UC Santa Cruz

UC Santa Cruz Electronic Theses and Dissertations

Title

Speed Map for Autonomous Rovers over Rough Terrain

Permalink

https://escholarship.org/uc/item/3wp1t9jq

Author

Loh, Jonathan Edau

Publication Date

2012

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

SANTA CRUZ

Speed Map for Autonomous Rovers over Rough Terrain

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Jonathan Loh

September 2012

The Thesis of Jonathan Loh
is approved:

Professor Gabriel Elkaim, Chair

Professor Dejan Milutinović

Professor Renwick Curry

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Copyright © by

Jonathan Edau Loh

2012

Table of Contents

List of Figures

List of Tables

Abstract

Speed Map for Autonomous Rovers over Rough Terrain

by

Jonathan Loh

All past NASA planetary rovers have only been able to traverse celestial surfaces at a maximum speed of $0.05 - 0.09m/s$ $(0.11 - 0.20mph)$. There is motivation to operate rovers more autonomously and to increase their speeds upwards to $3m/s$ on flat hard ground, which is considered fast for planetary rovers. In this thesis, a novel roughness metric is used to create a speed map and provide planetary rovers with information about the terrain. The provided information is intrinsic to the terrain regarding its roughness and speed allowing the rover to safely travel over smooth and rough terrain. The results of this method will benefit path planning algorithms and control operators in improving mission efficiency.

To my family:

Mom, Dad, Bettina, Chris, Theresa, Tiffany, and baby Lau

I love you.

Acknowledgments

I would like to thank Professor Gabriel Elkaim for his continual support and advice, this thesis would not be possible without it. I appreciate his vast knowledge and skills in all things is, and how quickly he is able to see multiple solutions to a problem. Special thanks to my faculty reading committee: Gabriel Elkaim, Renwick Curry, and Dejan Milutinović. Thank you for your countless hours of proofing and editing this thesis. ASL, thank you for keeping things light and always enjoyable during our Wednesday meetings.

I would like to thank Liam Pedersen and Terry Fong for giving me the opportunity to contribute to the research done at IRG, and to the folks at IRG– thanks for the technical support, the company, and keeping me entertained in the Pirate Lab (Oleg, Zak, Yoon, Taemin, Ara, Lorenzo).

Thank you family for supporting me always. To my best friend, solid rock, and partner whose love and encouragement makes me be a better person. Theresa, you are the reason I strive for excellence.

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Since the beginning of time, we humans have always been curious of our surroundings– always exploring the "new" frontier. We traveled by foot to find hunting grounds, rode on camel backs to discover ancient civilizations, sailed the oceans to find new lands, traveled by wagons through Western North America, submerged underwater to the depths of the ocean floor, and used rockets to explore outer space. The common theme in all of this, is the act of moving to discover our surroundings. We need to transport ourselves, either physically or remotely, in order to familiarize and understand what is around us.

The majority of Earth exploration can be done with our own two feet. However, exploring space and other planets can not be done so easily. Exploring these areas are mainly done remotely with satellites, landers, and rovers. Between 1957 and 1975, the former Soviet Union and the United States competed with each other for supremacy in space exploration by trying to be the first in all things space related. The United States won this race by being the first and only nation to send man to the moon on Apollo 11 in July 1969. In November 1970, the Soviet Union successfully landed Lunokhod 1, a remote controlled vehicle, to became the first to send and successfully land a rover on a celestial body. In July 1971, the United States

followed behind with a manned Lunar Roving Vehicle (LRV), a four-wheeled rover capable of holding two astronauts. The LRV was driven by an astronaut and was primarily used to extend the range of their activities on the moon [11]. The United States' accomplishments with lunar human exploration are great, however the risk of human life was always present. Launching astronauts to exo-planets and returning them back to Earth requires more resources than sending unmanned rovers one way to exo-planets. Exploring distant planets further increases the risk of human life, and therefore the need for rovers to explore is the safer approach.

Unmanned rovers have been used by the Soviet Union's Luna program to explore the Moon by moving across the surface, but was ineffective compared to the LRV of the Apollo program. The need for unmanned rovers became more apparent when deciding to explore Mars because there is no vessel capable of sending a flight crew to Mars and returning them safely to Earth. There have been 9 successful rover landings in all: 5 lunar (Lunokhod 1('69) & 2('73), LRV 1-2('71) & 3('72)) and 4 Martian (Mars Pathfinder Sojourner('96), MER Spirit & Opportunity ('03), MSL Curiosity('12)).

Over the past 40 years, all exo-planetary rovers have been tele-operated: a human controller commanding the rover to "drive forward 20 turns" or to "drive to this location." Since the Mars Pathfinder (1996), all exo-planetary rovers have some level of autonomy implemented in the form of hazard avoidance systems to navigate through unknown terrain [1]. In January 2004, the Mars Exploration Rover Spirit was the first exo-planetary rover to achieve autonomous driving and path planning to target destinations between 40-50m away[2, 17]. Within the past decade, research in autonomous navigation and path planning has focused on rover traversability over rough terrain [3, 4, 13, 14, 16, 18, 23, 24, 25, 26, 29, 30].

The study of rover speeds over rough terrain is important because the faster the rover can travel over terrain, the larger the area that can be discovered within a specified

time. This results in mission efficiency and more scientific information to study. All past NASA exo-planetary rovers have operated autonomously at a maximum speed of $0.05 - 0.09m/s$ $(0.11 - 0.20mph)$ [20, 21]. This speed is considered slow and is mainly due to communications lag through long distances in space. The communications lag slows down operations and requires the operator to drive the rover slowly. For this reason, there is a research push to operate rovers with greater autonomy and to increase their speeds to $3m/s$ on flat hard ground. This is considered fast for exo-planetary rovers because it is more than 30 times faster than previous rovers.

The fastest research rover, the NASA Ames Research Center (ARC) K10 platform for technology development, has a top speed of $\sim 0.9m/s$ on flat hard ground [5]. There is a constant strive to increase its top speed and to know in advance a safe speed to travel over rough terrain without crashing, damaging on-board electronics, and minimizing vibration effects on sensors. Two such systems used to aid in this research advancement are the NASA Ames Research Center's (ARC) KRex and the NASA Johnson Space Center's (JSC) Centaur2. Both of these rovers are Human Robotic Systems (HRS) that aid human exploration in reconnaissance of exo-planetary terrain.

1.2 Purpose of this Thesis

In autonomous mobile robotics, traversing through unknown terrain has generally been approached using an obstacle detection algorithm [28], wheel contact forces [15], and roughness cost functions [4, 6]. The purpose of this thesis is to present a novel method that creates a speed map and provides planetary rovers with local and global information about the terrain. The information will allow the rover to safely travel over both smooth and rough terrain. In addition, this information enables a new way to path plan; it not only provides hazard detection and terrain roughness, two elements important to path planning, but also allowable speed over terrain.

The algorithm for the speed map models the terrain with incoming sensor data

and use this information to calculate surface roughnesses. The surface roughness provides a recommended speed limit for the terrain with the goal to drive slowly over rough terrain (i.e.: rocks, ditches, peaks) and faster over relatively flat terrain. This speed map provides better path planning information and will also aid the control operator by illuminating areas of hazards while in tele-operation mode.

1.3 Thesis Outline

Speed Map for Autonomous Rovers Over Rough Terrain has seven chapters and four appendices. Chapter 1 introduces the background and motivation for this work.

Chapter 2 provides information on related work to traversability/roughness and speed control. Much of the work done with traversability/roughness has been centered around obstacle/hazard detection. With regards to speed control, this is a new area of research where reactive speed controller, fuzzy logic, and supervised/non-supervised machine learning have been used.

Chapter 3 describes the experimental setup (KRex rover) and the system in place for sensing and creating the speed map. It also covers grid maps and terrain roughness, fundamental concepts that are built upon in this thesis.

Chapter 4 details the speed map algorithm and is broken into three main sections. The first describes a way to efficiently model terrain from sensor data while using few resources and operating in real-time. The second explains the algorithm to define roughness. And lastly, the third section presents the recommended speed.

Chapter 5 and Chapter 6 show simulation and experimental results of the speed map method respectively. The algorithm was first coded in Matlab and tested using synthetic terrain. Afterwards, the Matlab code was ported to C and implemented inside the rover navigation software (rovernav) where it was tested using real sensor data from various field test sites.

Chapter 7 concludes the thesis by summarizing the results and contributions of this work, and further discusses potential future work as well as ways to continue this research.

Appendix A contains equations, Appendix B contains simulation results in table form, Appendix C contains code used for simulation and experimental results, and Appendix D contains the source code.

Chapter 2

Related Work

Research defining terrain roughness is a rich and well established area [3, 6, 10, 12, 13, 15, 19, 24, 28]. However, research in relating speed and roughness of terrain (with the assumption of no *a priori* maps and using only range scan sensor data) has been a fairly new field within autonomous mobile robots. The majority of this research has been influenced by the Mars rovers and the 2004/2005 DARPA Grand Challenge (DGC) [1, 4, 8, 18, 25, 26, 27, 29, 30]– autonomously controlled automobiles tasked with traversing hundreds of miles of desert terrain. In this chapter we will review related research within traversability and speed predictions.

2.1 Traversability/Roughness

Previous planetary rovers such as the Mars Exploration Rovers (MER) and the Mars Science Laboratory (MSL), have used cameras and stereo vision to detect obstacles/hazards and categorize terrain as traversable or not traversable [10, 19]. The subject rover, KRex, is a LiDAR based rover that uses Probabilistic Terrain Analysis (PTA), which is a fast and lightweight obstacle detection algorithm developed by Thrun et al. [28]. PTA is very useful for detecting obstacles to avoid. However, it is a binary detection algorithm and therefore can only be associated with two speeds: 0 and max velocity. Coupling this algorithm with the speed map could greatly improve rover speeds and mission efficiency, and add granularity to the path planning algorithm.

Hoffman and Krotkov [12] measured terrain roughness from prior elevation maps and made the assumption to measure roughness along specific directions of surfaces. They calculated surface roughness using variance of height, variance of slope, and structural similarity factor after filtering data through a Blackman window [12]. Similarly, El-Kabbany and Ramirez-Serrano [6] computed roughness based on terrain height from a range camera, however they do this with no *a priori* knowledge of terrain. The work done by Iagnemma and Dubowsky [15] focused on force analysis and wheel contact angles based on the type of terrain (i.e.: dry sand, soil, rock) to maximize wheel thrust and minimize slip. However, this approach results in very slow vehicle speed and assumes that both terrain and path are known. They defined terrain roughness (r) at global Cartesian coordinate (x, y) as the square root of the variance of all elevation points inside the convex hull, defined by the wheel-terrain contact points of the robot on flat ground. Each of these works looked only at the height of the local terrain, except for [12], which only gives limited information about the terrain height and its surroundings instead of the surface. Other work done within the DGC considered the importance of adapting speed to rough terrain [4, 25, 26, 27].

2.2 Speed Control

Estimating a safe speed to traverse through unknown and unstructured terrain is a relatively new area of research. The problem of estimating allowable speed over rough terrain is innately non-linear and difficult to estimate because rover vibrations are spatially dependent on previously traversed terrain and speed. Also, exoplanetary terrain is highly unpredictable and traversing over a particular section of terrain contains an infinite number of possible variations in terrain and paths. Three different strategies will be presented here: fuzzy logic, reactive control, and machine learning.

One of the approaches to speed control is to utilize fuzzy logic. Fuzzy logic was

developed during the 1960's, and has been used for systems that are complex and difficult to define exactly. A number of researchers, Howard and Seraji [13], Jin et al. [16], Seraji [23], have used this approach to control rover navigation and speed. This approach attempts to resemble human reasoning, where the controller is adjusted manually until the output speed matches the users preference. This approach is not adaptable to other rover systems and is impossible to tune for exo-planetary conditions since the terrain is random and unknown.

Castelnovi et al. use a reactive speed approach with four roughness calculations and changed the vehicle speed based on a laser line scan of the ground immediately in front of the vehicle. This approach is able to predict how fast to drive within its immediate line scan. This approach cannot be used to predict distant terrain speeds or be used for path planning. Stavens et al. [26], creators of *Stanley* (winner of the 2005 DGC), used a reactive method to control the speed of an automobile based on vertical acceleration (shock) of the vehicle. They set an acceleration threshold, which was determined using machine learning, to prevent damage to the vehicle and its electronics. This approach requires a training set to categorize rough terrain and smooth terrain, in which a person defines what is rough and what is not by driving over the training set. If *Stanley* were to drive over hazardous/rough terrain that exceeded the shock threshold, damage to the vehicle and electronics would have ensued before it is able to slow down.

Since the 2005 DGC, Stavens and Thrun [25] have updated their reactive speed control to predict and identify future shocks, which classify regions as rough/not rough through unsupervised learning. It then used this classifier along with the training set obtained from the DGC as its inference to trigger decelerations used in prior work. This method is still based on using a training set to supervise the learning. This approach will not work for exo-planetary systems because extraterrestrial terrain is unknown, and the risk of learning through failures would result in a damaged rover which is impossible to fix when not on Earth.

Another DGC inspired approach comes from team CalTech's *Alice*: elevation to speed limit conversion. Cremean et al. [4] fused three range scan sensors to create an elevation map and averaged the terrain height per cell. For each cell, a Gaussian filter is applied to it's neighboring cells to create a roughness metric. This roughness metric is then passed through a sigmoid function where the coefficients are heuristically tuned by comparing sample filter responses to real-life situations. This approach works well but requires supervised learning, the same problem as *Stanley*. It also lacks additional information about each cell and lacks a contiguous relationship with its neighboring cells.

The works relating to roughness/traversability do not look at the surface of the terrain and instead focus on relative heights and directional slopes. Both of these metrics are useful, however they lack a relationship of the surface with the local surroundings. Additionally, much of the prior work in speed control requires heuristic tuning to generate information on surface roughness and speed. These methods are not adaptable to other systems, cannot physically be tuned because exo-planetary terrain is unknown and difficult to define rough terrain, or cannot risk the chance of learning through failures that would result in a damaged rover. This leads us to pursue a new method that will look at the surface and define a roughness that contiguously relates the surrounding terrain. Because of the contiguous surface relationship with the local surroundings, we can transform this roughness into a speed.

Chapter 3

Background

Driving after many years of experience becomes an instinctual task; yet it requires quick reflexes and attention to avoid hazards on the road such as bumps, pot holes, rocks, road kill, wood planks, furniture, or other cars. Processing our environment (day/night, weather, winding mountain roads/straight city roads) and surroundings (road condition, other vehicles/pedestrians) is a simple task for the human brain. For a robot/computer, however, this task is quite complex.

In order to validate the algorithms for surface roughness and allowable speed– experimental data is required. As shown in the literature review in Chapter 2, much of the prior work requires heuristic tuning to generate good information on surface roughness and speed. Our training data comes from the NASA rover KRex.

3.1 KRex System

The system used for this thesis is NASA Ames Research Center's KRex rover (pictured in Fig. 3.1). Our algorithms have also been tested on NASA Johnson Space Center's Centaur2. KRex is a four-wheel drive all-wheel steering rover with a top speed of $3[m/s]$. It is $148.5ft^3$ $(4.22m^3)$, and has a modular mast with sensors attached (see Fig. 3.2).

