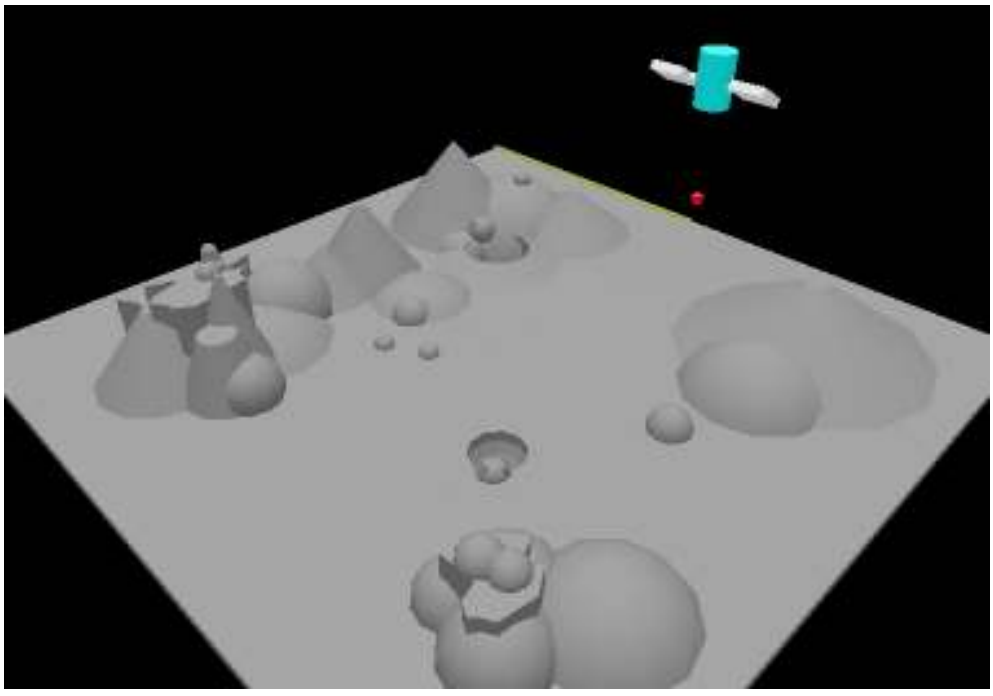

Figure 2.4: The lunar surface being scanned by the LRO.

Figure 2.4 shows the LRO as it scanned the lunar surface and generated surface map pixels. The laser beam's photon can be seen as it indexed towards the lunar surface. The red cube below the LRO is the laser beam's photon.

After the LRO had generated the map of the lunar surface, fifteen terrain points were visually identified on the map. These fifteen points were assigned coordinates relative to the world frame. The coordinates were obtained by using the invisible Cartesian robot to position small markers on the map's surface. As each marker was placed on top of an identifiable pixel, the robot's joint axis values were recorded. The values for the first three axes provided the position of the terrain point.

Position Determination

Using ROBOSIM a position determination simulation was developed to evaluate how accurate a lunar rover's position could be determined by measuring its distance to terrain features of known location. The LISP file used to create the simulation is listed in Appendix B. In this position determination simulation the method of three dimensional trilateration was used to calculate the lunar rover's position. The simulated rover used a laser range finder to measure its distance to the terrain features. Two constraints were applied when selecting terrain features to range. Each terrain feature had to be identifiable from the rover's perspective view. In addition each terrain feature must have been mapped and assigned coordinates from the previous LRO mapping simulation. This criterion was used because it simulated the circumstances a real lunar rover under supervised control by a remote operator would encounter when its position has to be determined. An Earth based operator would only be able to view the lunar surface

through the rover's camera. In addition the Earth based operator would have to identify on the LRO generated map, the lunar terrain features viewed through the rover's camera.

To begin the simulation the lunar surface and the rover were placed in the simulation environment. The lunar rover was positioned on the surface model. Its actual x, y, z coordinate position in meters was (0, 0, 30). A coordinate position of (0, 0, 30) meters placed the robot at the center of the horizontal lunar surface. Its elevation relative to the world frame was thirty meters along the positive z axis.

Three identifiable terrain features were chosen as points to measure. The terrain features' coordinates were retrieved from the LRO generated map and placed into the position determination algorithm. With the terrain features as the targets, the turret on the rover rotated and raised the laser so that it pointed at the first target point.

An algorithm very similar to the LRO's laser beam was used to generate and position the rover's laser beam. The photon from the laser was modeled as a cylinder that was two centimeters in diameter and twenty centimeters long. The photon's frame origin was at the center of the cylinder. By placing the frame's origin at the center of the modeled photon, the laser beam's measuring accuracy was set to plus or minus ten centimeters. To begin the distance measuring algorithm the laser beam's photon was placed at the base frame of the laser. The photon was then moved twenty centimeters away from the laser's base frame and toward the target. The photon moved along the direction of the modeled laser's z axis. The photon was oriented with the laser's z axis centered lengthwise through the twenty centimeter long cylinder. Once the photon was positioned it was checked for a collision with the lunar surface. If no collision was detected the photon was indexed another twenty centimeters toward the target. When a

collision was finally detected the distance between the base frame of the laser and the photon's frame was stored. Since the distance between the base of the rover and the terrain feature was needed, the simulation used the joint axis values of the turret and the photon's distance from the laser's base frame to calculate the net distance between the rover's base and the targeted terrain point. The laser measuring process was then repeated for two additional targets. Figure 2.5 shows the lunar rover measuring its distance to an identifiable target.

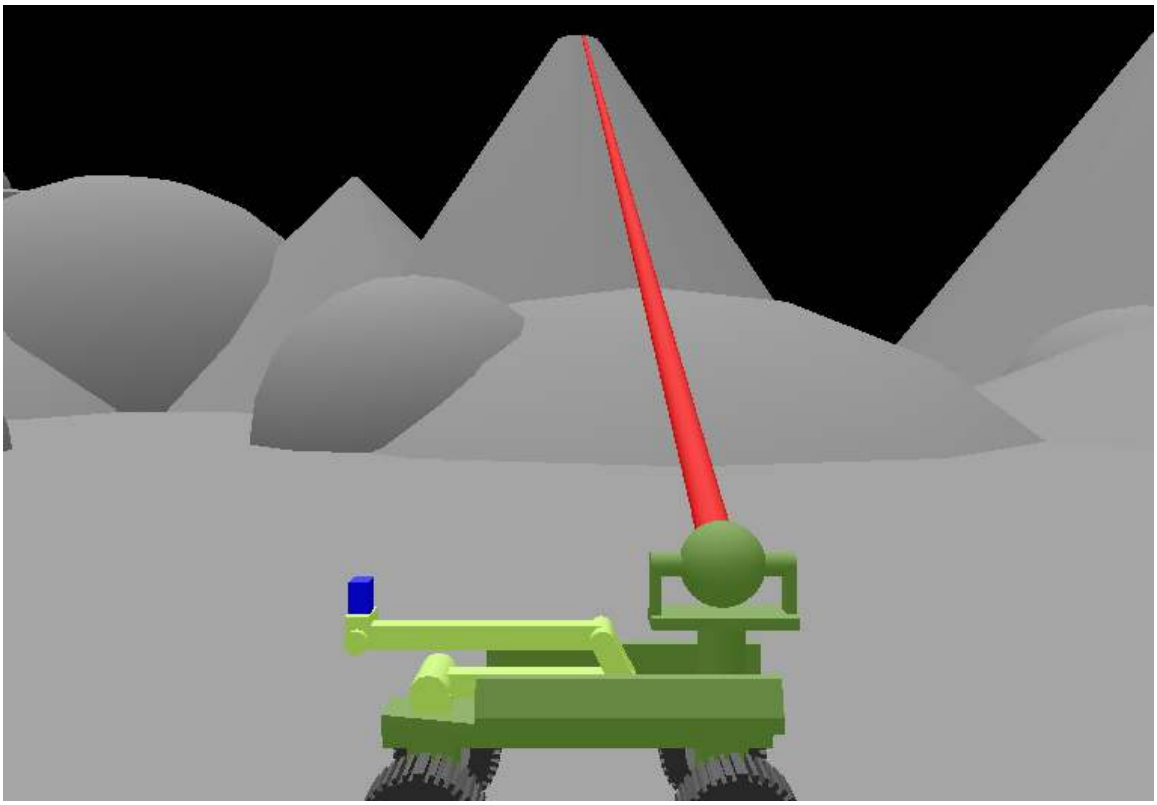

Figure 2.5: The lunar rover measuring its distance to a terrain feature.

After the distance to all three terrain points was measured, the simulation calculated the rover's position relative to those points using the three dimensional

trilateration method descried in Chapter I. The method translated and rotated the reference points until all three points were located on the world frame's x – y coordinate plane. The first reference point was translated to (0, 0) and the second reference point was positioned on the positive x axis. During all of the translations and rotations the position of the reference points relative to each other never changed. Once the reference points were manipulated to the described specifications, the corresponding position of the rover was calculated by solving for the intersection point of the three imaginary spheres that were created by the reference points and the distances to them. The rover's actual position relative to the world frame was finally determined by taking the prior calculated position and reversing the translations and rotations.

Dynamics and Navigation

To analyze the effects of the errors in the position determination simulation, a dynamics and navigation simulation was created. The simulation combined together the Simulink models of the rover's propulsion system, steering system, IMU system, dynamics system, and navigation system. Input and output signals from the models worked in conjunction to create a dynamic simulation of the rover's navigation about the lunar surface.

Before the simulation began a set of four predefined coordinates were entered into the navigation system. The four sets of coordinates established the planned path the rover was to travel. Next the calculated position of the rover was entered into the modeled IMU. The entered data was the rover's calculated position from the prior position determination simulation.

Once the simulation began the propulsion system provided a velocity signal to the dynamics system while the steering system provided a signal describing the front wheels' angular position. The rover's linear velocity propelled the rover forward while the front wheels turned the rover in order to change its directional heading. The position of the rover's front wheels was changed based upon an input signal from the navigation system. The navigation system controlled the directional heading of the rover. As the rover started moving forward the navigation controller would send a signal to the steering system commanding the front wheels to turn. The front wheels would turn the rover until the desired directional heading was maintained. Once the rover reached the first programmed position point, the navigation system then began to drive the rover to its next desired position point. The process continued until the rover reached its fourth programmed point.

The resulting dynamics of the rover were graphically displayed both in Simulink and ROBOSIM. Within the ROBOSIM environment the lunar rover was traversed about the lunar surface with the use of the invisible Cartesian robot. Output signals of position and directional heading from the dynamics system within Simulink were imported into the ROBOSIM environment as the joint axis values of the invisible Cartesian robot. The invisible Cartesian robot grasped the lunar rover and positioned it according to the signals. This provided a graphical animation of the lunar rover's motion. In addition plots of the rover's true position and the rover's measured position were generated within the Simulink environment.

CHAPTER III

RESULTS

Lunar Surface Mapping

As the LRO scan simulation was conducted a surface map of the modeled lunar surface was generated. Figure 3.1 shows the generated map from the LRO scan simulation. As can be seen the map was constructed of three dimensional pixels.

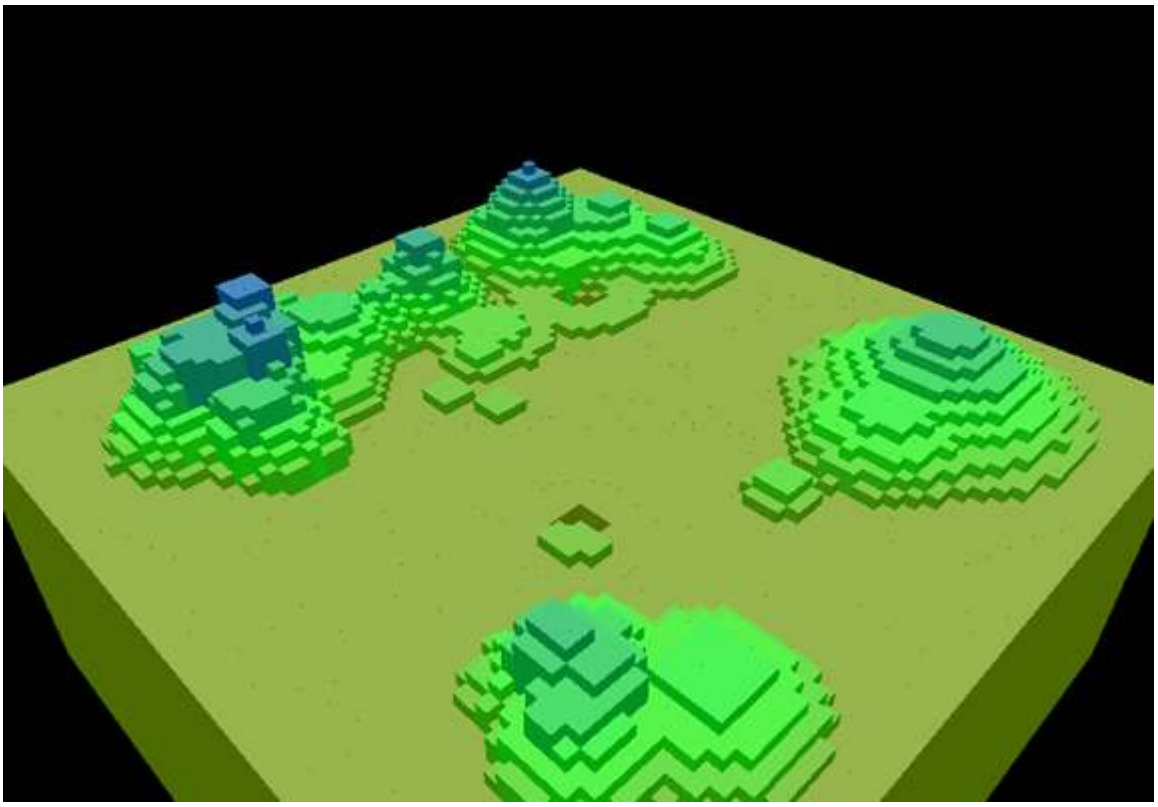

Figure 3.1: The LRO generated map of the lunar surface.

The map's main purpose was to provide reference points so that the lunar rover's position could be determined in the following simulations. A secondary purpose of the map was to provide a model that could be used to prepare for future lunar missions. The map could be used by mission planners to comprehend how the data from a lunar mapping mission might appear.

Upon observation of the map shown in Figure 3.1, prominent terrain features were distinguishable. Features that rose above the flat surface and reached a distinct point were very identifiable. For example, the four peaks in each of the corners of the map were clearly visible. The peaks' color assignments of green and blue clearly distinguished them from the lime colored flat surface that predominately filled the map. In addition, the opposite elevation extremes that were associated with the craters were easily detectable on the map. As the surface level dropped below the flat surface, reddish pixels appeared. These reddish pixels indicated a lower surface elevation of approximately eighteen meters when compared to the blue pixels that represented the elevated surface peaks.

Figure 3.2 provided another view of the LRO generated map. This view is similar to traditional satellite reconnaissance images. If the technology that creates a three dimensional pixel were not available, the map would have to be evaluated in the form similar to what is shown in Figure 3.2. From this view the pixel colors became very important when evaluating elevation changes in the lunar surface.

From the view above, rapid elevation changes were very distinguishable. With the rapid elevation change the color among nearby pixels did not blend together. The jagged rock formation in the lower right corner of the map provided a good example.

The sharp change in color with nearby pixels indicated a significant change in elevation. From the lunar model itself, this jagged rock formation had vertical walls and rose sharply from the lunar floor. On the other hand, the pixels representing the rolling hillside in the upper left corner of Figure 3.2 tended to blend together. The hillside could be distinguished but it was not as prominent as the jagged rocks.