When humans drive, determining what speed to drive is intuitive because we have

Figure 3.1: KRex

years of experience and have learned what rough roads are. However, the number of car crashes indicates that we are not perfect. Our eyes give us feedback on both the near fields and far fields and we process this information to determine the presence of obstacles in our path. The KRex rover uses a Velodyne LiDAR to detect path conditions and obstacles. It has 32 beams that provide simultaneous range measurements from -30° to +10° from horizontal body axis, with a complete 360° scan occuring at $10Hz$ [22]. The Velodyne sensor has a range of 70 meters and produces $700,000$ points per second. When accumulated, these points form a 3-D point cloud which is registered to a global grid map using a position and orientation (pose) estimate. After the point cloud data (PCD) has been updated with pose, the PCD is considered aligned. These pose updates are based upon compass, inclinometer, GPS, and Inertial Measurement Unit (IMU) measurements as well as matching points to their corresponding plane. In this thesis it is assumed that we already have aligned PCD.

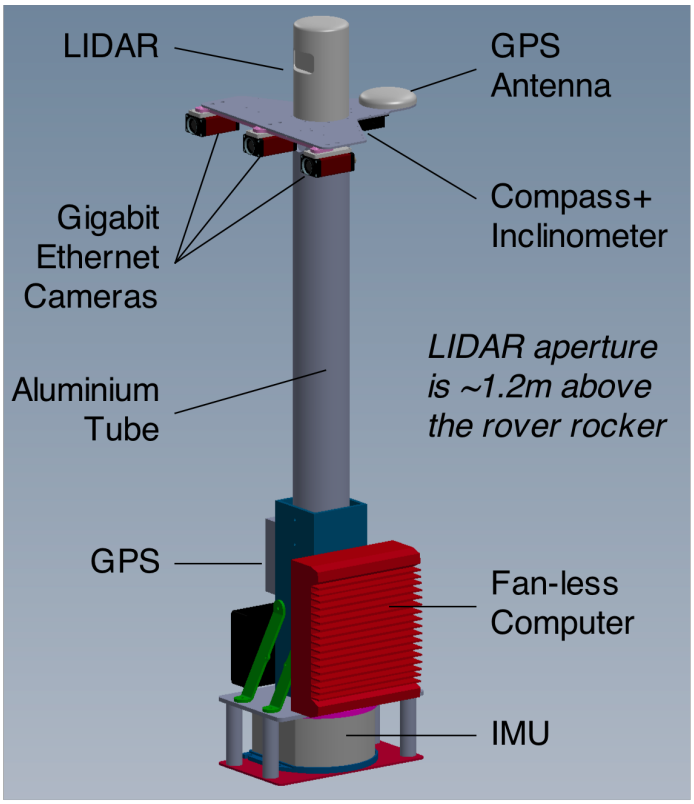


Figure 3.2: Navigation sensor mast, installed on Centaur2 and KRex

3.1.1 Grid Map

The grid map (similar to an occupancy map) maps the environment as an array of cells, with cell sizes ranging in size from 5cm to 50cm depending on the application requirements. For occupancy maps, each cell holds a probability value for likelihood of being occupied. This technique makes no assumptions about the type of feature that is occupying the cell. Fig. 3.3 is an example of an occupancy map for obstacles/hazards on a road. The KRex rover currently implements PTA [28] which is an obstacle detection algorithm that utilizes an occupancy map. For this thesis, we improve upon the PTA and use a grid to store accumulated statistics for each cell. Each cell of the grid contains the estimates of surface roughness and speed. Using grid representation helps discretize the environment uniformly and organize the sensor data.

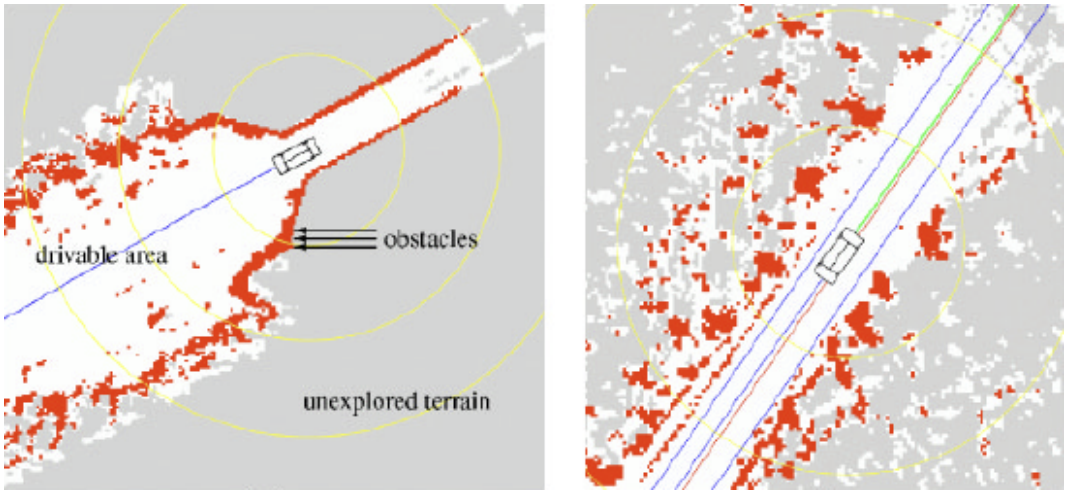


Figure 3.3: Occupancy map of road, where red dots are considered obstacles [27]

3.2 Terrain Roughness

Before creating a speed map, we must first define terrain roughness. Roughness can be defined as differences on a surface relative to its local surroundings (via rotation or translation). According to Hoffman and Krotkov [12], roughness measurements must have the following qualities:

(1) Must discriminate between surfaces of different amplitudes, frequencies, and correlation.

(2) Be an intrinsic property of the surface, invariant to direction of travel

(3) Be a local, not global measure of the surface.

(4) Have intuitive or physical meaning.

Even though this definition was suggested in 1989 [12], most of the later work in creating roughness calculations failed to meet all of the above criteria. The literature review on roughness/traversability in Chapter 2, only considered height differences and slope calculations which lack a contiguous relationship of surfaces within the local frame.

Roughness should also include the vehicle's obstacle height, which is the wheel radius, to non-dimesionalize and scale the metric to the robot size. For example, the final roughness threshold for a monster truck with 5 foot wheels will care less about smaller roughnesses than a 4-door compact sedan with 14in wheels. The next chapter will describe and show the roughness measurement used for the speed map in this thesis.

Chapter 4

Speed Map

This chapter discusses the development of the speed map in three parts: 1) Terrain Model Representation, 2) Roughness Calculation, and 3) Speed Recommendation.

4.1 Terrain Model Representation

The first step in creating the speed map is to recast the point cloud data (PCD) into a form that will best model and represent the terrain. Previous work, as stated in Ch. 2, used the PCD directly to represent the terrain and compute a roughness metric. The KRex system uses this terrain model representation for two purposes: 1) creating a global map and 2) calculating a speed map. With over 700,000 points per second coming from the Velodyne, storing the PCD requires a massive storage unit with its attendent increase in payload. Since the KRex system is designed for off-planet missions, payload must be minimized and storage space must be limited. Therefore, we store the statistics into a matrix (Eq. 4.4) for each cell of a given terrain– known as a growing set; this integrates seamlessly into the grid map infrastructure. Below, in Fig. 4.1, is an example grid map used for the KRex system where the tile size is the discoverable region around the rover as it traverses through the environment.

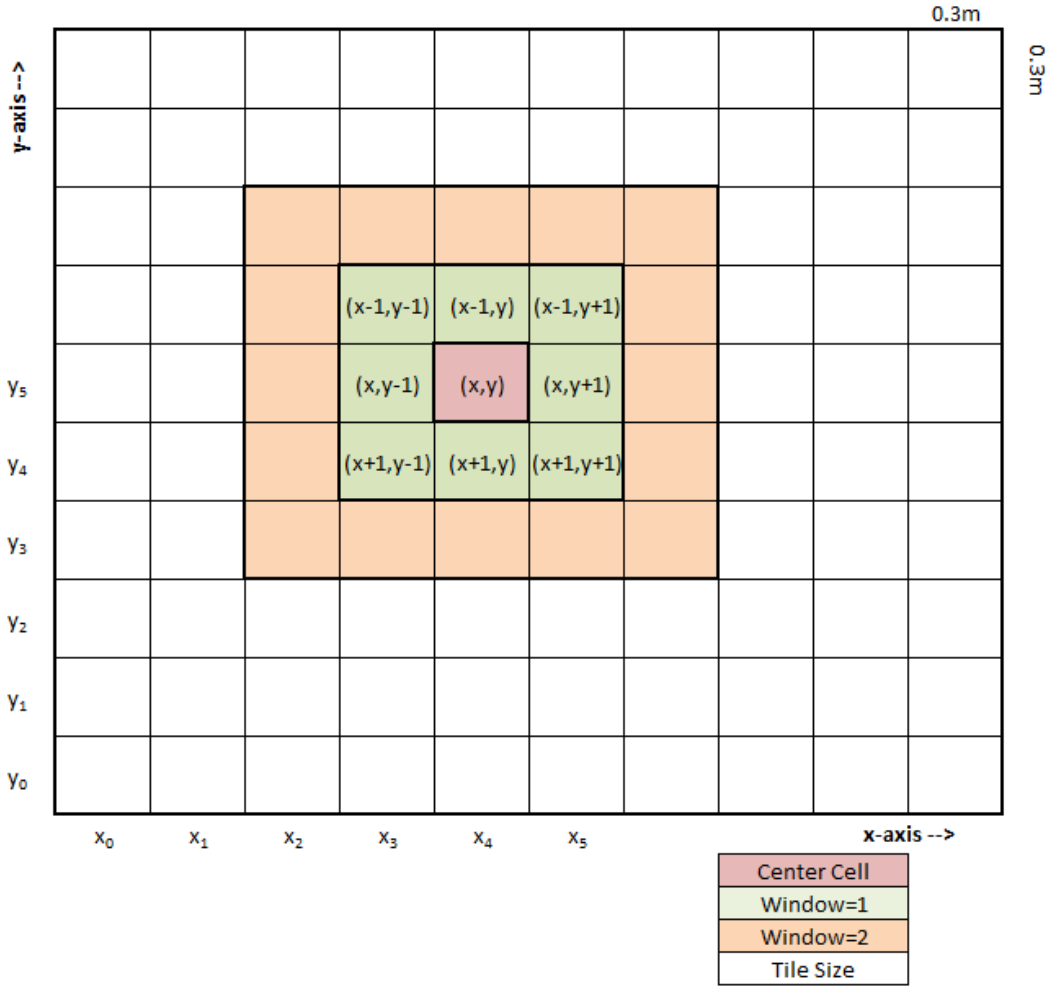


Figure 4.1: Grid Map showing Tile Size, Cell Size, and Window Sizes where Window=1 is in green and includes the Center Cell; Window=2 is in orange and includes Window=1 and Center Cell

The accumulated statistics for each cell are used to compute a least squares fitted plane, normal vector, and other information in real time as new/updated terrain data arrives.

First, we begin with the equation of a plane where (x, y, z) are the global Cartesian coordinates of each point within the PCD:

$$z = Ax + By + C \tag{4.1}$$

Next, minimize the sum of squared errors:

$$E(A, B, C) = \sum_{i=1}^{n} [(Ax_i + By_i + C) - z_i]^2 \tag{4.2}$$

The LS best fit plane is satisfied when the gradient of $E(A, B, C)$ is equal to 0:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \nabla E = 2 \sum_{i=1}^{n} [(Ax_i + By_i + C) - z_i] \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{4.3}$$

That can be written in $Fx = y$ form, i.e.:

$$\underbrace{\begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix}}_{F} \underbrace{\begin{bmatrix} A \\ B \\ C \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix}}_{y} \tag{4.4}$$

The plane parameters are found by taking the inverse of matrix F:

$$x = F^{-1} y \tag{4.5}$$

Once the LS best fit plane is determined, the surface normal is easily derived from the plane equation:

$$N = \frac{(-A, -B, 1)}{||(-A, -B, 1)||} \tag{4.6}$$

Where $||(-A, -B, 1)||$ is the length of the surface normal vector:

$$||(-A, -B, 1)|| = \sqrt{A^2 + B^2 + 1^2} \tag{4.7}$$

The accumulated statistics method is used, as opposed to Cremean et al.'s approach of averaging terrain heights [4], because the LS fit plane and surface normal give additional insight about surface direction, and provide a contiguous relationship between neighboring cells when computing terrain smoothness (roughness).

4.2 Roughness Calculation

This section introduces a new metric to define roughness (r) using surface normals and blending in other known metrics to fine tune and compensate for unusual cases. Roughness is defined as the compliment of the sum of products of smoothness cost functions ($s_{(.)}$): $r = 1 - (\Sigma\Pi s_{(.)})^{\psi}$.

Note: all incremental smoothness cost functions (denoted as $s_{(.)}$) have a value between 0 and 1, where 0 equates to rough and 1 equates to smooth. The roughness value (r) has a value between 0 and 1, where 0 equates to smooth and 1 equates to rough. Smoothness and roughness are complimentary metrics that can be used interchangeably to define each other.

4.2.1 Surface Normals (N)

We use CalTech's *Alice* framework of a center cell and it's neighboring cells within a window size to create a contiguous relationship with the local surrounding. In the *Alice* design, the average PCD heights within each cell are used to represent the terrain and roughness [4]. For our method, fitted planes and surface normals computed from the PCD are used to represent the terrain and smoothness.

From the surface normals (N), a smoothness scale (s_N) is created by taking the dot products of the unit normals from the center cell and its neighboring cells (Eq. 4.8) and applying an averaging filter over a given window size, K, where each side of the window has a length of $2K + 1$ cells (see Fig. 4.1).

$$s_N = \left| N_{(x,y)} \cdot N_{(x+i,y+j)} \right| \tag{4.8}$$

$$s(x,y) = \frac{1}{n_{cells}} \sum_{i=-K}^{K} \sum_{\substack{j=-K \\ (i,j)\neq(0,0)}}^{K} s_N \tag{4.9}$$

Where $n_{cells} = (2K+1)^2 - 1$ is the number of cells inside a window size minus

the center cell and (x, y) are Cartesian coordinates of a global grid map. The relationship focused upon is the θ from the dot product, $\frac{a \cdot b}{||a||||b||} = cos\theta$. This tells us how much the neighboring cell planes are rotated relative to the center cell. If the unit normals are parallel (the same), then the dot product is 1 ($\theta = 0$) and the planes are identical. If the unit normals are perpendicular, then the dot product is 0 ($\theta = \pi/2$) and the planes are rotated 90 degrees from each other. The range of the dot products absolute value is between 0 and 1, and is used as the smoothness cost function, s_N, in Eq. 4.8.

Using the dot product of the surface normals as a smoothness metric provides better intrinsic information about the terrain compared to [4] and [15]. This approach looks at the surface and calculates surface differences between cells while maintaining a contiguous relationship within its local window. In [4] and [15], they use cell heights which have no relationship between each other besides above or below the center cell. However, this is still important information and will be used to supplement the surface normal approach.

4.2.1.1 Cell Size

The cell size determines the resolution of the modeled terrain. The smaller the cell size the higher the resolution and correspondingly, more computations are required to process all cells within tile frame. If the cell size is too small, then there is a higher chance of having no PCD within the cell which would result in a high number of unknown cells. It would also cause PCD to be sparse within each cell, which would create fitted planes with poor confidence. The software module in KRex responsible for pose estimation, iterative closest point (ICP), and point to plane matching requires the cell sizes to have a minimum size of 30cm. Due to the this system constraint, the speed map will also use a 30cm cell size.