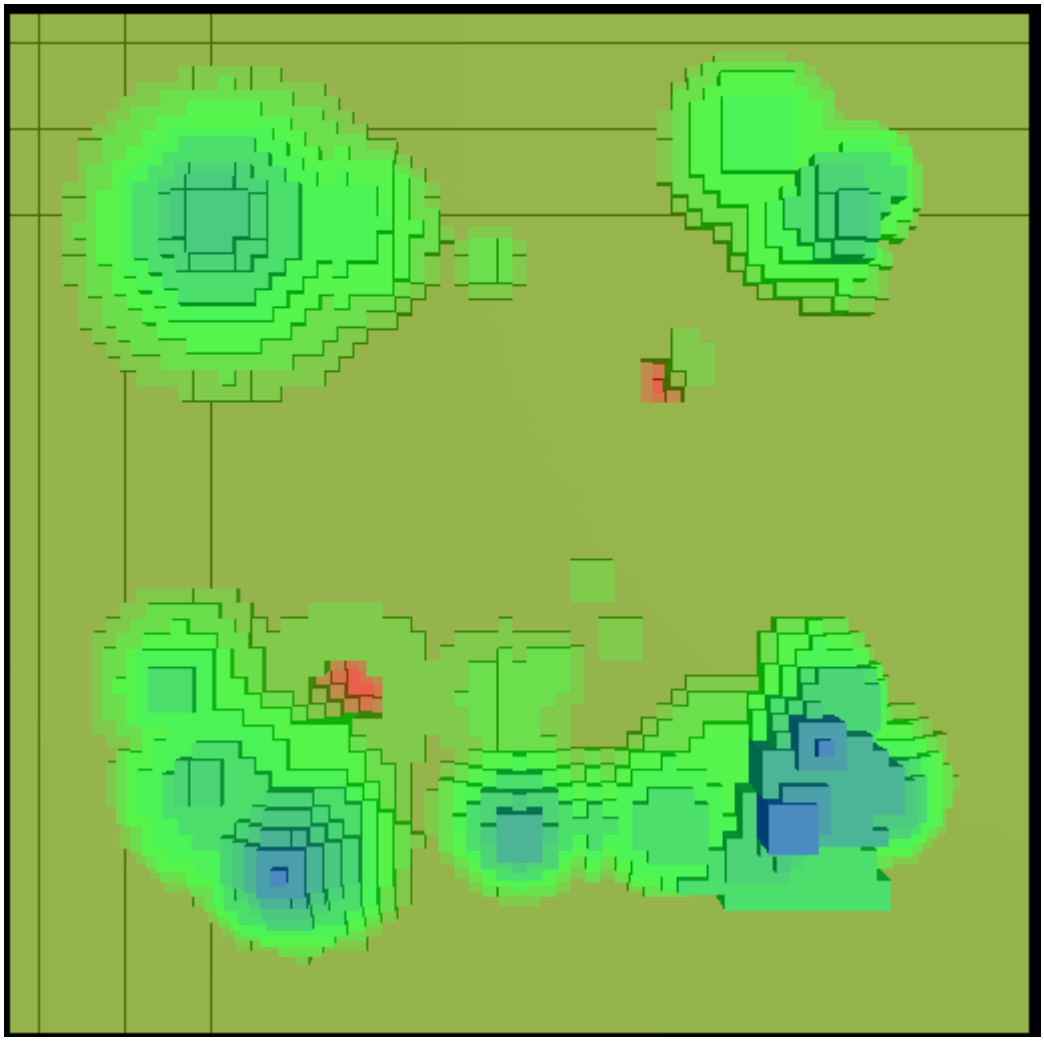

Figure 3.2: The LRO generated model of the lunar surface as viewed from above.

Small rocks a half meter or greater in elevation and that were isolated from other terrain features were also noticeable. However rocks that were joined with other features were undetectable. For example the small rocks located at the edge of the crater shown in the lower left corner of Figure 3.2 blended in with the hillside.

Since the map was generated with a limited accuracy of one half meter in elevation for each square meter of horizontal surface area, a comparison between the model and the map was conducted. The results can be viewed in Figure 3.3.

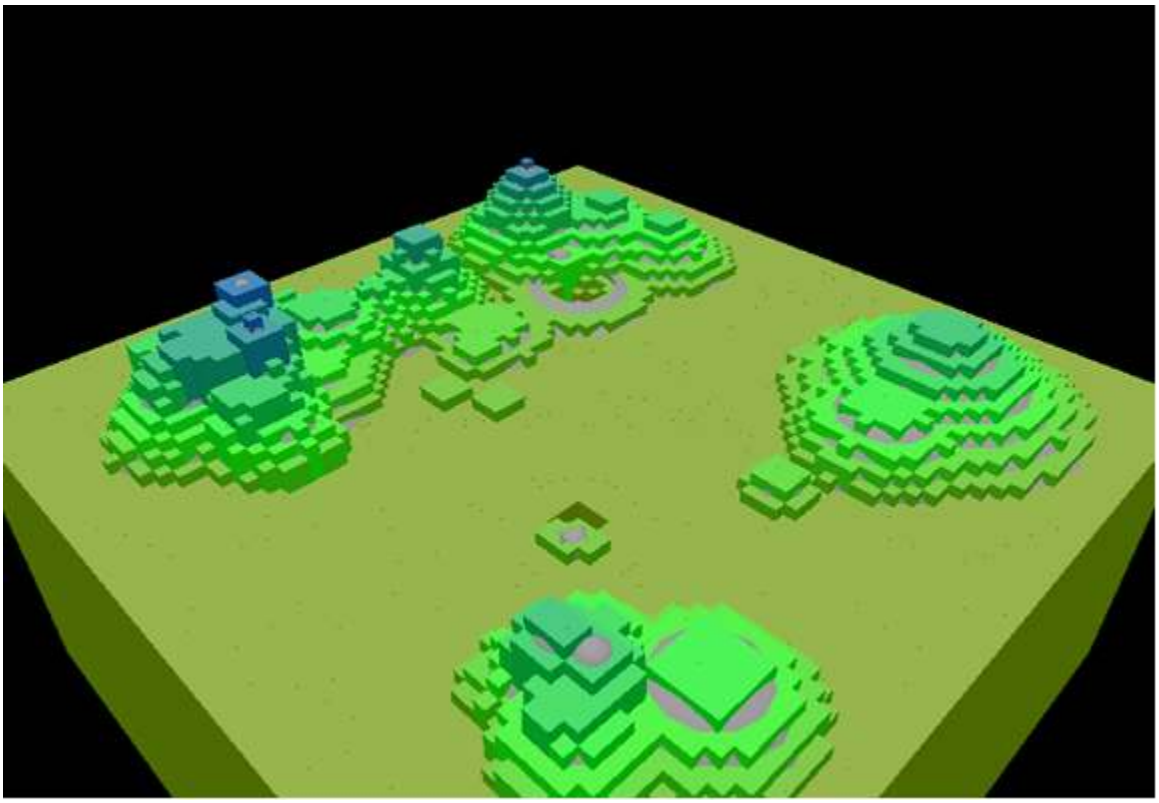

Figure 3.3: The LRO generated data superimposed over the lunar surface model.

Figure 3.3 shows the map superimposed over the lunar surface model. This comparison revealed the inaccuracies of the map. The most noticeable of these errors

was the surfaces that were curved. Small portions of the hillsides and crater regions protruded past the map pixels. This was due to the limited horizontal resolution of the mapping. In addition the limited vertical resolution was evident by the exposure of the top surfaces of rocks when the map was superimposed over the lunar surface model.

After the lunar surface map was generated, distinguishable terrain features were identified as points and assigned coordinates. The points were determined based upon two criteria. The first criterion was the points had to be distinguishable on the map. Second, the points had to be visible from the rover's view point. These two conditions were required to maintain the simulation's authenticity of a rover that was under supervised control by a remote operator.

A total of fifteen points, (P1 - P15) were chosen on the map. The points were the tops of rocks, boulders and hillsides since these were the easiest targets to identify from the map and from the rovers line of sight. A list of the points and their assigned coordinates are contained in Table 3.1. Figures 3.4 and 3.5 also show the points with their coordinates except they are illustrated on the lunar surface map.

Table 3.1: Identified mapped lunar terrain features.

Feature	X (m)	Y (m)	Z (m)
P1	12.0	11.5	31.0
P2	21.5	20.0	37.0
P3	16.0	25.5	34.0
P4	-2.0	18.0	32.0
P5	-20.0	20.0	37.0
P6	-23.0	-11.0	35.0
P7	-21.0	-17.0	36.0
P8	-15.0	-22.0	40.0
P9	-13.0	-15.0	33.0
P10	0.0	-20.0	38.0
P11	5.0	-4.0	31.0
P12	7.0	-8.0	31.0
P13	17.0	-19.0	40.0
P14	18.0	-17.0	39.0
P15	19.0	-14.0	40.0

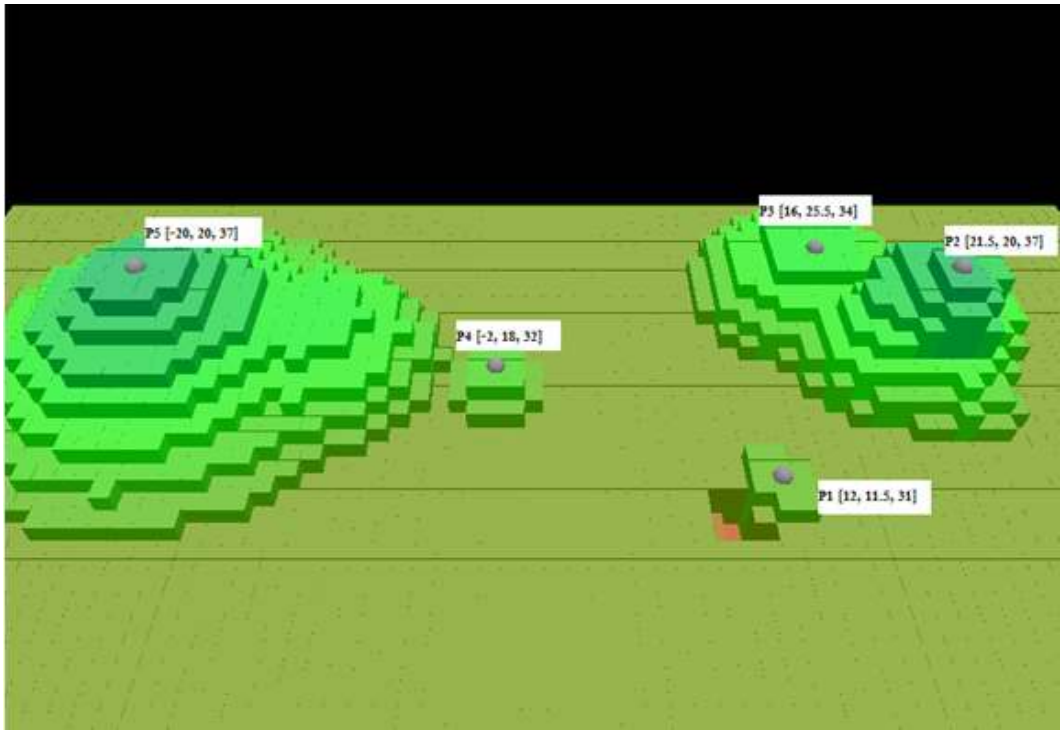

Figure 3.4: Mapped lunar terrain points P1 through P5.

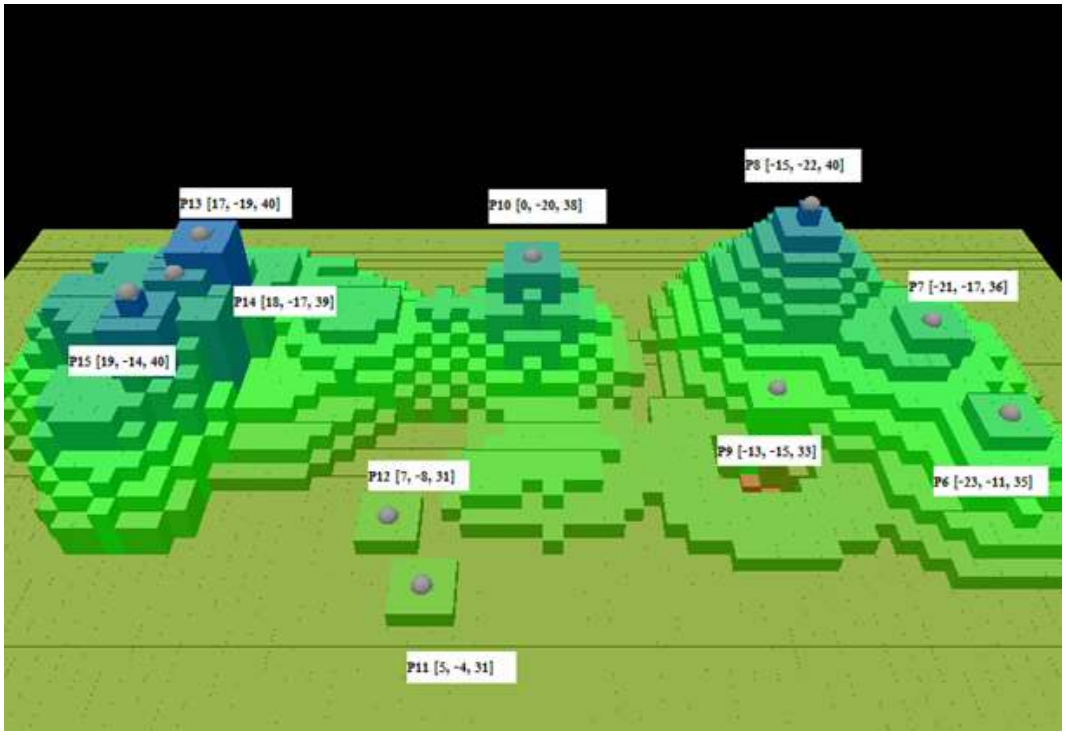

Figure 3.5: Mapped lunar terrain points P6 through P15

Lunar Rover's Position Determination

The position determination simulation was executed ten times to obtain the average position of the lunar rover. Each time the simulation was conducted a different set of terrain points was used. The experimentally obtained results from the ten simulations are shown in Table 3.2.

Table 3.2: Experimental results from the position determination simulations

Test Number	Distance to LRO Scan Targets			Calculated Position (m)			Total Positioning Error (m)
	Target One Distance (m)	Target Two Distance (m)	Target Three Distance (m)	X	Y	Z	
0	Point1	Point2	Point3	0	0	30	0
	28.83	25.31	28.14				
1	P2	P15	P8	-0.33	1.35	34.39	4.6
	28.83	25.31	28.14				
2	P3	P13	P6	-0.31	1.68	35.55	5.81
	28.46	28.86	25.5				
3	P1	P14	P7	-1.03	0.76	32.84	3.11
	16.2	26.25	26.42				
4	P1	P11	P12	0.87	-0.13	29.15	1.23
	16.2	5.95	10.14				
5	P4	P12	P9	-0.05	0.38	31.74	1.78
	17.23	10.14	19.71				
6	P5	P15	P10	-0.53	0.21	32.44	2.51
	28.13	25.31	20.97				
7	P5	P1	P8	0.08	0.71	33.02	3.1
	28.13	16.2	28.14				
8	P4	P13	P8	0.08	1.04	34.19	4.32
	17.23	28.86	28.14				
9	P15	P11	P12	1.79	0.64	28.47	2.44
	25.31	5.95	10.14				
10	P2	P15	P1	-0.24	1.31	33.98	4.19
	28.83	25.31	16.2				
			Average	**0.03**	**0.80**	**32.58**	**3.31**
			True Position	**0**	**0**	**30**	

Before the first test was conducted the position determination simulation with its algorithms was tested to validate it was properly programmed. The program was validated by the control test number 0. Test number 0 used the exact position of the photon as the reference point's coordinates each time the photon collided with the lunar

terrain. By using this technique the returned position of the rover would match its true position since there were no errors in the reference coordinates. The rover's laser beam was targeted toward points P2, P15 and P8. Once the laser's photon collided with those points the actual position of the photon's frame was used as the reference points. The photon was positioned at point1, point2 and point3 when it collided with the surface. This is evidenced by the fact that distance measurements from test number 0 and test number 1 were the same for the two sets of reference points. As can be seen by the results of test number 0, the rover's position was determined to be its true position. Thus the program was verified to be correct.

After ten simulations with various combinations of three reference points the average position of the rover was calculated. The results show the lunar rover's calculated average position in meters with respect to the world frame was (0.03, 0.80, 32.58). When compared to the rover's true position of (0, 0, 30) the rover had 3.31 meters of total position error. However if just the horizontal coordinates of x and y were evaluated the rover's calculated position was quite accurate. By examining the method of three dimensional trilateration presented in Chapter I, the improved accuracy in the horizontal coordinates can be explained. The first two coordinates to be solved were the x and y coordinates. Both of these values had some error. Their errors carried over to the solution of the z coordinate because the calculated values for the x and y coordinates were entered into the equation to calculate the z coordinate.

Test number 4 and number 5 produced the best results. These tests used the small rocks as their reference points. The rocks that are labeled as points P1, P11 and P12 were used for test number one. The points P4, P12 and P9 were used for test number 2. The

terrain points had three common characteristics that the other terrain points did not have. All of the points were small rocks two to three meters in diameter. Each one of the rocks was approximately the same elevation as the rover. Thus the tops of their surfaces where the LRO scanned were sensed by the rover's laser beam. Lastly the rocks were closer than the other terrain points.

Test number 1 and number 2 produced the most error. The terrain points referenced were P2, P8 and P15 for test number 1. Test number 2 referenced points P3, P13 and P6. Each of the points had three common characteristics. Each of the points was elevated four to ten meters above the rover. Each point resided at the top of a boulder or rock formation where the rover's laser could not sense the surface that was mapped by the LRO. Last, each one of the points was twenty five meters or further away from the rover.

Lunar Rover's Dynamics and Navigation

The dynamics and navigation simulation demonstrated, with animated graphical output, the effect of the position determination error. The average value obtained from the position determination simulations was entered into the rover's IMU as its starting position. In addition to the initial position error, the IMU was modeled with a typical drift velocity that is inherent to any IMU. When all of these errors were taken into account the rover's performance became limited. Since the errors continually grew the accuracy in the measured position steadily decayed over time.

To validate the simulation and to verify the models were setup correctly a control test was performed. The test consisted of navigating the rover to a set of sequenced

horizontal coordinates with no initial position error or IMU error. The results of this control test are shown in Figure 3.6. The rover started at position (0, 0) and then drove to (20, 10). Once at this position the rover changed its directional heading and proceeded to point (10, 20). Once it reached the third program point the rover turned again and returned to its starting position. As can be seen by the graph, the simulation and its models were validated.

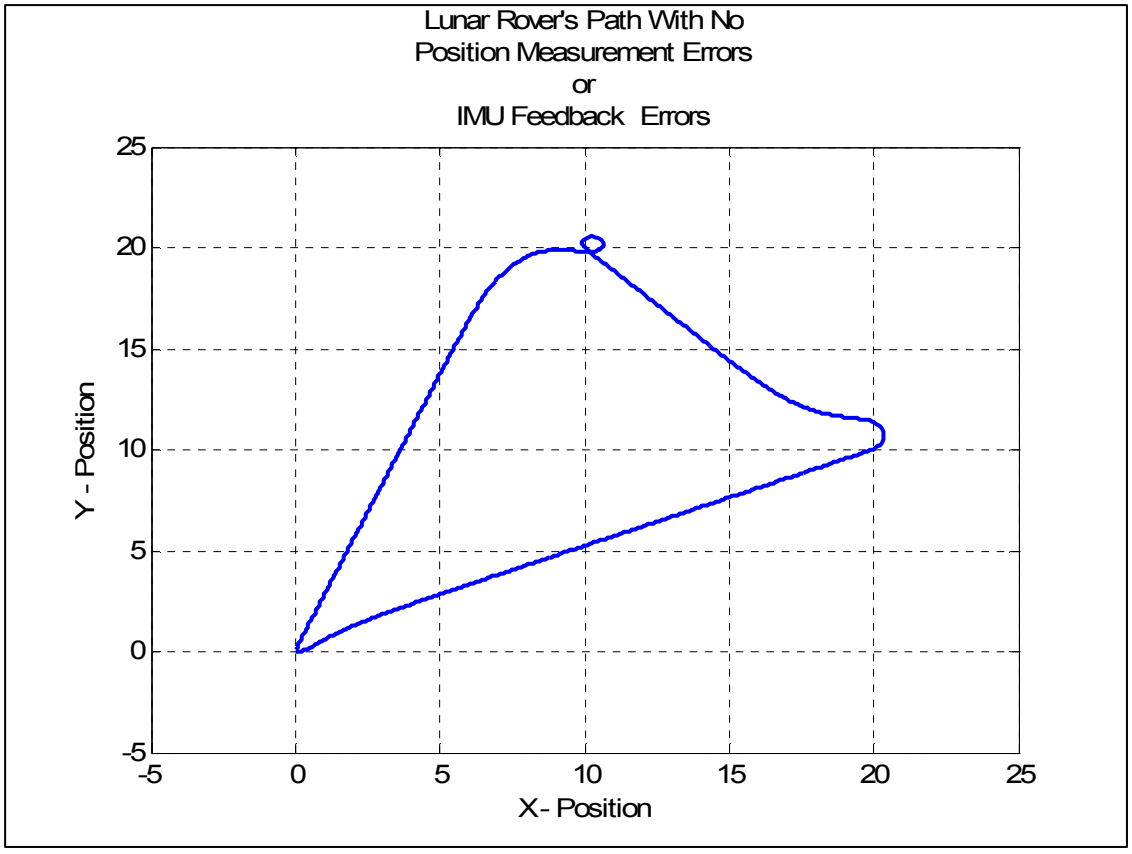

Figure 3.6: The lunar rover's path with no measurement errors

The limited performance began with the error associated with the initial position determination. To comprehend this error test number 1 of the simulation was run with the starting position of the rover at (0.03, 0.80). This position was entered into the IMU

but the true starting position of the rover was at (0, 0). The sequenced navigation points were the same as those of the previous test simulation. The results of the simulation are shown in Figure 3.7.

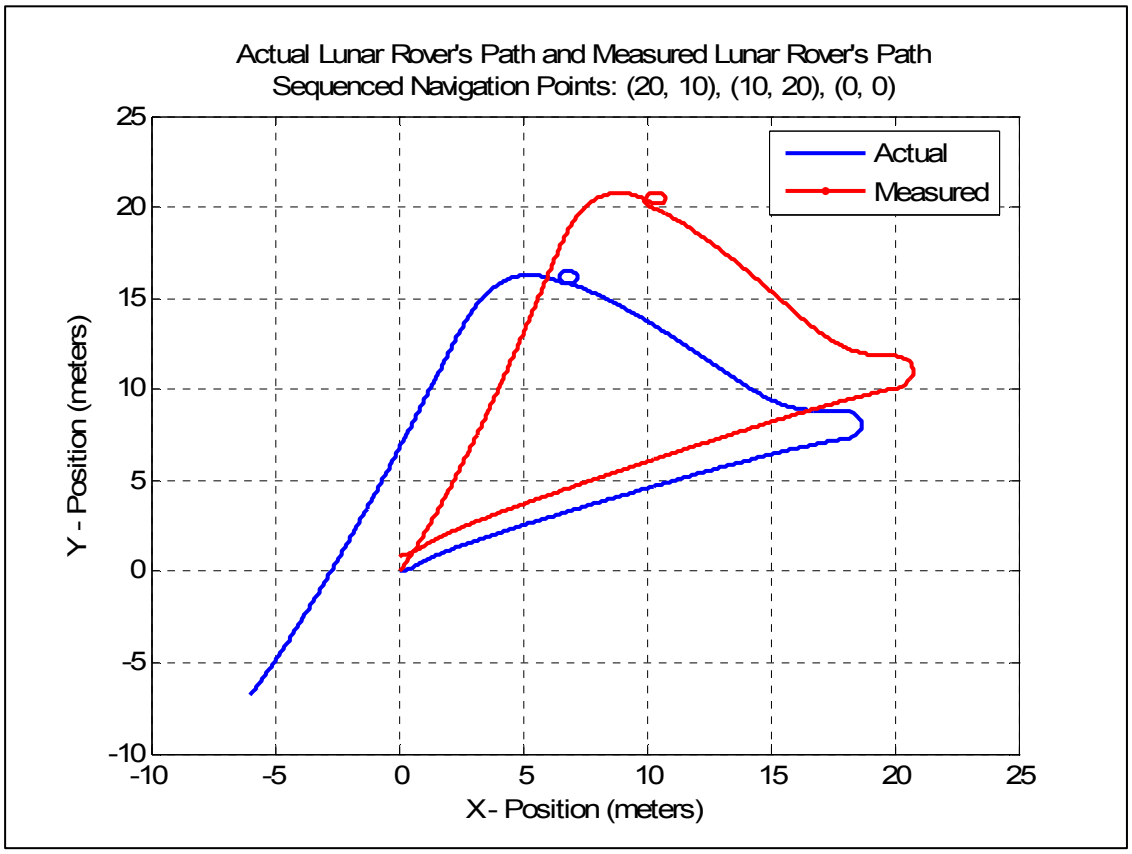

Figure 3.7: The lunar rover's path for test number 1.

As shown in Figure 3.7 the error between the rovers's actual position and its measured position steadily grew as the rover operated. Feedback to the controller via the IMU produced data that led the controller to manipulate the rover off its intended path. As can be seen by the graph, the rover never made it to any of its programmed points. In addition to not reaching its desired points the rover collided with the rock located near the

coordinates (12, 12). Figure 3.8 shows the graphical output of the rover navigating past the rock in the controlled test but colliding with it in test number 1.

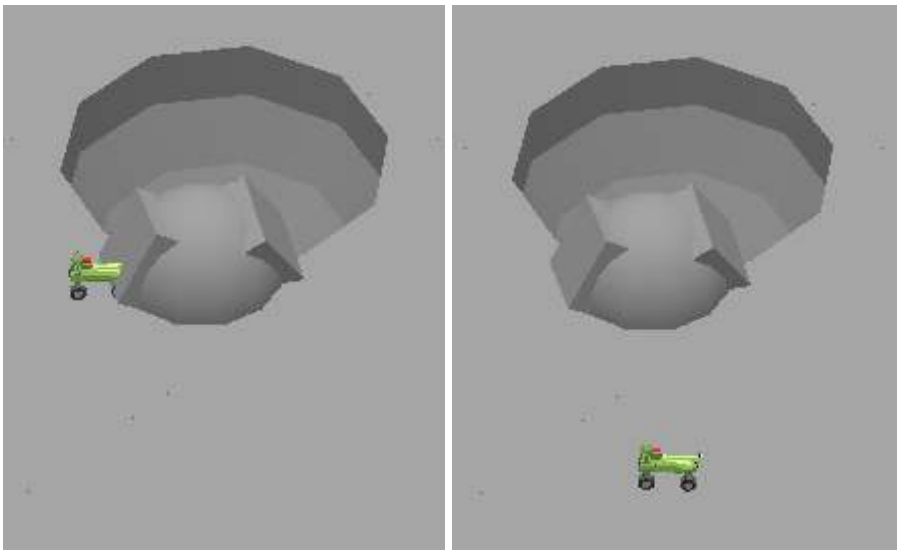

Figure 3.8: Rover navigation near a rock with a collision and then without a collision.

To further evaluate the effect of the errors a second test was run with a different set of navigation points. The rover once again started at (0.03 0.80) but it drove to points: (0 -10), (-12, 10) and (0, 0). Once the rover reached its final destination it was approximately seven meters from its intended location. The results of test number 2 are shown in Figure 3.9. As can be seen in Figure 3.9 the position error of the rover steady increased as the simulation progressed.

The results of the tests showed the position determination and IMU errors will drastically alter the rover performance. If the rover's IMU is not recalibrated regularly on a frequent schedule, the errors will continue to grow. These results also demonstrated the usefulness of the produced simulation. The graphical nature of the results produced easily comprehendible pictures. The simulation was constructed as a platform for

research and mission development that can easily be analyzed with an accurate lunar surface model.

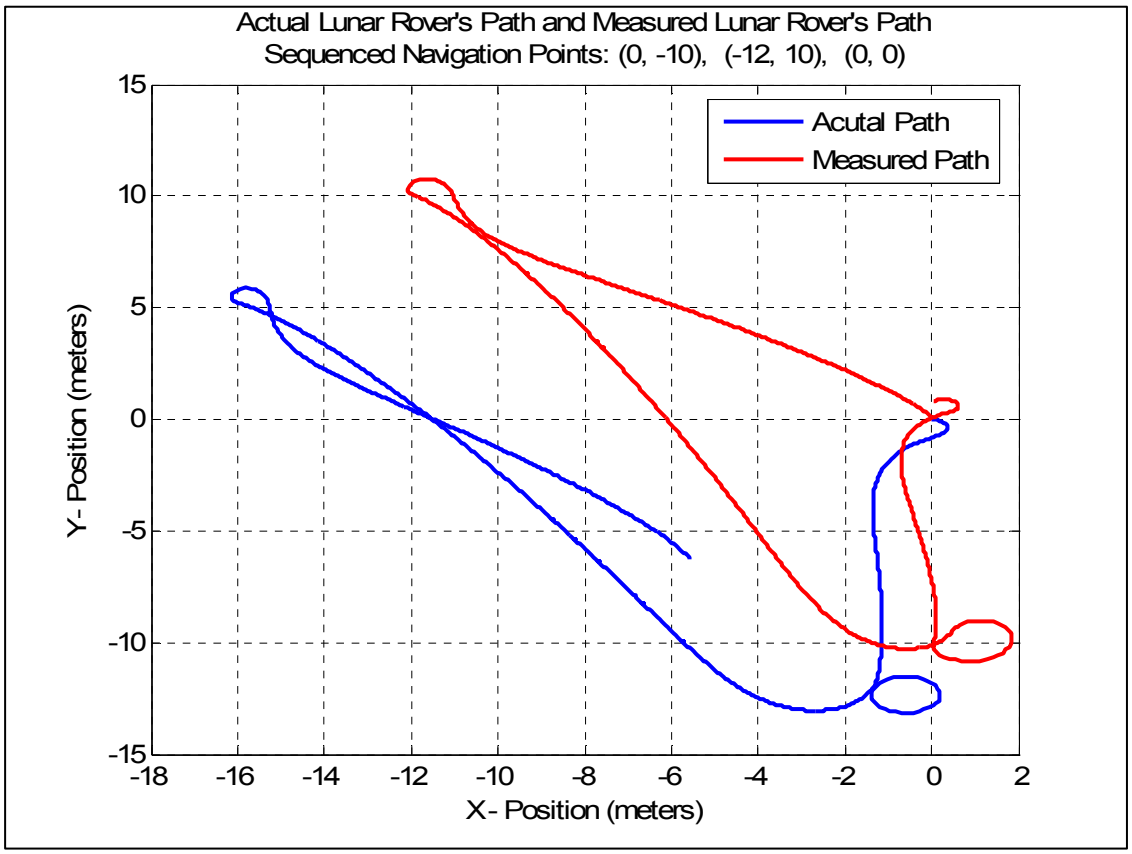

Figure 3.9: The lunar rover's path for test number 2

CHAPTER IV

CONCLUSIONS AND FUTURE EFFORTS

Position Determination

If only the relative position of the rover is needed, the use of a laser range finder to measure its distance to terrain points of known location does lead to an adequate position determination. With an average error position of 3.31 meters, the rover would be limited in the type of tasks it could perform. With this size of error, the rover would have to rely on the supervised control from a remote operator quite frequently for many tasks. Any autonomous task that requires knowledge of an accurate position, such as precise navigation, map generation, rock retrieval or assembly tasks associated with the construction of a lunar base would not be possible without the assistance of supervised control. On the other hand if the rover's mission was exploratory in nature with tasks such as the photographic reconnaissance of certain regions of the lunar surface, position accuracy would not be as crucial. Exploration, mapping and general surveying of the lunar surface with this limited accuracy could be accomplished using prior mapped terrain features as the points of reference.

As a potential application, this position determination method could be used for the placement of positioning beacons. Once a rover has placed several positioning beacons on the lunar surface, other rovers would not have to rely on supervised control to establish their position. They would simply receive positional data from the previously placed beacons. With the positioning beacons, the rovers would be able to operate

autonomously without the need to reestablish their position relative to the local terrain. This position determination method could be used on the lunar surface with the SCPA navigation system presented in Chapter I.

When compared to the accuracy of GPS used here on Earth, the position determination results are not much different. According to information posted on the National Space Based Positioning, Navigation, and Timing Executive Committee's web site (www.pnt.gov), the GPS signal available for civilian use is guaranteed to provide a horizontal position accuracy of 36 meters, 95% of the time. Typically the results are better than 16 meters depending on the signal. The accuracy of GPS is improved significantly by augmenting the signal with a correction. The United States Coast Guard Navigation Center operates a maritime Differential GPS (DGPS) that provides correction data from land based broadcast sites. According to information posted on their web site (www.navcen.uscg.gov), the DGPS signal provides a 10 meter or better accuracy. Typically the signal provides a 1 to 3 meter positioning error. When the results listed in Table 3.2 are compared to the error of DGPS, it can be concluded that accurate positioning is possible by the presented method.

Using three dimensional trilateration with terrain features as the reference points to determine the position of a rover does provide some attractive benefits. One major benefit is the cost. The cost investment to design and build a laser range finder would be significantly cheaper when compared to systems similar to GPS. With a lunar GPS, several satellites would have to be placed in orbit around the moon. The cost to place and maintain these satellites would be enormous. However with a laser range finder there would be significantly less development cost and practically no maintenance. The only

significant development cost would be with the LRO used to map the lunar surface. Another benefit would be its reliability. Since the system only consists of a camera and a laser mounted on a turret, there would be fewer components that could malfunction. With the rover operating on the lunar surface it would need to be maintenance free since there would be nobody there to repair it. In addition the system would be independent of an array of orbiting satellites or positioning beacons.

The major drawback to using terrain points as the known reference points is that they might not always be present in the local area of operation. If the local lunar terrain is featureless then there would not be any targets available. Even with limited terrain features, they still would have to be identifiable on the LRO generated map as well as through the lens of the rover's camera.

Inertial Measurement Unit Modeling

Based upon the results from the dynamics and navigation simulation, the measurement errors that led to drift in the IMU, significantly degraded the rover's navigation performance. The drift of the IMU had more of an impact on the performance than the initial position error. If the rover was able to recalibrate its IMU on a frequent basis, the drift could be eliminated. With GPS and its ever present signal available near the Earth's surface, an IMU is able to recalibrate before the errors become too large. For a lunar rover, a similar GPS would probably not be available for many years.

Current research and development is being conducted in order to set up localized navigation beacons (LeMaster, E. and Rock, S., 2004). The beacons would be placed in an array were a rover could constantly receive positional data from the beacons. With

this type of system a rover could easily acquire the positioning accuracy to successfully operate in an autonomous mode. When compared to the position determination method presented in this thesis, the navigation beacons offer two major advantages. The beacon's location coordinates are very accurate and the signal to the rover is continuously transmitted.