4.2.1.2 Window Size

The window size determines the region of area that will be used to associate neighboring cells to the center cell through the dot product of surface normals. See Fig. 4.1 for window size reference. The minimal window size is based upon the wheel diameter of the rover ($\approx 0.6m$). With a cell size of $0.30m$, the window size of $K = 1$ covers a length of $0.9m$– which is larger than the wheel size. Theoretically there is no maximum window size, however the larger the window size the more computations required and thus the slower the algorithm. A window size of $K = 3$ covers a length of $2.1m$ which encompasses slightly more than the entire rover.

4.2.2 Adjustments within Cell

When fitting a plane to the PCD within a cell, it sometimes contains bumpy terrain (see Fig. 4.2), and the resulting plane will not capture the terrain shape accurately. In these cases, the residual is used as a smoothness and confidence factor for the cell. The spread of the PCD along the surface of the terrain can sometimes look like a line (see Fig. 4.4) and result in poorly fit planes. The coverage measures the spread of the PCD and uses it as a confidence factor. The slope cost uses the slope of the plane and incorporates the rover's roll and pitch limitations to calculate the traversability of the cell and uses it as a smoothness factor. This subsection will detail adjustments using residual, coverage, and slope cost functions for the smoothness metric within a cell.

4.2.2.1 Residual

The residual is the sum of the squares of the errors (or offsets) of the points from the plane (variance of plane errors). As the measurement is the z_i's, the offset is in the vertical dimension. This is used to test the quality of the plane fit along the z-axis (Fig. 4.2).

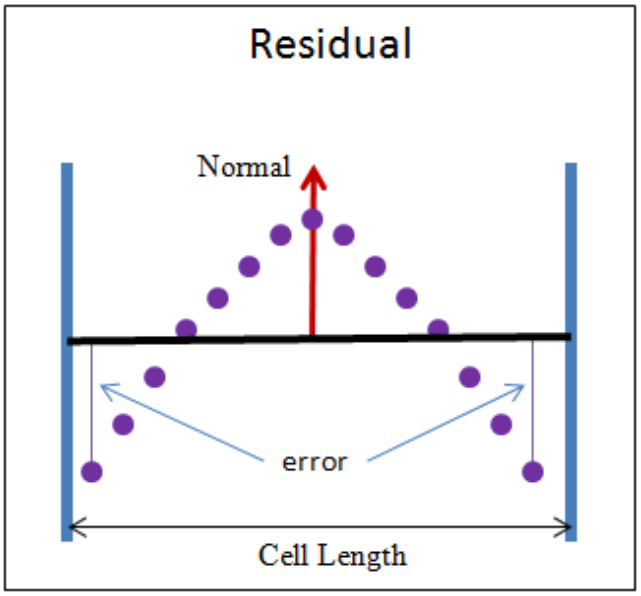


Figure 4.2: Residual of a bump

The normalized residual for each cell, (x, y), is found using the following equation:

$$residual = \frac{1}{n}\sum_{i=1}^{n}(z_i - (Ax_i + By_i + C))^2 \tag{4.10}$$

$$\begin{aligned} = &\frac{1}{n}(S_{zz} + A^2S_{xx} + B^2S_{yy} + C^2 \\ &+ 2(A(CS_x + BS_{xy} - S_{xz}) \\ &+ B(CS_y - S_{yz}) - CS_z)) \end{aligned} \tag{4.11}$$

Where n is the number of points and (A, B, C) are the plane coefficients in the associated cell (x, y) and the x_i, y_i, and z_i's are the individual points within that cell. This residual cost will be used as a weighting function in the total smoothness cost, using a logarithmic scale of the residual:

$$q = \frac{log_{10}(residual^{-1})}{2.6021} \tag{4.12}$$

$$residualCost = \begin{cases} 0 & \text{if } q \leq 0 \\ 1 & \text{if } q \geq 1 \\ q & \text{else} \end{cases} \tag{4.13}$$

Applying a logarithmic scale to the residual condenses the range, which makes con-

verting it into a cost function manageable. The scaling factor of $\frac{1}{2.6021} = \frac{1}{log_{10}(0.05^{-2})}$ is used to put a lower limit on the residual and defines that residuals less than $5^2 cm^2$ are well fit planes.

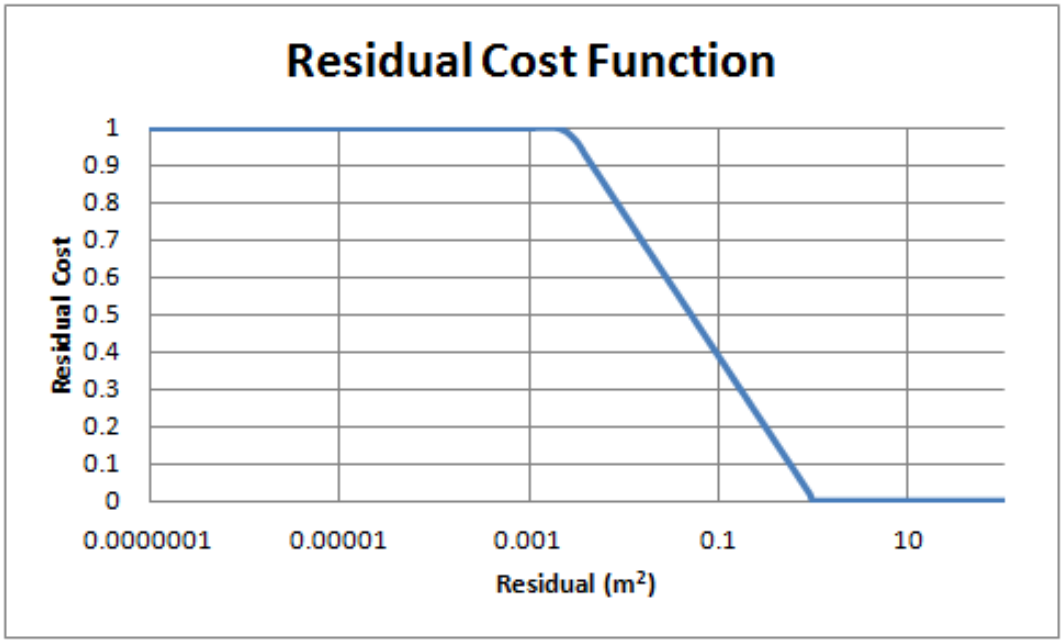


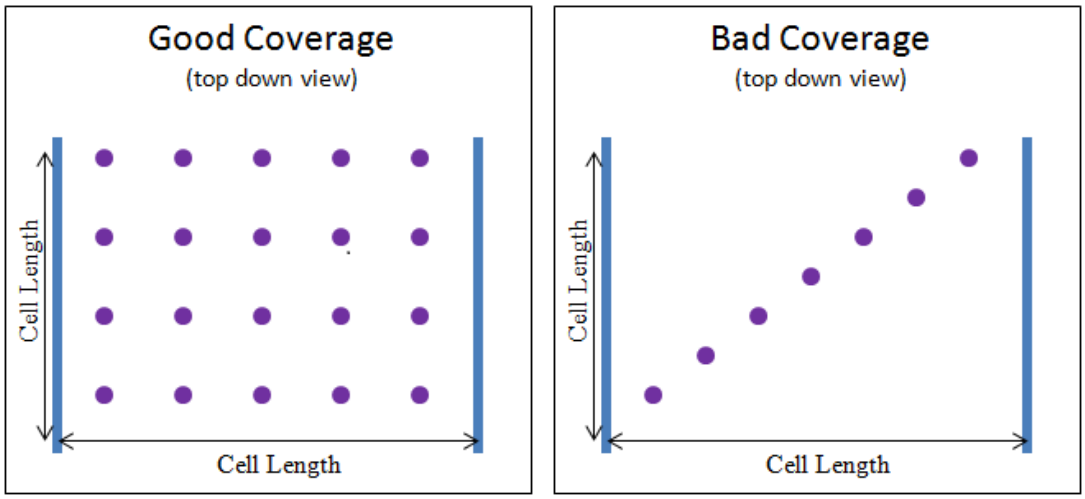
Figure 4.3: Residual Cost Function

Poorly fit planes are penalized with lower weights associated with lower speeds, and well fit planes are rewarded with higher weights associated with higher speeds.

4.2.2.2 Coverage

Coverage is how well the PCD is spread over the grid cell in the X-Y plane (see Fig. 4.4). Because the laser scan is a line scanner rotating 360 degrees, there is potential for the PCD to form a singular line across the grid cell. If this were to occur, the plane fitting would be based upon the singular line and would be a poor representation due to the lack of coverage.

In order to test the quality of the coverage, or planarity, the covariance method of principal component analysis (PCA) will be used. The covariance matrix ($c3x3$) of



(a) Good coverage (b) Bad coverage

Figure 4.4: Comparing (a) good spread v.s. (b) bad spread

each cell at position (x, y) is computed first:

$$c3x3_{(x,y)} = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix} \tag{4.14}$$

where the $cov(x, x)$ is the $var(x)$, which is the average of squared differences from the mean $(\bar{x})$ of each cell (x, y).

$$cov(x,x) = var(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n}S_{xx} - \left(\frac{S_x}{n}\right)^2 \tag{4.15}$$

where n is the number of points in the cell.

The covariance is the correlation between two variables and defines the linear dependence to each other. Next, the eigenvalues are determined via the quadratic equation from the determinant of the covariance matrix C.

$$det|C - \lambda I| = 0 \tag{4.16}$$

$$(C_{0,0} - \lambda) * (C_{1,1} - \lambda) - (C_{0,1} * C_{1,0}) = 0 \tag{4.17}$$

$$\lambda^2 - (C_{0,0} + C_{1,1})\lambda + C_{0,0} * C_{1,1} - C_{0,1} * C_{1,0} = 0 \tag{4.18}$$

Where C is a $2x2$ matrix

$$C(x,y) = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix} \tag{4.19}$$

and the eigenvalues are λ_1 and λ_2

$$a = C_{0,0} + C_{1,1} \tag{4.20}$$

$$b = \sqrt{4 * C_{0,1} * C_{1,0} + (C_{0,0} - C_{1,1})^2} \tag{4.21}$$

$$\lambda_1 = (a+b)/2 \tag{4.22}$$

$$\lambda_2 = (a-b)/2 \tag{4.23}$$

$$a \geq b \geq 0 \tag{4.24}$$

When testing for planarity, if either eigenvalue is less than or equal to ω then the PCD does not have good coverage of the grid cell. Where $(\lambda_{(.)} \geq 0)$ and $(0 \leq \omega < cellSize)$.

$$coverage = \begin{cases} 0 & \text{if } \lambda_1 \leq \omega \text{ or } \lambda_2 \leq \omega \\ 1 & \text{else} \end{cases} \tag{4.25}$$

Experimentally, $\omega = 0.0008$ gave acceptable results as the coverage threshold, but we leave it to further research to determine what threshold for ω is considered bad coverage with respect to *cellSize*.

4.2.2.3 Slope (ϕ,θ)

The slope is used to calculate traversability for singular cells. This incorporates the rover's roll (ϕ) and pitch (θ) limitations. The slope cost is independent of the surface normal cost, which is a purely local cost function (dependent on surrounding). Only the larger of the two ratios is required (eq. 4.26) since the larger ratio is the limiting

factor for the individual cell.

$$slope = max\left[\frac{|A|}{\phi_{thrs}}, \frac{|B|}{\theta_{thrs}}\right] \tag{4.26}$$

Where A and B are the plane coefficients, and ϕ_{thrs} and θ_{thrs} are the vehicle roll and pitch thresholds. Creating the slope cost function is easily done by taking the difference of a parabolic function.

$$s_{slope} = \begin{cases} 0 & \text{if } slope \geq 1 \\ 1 - slope^{\eta} & \text{else} \end{cases} \tag{4.27}$$

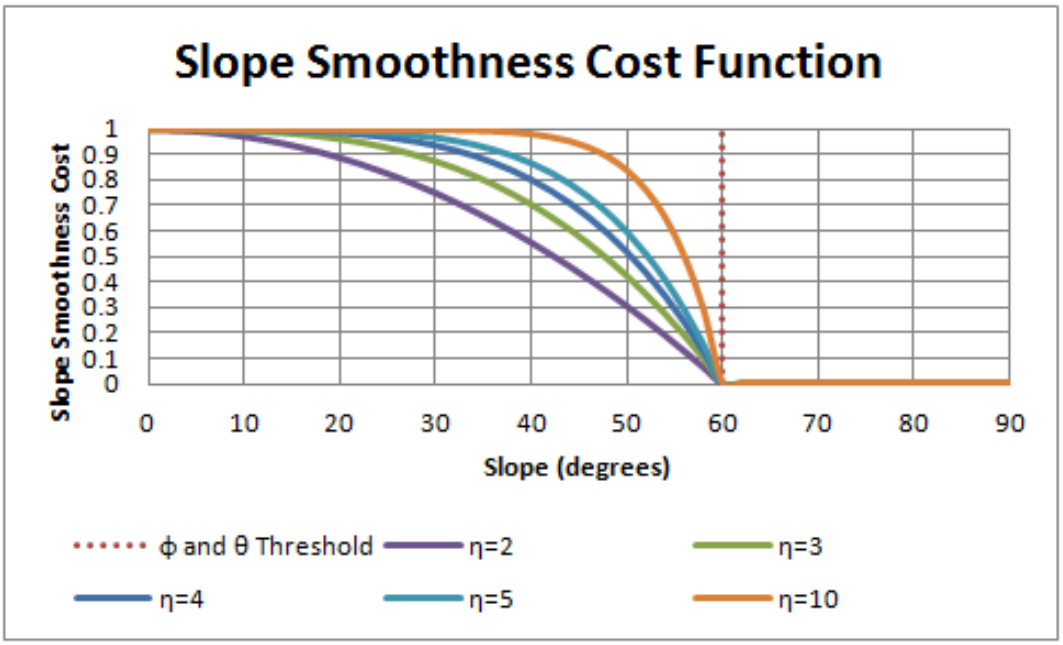


Figure 4.5: Slope Smoothness Cost Function with varying η's

Experimentally we determine $\eta = 4$ because the shape of the curve extended the s_{slope} value of 1 and has an equivalent smoothness cost of 0.5 at 50°. We leave it to further research to determine the best η to use. In Fig. 4.5, the slope smoothness cost is shown using varying η values of 2, 3, 4, 5, and 10.

The slope cost function is of the form in Fig. 4.5 because we want the rover to traverse at maximum speed over sloped terrain as long as the slope is less than the

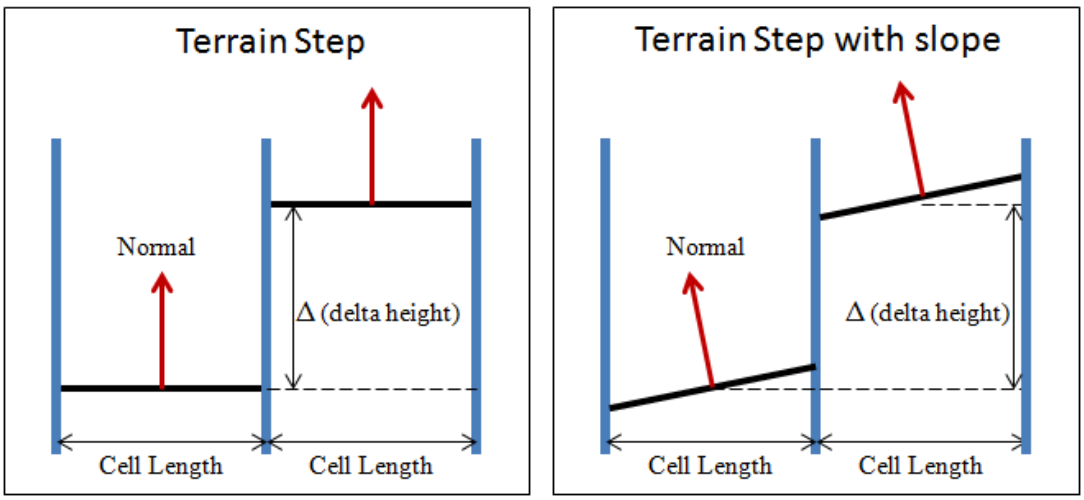
thresholds for rover pitch and roll.

4.2.3 Adjustments between Cells

The framework of our method provides a way to relate the local surroundings with the center cell. Using just the surface normals for the smoothness lacks information on terrain heights, which are important in detecting obstacles for the rover. The projected height is used to distinguish step terrain from continuously sloped terrain; incorporating the rover's obstacle height allows this metric to determine obstacles. The following subsection will detail adjustments using projected heights for the smoothness metric between neighboring cells.

4.2.3.1 Projected Height (μ)

The surface normal metric has difficulty distinguishing a step in the terrain if the step happened to be perfectly aligned with the grid map as seen in Fig. 4.6a and Fig. 4.6b.



(a) Terrain Step (b) Terrain Step with Slope

Figure 4.6: Step terrain (a) without slope and (b) with slope

In Fig. 4.6a and 4.6b, the surface normals are identical and therefore the planes are indistinguishable from each other. Using the mean height of the planes solves this

problem. The mean height μ is calculated as:

$$\mu = S_z/n \tag{4.28}$$

Where n is the number of points in the cell, and S_z is the summation of all the heights in the cell. By taking the difference between the center cell average height (μ_{center}) and its neighboring cell average height (μ_{nbr}), the Δ height is inversely proportional to smoothness. As the Δ height increases, the smoothness decreases.

$$\Delta = |\mu_{center} - \mu_{nbr}| \tag{4.29}$$

Using Δ gives a range of $[0 : \infty]$. Introducing the rover's obstacle height, which is the relative terrain height ($\delta = 0.3m$) the rover cannot drive over, into the equation will reduce the range to $[0 : 1]$. The mean height smoothness cost (s_μ) is then:

$$s_\mu(x, y) = 1 - min[\frac{\Delta}{\delta}, 1] \tag{4.30}$$

This mean height smoothness works well with step terrain, however for continuously sloped terrain this method will contain a rough cost related to the delta height (see Fig. 4.7). We want to prevent this rough cost since the terrain is smooth.

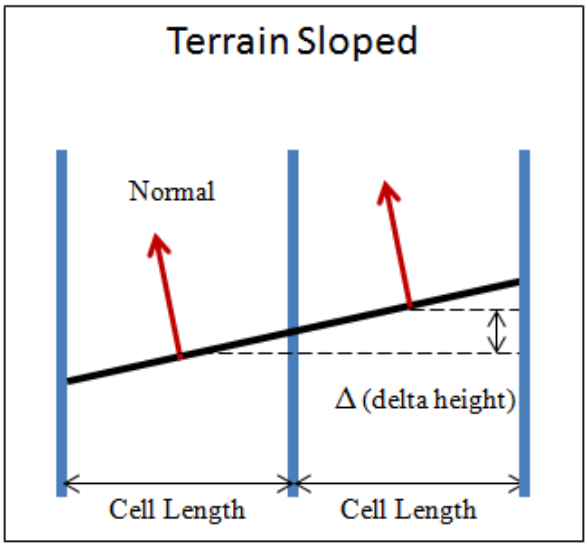


Figure 4.7: Terrain Sloped

To handle this characteristic, we extend the mean height cost to use a projected height of the plane (μ_{proj}) (see Fig. 4.8). Since the plane of each cell is centered at (0,0) and the cell size is known, finding the projected height is done by computing the height at the middle of the neighboring plane using the center plane equation as such:

$$\mu_{proj} = A_{center}(i * cellSize) + B_{center}(j * cellSize) + C_{center} \tag{4.31}$$

where i and j iterate through the neighboring cells. The neighboring cell height is found using the neighboring plane equation:

$$\mu_{nbr} = A_{nbr}(0) + B_{nbr}(0) + C_{nbr} \tag{4.32}$$

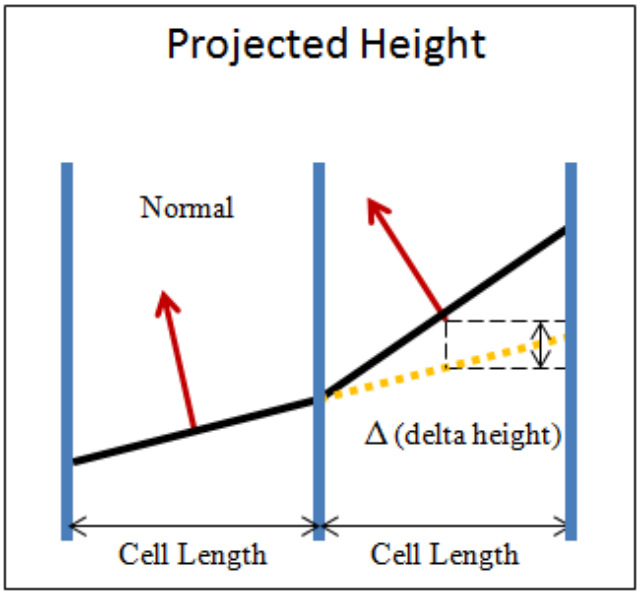


Figure 4.8: Projected Height

The new cost function is then:

$$\Delta = |\mu_{proj} - \mu_{nbr}| \tag{4.33}$$

$$s_{\mu}(x, y) = 1 - min[\frac{\Delta}{\delta}, 1] \tag{4.34}$$

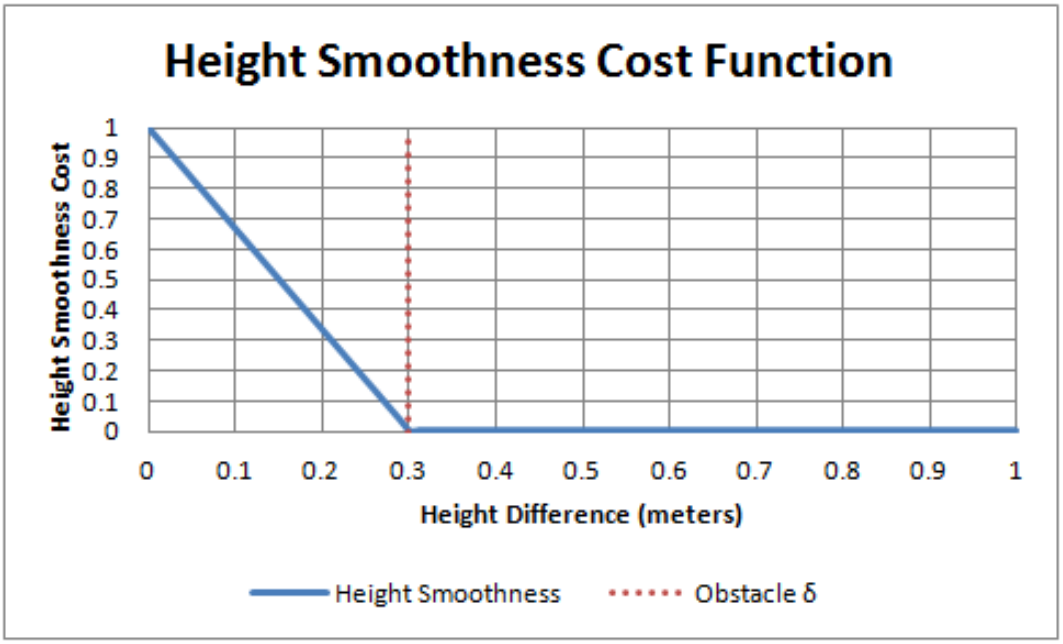


Figure 4.9: Height Smoothness Cost Function

Using projected height provides continuity between cells and checks if the terrain is a continuous slope or sloped step.

4.2.4 Total Smoothness Cost

The next step in the smoothness calculation is combining all the smoothness metrics together. To do this, a confidence factor (β) is introduced into the equation to help weigh the smoothness value between each cell more heavily for well-fit planes (low $residual$), good coverage (non-zero eig), and large cluster of points in cell(large $numPts$). Additionally, the center cell has its own confidence factor (γ) that will weigh the entire smoothness value more heavily for a well fit plane ($residualCost$), good $coverage$, and traversable slope (s_{slope}). This gives us our updated smoothness

equation $s_{center}(x, y)$:

$$s_{center}(x,y) = \frac{\gamma(x,y)}{\beta^*(x,y)} \sum_{i=-K}^{K} \sum_{\substack{j=-K \\ (i,j)\neq(0,0)}}^{K} (s_N * s_\mu * s_{slope} * \beta)(x+i, y+j) \tag{4.35}$$

$$\gamma(x,y) = residualCost(x,y) * coverage(x,y) * s_{slope}(x,y) \tag{4.36}$$

$$\beta^*(x,y) = \sum_{i=-K}^{K} \sum_{\substack{j=-K \\ (i,j)\neq(0,0)}}^{K} \beta(x+i, y+j) \tag{4.37}$$

$$\beta(x,y) = numPts(x,y) * residualCost(x,y) * coverage(x,y) \tag{4.38}$$

The computation for the smoothness skips over the center cell in the summation because the dot product and projected height of the center cell to itself will always result in a value of 1. This will slightly increase the smoothness value towards 1 especially when the cell is surrounded by perpendicular planes. Eliminating this calculation also slightly reduces computation at load. To reduce computation time even further, the smoothness of the center cell (x, y) is computed only if the number of points within the center cell is greater than 10 and the $coverage(x, y) \neq 0$, otherwise a default smoothness of 0 is used.

The inner term, $s_N * s_\mu * s_{slope}$, is computed to guarantee any smoothness cost equal to 0 ($s_N = 0$, $s_\mu = 0$, or $s_{slope} = 0$) will force the entire product to be rough. Additionally, if $\gamma(x, y) = 0$, the whole smoothness term ($s_{center}(x, y)$) is considered rough either due to $residualCost(x, y) = 0$, $coverage(x, y) = 0$, or $s_{slope}(x, y) = 0$. If $residualCost(x, y) = 0$ or $s_{slope}(x, y) = 0$, the terrain is untraversable due to bumpy terrain or slope exceeding the roll/pitch threshold. If $coverage(x, y) = 0$, the terrain is considered untraversable due to a lack of data quality and not actual terrain roughness. The data quality is included because for uncertain terrain we do not want the rover to traverse these areas.

Once the smoothness cost is computed for all the cells within the tile frame, the smoothness is smoothed using the neighboring smoothness' via the α term which is

bounded from $[\frac{1}{n+1} : 1]$ where n is the number of neighboring cells used.

$$s(x,y) = \alpha * s_{center}(x,y) + (1-\alpha) * s_{nbr}(x,y) \tag{4.39}$$

$$s_{nbr}(x,y) = \frac{1}{n} \sum_{i=-K}^{K} \sum_{\substack{j=-K \\ (i,j)\neq(0,0)}}^{K} s_{center}(x+i, y+j) \tag{4.40}$$

At the upper limit of $\alpha = 1$, the resulting smoothness ($s(x,y)$) will only contain the center cell smoothness (s_{center}). The lower limit is derived by setting the center term equal to the neighbor term. That is, the lower limit of α is $\frac{1}{n+1}$, which provides equal weighting among all the $(n+1)$ cells used within the window size including the center cell. The maximum number of neighboring cells within the window size is $n = (2K+1)^2 - 1$.

4.2.4.1 Smoothness Scaling

It is safe to assume terrain areas with low smoothness (large roughness) are untraversable and therefore these regions are penalized without adversely affecting obstacle representation. To accomplish this, the smoothness s is modified:

$$s'(x,y) = s(x,y)^{\psi} \tag{4.41}$$

Where $\psi > 1$. Raising the power of s greater than 1 maintains the range $[0,1]$ while increasingly penalizing terrain regions that are rough (see Fig. 4.10). This concept is taken from Iagnemma and Dubowsky [15]. Experimentally, we determined $\psi = 2$, but leave it for further research to optimize ψ for the mission requirements.

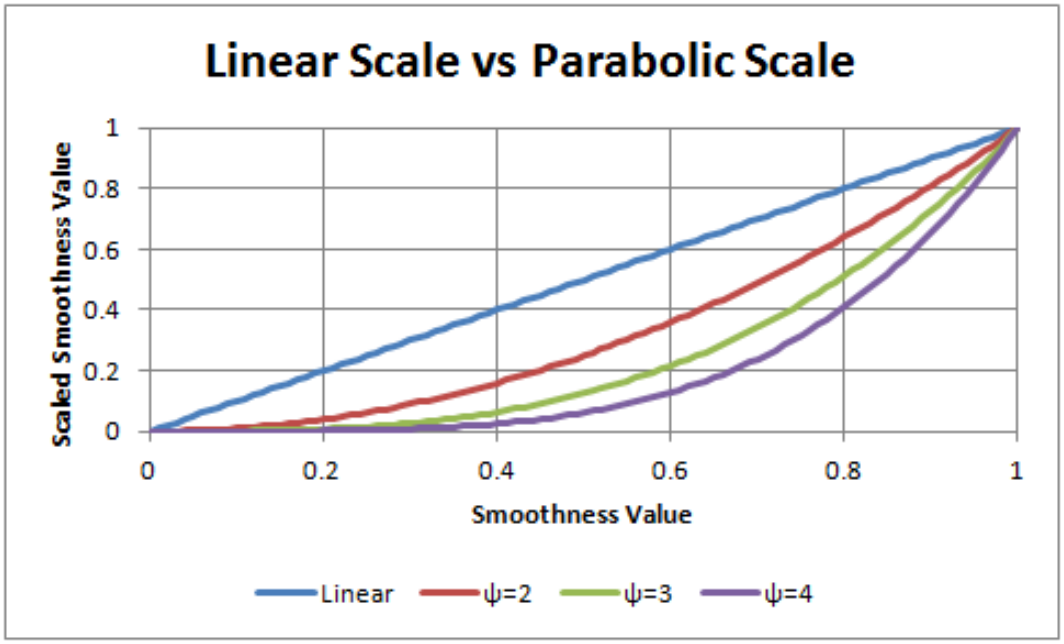


Figure 4.10: Smoothness Scaling: Linear vs Parabolic

4.2.5 Final Roughness Cost

The final roughness cost (r) is defined as the compliment of smoothness s', so that a roughness of 1 means rough, and a roughness of 0 means smooth.

$$r(x, y) = 1 - s'(x, y) \tag{4.42}$$

4.3 Speed Recommendation

While we have been concentrating on roughness, in the end, allowable speed is what is important. From our literature review in Ch. 2, much of the prior work requires heuristic tuning to generate quality information on surface roughness and speed. These methods are not adaptable to other systems and cannot risk the chance of learning what rough terrain is through failures.