Using a GPS to provide an updated positional reference has two drawbacks. In the case of a lunar positioning system it would be extremely expensive to build and operate. Secondly a GPS signal might not always be present due to blockage or maintenance issues with a satellite. An ideal positioning system to have would be one that did not rely on an external reference such as satellites or beacons.

Future efforts could be put forth to develop an IMU that did not rely on a GPS signal to recalibrate its position. An IMU has many attractive advantages that if harnessed would lead to great advancements in autonomous robot navigation. An IMU is immune to jamming and deception. It is completely independent of outside systems. However the open loop nature of the IMU compounds the measurement errors. The challenge would be to develop an integrated navigation system that used an IMU and other internal sensors onboard a rover to perform accurate navigation.

To aid in this development, the IMU model created for this thesis could be developed much further to better predict its measurement errors. The model could be studied to develop rover maneuvers that minimized the error of the IMU. Additionally the model could be coupled with other sensors to crosscheck the IMU's position determination and make the necessary corrections. The sensors could include drive shaft position sensors, wheel slippage sensors, or magnetic directional compasses. The goal

would be to develop an IMU that could operate for longer durations without the need to be recalibrated. In the case of a robotic lunar rover that was equipped with an IMU, it could drive to an area on the lunar surface where it could then measure its distance to terrain features and use those measurements to recalculate its position.

APPENDIX A

INVISIBLE CARTESIAN ROBOT INVERSE-KINEMATICS EQUATIONS DERIVATION

The inverse-kinematics equations determine a robot's joint axis values for a required end-effector frame position and orientation relative to a reference frame. For the invisible Cartesian robot the joint axis values are d_1, d_2, d_3, θ_4, θ_5 and θ_6. Each transformation from adjoining frames can be determined by using the data from the link parameter table given in Table 1.1 and applying those parameters to equation (A.1).

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.1)} $$

Since the overall transformation that gives the position and orientation of the frame for axis 6 relative to the reference frame is known, both sides of equation (A.2) are known.

$$ {}^{0}_{6}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{21} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(A.2)} $$

By inspection of equation (A.2) and knowing that the first three axes of the Cartesian robot position the end-effector, it can be determined that $d_1 = P_x$, $d_2 = P_y$ and $d_3 = P_z$. To solve for the angle values of the remaining axes we must expand equation A.2 to the form shown in equation (A.3).

$$ {}^{0}_{1}T \; {}^{1}_{2}T \; {}^{2}_{3}T \; {}^{3}_{4}T \; {}^{4}_{5}T \; {}^{5}_{6}T = {}^{0}_{6}T \qquad \text{(A.3)} $$

By rearranging equation (A.3) to the form of equation (A.4) the solution to θ_4 can be obtained from the equating elements of each side of the equation.

$$ {}^{1}_{2}T^{-1} \; {}^{0}_{1}T^{-1} \; {}^{0}_{6}T = {}^{2}_{3}T \; {}^{3}_{4}T \; {}^{4}_{5}T \; {}^{5}_{6}T \qquad \text{(A.4)} $$

The value of the element [1, 3] on the left side of the equation is the known quantity r_{33}. The value of the element [3, 3] on the left side of the equation is the known quantity r_{23}. The corresponding elements [1, 3] and [3, 3] on the right side of the equation contain the values of $\cos\theta_4 \sin\theta_5$ and $\sin\theta_4 \sin\theta_5$. The solution to θ_4 can be found by equating these two elements and getting the inverse tangent of r_{23} / r_{33}. Since the joint axis can have two solutions the coefficient wrist will defined as: wrist equals +1 for clockwise rotation and wrist equals -1 for counterclockwise rotation. The final solution to θ_4 is stated in equation (A.5).

$$ \theta_4 = \text{Atan2}(\text{wrist} \sin\theta_4 \sin\theta_5, \text{wrist} \cos\theta_4 \sin\theta_5) \text{ for } \theta_5 \neq 0 \text{ or } \theta_5 \neq 180 \qquad \text{(A.5)} $$

A singularity configuration results when θ_5 goes to 0 or 180 degrees. To avoid this problem whenever θ_5 gets within plus or minus ½ degrees of either 0 or 180 degrees, the value of θ_4 will be set and held at the corresponding value of θ_5.

The solution to θ_5 and θ_6 can be found in similar method as θ_4. The solution to θ_5 lies within equation (A.6). The [1, 3] and [3, 3] elements on each side of the equation can be equated as shown in equation (A.7) and (A.8). The solution to θ_5 is given in equation (A.9).

$$ {}^{3}_{4}T^{-1}\ {}^{2}_{3}T^{-1}\ {}^{1}_{2}T^{-1}\ {}^{0}_{1}T^{-1}\ {}^{0}_{6}T = {}^{4}_{5}T\ {}^{5}_{6}T \qquad \text{(A.6)} $$

$$ \sin\theta_5 = r_{33} \times \cos\theta_4 + r_{23} \times \sin\theta_4 \qquad \text{(A.7)} $$

$$ \cos\theta_5 = -r_{13} \qquad \text{(A.8)} $$

$$ \theta_5 = \text{Atan2}\ (\sin\theta_5, \cos\theta_5) \qquad \text{(A.9)} $$

The solution to θ_6 also lies within equation (A.6). The [2, 1] and [2, 2] elements on each side of the equation can be equated as shown in equations (A.10) and (A.11). The solution to θ_5 is given in equation (A.12).

$$ \sin\theta_6 = r_{21} \times \cos\theta_4 - r_{31} \times \sin\theta_4 \qquad \text{(A.10)} $$

$$ \cos\theta_6 = r_{22} \times \cos\theta_4 - r_{32} \times \sin\theta_4 \qquad \text{(A.11)} $$

$$ \theta_6 = \text{Atan2}\ (\sin\theta_6, \cos\theta_6) \qquad \text{(A.12)} $$

APPENDIX B

LISP FILES USED TO CREATE KEY ROBOSIM MODELS

Invisible Cartesian Robot

```
;Invisible_Cartesian_Robot.lsp
(defun make-cartesian-robot nil (make-serial-agent
;*****
;Base Frame
;*****
        (make-fixed-link)
        (rotatey 90)  ; Sets orientation of base wrt to world frame
;*****
;Link 1
;*****
(make-prismatic-link nil nil)
        (matmult (rotatex 0) (translate 0 0 0)
        (rotatez 0) (translate 0 0 0))  ; 1 wrt base
;*****
;Link 2
;*****
        (make-prismatic-link nil nil)
        (matmult (rotatex -90) (translate 0 0 0)
                (rotatez -90) (translate 0 0 0)) ; 2 wrt 1
;*****
;Link 3
;*****
        (make-prismatic-link nil nil)
        (matmult (rotatex 90) (translate 0 0 0)
                (rotatez 0) (translate 0 0 0)) ; 3 wrt 2
;*****
;Link 4
;*****
        (make-revolute-link nil nil)
        (matmult (rotatex 0) (translate 0 0 0)
                (rotatez 0) (translate 0 0 0)) ; 4 wrt 3
;*****
;Link 5
;*****
        (make-revolute-link nil nil)
        (matmult (rotatex -90) (translate 0 0 0)
```

```
                (rotatez 0) (translate 0 0 0)) ; 5 wrt 4
;*****
;Link 6
;*****
        (make-revolute-link nil nil)
        (matmult (rotatex 90) (translate 0 0 0)
                (rotatez 0) (translate 0 0 0)) ; 6 wrt 5
))
;****** Place robot in simulation ********
(setq cartesian (make-cartesian-robot))
(use-objects cartesian)
```

Lunar Surface

```
; Lunar_Surface.lsp
;******************
; Set Viewing Parameters
(clear-simulation)
(set-look-from 45 45 45)
(set-look-at 0 0 0)
(set-camera 40 4000 100)
(enable-smooth-shading t)
(setq lunarsurface
; Rolling hills.
(make-composite
        (make-dome 1)(translate 5 -4 0)
        (make-dome 1)(translate 7 -8 0)
        (make-dome 2)(translate 2 -10 0)
        (make-dome 2)(translate -2 18 0)
        (make-dome 8)(translate -12 20 -4)
        (make-truncated-cone 12 1 7)(translate -20 20 0)
        (make-dome 8)(translate -1 -12 -6)
        (make-dome 10)(translate 15 -17 -7)
        (make-dome 9)(translate -20 -18 -4)
        (make-dome 2)(translate -20 -20 0)
; Crater #1 (pos x)(pos y).
        (make-revolve-surface
                (make-primitive (list
                                #(2.5 0 0)
                                #(2.5 0 -1)
                                #(2 0 -2)
                                #(.5 0 -4)
                                #(0 0 -5)
                                #(0 0 -10)
```

```
                                #(5 0 -10)
                                #(5 0 0)
                                #(2.5 0 0)
                                )
                        )
                10)(translate 10 10 0)
                (make-sphere 1.25)(translate 12 11.5 0)
                (make-box 1.25 1.25 1.25)(matmult (translate 12 11.5 0)(rotatex 45)
                (rotatey 45)(rotatez 45))
                (make-box 1.25 1.25 1.25)(matmult (translate 12 11.5 0)(rotatex -45)
                (rotatey -45)(rotatez -45))
; Boulder group #1 - (pos x)(neg y).
                (make-extrude-surface
                        (make-primitive (list
                                #(4 0 0)
                                #(3.5 1 0)
                                #(3 1.5 0)
                                #(2.9 3 0)
                                #(1 4.3 0)
                                #(0 5 0)
                                #(-2 5 0)
                                #(-1 4.1 0)
                                #(-5 3.4 0)
                                #(-8 3 0)
                                #(-4 2.5 0)
                                #(-7 1.5 0)
                                #(-9 1 0)
                                #(-5 .3 0)
                                #(-2 0 0)
                                #(4 0 0)
                                )
                        )
                5)(translate 20 -25 0)
                (make-extrude-surface
                        (make-primitive (list
                                #(4 0 0)
                                #(1.5 1 0)
                                #(2 1.5 0)
                                #(4 2 0)
                                #(5 3 0)
                                #(3.2 3.5 0)
                                #(4 5 0)
                                #(2.2 6 0)
                                #(-3 5 0)
                                #(-3.2 3 0)
                                #(0 0 0)
```

```
                        #(4 0 0)
                        )
                )
        8)(translate 19 -20 0)
        (make-truncated-cone 3 .75 9.5)(translate 17 -19 0)
        (make-sphere 1.1)(translate 18 -18 8)
        (make-sphere 1)(translate 17 -18.5 8)
        (make-box 3 3 3)(matmult (translate 16 -20 4)(rotatex 45)(rotatey 45))
        (make-sphere .75)(translate 17 -19 9.5)
        (make-truncated-cone 4 .25 10)(translate 19 -14 0)
        (make-sphere 1.0)(translate 15 -22 5)
        (make-truncated-cone 4 1.7 6)(translate 21 -12 0)
        (make-sphere 3)(translate 20 -10 1)
        (make-sphere 2)(translate 20.5 -12 4)
        (make-truncated-cone 5 .1 5)(translate 5 -20 0)
        (make-dome 5)(translate 10 -20 0)
        (make-truncated-cone 6 .5 8)(translate 0 -20 0)
        (make-truncated-cone 6 .5 7)(translate 24 -18 0)
; Boulder group #2 (pos x)(pos y)
        (make-extrude-surface
                (make-primitive (list
                        #(0 0 0)
                        #(1.5 0.5 0)
                        #(2 1.5 0)
                        #(1.5 2 0)
                        #(1 3 0)
                        #(-2 3.4 0)
                        #(-4 2.1 0)
                        #(-3 1 0)
                        #(-5 0 0)
                        #(-3.2 -2.5 0)
                        #(-1.6 -3 0)
                        #(0 -3.2 0)
                        #(0 0 0)
                        )
                )
                5)(translate 23 21 0)
        (make-dome 1.5)(translate 21 22 5)
        (make-sphere 1.7)(translate 21.5 20 5)
        (make-dome 4)(translate 19 20 0)
        (make-dome 3)(translate 22 18 0)
        (make-dome 4)(translate 23 23 0)
        (make-dome 7)(translate 16 26 -3)
; Crater #2 + rocks (neg x)(neg y).
        (make-revolve-surface
                (make-primitive (list
```

```
                                        #(3.5 0 0)
                                        #(3.5 0 -1)
                                        #(3 0 -2)
                                        #(1.5 0 -4)
                                        #(0 0 -5)
                                        #(0 0 -10)
                                        #(7 0 -10)
                                        #(7 0 0)
                                        #(3.5 0 1.5)
                                        #(3.5 0 0)
                                        )
                        )
                20)(translate -12 -12 0)
                (make-sphere 1.5)(translate -13 -15.5 2)
                (make-box 1.5 1.25 1.5)(matmult (translate -13 -15.5 2)(rotatex 45)
                (rotatey 45)(rotatez 45))
                (make-box 1.25 1.5 1.25)(matmult (translate -13 -15.5 2)(rotatex -45)
                (rotatey -45)(rotatez -45))
                (make-sphere 3)(translate -11 -17 -1)
                (make-truncated-cone 7 .25 5)(translate -23 -11 0)
                (make-truncated-cone 8 .1 10)(translate -15 -22 0)
                (make-sphere 1)(translate -21 -17 5)
; Flat surface.
                (make-box 35 35 1)(translate -17.5 17.5 -1)
                (make-box 35 35 1)(translate 17.5 -17.5 -1)
                (make-box 35 10 1)(translate -17.5 -3 -1)
                (make-box 8 35 1)(translate -4 -17.5 -1)
                (make-box 35 20 1)(translate -17.5 -25 -1)
                (make-box 10 35 1)(translate -30 -17.5 -1)
                (make-box 35 8 1)(translate 17.5 3.5 -1)
                (make-box 8 35 1)(translate 3.5 17.5 -1)
                (make-box 35 22 1)(translate 17.5 24 -1)
                (make-box 22 35 1)(translate 24 17.4 -1)
)
)
(set-position lunarsurface (translate 0 0 0))
(use-objects lunarsurface)
```