Estimating allowable speed is a non-linear problem that is spatially dependent on the history of speed and position as well as the wheel-to-ground interaction [9]. The KRex rover is built as a rigid body with four wheels and no suspension system (see

Fig. 3.1), though the tires act as a passive suspension and can be modeled as a linear dynamical system (LDS) with a spring and damper. However, this is not done in our approach because suspension systems are complex and vary from rover to rover; we want our method to be adaptable to other systems, such as the JSC C2 rover. Additionally, modeling wheel-to-ground interaction is difficult when multiple wheel contact points occur with the ground.

We simplify the allowable speed prediction problem by linearizing wheel-to-ground interaction and suspension system model with the following assumptions. We replace the wheel-to-ground interaction with the residual cost function and the projected height (s_μ) cost function. Using these two cost functions allows the system to define "pot holes" and step terrain as rough, where multiple wheel contact points occur. We also replace the suspension system model with the dot product surface normal cost function (s_N) to detect surface changes that define the smoothness. The s_N function is used in the same way as comparing a vibration threshold to the acceleration of the LDS's step response to capture the roughness. The more the surface changes within a window size, the more vibrations occur in an LDS; in the same way the more rotated cell planes are relative to the center cell.

Terrain with the least amount of resistance (roughness) is assigned faster speeds. We scale the roughness metric defined in the previous section with the maximum rover velocity $v_{max}(\approx 3.0m/s)$:

$$v_r = v_{max} * (1 - r(x, y)) = v_{max} * s'(x, y) \tag{4.43}$$

The more planes that are similar to the center cell, the closer $r(x, y)$ approaches 0, and the faster the rover can drive over the terrain. However, the more planes that are disjoint from the center cell, the closer $r(x, y)$ approaches 1, and the slower the rover can drive over that terrain. The framework of the smoothness calculation allows the patch of terrain within a window size to have a contiguous relationship with

the local surrounding. This relationship is important because it allows the surface to be defined by a smoothness metric that integrates its local surrounding.

The approach to determine an allowable speed for diverse terrain (Eq. 4.43) is convenient and over simplified. It uses a naive model in order to make roughness an intuitive feel for the reader by linearly converting to speed. The idea is: the smoother the surface, the faster the rover can traverse. Speed over rough terrain is not unique to this work, however there is a reasonable way to get there. As a first cut, this linearization is acceptable but can increase in complexity to include vibration in future work. This method is adaptable, does not require supervised learning, and does not risk the damage of the rover.

Chapter 5

Simulations

This chapter will discuss and show simulation results using synthetically generated terrain. The simulations will test rover specific parameters and algorithm specific performance. The utility of simulation data is that it allows for a full end-to-end test of the algorithm, and call qualitatively anew performance.

Matlab is used to generate terrain, test, and verify the algorithms effectiveness/correctness. The synthetically created terrain include step terrain (Fig. 5.4) and ramp terrain (Fig. 5.1). No measurement noise was included in the created terrains. These generated patterns are used to simulate rocks and crater like terrain. The following sections will discuss the parameters of the algorithm and its result using synthetic terrain to illustrate the changes.

Note: The roughness metric (r) is in the interval $[0, 1]$, 1 being very rough terrain and 0 being very smooth terrain.

5.1 Rover Specific Parameters

5.1.1 Roll and Pitch Threshold

The roll (ϕ) and pitch (θ) thresholds, in the slope cost function, can be changed to meet rover specifications/limits. However, for simulation and testing purposes a 60°

threshold is used for both roll and pitch. See Fig. 4.5 for cost function graph and Section 4.2.2.3 for more details on cost function itself. Fig. 5.1 is an example ramp input used to test the roll/pitch threshold. All ramp inputs cover a $2m \times 2m$ area.

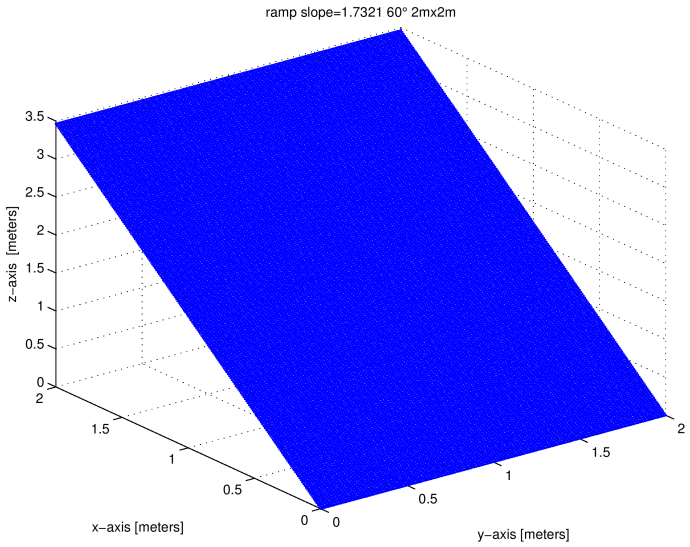


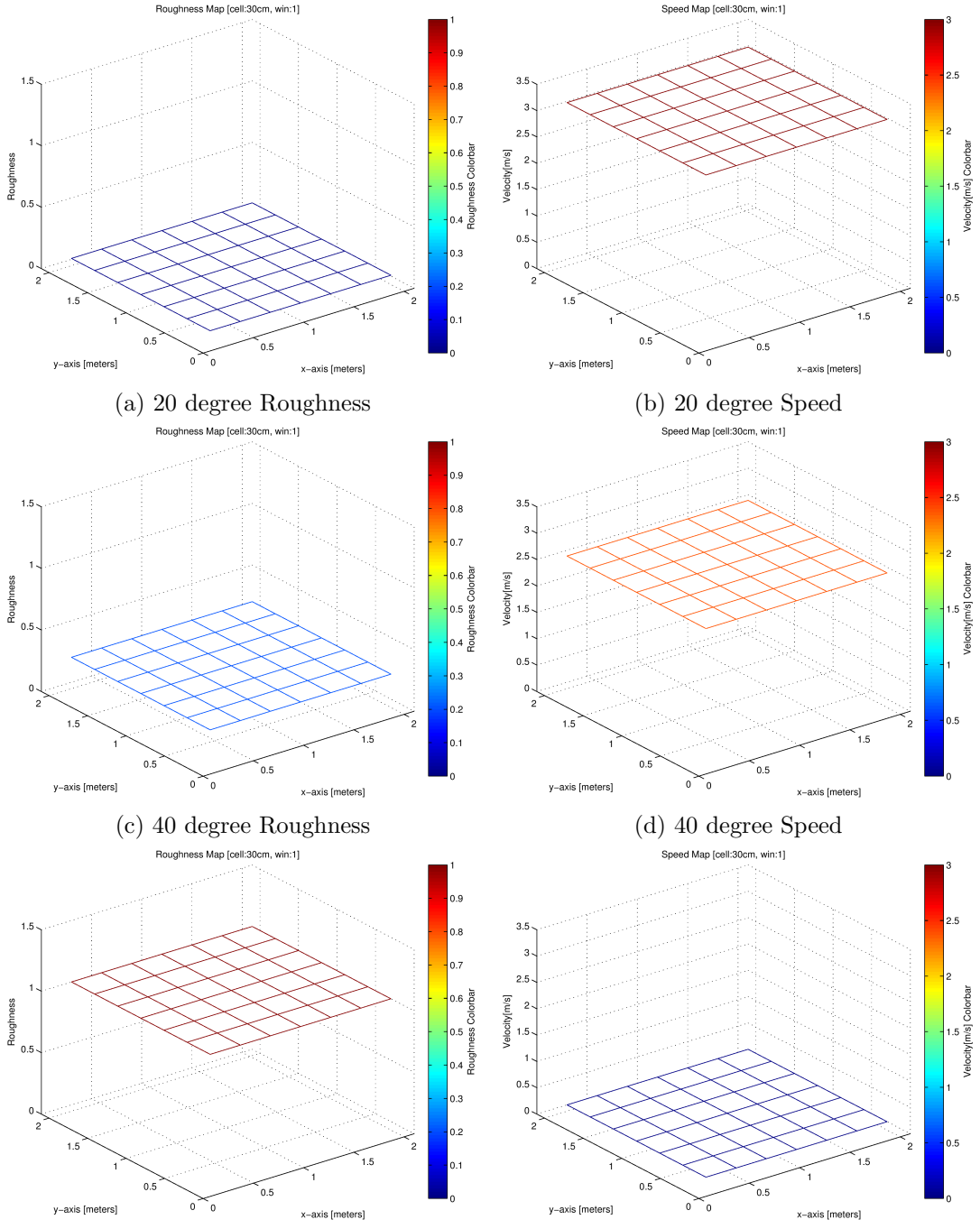
Figure 5.1: Ramp input with slope $= tan(60°) = 1.7321$ in a $2m \times 2m$ area

The ramp input simulations uses the following parameters:

Parameter	Value
Cell Size	$0.3m$
Window Size (K)	1
Alpha Weight (α)	1
Obstacle Height (δ)	$0.3m$
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Roll/Pitch Threshold (ϕ/θ)	60°

Table 5.1: Speed map conditions used for ramp input simulations

Below are three ramp input results that show that roughness and speed follow the slope cost function. All slopes are traversable at maximum speeds, however as the slope approaches the roll/pitch threshold, the roughness will increase exponentially towards 1 as seen in Tbl. 5.2 and Fig. 5.3. The uniform slope for the ramp inputs are essentially flat terrain rotated about the x-axis or y-axis. To verify the algorithms correctness for the ramp inputs, the resulting roughnesses and speeds are constant depending on the slope of the ramp.



(a) 20 degree Roughness

(b) 20 degree Speed

(c) 40 degree Roughness

(d) 40 degree Speed

(e) 60 degree Roughness

(f) 60 degree Speed

Figure 5.2: Ramp roughness (left) and speed (right) results at 20°, 40°, and 60°

Fig. 5.2 shows 20°, 40°, and 60° ramp roughness and speed results in the left and right hand columns respectively. With a 20° ramp, the terrain is considered traversable with 0 roughness and $3m/s$ speed (see Fig. 5.2a and Fig. 5.2b). As the ramp angle

increases to 40°, the roughness slightly increases to 0.2028 and 2.3916m/s speed (see Fig. 5.2c and Fig. 5.2d). When the ramp angle reaches or exceeds the roll/pitch threshold of 60°, the terrain is considered untraversable with roughness of 1 and 0m/s speed (see Fig. 5.2e and Fig. 5.2f). All the roughness and speed results in Fig. 5.2 are a constant value and appear as a flat plane. This verifies that the algorithm does not confuse sloped terrain with stepped terrain, otherwise the roughness values for 20° and 40° would be much higher and look more like Fig. 5.5. Running these simulations verify our algorithm along sloped terrain as evidence by Tbl. 5.2 and Fig. 5.3, where slope is the tangent function of the angle.

Input	Window Size (K)	Alpha (α)	Angle ($\Theta°$)	Roughness	Speed (m/s)	Slope ($tan(\Theta)$)
ramp 0	1	1	0	0	3	0
ramp 10	1	1	10	0.0004	2.9988	0.1763
ramp 20	1	1	20	0.0078	2.9766	0.3640
ramp 30	1	1	30	0.0275	2.9175	0.6080
ramp 40	1	1	40	0.2028	2.3916	0.8391
ramp 50	1	1	50	0.6376	1.0872	1.1918
ramp 60	1	1	60	1	0	1.7321
ramp 70	1	1	70	1	0	2.7475

Table 5.2: Ramp simulation results using various slopes

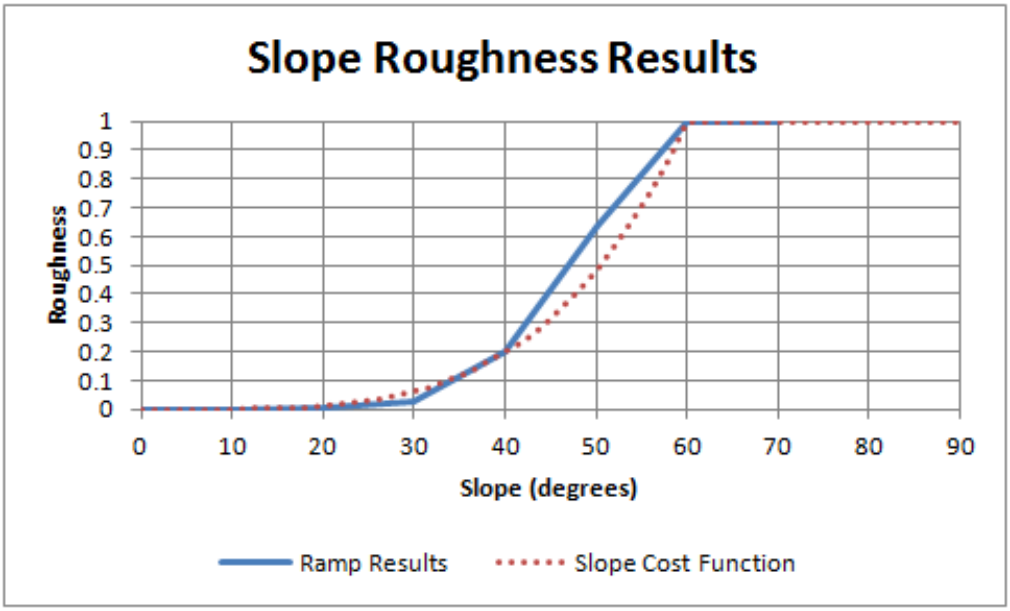


Figure 5.3: Ramp Results

The roughness results, from Tbl. 5.2, plotted in blue in Fig. 5.3, follow the red dotted slope cost function line which validates the algorithm for sloped terrain.

5.1.2 Obstacle Height

The vehicle obstacle height threshold (δ), in the projected height cost function (s_μ), can also be changed to meet rover specifications/limits. For simulation and testing $\delta = 0.3m$ is used. See Fig. 4.9 for cost function graph and Section 4.2.3.1 for more details on cost function.

The generated step terrains are $2m \times 2m$ with the step taking place at $1m$ in the x-direction. The step takes place near the middle of the cell, this is considered a misaligned step. Fig. 5.4 is an example step input used to test the obstacle height parameter. All step inputs cover a $2m \times 2m$ area.