Lunar Rover

```
;Lunar_Marcbot.lsp
(defun make-lunar-robot nil (make-parallel-agent
;**********
;       Body of Robot
```

```
;**********
(set-object
        (make-fixed-link
        (make-box .8 .1 .05)(translate 0 0 .2)
        (make-box .1 .1 .08)(translate .3 0 .12)
        (make-box .1 .1 .08)(translate -.3 0 .12)
        (make-cylinder .025 .38)(matmult (rotatex 90)(translate .3 .11 -.19))
        (make-cylinder .025 .38)(matmult (rotatex 90)(translate -.3 .11 -.19))
        (make-extrude-surface
                (make-primitive (list
                        #(-.4 -.1 0)
                        #(.2 -.1 0)
                        #(.4 -.05 0)
                        #(.4 .05 0)
                        #(.2 .1 0)
                        #(-.4 .1 0)
                        #(-.4 -.1 0)
        )) .02)(translate 0 0 .25)
        (make-extrude-surface
                (make-primitive (list
                        #(0 0 0)
                        #(.05 0 0)
                        #(.05 .03 0)
                        #(.025 .1 0)
                        #(0 .1 0)
        )) .6)(matmult (translate -.4 .1 .27)(rotatez 90)(rotatex 90))
        (make-extrude-surface
                (make-primitive (list
                        #(0 0 0)
                        #(.05 0 0)
                        #(.05 .03 0)
                        #(.025 .1 0)
                        #(0 .1 0)
        )) .6)(matmult (translate .2 -.1 .27)(rotatez -90)(rotatex 90))
        (make-cylinder .05 .2)(translate -.3 0 .27)
)
:color '(100 150 50)
)
;******Base-to-Station transformation.******
        (translate 0 0 0)
;**********
; Wheel Creation.
;**********
        (setq lfw (make-wheel))
                (translate .3 .19 .11)
        (setq rfw (make-wheel))
```

```
                (translate .3 -.19 .11)
        (setq lrw (make-wheel))
                (translate -.3 .19 .11)
        (setq rrw (make-wheel))
                (translate -.3 -.19 .11)
;************
; Laser Creation.
;************
        (setq laser (make-laser))
        (translate 0 0 0);Laser base with respect to the base of the robot.
;*************************
; Arm and Swiss Ranger Sensor.
;*************************
        (setq arm (make-arm))
                (translate .3 0 .27)
));End of make parallel agent and make lunar robot.
;*************************************
; Define make-wheel.
;*************************************
(defun make-wheel nil (make-serial-agent
(make-fixed-link)
(translate 0 0 0)
(set-object
(make-revolute-link nil nil
(make-cylinder .11 .1)(matmult (rotatex 90)(translate 0 0 -.05))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 10))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 20))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 30))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 40))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 50))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 60))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 70))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 80))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 90))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 100))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 110))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 120))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 130))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 140))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 150))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 160))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 170))
(make-box .24 .1 .01)(matmult (rotatey 90) (translate 0 0 -.005)(rotatey 180))
)
:color '(50 50 50))
```

```
(translate 0 0 0);1 with respect to base.
(set-object
        (make-fixed-link
                (make-cylinder .06 .12)(matmult (rotatex 90)(translate 0 0 -.06))
        )
        :color '(150 150 150))
        (translate 0 0 0); 2 with respect to 1.
        ); End of make-serial agent.
); End of defun make-wheel.
;***************
; Define make-laser.
;***************
(defun make-laser nil (make-serial-agent
        (make-fixed-link)
        (translate -.3 0 .57); Position laser base with respect to robot base.
; Laser link1.
        (set-object
        (make-revolute-link nil nil
                (make-box .1 .3 .025)(translate 0 0 -.125)
                (make-box .05 .02 .1)(translate 0 .14 -.1)
                (make-box .05 .02 .1)(translate 0 -.14 -.1)
        )
        :color '(100 150 50))
        (translate 0 0 0);link1 wrt laser base
; Laser link 2.
        (set-object
        (make-revolute-link nil nil
                (make-cylinder .025 .3)(matmult (translate 0 0 -.15)(rotatex 0))
                (make-sphere .09)(translate 0 0 0)
;               (make-cylinder .04 .2)(rotatey 90)
        )
        :color '(100 150 50))
; Set laser link 2 with respect to laser link 1.
        (matmult(rotatex 90)(translate 0 0 0)(rotatez 0)(translate 0 0 0))
; Laser link 3 - end effector.
        (set-object
                (make-fixed-link
                        (make-cylinder .04 .2)(translate 0 0 0)
                )
                :color '(255 0 0))
; Laser link3 with respect to link 2.
        (rotatey 90)
        );End of make-serial-agent.
); End of defun make-laser.
;***************
; Define make-arm.
```

```
;***************
(defun make-arm nil (make-serial-agent
(set-object
(make-fixed-link
        (make-box .08 .02 .04)(translate 0 0.04 0)
        (make-box .08 .02 .04)(translate 0 -0.04 0)
        (make-cylinder .04 .15)(matmult(translate 0 0.075 0.05)(rotatex 90))
        (make-box .4 .05 .05)(translate -.2 0 0.025)
        (make-cylinder .025 .12)(matmult (translate -.4 .06 0.05)(rotatex 90))
        (make-box .05 .02 .11)(matmult(translate -.4 0.035 0.05)(rotatey 27))
        (make-cylinder .025 .12)(matmult (translate -.35 .06 0.15)(rotatex 90))
        (make-box .05 .02 .11)(matmult(translate -.4 -0.035 0.05)(rotatey 27))
        (make-box .5 .05 .05)(translate -.1 0 0.125)
        (make-cylinder .025 .12)(matmult (translate .15 .06 0.15)(rotatex 90))
        (make-box .05 .02 .05)(translate .15 .035 .15)
        (make-box .05 .02 .05)(translate .15 -.035 .15)
        (make-box .05 .07 .02)(translate .15 0 .18)
)
:color '(200 255 100))
(translate 0 0 0) ;Arm with respect to base.
; Make Swiss Ranger Sensor.
        (set-object
        (make-fixed-link
        (make-box .0423 .05 .067)(translate .15 0 .2)
        )
:color ' (0 0 255))
(translate 0 0 0)
); End of make-serial-agent.
); End of defun make-arm.
(setq r (make-lunar-robot))
(use-objects r)
```

Lunar Reconnaissance Orbiter Scan

```
;LRO_Scan.lsp
(clear-simulation)
;Load Associated Programs
(load "Lunar_Surface.lsp")
(load "gpos-cartesian.lsp")
(load "cartesian_invkin.lsp")
(load "Invisible_Cartesian_Robot.lsp")
(load "Lunar_Reconnaissance_Orbiter.lsp")
;Set Viewing Parameters
(set-look-at 0 0 30)
(set-look-from 45 45 75)
```

```
(set-camera 1 4000 100)
(enable-smooth-shading t)
(setq d 1); LRO speed during laser scan.
(setq f .999); Laser beam length index value.
(setq ii 1.0); Satellite index movement along the x axis.
(setq jj 1.0); Satellite index movement along the y axis.
(setq xx -35);Scan start position in the x direction.
(setq yy -35);Scan start position in the y direction.
(setq xxx 35);Scan stop position in the x direction.
(setq yyy 35);Scan stop position in the y direction.
(set-position LRO (translate 10 10 50))
(set-position lunarsurface (translate 0 0 30))
(setq lz (make-composite
        (make-box 1 1 .5)(translate 0 0 0)
        (make-box 1 1 .5)(translate 0 0 -.5)
        )
)
(set-object lz :color'(255 0 0))
(use-objects lz)
(setq picture1 (make-sphere .001))
(set-update nil)
(move-inter-to cartesian (translate 10 10 50) 0 d)
(grasp cartesian LRO)
(move-inter-to cartesian (translate 5 5 50)0 d)
(let
(
(x xx)
)
(loop
        (let
        (
        (y yy)
        )
        (loop
        (move-inter-to cartesian (matmult(translate x y 50)(rotatex 180)) 0 d)
                (let
                (
                (length 1)
                (point)
                )
                (loop
                (setq length (+ length f))
                (set-position lz (matmult(gpos-cartesian cartesian)(translate 0 0 (+ 0
                length))))
                (if
                        (and
```

```
(= (check-collision lz lunarsurface) nil)
)
        (progn
        ()
        )
        (progn
        (setq point (make-box 1.0 1.0 (- 50 length)))
        (set-position point (matmult(translate 0 0 (* -1 (- 50
        length)))
        (gpos-cartesian cartesian)(translate 0 0 (+ 0 length))(rotatex
        180)))
        (if
                (>= length 24.5)
                (set-object point :color'(255 0 0))
        )
        (if
                (and
                (>= length 24)(< length 24.5))
                (set-object point :color'(240 15 0))
        )
        (if
                (and
                (>= length 23.5)(< length 24))
                (set-object point :color'(225 30 0))
        )
        (if
                (and
                (>= length 23)(< length 23.5))
                (set-object point :color'(210 45 0))
        )
        (if
                (and
                (>= length 22.5)(< length 23))
                (set-object point :color'(195 60 0))
        )
        (if
                (and
                (>= length 22)(< length 22.5))
                (set-object point :color'(180 75 0))
        )
        (if
                (and
                (>= length 21.5)(< length 22))
                (set-object point :color'(165 90 0))
        )
        (if
```

```
        (and
        (>= length 21)(< length 21.5))
        (set-object point :color'(150 105 0))
)
(if
        (and
        (>= length 20.5)(< length 21))
        (set-object point :color'(135 120 0))
)
(if
        (and
        (>= length 20)(< length 20.5))
        (set-object point :color'(120 135 0))
)
(if
        (and
        (>= length 19.5)(< length 20))
        (set-object point :color'(105 150 0))
)
(if
        (and
        (>= length 19)(< length 19.5))
        (set-object point :color'(90 165 0))
)
(if
        (and
        (>= length 18.5)(< length 19))
        (set-object point :color'(75 180 0))
)
(if
        (and
        (>= length 18)(< length 18.5))
        (set-object point :color'(60 195 0))
)
(if
        (and
        (>= length 17.5)(< length 18))
        (set-object point :color'(45 210 0))
)
(if
        (and
        (>= length 17)(< length 17.5))
        (set-object point :color'(30 225 0))
)
(if
        (and
```

```
        (>= length 16.5)(< length 17))
        (set-object point :color'(15 240 0))
)
(if
        (and
        (>= length 16)(< length 16.5))
        (set-object point :color'(0 255 0))
)
(if
        (and
        (>= length 15.5)(< length 16))
        (set-object point :color'(0 240 15))
)
(if
        (and
        (>= length 15)(< length 15.5))
        (set-object point :color'(0 225 30))
)
(if
        (and
        (>= length 14.5)(< length 15))
        (set-object point :color'(0 210 45))
)
(if
        (and
        (>= length 13)(< length 14.5))
        (set-object point :color'(0 195 60))
)
(if
        (and
        (>= length 12.5)(< length 13))
        (set-object point :color'(0 180 75))
)
(if
        (and
        (>= length 12)(< length 12.5))
        (set-object point :color'(0 165 90))
)
(if
        (and
        (>= length 11.5)(< length 12))
        (set-object point :color'(0 150 105))
)
(if
        (and
        (>= length 11)(< length 11.5))
```

```
        (set-object point :color'(0 135 120))
)
(if
        (and
        (>= length 10.5)(< length 11))
        (set-object point :color'(0 120 135))
)
(if
        (and
        (>= length 10)(< length 10.5))
        (set-object point :color'(0 105 150))
)
(if
        (and
        (>= length 9.5)(< length 10))
        (set-object point :color'(0 90 165))
)
(if
        (and
        (>= length 9)(< length 9.5))
        (set-object point :color'(0 75 180))
)
(if
        (and
        (>= length 8.5)(< length 9))
        (set-object point :color'(0 60 195))
)
(if
        (and
        (>= length 8)(< length 8.5))
        (set-object point :color'(0 45 210))
)
(if
        (and
        (>= length 7.5)(< length 8))
        (set-object point :color'(0 30 225))
)
(if
        (and
        (>= length 7)(< length 7.5))
        (set-object point :color'(0 15 240))
)
(if
        (and
        (>= length 1)(< length 7))
        (set-object point :color'(0 0 255))
```

```
                        )
                    (use-objects point)
                    (return)
                    )
                )
                    (if
                        (> length 100)
                        (return)
                    )
                )
            )
            (setq y (+ y jj))
                (if
                (> y yyy)
                (return)
                ); End of if statement.
            ); End of loop to drive robot along the y axis.
); End of let to drive robot along the y axis.
(setq x (+ x ii));Index along the x axis.
            (if
            (> x xxx); End of scan along the x axis
            (return)
            ); End of if statement.
); End of loop to drive robot along the y axis.
); End of let to drive robot along the x axis.
(remove-objects lz)
```

Position Determination

```
; 3D_Trilateration_1.lsp
(clear-simulation)
; Load Associated Programs
(load "Lunar_Surface.lsp")
(load "Lunar_Marcbot.lsp")
(load "gpos-laser.lsp")
; Set Viewing Parameters
(set-look-at 0 0 30)
(set-look-from 45 45 75)
(set-camera 1 4000 100)
(enable-smooth-shading t)
; Set the position of the lunar surface.
(set-position lunarsurface (translate 0 0 30))
; Set the position of the robotic lunar rover.
(set-position r (translate 0 0 30))
```

```
(message-box "Position determination of a robotic lunar rover using 3-D trilateration."
"View of the lunar surface.")
(drive-agent laser '(360) 200)
(set-camera 40 4000 3000)
(set-clipplane 40 4000)
(message-box "Position Determination of a robotic lunar rover using 3-D trilateration."
"View of the robotic lunar rover with its laser range finder.")
(drive-agent laser '(0) 800)
; Create laser beam photon (lz). Cylinder size determines laser's detection accuracy.
(setq lz (make-composite
                (make-cylinder .01 .1)(translate 0 0 0)
                (make-cylinder .01 .1)(translate 0 0 -.1)
                                )
);Lasers accuracy set to +/- .1 meters.
(set-object lz :color'(255 0 0))
; Set view point behind rover.
(set-look-at 20 20 30)
(set-look-from -1 -1 31)
(set-clipplane 1 4000)
(set-camera 1 4000 100)
;*************************************************
; Drive laser range finder to first point to measure.
;*************************************************
(drive-agent laser '(42.5 11) 500)
; Place laser on end of laser base.
(setq lzsight (make-cylinder .039 30))
(set-position lzsight (matmult(translate 0 0 30)(gpos-laser laser)))
(set-object lzsight :color '(255 0 0))
(use-objects lzsight)
; Algorithm to measure distance with laser beam.
(let
(
(length 1)
)
(loop
(setq length (+ length .19))
; Position laser beam at the base of the laser + indexed length.
(set-position lz (matmult(translate 0 0 30)(gpos-laser laser)(translate 0 0 (+ 0 length))))
(use-objects lz)
        (if
        (and ;Check for collision between the laser beam and the moon.
        (= (check-collision lz lunarsurface) nil)
        )
                (progn
                ()
                )
```

```
                (progn
                        (setq point1 (make-sphere 0.5))
                        (set-object point1 :color '(255 0 0))
; Position the data point;
; Matrix Multiplication (Position of the Lunar Rover)(Laser's Base Position)(Length of
laser beam).
                        (set-position point1 (matmult(translate 0 0 30)(gpos-laser
                        laser)(translate 0 0 (+ 0 length))))
; Get position of laser tip relative to rover base and determine distance between.
                        (setq pos1 (matmult (gpos-laser laser)(translate 0 0 (+ 0 length))))
                        (setq px1 (tref pos1 0 3))
                        (setq py1 (tref pos1 1 3))
                        (setq pz1 (tref pos1 2 3))
; Set r1 to the distance between the laser tip and the rover's base.
                        (setq r1 (sqrt (+ (* px1 px1) (* py1 py1) (* pz1 pz1))))
                        (return)

                )
        )
        (if
        (> length 50);max length of laser beam.
        (return); If no collision with beam at length of 50, then end laser beam index.
                        )
        )
        )
)
(remove-objects lz)
(remove-objects lzsight)
(message-box "Calculated distance from the lunar rover's base frame to the identified
LRO scan target P2 ."
        "Distance in meters :" (+ 0 r1))
; Reposition view point behind rover.
(set-look-at 20 -20 30)
(set-look-from -1 1 31)
(set-clipplane 1 4000)
(set-camera 1 4000 100)
;*****************************************************
;
; Drive laser range finder to second point to measure.
;*****************************************************
;
(drive-agent laser '(-36 21.5) 500)
;Place laser on end of laser base.
(setq lzsight (make-cylinder .039 27))
(set-position lzsight (matmult(translate 0 0 30)(gpos-laser laser)))
(set-object lzsight :color '(255 0 0))
(use-objects lzsight)
; Algorithm to measure distance with laser beam.
```

```
(let
(
(length 1); Initial length of the laser beam.
)
(loop
	(setq length (+ length .19)); Index the length of the laser beam.
	; Position laser beam at the base of the laser + indexed length.
	(set-position lz (matmult(translate 0 0 30)(gpos-laser laser)(translate 0 0 (+ 0
	length))))
	(use-objects lz)
	(if
	(and ;Check for collision between the laser beam and the moon.
	(= (check-collision lz lunarsurface) nil)
	)
		(progn
		()
		)
		(progn
			(setq point2 (make-sphere 0.5))
; If there is a collision - create data point.
			(set-object point2 :color '(0 255 0))
; Position the data point;
; Matrix Multiplication (Position of the Lunar Rover)(Laser's Base Position)(Length of
laser beam)
			(set-position point2 (matmult(translate 0 0 30)(gpos-laser
			laser)(translate 0 0 (+ 0 length))))
; Get position of laser tip relative to rover base and determine distance between.
			(setq pos2 (matmult (gpos-laser laser)(translate 0 0 (+ 0 length))))
			(setq px2 (tref pos2 0 3))
			(setq py2 (tref pos2 1 3))
			(setq pz2 (tref pos2 2 3))
; Set r2 to the distance between the laser tip and the rover's base.
			(setq r2 (sqrt (+ (* px2 px2) (* py2 py2) (* pz2 pz2))))
			(return)
		)
	)
	(if
	(> length 50); Max length of laser beam.
	(return); If no collision with beam at length of 50, then end laser beam index.
	)
); end of loop
); end of let
(remove-objects lz)
(remove-objects lzsight)
(message-box "Calculated distance from the lunar rover's base frame to the identified
LRO scan target P15." "Distance in meters: " (+ 0 r2))
```

```
; Reposition view point behind rover.
(set-look-at -20 -20 30)
(set-look-from 1 1 31)
(set-clipplane 1 4000)
(set-camera 1 4000 100)
;*************************************************
; Drive laser range finder to third point to measure.
;*************************************************
(drive-agent laser '(-123.7 19.5) 500)
; Place laser sight on end of laser base.
(setq lzsight (make-cylinder .039 30))
(set-position lzsight (matmult(translate 0 0 30)(gpos-laser laser)))
(set-object lzsight :color '(255 0 0))
(use-objects lzsight)
; Algorithm to measure distance with laser beam.
(let
(
(length 1); Initial length of the laser beam.
)
(loop
        (setq length (+ length .19));Index the length of the laser beam.
        ; Position laser beam at the base of the laser + indexed length.
        (set-position lz (matmult(translate 0 0 30)(gpos-laser laser)(translate 0 0 (+ 0
        length))))
        (use-objects lz)
        (if
        (and ;Check for collision between the laser beam and the moon.
        (= (check-collision lz lunarsurface) nil)
        )
                (progn
                ()
                )
                (progn
                (setq point3 (make-sphere 0.5))
; If there is a collision - create data point.
                (set-object point3 :color '(0 0 255))
; Position the data point;
; Matrix Multiplication (Position of the Lunar Rover)(Laser's Base Position)(Length of
laser beam)
                (set-position point3 (matmult(translate 0 0 30)(gpos-laser laser)(translate 0
                0 (+ 0 length))))
; Get position of laser tip relative to rover base and determine distance between.
                (setq pos3 (matmult (gpos-laser laser)(translate 0 0 (+ 0 length))))
                (setq px3 (tref pos3 0 3))
                (setq py3 (tref pos3 1 3))
                (setq pz3 (tref pos3 2 3))
```

```
; Set r3 to the distance between the laser tip and the rover's base.
                (setq r3 (sqrt (+ (* px3 px3) (* py3 py3) (* pz3 pz3))))
                (return)
                )
        )
        (if
                (> length 50); Max length of laser beam.
                (return); If no collision with beam at length of 50, then end laser beam
index.
        )
); end of loop
); end of let
(remove-objects lz)
(remove-objects lzsight)
(message-box "Calculated distance from the lunar rover's base frame to the identified
LRO scan target P8." "Distance in meters: " (+ 0 r3))
;*****************************
; Target Data from LRO Scan.
;*****************************
; Point1 is target P2 [21.5, 20, 37]
(setq point1 (make-sphere 1))
(set-position point1 (translate 21.5 20 37))
(setq point1x 21.5)
(setq point1y 20)
(setq point1z 37)
; Point2 is target P15 [19, -14 40]
(setq point2 (make-sphere 1))
(set-position point2 (translate 19 -14 40))
(setq point2x 19)
(setq point2y -14)
(setq point2z 40)
; Point3 is target P8 [-15, -22, 40]
(setq point3 (make-sphere 1))
(set-position point3 (translate -15 -22 40))
(setq point3x -15)
(setq point3y -22)
(setq point3z 40)
;*************************************************************************
; Move Point1 to [0, 0, 0] & translate Point2 and Point3 the corresponding value.
;*************************************************************************
(set-position point1 (translate 0 0 0))
(set-position point2 (translate (- point2x point1x) (- point2y point1y) (- point2z point1z)))
(set-position point3 (translate (- point3x point1x) (- point3y point1y) (- point3z point1z)))
(use-objects point1 point2 point3)
;*****************************************
; Determine Point2's position relative to Point1.
```

```
;*********************************************
(setq pos21 (get-position point2 :relative-to point1))
(setq px21 (tref pos21 0 3))
(setq py21 (tref pos21 1 3))
(setq pz21 (tref pos21 2 3))
;*************************************************
; Rotate Point2 to where its origin lies on the + x-axis.
;*************************************************
(setq thetaz (* (atan py21 px21) (/ 180 pi)))
(setq thetay (* (atan pz21 (sqrt (+ (* px21 px21) (* py21 py21)))) (/ 180 pi)))
(set-position point2 (matmult (rotatey (* 1 thetay))(rotatez (* -1 thetaz))
        (translate (- point2x point1x) (- point2y point1y) (- point2z point1z))))
;*************************************************************************
; Rotate Point3 through the same rotations as Point2 previously was rotated.
;*************************************************************************
(set-position point3 (matmult (rotatey (* 1 thetay))(rotatez (* -1 thetaz))
        (translate (- point3x point1x) (- point3y point1y) (- point3z point1z))))
;*********************************************
; Determine Point3's position relative to Point1.
;*********************************************
(setq pos31 (get-position point3 :relative-to point1))
(setq px31 (tref pos31 0 3))
(setq py31 (tref pos31 1 3))
(setq pz31 (tref pos31 2 3))
;*************************************************************************
; Rotate Point3 so that it lies on the Z=0 plane with Point1 and Point2.
; Check to see if Point3 is in the III or IV quadrant so the rotation about the x axis
; will be correct.  If py31 is negative or equal to zero the angle calculated from atan
;function will rotate to the z=0 plane.  If it is in the I or II quadrant then Point3 will need
;to be rotated in the opposite direction of the calculated angle.
;*************************************************************************
(if (<= py31 0)
        (setq thetax (* (atan pz31 (sqrt (+ (* 0 px31 px31) (* py31 py31)))) (/ 180 pi)))
        (setq thetax (* (atan pz31 (sqrt (+ (* px31 px31) (* py31 py31)))) (/ 180 pi) (-1)))
)
;*******************************************
; Rotate Point3 so that it is on the z=0 plane.
(set-position point3 (matmult (rotatex (* 1 thetax))(rotatey (* 1 thetay))(rotatez (* -1
thetaz))(translate (- point3x point1x) (- point3y point1y) (- point3z point1z))))
;*************************************************************
; Determine Point3's new position relative to Point1 (or origin).
;*************************************************************
(setq pos30 (get-position point3))
(setq px30 (tref pos30 0 3))
(setq py30 (tref pos30 1 3))
(setq pz30 (tref pos30 2 3))
```

```
;*************************************************************
; Determine Point2's new position relative to Point1 (or origin).
;*************************************************************
(setq pos20 (get-position point2))
(setq px20 (tref pos20 0 3))
(setq py20 (tref pos20 1 3))
(setq pz20 (tref pos20 2 3))
;*************************************************************************
; Now we have three spheres located on the z=0 plane with radiuses of r1, r2 & r2.
; Take the formula for each of the spheres and set them equal to each other.
; This will allow you to solve for the unknown x, y, & z coordinates.
;*************************************************************************
(setq x0 (/ (+ (* r1 r1) (* -1 r2 r2) (* px20 px20)) (* 2 px20)))
(setq y0 (- (/ (+ (* r1 r1) (* -1 r3 r3) (* px30 px30) (* py30 py30)) (* 2 py30)) (/ (* px30
x0) py30)))
(if (< (+ (* r1 r1) (* -1 x0 x0) (* -1 y0 y0)) 0)
        (progn (setq z0 0); Set z0 = 0 because there is too much noise/error in the data.
        (message-box "Caution! Noisy Data. Possibly high level of error in results.")
        )
        (progn (setq z0 (* -1 (sqrt (+ (* r1 r1) (* -1 x0 x0) (* -1 y0 y0))))))
; Use negative solution since rover is below targets.
)
;*************************************************************************
; Set a point at this position (x0, y0, z0) and then re-position the point by reversing
; the previous translations.  The re-positioned point's location will be the calculated
; position of the lunar rover's base.
;*************************************************************************
(setq rover-position (make-sphere 1))
(set-position rover-position (matmult (translate (* 1 point1x) (* 1 point1y) (* 1 point1z))
(rotatez (* 1 thetaz)) (rotatey (* -1 thetay)) (rotatex (* -1 thetax))
(translate (* 1 x0) (* 1 y0) (* 1 z0))))
(setq rover (get-position rover-position))
(setq rover-x (tref rover 0 3))
(setq rover-y (tref rover 1 3))
(setq rover-z (tref rover 2 3))
(setq error (* (/ (- (sqrt (+ (* rover-x rover-x) (* rover-y rover-y) (* rover-z rover-z)))
(sqrt (* 30 30))) (sqrt (* 30 30))) 100))
(setq diste (sqrt (+ (* rover-x rover-x) (* rover-y rover-y) (* (- rover-z 30) (- rover-z
30)))))
(message-box "(X,Y,Z) position of the lunar rover's base frame."
"Calculated position using 3-D Trilateration method :" "(" (* 1 rover-x) ", " (* 1 rover-y)
", " (* 1 rover-z) ") meters."
"
Actual position: (0, 0, 30) meters."
"
Total distance of error: "(* 1 diste) " meters.")
```

APPENDIX C

MATLAB SIMULINK MODELS

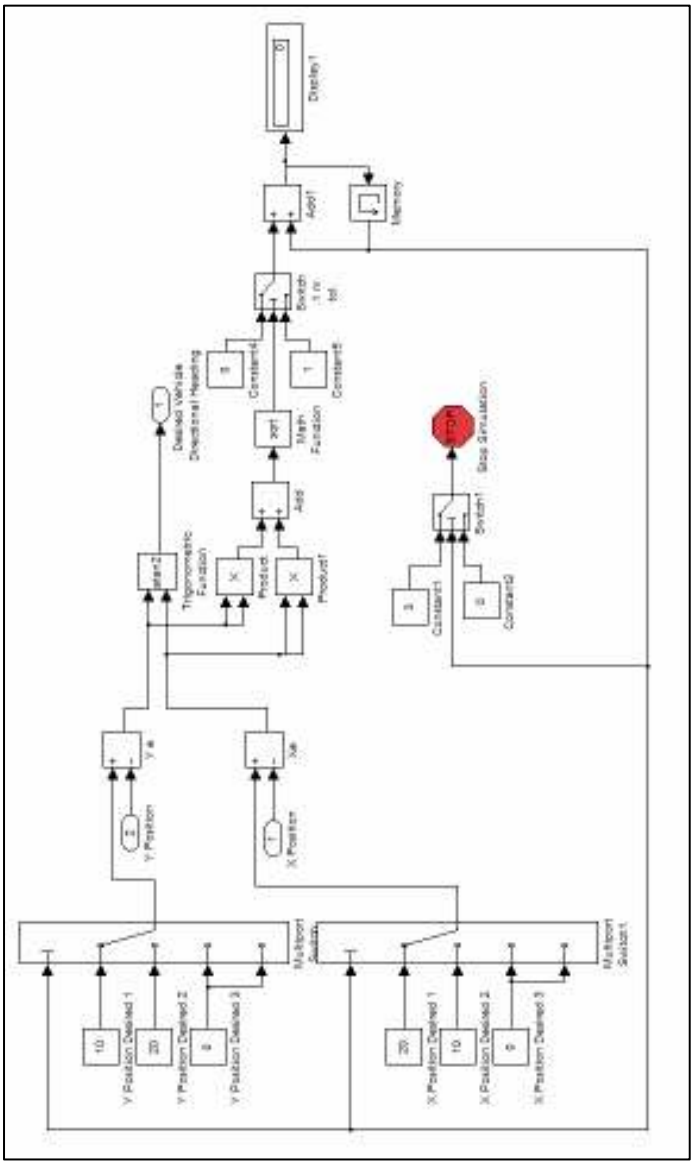

Figure C.1: The navigation central processing unit system.

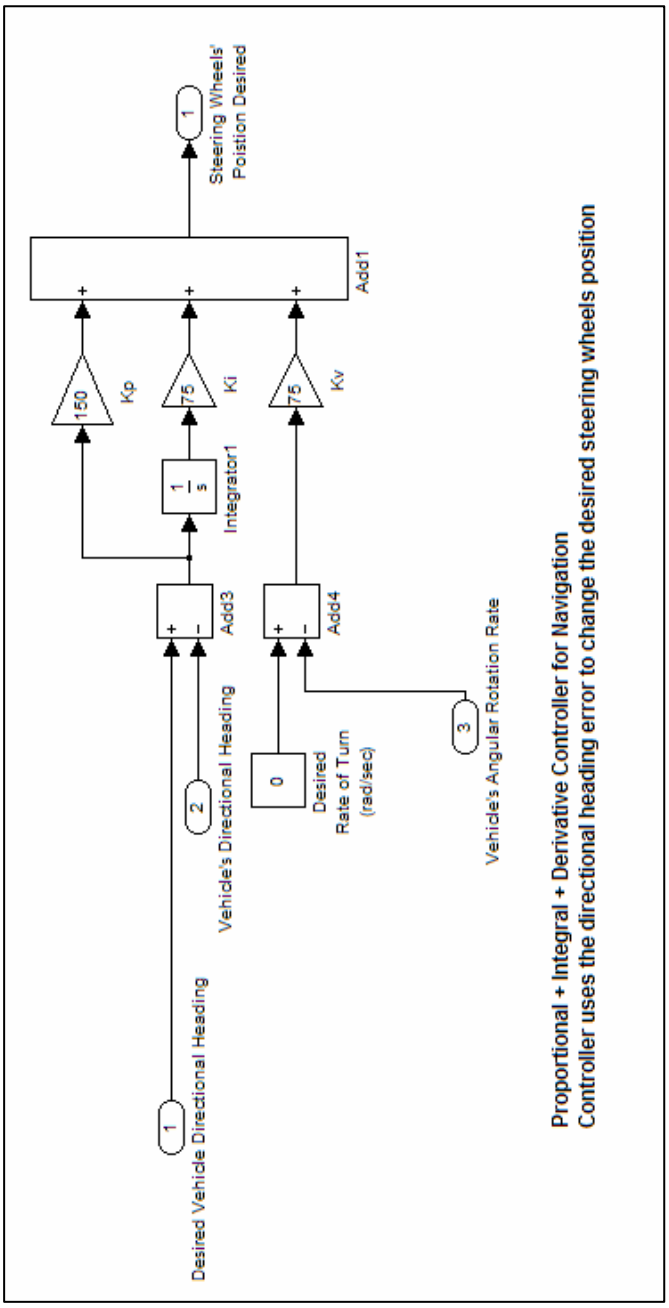

Figure C.2: The navigation controller system.

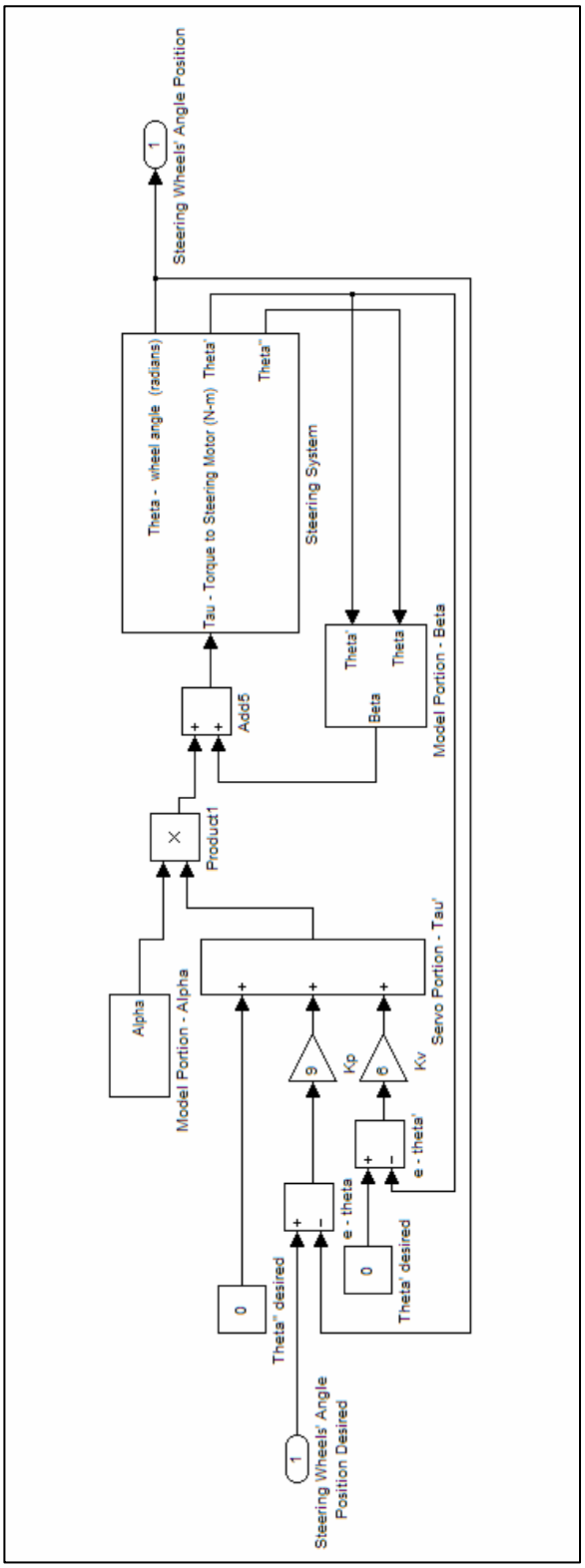

Figure C.3: The steering system.