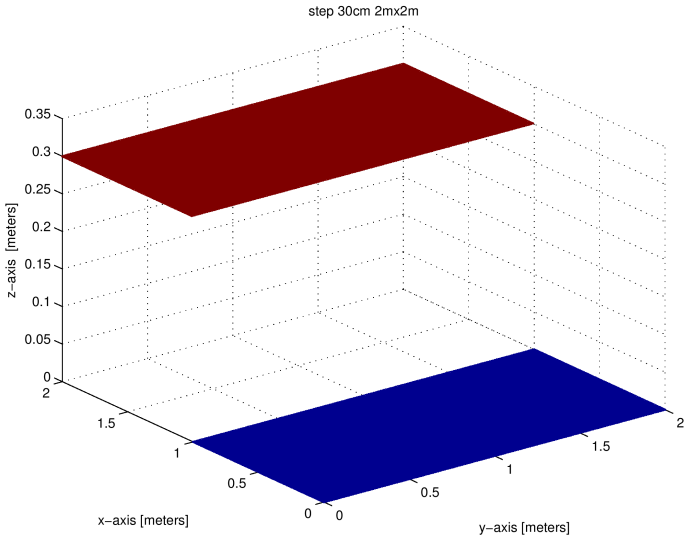


Figure 5.4: $30cm$ step input with step at $1m$ in a $2m \times 2m$ area

The step input simulations uses the following parameters:

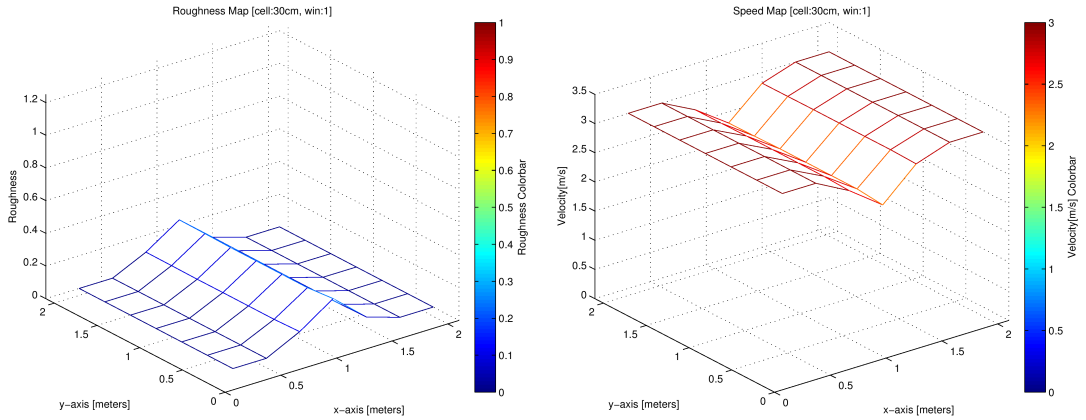
Parameter	Value
Cell Size	$0.3m$
Window Size (K)	1
Alpha Weight (α)	1
Obstacle Height (δ)	$0.3m$
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Roll/Pitch Threshold (ϕ/θ)	60°

Table 5.3: Speed map conditions used for step input simulations

In Fig. 5.5 are three step input results that show that roughness and speed follow the height cost function. The sharp bump in the roughness and speed figures is where the step is positioned and reflects the roughness and speed of the step. The roughness of the step will increase while the speed will decrease as the step height approaches the rover's obstacle height, δ.

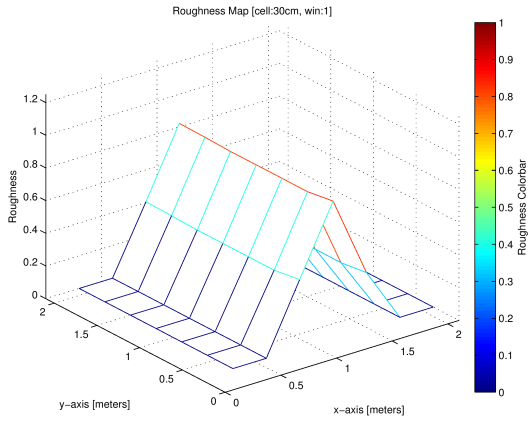
Fig. 5.5 shows $5cm$, $20cm$, and $1m$ step roughness and speed results in the left and right hand columns respectively. With a $5cm$ step, the terrain is considered traversable with 0.2404 roughness and $2.2788m/s$ speed (see Fig. 5.5a and Fig. 5.5b). As the step size increases to $20cm$, the roughness increases to 0.8367 and $0.4899m/s$ speed (see Fig. 5.5c and Fig. 5.5d). When the step height reaches or exceeds the obstacle height of $0.3m$, the terrain is considered untraversable with a roughness of 1 and $0m/s$ speed (see Fig. 5.5e and Fig. 5.5f).

These results verify that the roughness increases as the step size approaches δ. The cells that are neighboring the step cells (denoted as *Left* and *Right* in Tbl. 5.4) are only slightly affected because, for window size of 1, there are 5 similar neighboring cells that will outweigh the influence of the remaining 3 cells that are on the step. However, for larger window sizes these columns ($Left, Center, Right$) will be affected more, which will be discussed in a later section. The column marked *Center* in Tbl. 5.4, is the roughness measurement of the cell that contains the step.

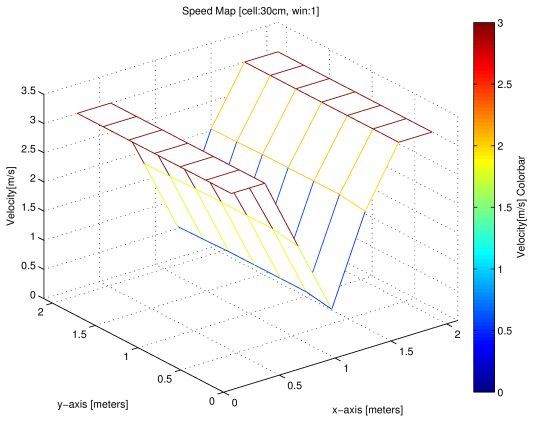


(a) 5cm Roughness

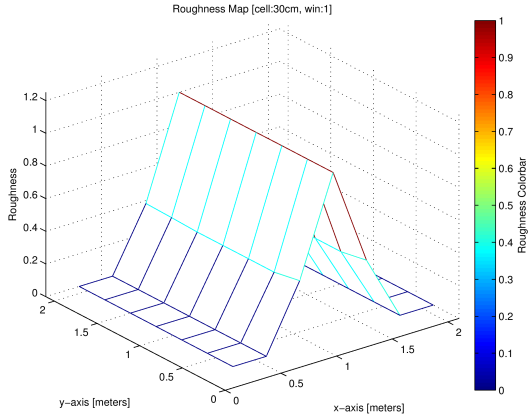
(b) 5cm Speed



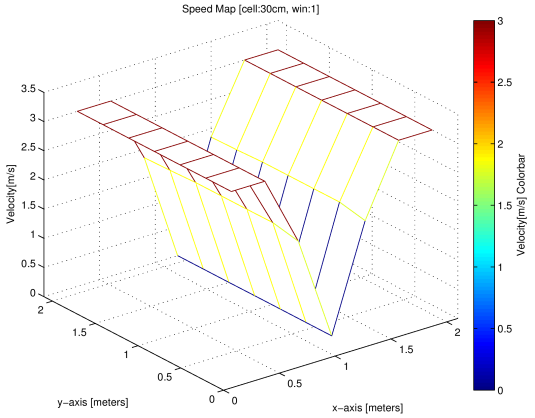
(c) 20cm Roughness



(d) 20cm Speed



(e) 1m Roughness



(f) 1m Speed

Figure 5.5: Step roughness (left) and speed (right) results at $5cm$, $20cm$, and $1m$

Note: Matlab's mesh function was used to draw figures to give better perspective of results than using scatter3. The coloring of the lines may not correspond with cell area, but the coloring of the vertex corresponds with the roughness and speed of the cell area.

Input	Window Size	Alpha	Roughness Left	Roughness Center	Roughness Right	Speed (m/s) Center
step 0.00m	1	1	0	0	0	3
step 0.05m	1	1	0.1008	0.2404	0.0647	2.2788
step 0.10m	1	1	0.2162	0.4778	0.1527	1.5666
step 0.15m	1	1	0.3277	0.6761	0.2482	0.9717
step 0.20m	1	1	0.4195	0.8367	0.3363	0.4899
step 0.25m	1	1	0.4804	0.9443	0.4055	0.1671
step 0.30m	1	1	0.5235	0.9891	0.4663	0.0327
step 1.00m	1	1	0.4004	1	0.4004	0

Table 5.4: Step simulation results using Window=1 and Alpha=1

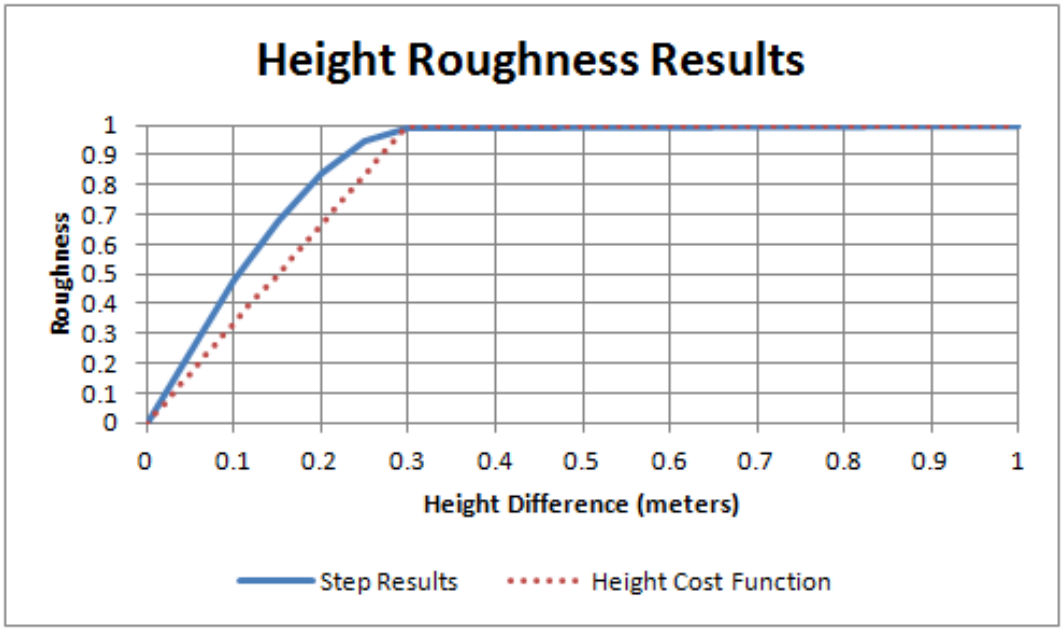


Figure 5.6: Step Results

Looking at these step input results, the roughness increases and speed decreases while the step height increases. The trend of the blue step result line in Fig. 5.6, is somewhat linear as the step height approaches δ. This verifies the height cost function is working properly when compared to the cost function.

5.2 Algorithm Specific Parameters

5.2.1 Window Sizes

Window size is the area in which to calculate the center cell roughness. Increasing the window size allows the algorithm to compute the dot product surface normals with a larger area of terrain. Refer to Section 4.2.1.2 and Fig. 4.1 for more information. We varied the window size to test the effects on the roughness calculation. We used step terrain similar to Fig. 5.4 along with the following parameters:

Parameter	Value
Cell Size	$0.3m$
Alpha Weight (α)	1
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Obstacle Height (δ)	$0.3m$
Roll/Pitch Threshold (ϕ/θ)	60°

Table 5.5: Speed map conditions for step input simulations with varying window size

The results of the varied window size are in Tbl. 5.6:

			Roughness			Speed (m/s)
Input	Window Size	Alpha	Left	Center	Right	Center
step 0.05m	1	1	0.0689	0.2475	0.0441	2.2575
step 0.05m	2	1	0.1078	0.4206	0.1036	1.7382
step 0.05m	3	1	0.1446	0.5554	0.1440	1.3338
step 0.15m	1	1	0.1856	0.7101	0.1389	0.8697
step 0.15m	2	1	0.3023	0.9112	0.3049	0.2664
step 0.15m	3	1	0.4007	0.9528	0.4095	0.1416
step 0.20m	1	1	0.2288	0.8580	0.1807	0.4260
step 0.20m	2	1	0.3861	0.9561	0.3953	0.1317
step 0.20m	3	1	0.5090	0.9766	0.5240	0.0702
step 1.00m	1	1	0.1929	1	0.1929	0
step 1.00m	2	1	0.5037	1	0.5446	0
step 1.00m	3	1	0.6767	1	0.7138	0

Table 5.6: Step simulation results using various window sizes

The step results show the roughness values of (*Left*, *Center*, and *Right*) increased as the window size increased; allowing differing cells to be added into the computation causing the roughness value to approach one (rough surface). However, just the opposite would occur if similar cells were added to the computation, causing the roughness value to approach zero (smooth surface).

The significance of window sizing is to choose how large of an area is needed to estimate the roughness and speed for the rover. For small window size, the minimum size is defined by the rover's wheel diameter since the maximum wheel-terrain contact points would occur in a wheel diameter length by wheel radius deep hole, covering half the surface of the wheel. For larger window sizes, it could be defined by the rover footprint (area beneath the rover) in order to provide roughness and speed estimates for the entire rover as a rigid body and also for potential hang-up failures, where the rover body becomes lodged atop an obstacle [15].

5.2.2 Alpha Weight

The alpha term is a weight that fuses neighboring roughness values with the center roughness value. Decreasing the alpha term will merge the neighboring cells' center roughness value into the current centers' roughness value. If $\alpha = 1$, then the roughness value for the center cell is only the center cell. Whereas, if $\alpha = 0.8$, 80% of the center roughness is used and the remaining 20% comes from the neighboring center cells within the window size. We varied α to test the effects it had on the roughness calculation. We used step terrain similar to Fig. 5.4 along with the following parameters:

Parameter	Value
Cell Size	$0.3m$
Window Size (K)	2
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Obstacle Height (δ)	$0.3m$
Roll/Pitch Threshold (ϕ/θ)	$60°$

Table 5.7: Speed map conditions for step input simulations with varying alpha

The results of varying α are in Tbl. 5.8:

			Roughness			Speed m/s
Input	Window Size	Alpha	Left	Center	Right	Center
step 0.05m	2	1	0.1078	0.4206	0.1036	1.7382
step 0.05m	2	0.8	0.1152	0.3684	0.1115	1.8948
step 0.05m	2	0.5	0.1264	0.2859	0.1232	2.1423
step 0.15m	2	1	0.3023	0.9112	0.3049	0.2664
step 0.15m	2	0.8	0.3215	0.8431	0.3227	0.4707
step 0.15m	2	0.5	0.3497	0.7047	0.3491	0.8859
step 0.20m	2	1	0.3861	0.9561	0.3953	0.1317
step 0.20m	2	0.8	0.4011	0.8998	0.4077	0.3006
step 0.20m	2	0.5	0.4232	0.7723	0.4259	0.6831
step 1m	2	1	0.5037	1	0.5446	0
step 1m	2	0.8	0.5160	0.9818	0.5486	0.0546
step 1m	2	0.5	0.5341	0.8861	0.5545	0.3417

Table 5.8: Step simulation results using various alpha values

In contrast to the roughness calculation effects of changing the window size for the step inputs, the *Center* roughness ($r(x, y)$), in Tbl. 5.8, approaches the neighboring cells' roughness as alpha approaches zero. And as alpha approaches one, the cell roughness ($r(x, y)$) will only be the center cell roughness value. This is why the roughness values in *Left* and *Right* approach its neighboring cells' roughness as the alpha weight approaches zero. The significance of α is to smooth roughness measurements within a window size together.

Chapter 6

Experimental Results

The experimental section will show and discuss the results of the speed algorithm using real sensor data from several field tests. Similar to the simulation section, various results will be shown for parameter changes such as: window size and alpha weight. Additionally, this section will compare the speed map method to other existing methods.

To run the experiments on real data, the Matlab simulation code was ported to C, then integrated into *RoverNav* (NASA's rover navigation software framework which is currently being used on ARC's KRex rover and JSC's C2 rover). The rover was simulated using Visual Environment for Robotic Virtual Exploration (VERVE), a visual interface for real-time situational awareness and exploration of new mission operations. To test and verify the algorithm using real world environments, all experiments used replay data (actual sensor data) from various field test locations: Johnson Space Center (rockyard, crater), and Ames Marscape.