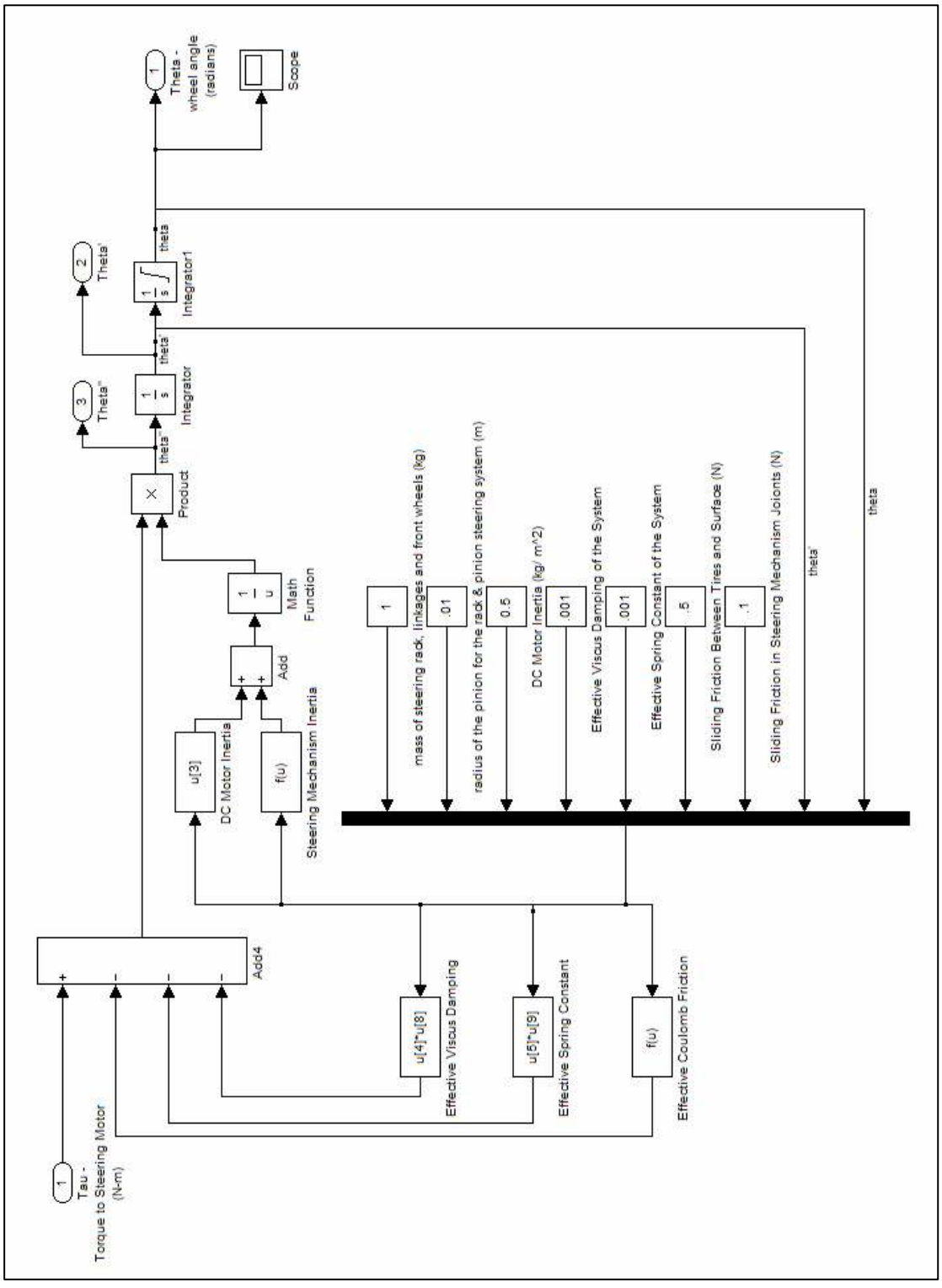

Figure C.4: The steering dynamic equation subsystem.

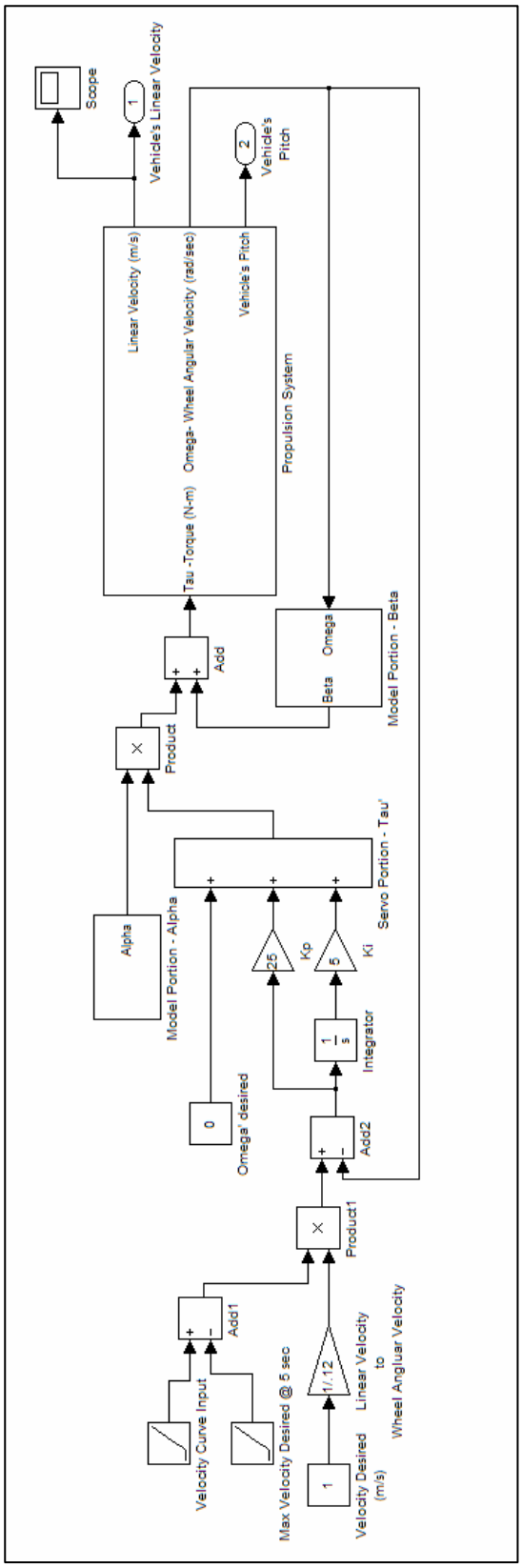

Figure C.5: The propulsion system.

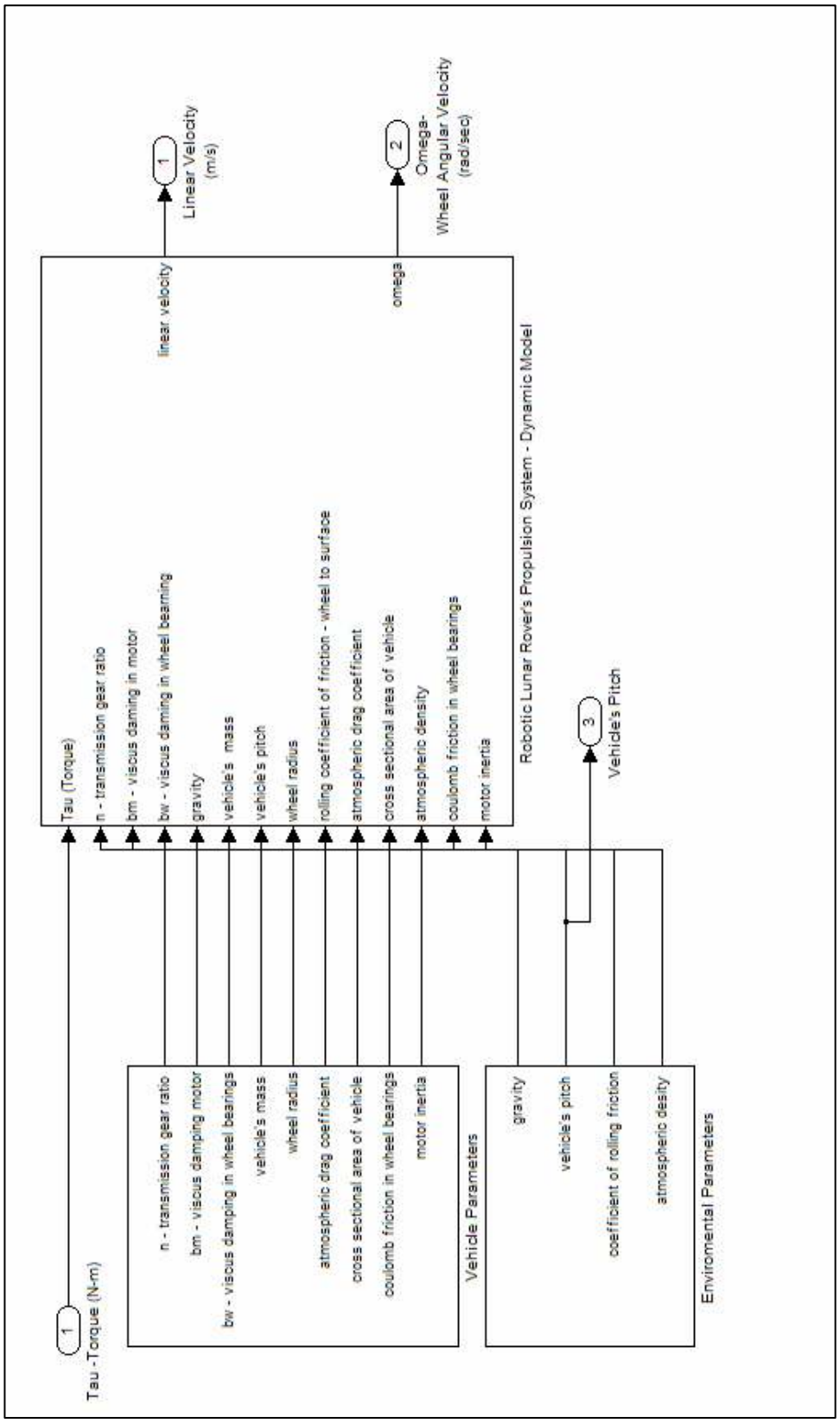

Figure C.6: The propulsion dynamics subsystem.

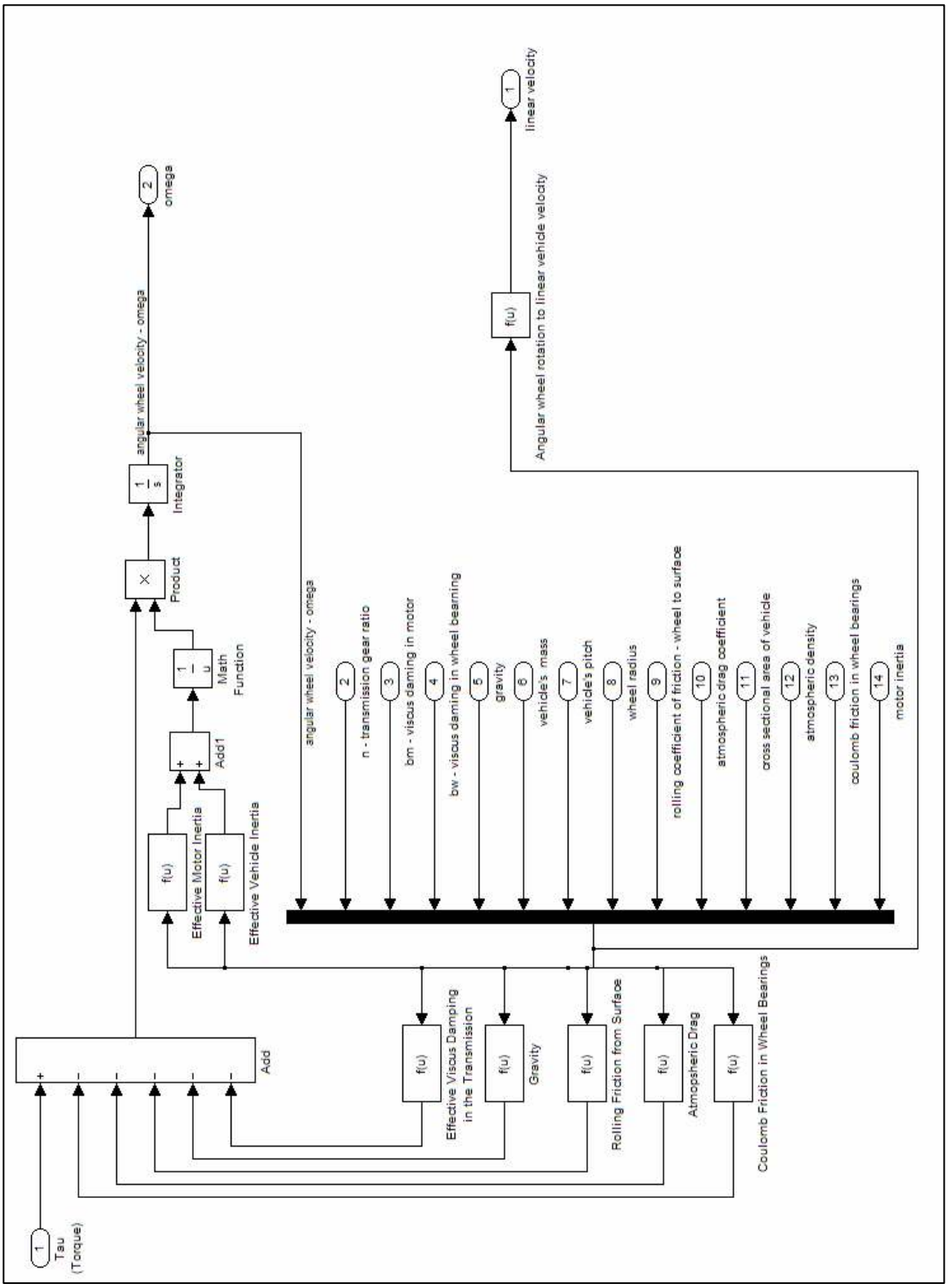

Figure C.7: The propulsion dynamic equation subsystem.

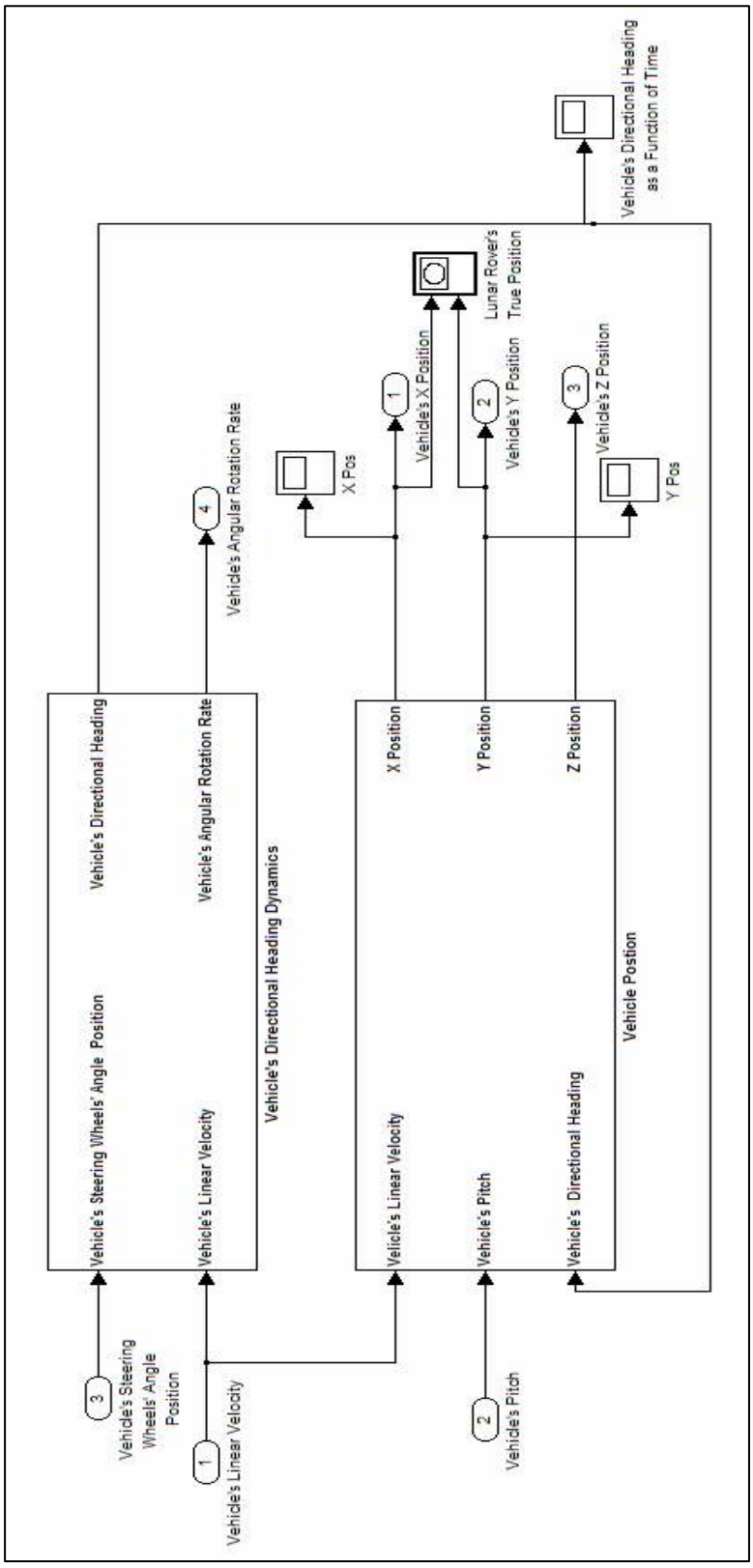

Figure C.8: The dynamics system.

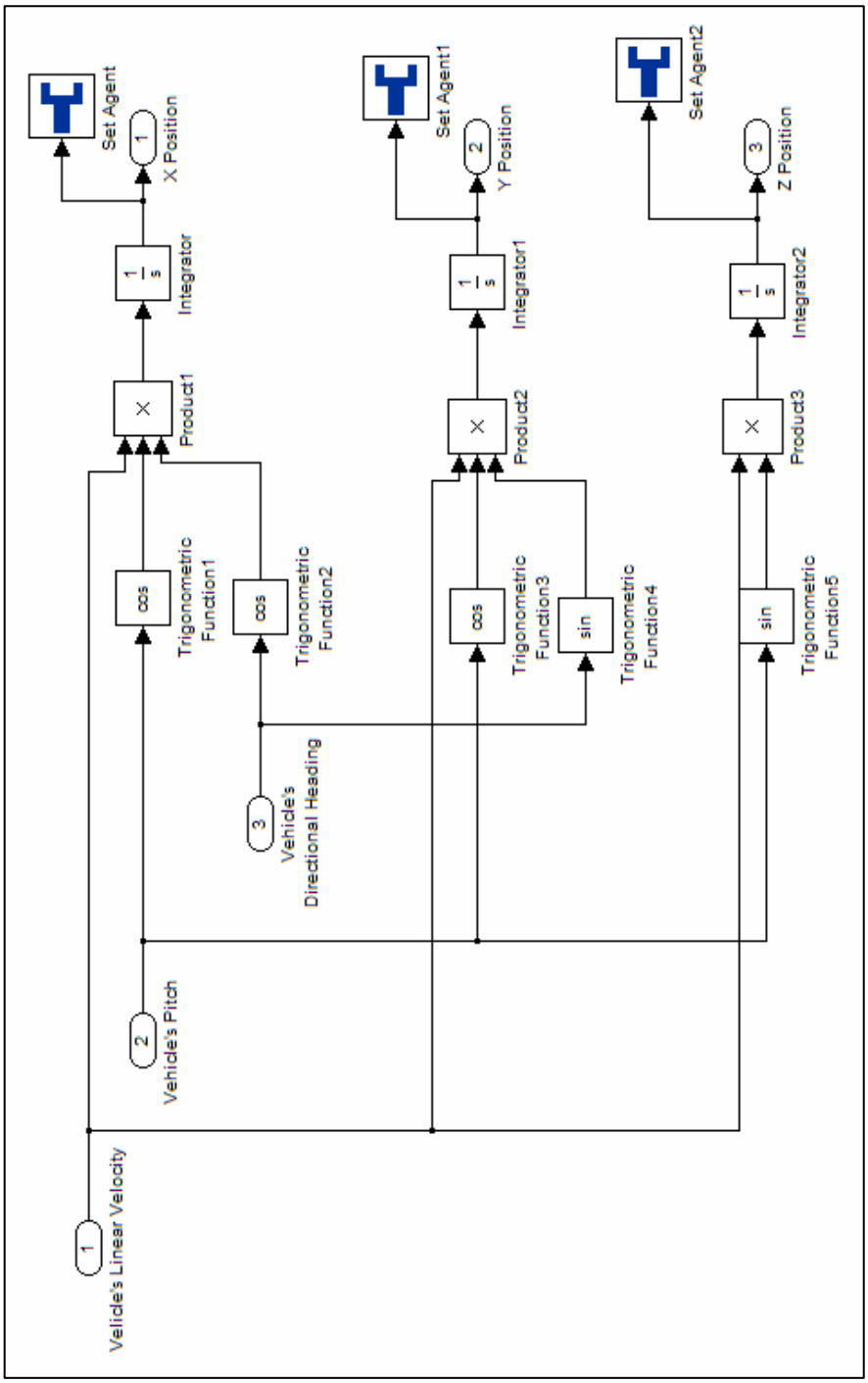

Figure C.9: The position subsystem.

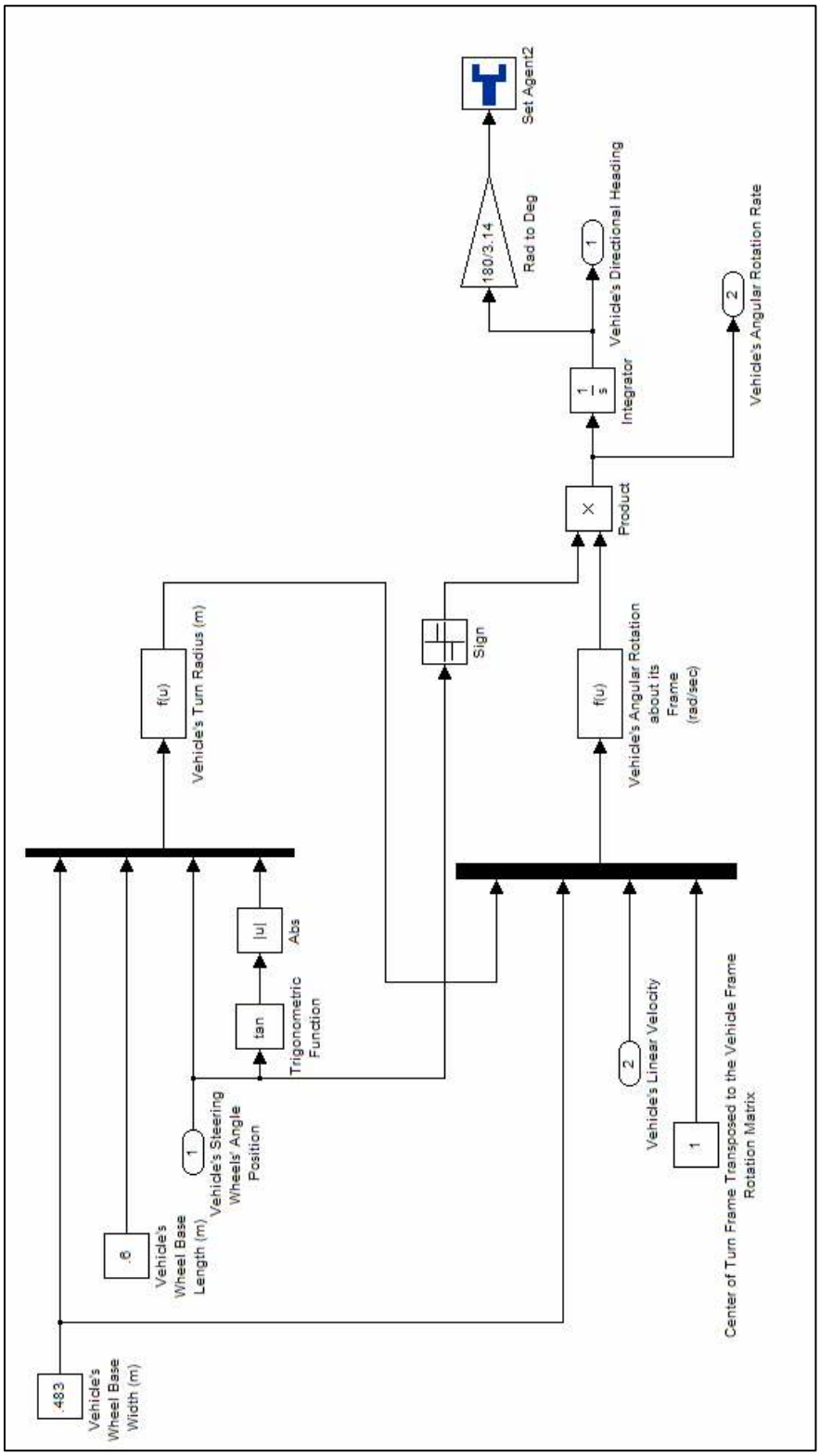

Figure C.10: The directional heading subsystem.

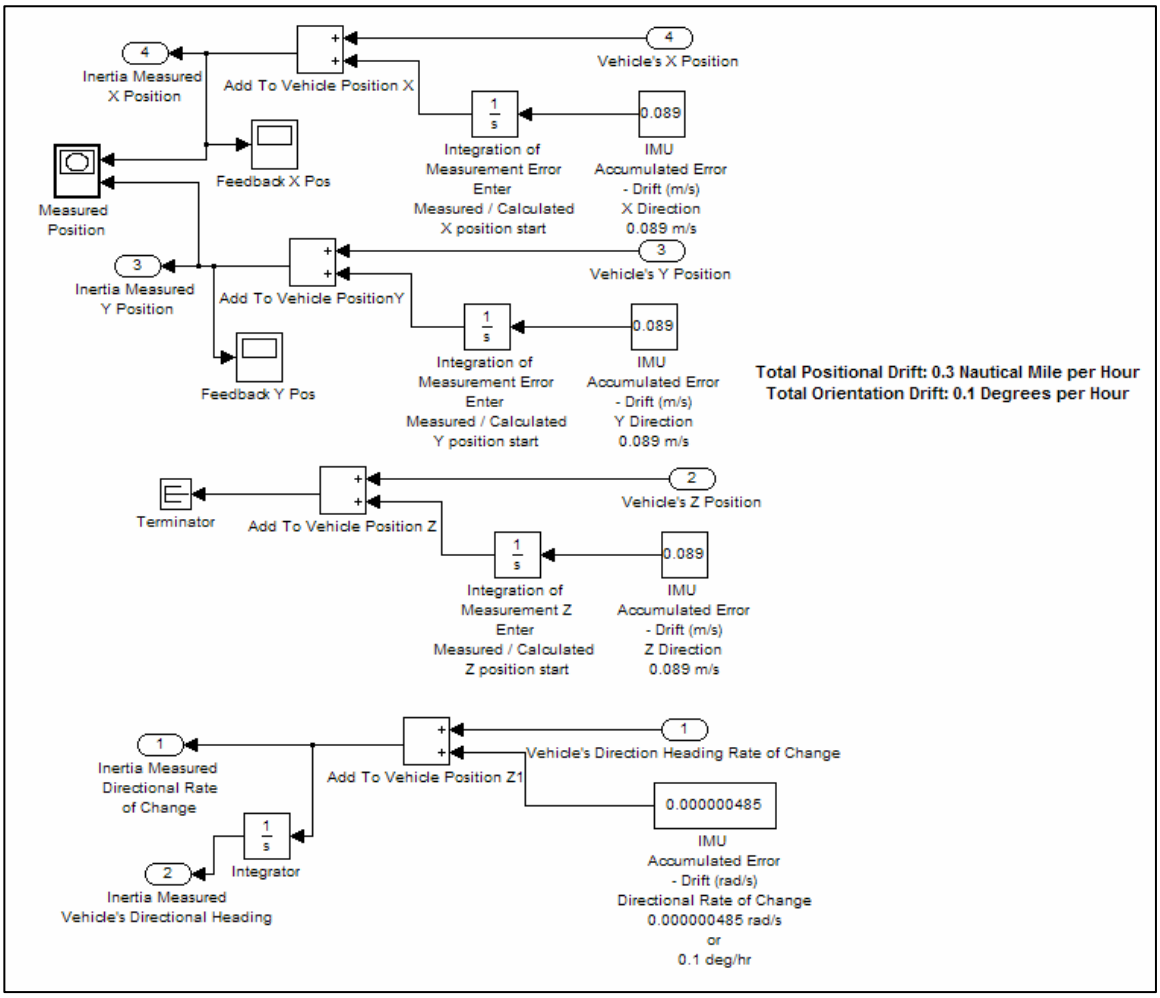

Figure C.11: The inertial measurement unit system.

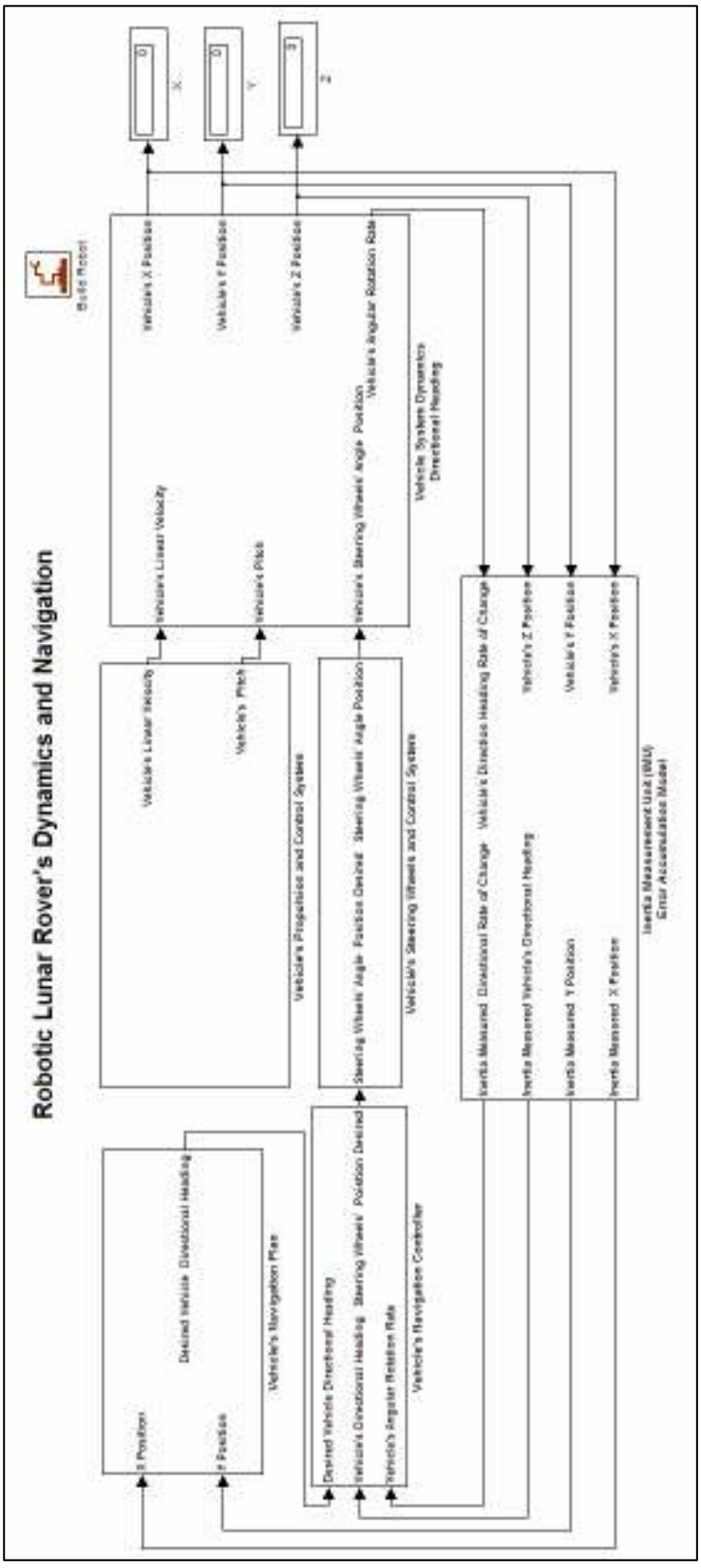

Figure C.12: The dynamics and navigation simulation system.

REFERENCES

Bais, A., Sablatnig, R., Gu, J., and Mahlknecht, S., Active Single Landmark Based Global Localization of Autonomous Mobile Robots. Advances in Visual Computing, Lectures Notes in Computer Science, Volume 4291, pages 202-211, 2006.

Bejczy, A., Robot Arm Dynamics and Control, Jet Propulsion Laboratory Technical Memo 33-669, 1974.

Bhatti, U., Ochieng, W. and Feng, S., Integrity of an Integrated GPS/INS System in the Presence of Slowly Growing Errors. Part I: A critical review. GPS Solutions, Volume 11, Number 3, pages 173-181, 2007.

Craig, J., Introduction to Robotics Mechanics and Control. Pearson Prentice Hall, Third Edition, 2005.

Denavit, J. and Hartenberg, R., A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. Journal of Applied Mechanics, pages 215-221, June 1955.

Di Marco, M., Garulli, A., Giannitrapani, A. and Vicino, A., A Set Theoretic Approach to Dynamic Robot Localization and Mapping. Autonomous Robots, Volume 16, Number 1, pages 23-47, 2004.

Fernandez, K., Robotic Simulation and a Method for Jacobian Control of a Redundant Mechanism with Imbedded Constraints. PhD Dissertation, Vanderbilt University, 1988.

Kwon, Y. and Lee, J., A Stochastic Map Building Method for Mobile Robots using 2-D Laser Range Finder. Autonomous Robots, Volume 7, Number 2, pages 187-200, 1999.

Markiewicz, B., Analysis of the Computed Torque Drive Method and Comparison with Conventional Position Servo for a Computer-Controlled Manipulator. Jet Propulsion Laboratory Technical Memo 33-601, 1973.

LeMaster, E. and Rock, S., A Local-Area GPS Pseudolite-Based Navigation System for Mars Rovers. Autonomous Robots, Volume 14, Numbers 2-3, pages 209-224, 2003.

Paul, P., Modeling, Trajectory Calculation and Servoing of a Computer Controlled Arm. Technical Report AIM-177, Stanford University Artificial Intelligence Laboratory, 1972.

Springfield, J., An Open, Extensible System for Robot Simulation. PhD Dissertation, Vanderbilt University, 1993.

The MathWorks, Learning Simulink. 2004

Zhang, Y. and Gao, Y., A Method to Improve the Alignment Performance for GPS-IMU System. GPS Solutions, Volume 11, Number 2, pages 129-137, 2007.