Note: The color scheme for speed ranges from red (rough/zero speed) to green (smooth/max speed) for all VERVE results. Matlab results will specify a colorbar and its associated speed in figures.

6.1 Algorithm

Storing the point cloud data and modeling the terrain requires a fast and efficient technique that can be updated in real-time and use minimal memory. A growing set least squares approach is used to accomplish this. The algorithm ran in real-time updating a simple running sum rather than having to store every single PCD. Using this approach cuts down on storage space, which is a large benefit because it simplifies the rover and does not require the system to worry about optimizing storage space or data retrieval.

After new point cloud data has been added and updated to the system, the roughness will then be updated as follows:

Algorithm 1 Update Tile Frame

```
function UPDATE()
    for all cells (x, y) in TileFrame do
        CalcRoughness(x,y,false): compute center cell roughness
    end for
    for all cells (x, y) in TileFrame do
        CalcRoughness(x,y,true): gather neighboring cell roughnesses
    end for
end function
```

Algorithm 2 Calculate Roughness

```
function CALCROUGHNESS(x, y, getNbr)
    get corresponding map cell k
    for all cells (i, j) in WINDOW do
        if getNbr != true then
            get corresponding map cell l
            accumulate smoothness cost with dot product surface normals, height
                cost, and weight factor between k and l: s+ = s_N * s_μ * weight
        else
            accumulate neighboring cell smoothness: (x, y).sNbr += (i, j).sCenter
        end if
    end for
    if getNbr != true then
        (x, y).sCenter = s/weight
    else
        (x, y).roughness = 1 − (α * (x, y).sCenter + (1 − α) * (x, y).sNbr)^2
    end if
end function
```

In order to compute the roughness accurately for each update, *CalcRoughness* runs twice. The first run is to compute the center roughnesses for all cells within the tile frame, and the second run is to accumulate neighboring roughnesses. If the first run is not done before the neighboring roughness accumulation, then the roughness for each cell will not be accurate or up to date. The running time for this implemented algorithm is at worst $O(2[(2K+1)^2-1]N)$, where N is the number of cells within a tile and K is the number of cells within a window size.

6.2 Speed Map Method Results

6.2.1 Dot Product Normal Results

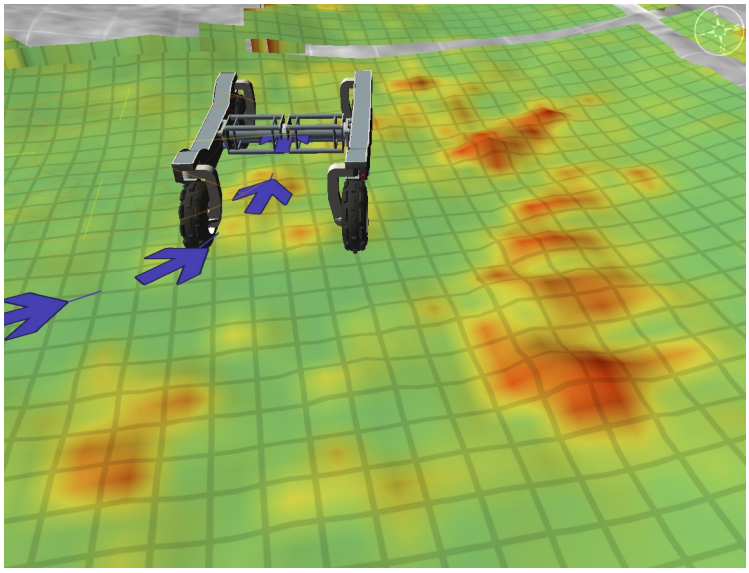


Figure 6.1: Marscape - uneven terrain patch in red, showing dot product technique

Note: The color scheme of [red→green]∼[rough→smooth]∼[zero speed→max speed].

The effectiveness of the dot product normal in the speed map method is seen in Fig. 6.1 of an uneven terrain patch on Marscape, where the bumps are less than $0.1725m$ high. The red-orange patches in the figure below are defined as rough surfaces where low speeds are assigned to these areas for traversal. These patches are considered rough because the dot product normal (s_N) is more prevalent in detect-

ing changes in direction of the uneven surfaces than the slopes of the cells (s_{slope}) or projected heights (s_μ). The green colored cells are smooth surfaces and have a high speed associated with them.

6.2.2 Obstacle Height Results

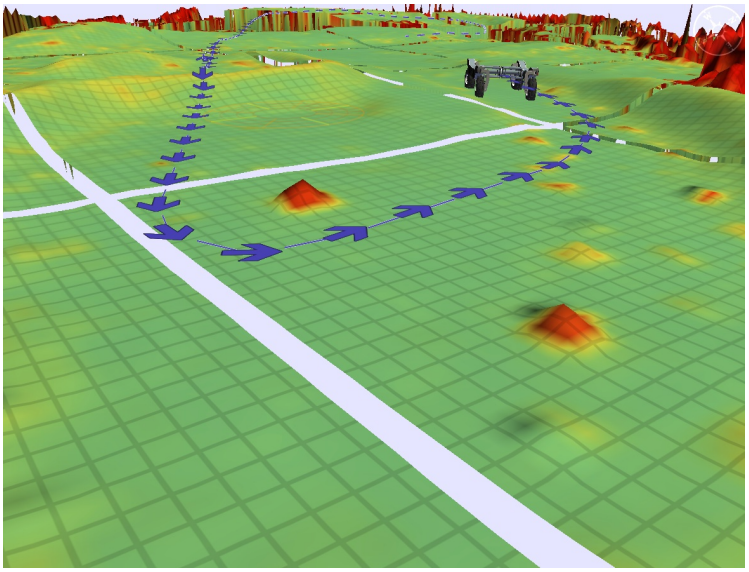


Figure 6.2: Marscape - two large rocks $0.4572m$ tall, showing step terrain

Note: The color scheme of [red→green]∼[rough→smooth]∼[zero speed→max speed].

We tested the obstacle height ($\delta = 0.3m$) using two singular large rocks, approximately $0.4572m$ tall and $0.60m$ wide, as seen in Fig. 6.2 by the red colored cells. All three smoothness cost functions (s_N, s_μ, s_{slope}) contributed to the roughness of the rocks because 1) the surface normals between rock cells have a large angle of rotation, 2) the projected height cost (s_μ) within the rocks exceeded δ, and 3) the rock slopes are greater than the roll/pitch thresholds. The two rocks have been computed as rough obstacles and have been assigned a $0m/s$ speed, which is noted by the red coloring of the rocks.

6.2.3 Roll/Pitch Threshold Results

We tested the slope smoothness (s_{slope}) using the crater like area of Marscape (see Fig. 6.3). Since craters are naturally continuously sloped, the majority of the terrain is computed as smooth and assigned high speeds, as evidenced by the green coloring, because the majority of the surface normals between crater cells have a small angle of rotation and the s_μ within the crater are near 0. Several areas near the apex of the crater, have been marked rough by the orange-yellow coloring and assigned lower speeds due to slopes approaching roll/pitch thresholds, and curvature changes of the terrain that have been marked rough in s_N. The consistent speed throughout the rovers path verifies this type of terrain is not confused with step terrain, like that in Fig. 6.2.

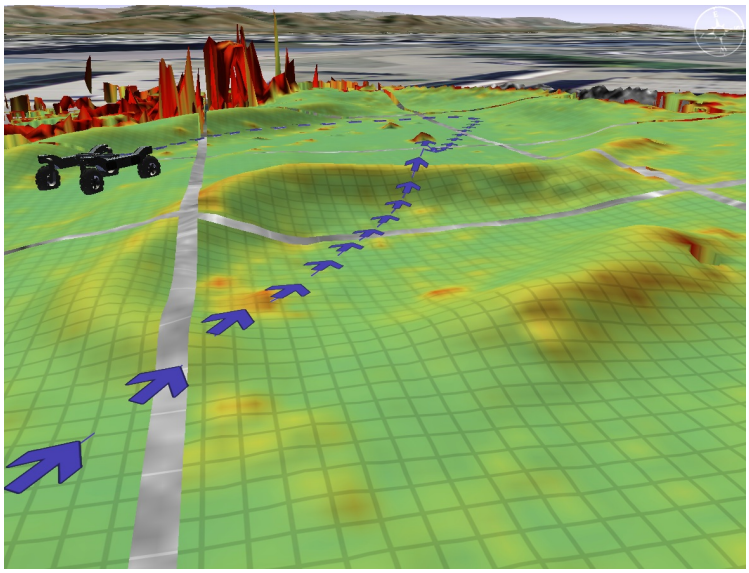


Figure 6.3: Marscape - crater, showing continuous sloped terrain

Note: The color scheme of [red→green]∼[rough→smooth]∼[zero speed→max speed].

6.2.4 Window Size

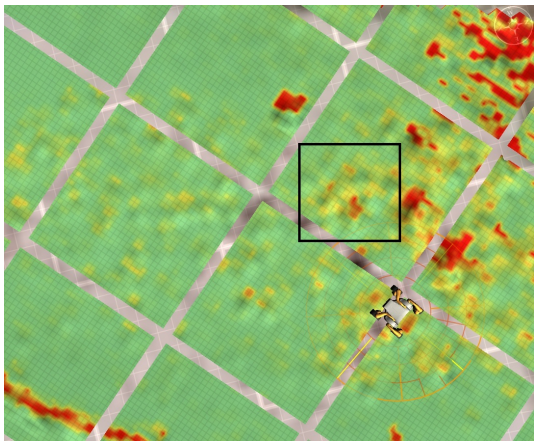
The JSC rockyard field site was used to test the effects of window sizes. Below, in Fig. 6.4, are two photos of the JSC rockyard used as a reference for size and cluster of rocks. The photos show small rocks scattered on the west end of the rockyard, and a condensed cluster of large rocks on the east end, near Mt. Kosmo. This is

also evidenced in Fig. 6.5, where the boxed region is the same region pictured in Fig. 6.4b.

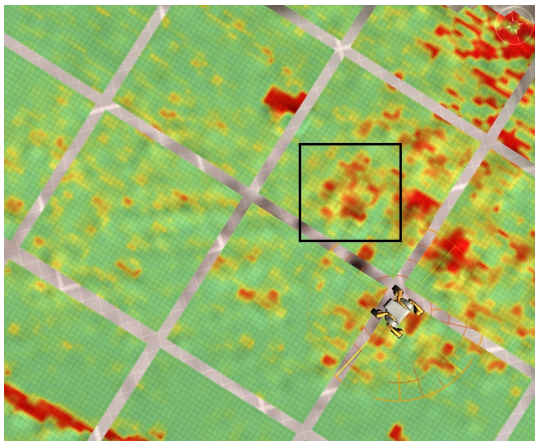
(a) West end - Small Rocks

(b) East end - Large Rocks

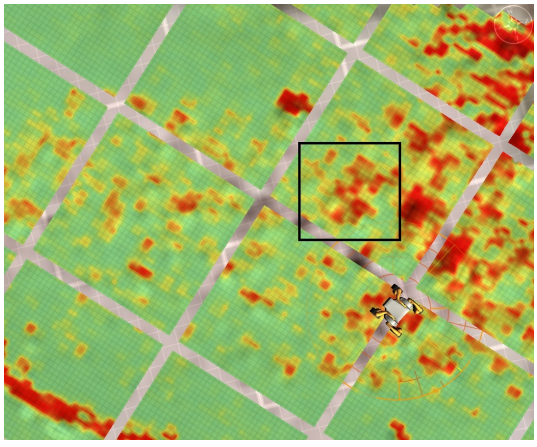
Figure 6.4: JSC rockyard terrain



(a) Alpha = 1, Window = 1



(b) Alpha = 1, Window = 2



(c) Alpha = 1, Window = 3

Figure 6.5: Window size comparison using JSC rockyard. Boxed region shows roughness/speed increase(decrease) as window size increases(decreases)

Note: color scheme of [red→green]∼[rough→smooth]∼[zero speed→max speed].
We varied the window sizes and tested the effects it had on the roughness calculation. Using VERVE to simulate the experimental data shows the window size effects more noticeably. The roughnesses in the boxed region of Fig. 6.5a-c, increases in magnitude and area as the window size increases. This is a direct effect of including more differing cells into the roughness computation, and results in roughnesses approaching one. It is also evident in the double summation of Eq. 4.35; as K increases, more neighboring cells are included in the roughness cost, thereby increasing the affected area and changes the roughness depending on how different the neighboring cells are from the center cell. For similarly rough neighboring cells, the center cell roughness is invariant to K.

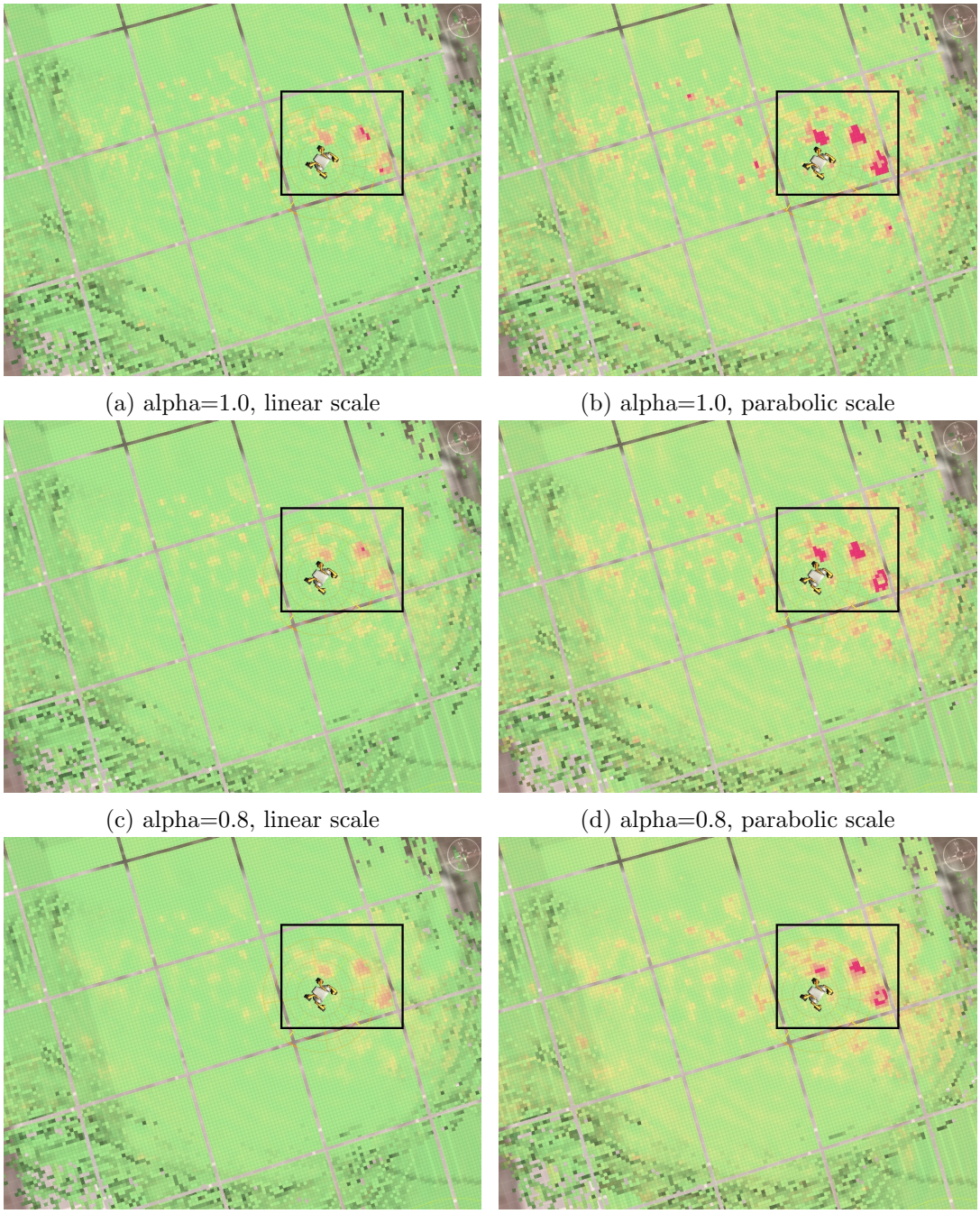
Experimentally we chose a window size of $K = 2$ because the surface area covered is similar to the dimensions of the KRex rover. See Tbl. 6.1 for corresponding window sizes with surface areas. Choosing the best window size to use is left for further research.

Window Size (K)	Window Dimension $[cell \times cell]$	Surface Area $[m \times m]$
1	3×3	0.9×0.9
2	5×5	1.5×1.5
3	7×7	2.1×2.1
4	9×9	2.7×2.7

Table 6.1: Corresponding Window Size with Surface Area, using cell size = 0.3

6.2.5 Alpha Weights

As mentioned in Section 4.2.4, the roughness calculation is based on a weighted sum of the center cell and its neighbors. To better relate the neighbors with the center cell, the neighbors' center roughness is fused into the equation and weighted with α (Eq. 4.39)



(a) alpha=1.0, linear scale

(b) alpha=1.0, parabolic scale

(c) alpha=0.8, linear scale

(d) alpha=0.8, parabolic scale

(e) alpha=0.5, linear scale

(f) alpha=0.5, parabolic scale

Figure 6.6: Roughness α comparisons with JSC rock-yard data set

Fig. 6.6 shows a comparison of alpha weights and its roughness scale factor between linear scale and parabolic scale. The clustered large rocks, marked by the black box, maintain the shape of the rough area while the intensity of the roughness and speed decreases as alpha approaches $\frac{1}{n+1}$. This is different than the effects the window size has on the roughness and speed, as mentioned in the previous section 6.2.4.

The effects of changing α stem directly from Eq. 4.39. Since window size does not change, the affected region is constant throughout and therefore only the roughness value changes and approaches the neighboring cells' center roughness (s_{center}) as α approaches the lower limit $\frac{1}{n+1}$. For our experiments, $\alpha = 0.8$ gave excellent results, however other missions will need to determine the optimal α band for their specific requirements.

The parabolic scale with $\psi = 2$ (Eq. 4.41) is used for the roughness scale. This scaling is computed by raising the power of the total smoothness cost ($s(x, y)$) with ψ to increasingly penalize rough terrain. This is seen in Fig. 4.10 and when comparing the boxed area of Fig. 6.6a to 6.6b, 6.6c to 6.6d, and 6.6e to 6.6f.

6.3 DEM vs. LiDAR

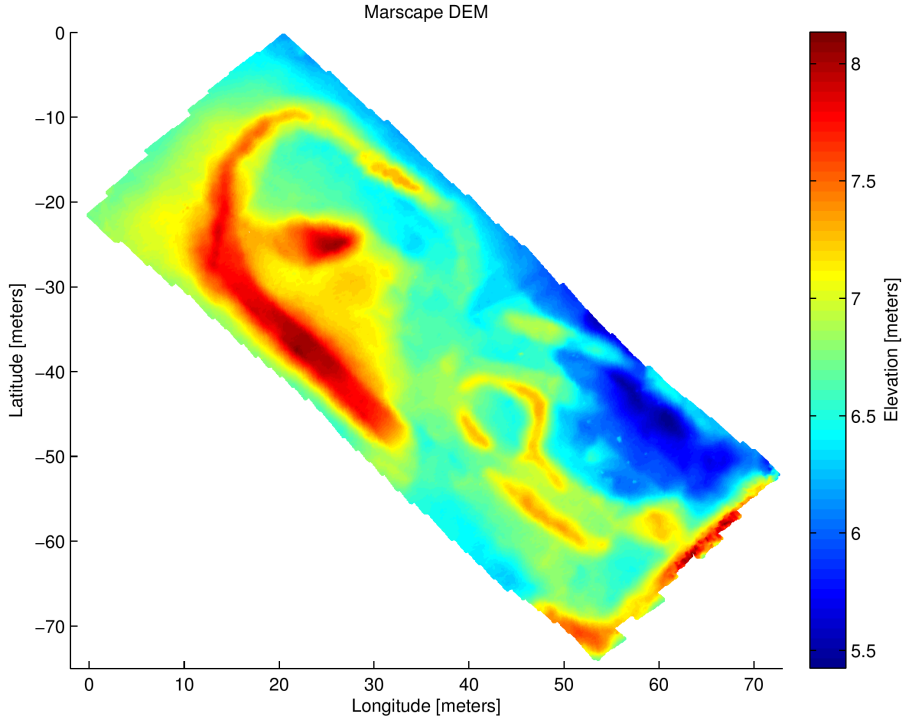


Figure 6.7: Marscape DEM 10cm resolution

In this section we compare the DEM (Digital Eleveation Map) of NASA Ames Marscape with a 10cm resolution (see Fig. 6.7) to the results from the KRex LiDAR

system, using the same terrain area. This comparison is done in order to verify the KRex system is detecting surface roughnesses and speeds similar to the DEM results. The roughness and speed of the DEM and LiDAR both used the following conditions in Tbl. 6.4:

Parameter	Value
Cell Size	$0.3m$
Window Size (K)	2
Alpha Weight (α)	0.8
Obstacle Height (δ)	$0.3m$
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Roll/Pitch Threshold (ϕ/θ)	60°

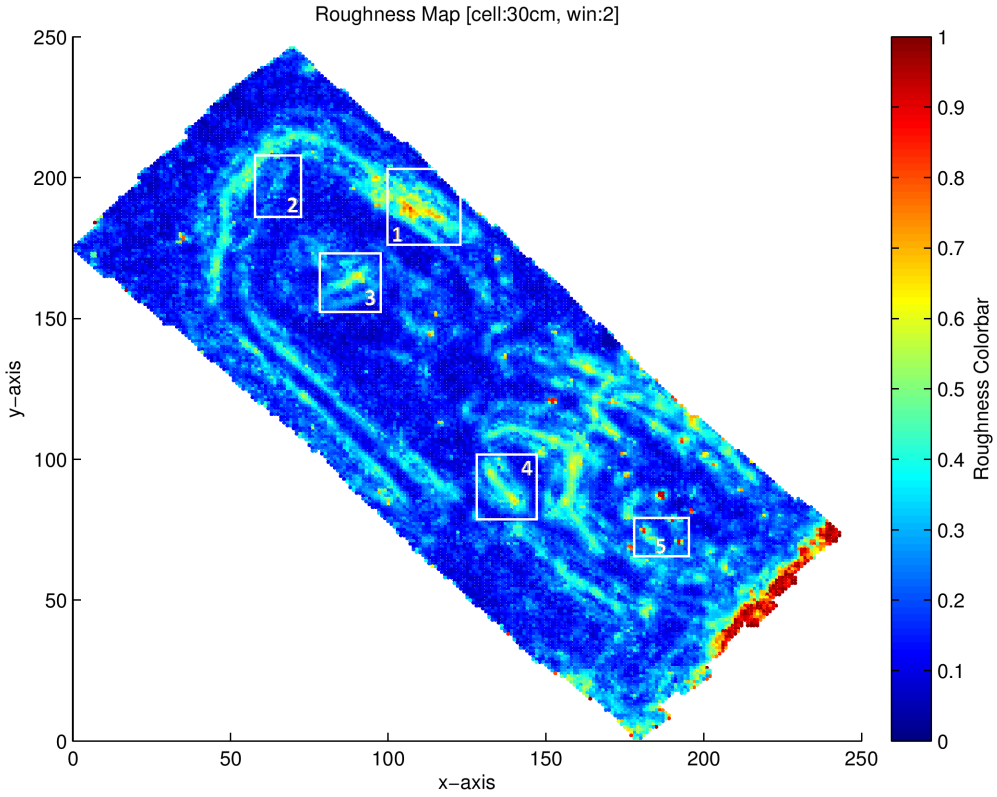
Table 6.2: Speed map conditions used for DEM and LiDAR

The speed map algorithm was applied to the Marscape DEM in Matlab and the results are shown below in Fig. 6.8. The roughness and speed map both use the same color gradient– [dark blue→red] for [low speed→ high speed] and [rough→ smooth].

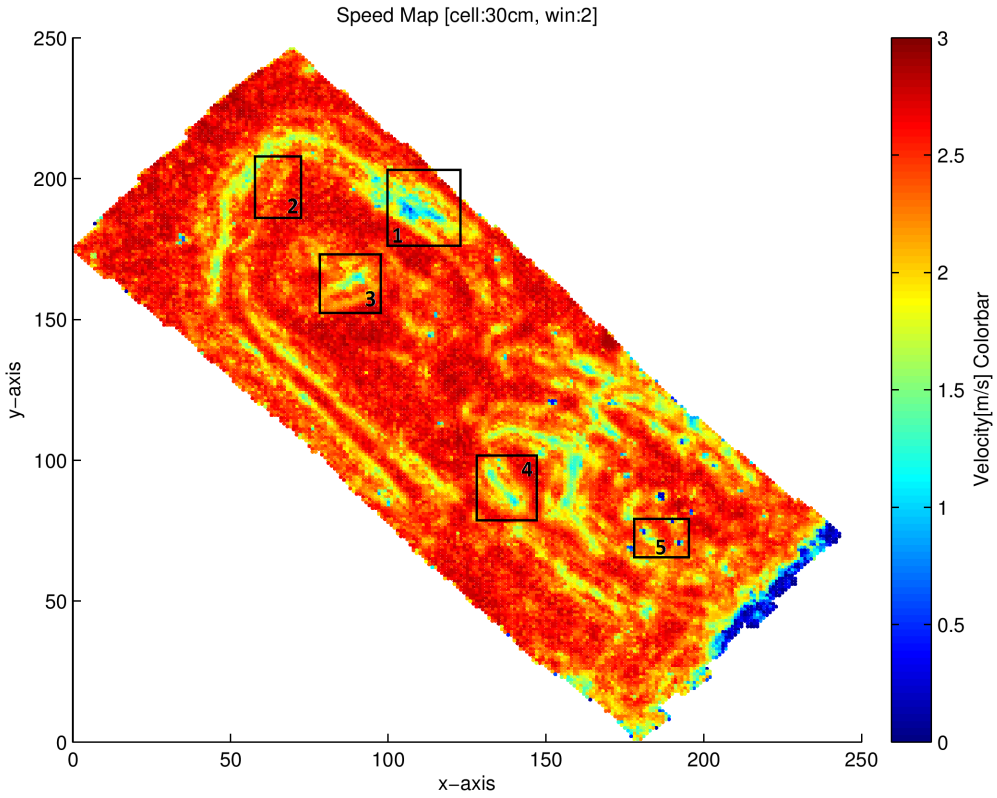
The numbered boxes in Fig. 6.8a and 6.8b, are terrain features that will be used to verify the KRex LiDAR system results. The terrain features are categorized as follows:

Feature	Description
1	Rough hillside
2	Uneven terrain on on level ground
3	Summit of hill
4	Crater wall
5	2 large rocks standing $0.4572m$ tall

Table 6.3: Category of Terrain Features



(a) Roughness Map of Marscape DEM



(b) Speed Map of Marscape DEM

Figure 6.8: Marscape DEM produced roughness and speed map

The real-time results of the speed map algorithm using the PCD from the LiDAR scans were simulated in VERVE, and the results of this are shown below in Fig. 6.9. The color gradient of the speed map is [red→ green] which is equivalent to [no/low speed → max/high speed].

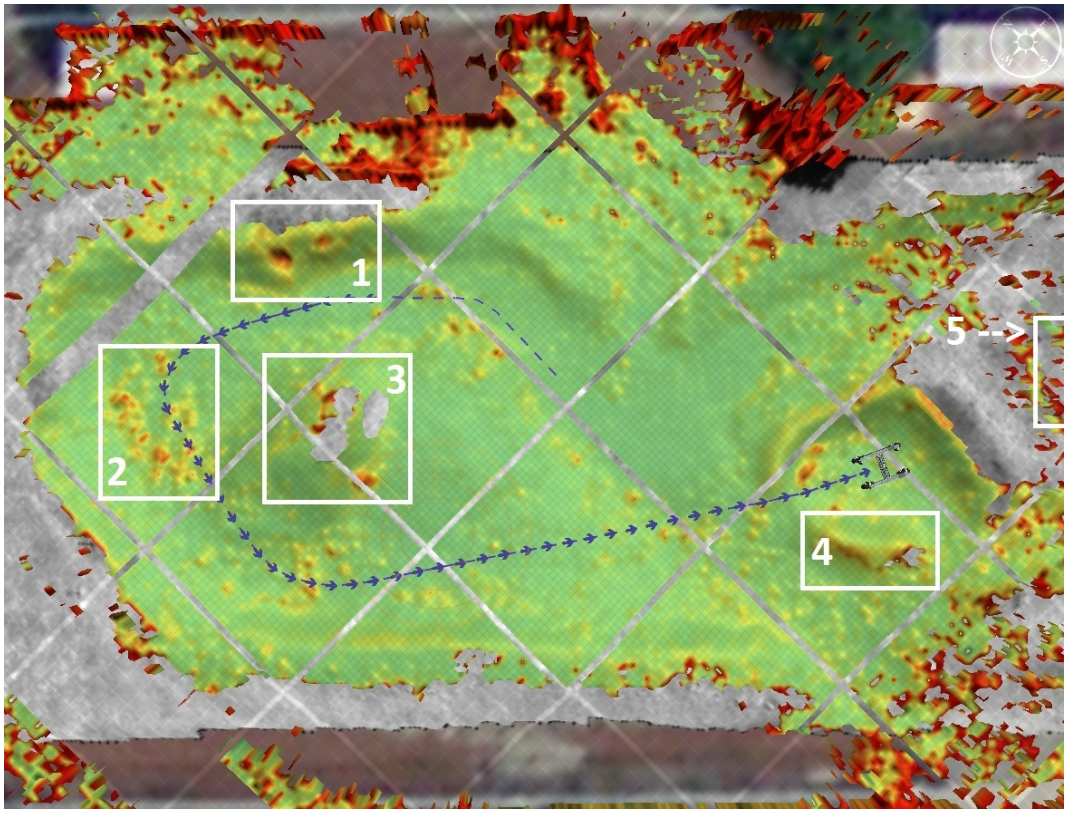


Figure 6.9: Marscape Overhead

The results of the DEM and LiDAR scan are nearly identical, which validates the consistency of the algorithm. Boxes 1-5 in Fig. 6.9 show features detected on the KRex system match the true data of the DEM in Figs. 6.8a-b. The box 5 features are not shown well in Fig. 6.9 but is captured in Fig. 6.2. Marscape terrain is permanent except for the rocks, which are moved around on a regular basis. For this reason the rock features did not match the position exactly but the roughnesses and speeds were captured in both the DEM and the KRex LiDAR system.

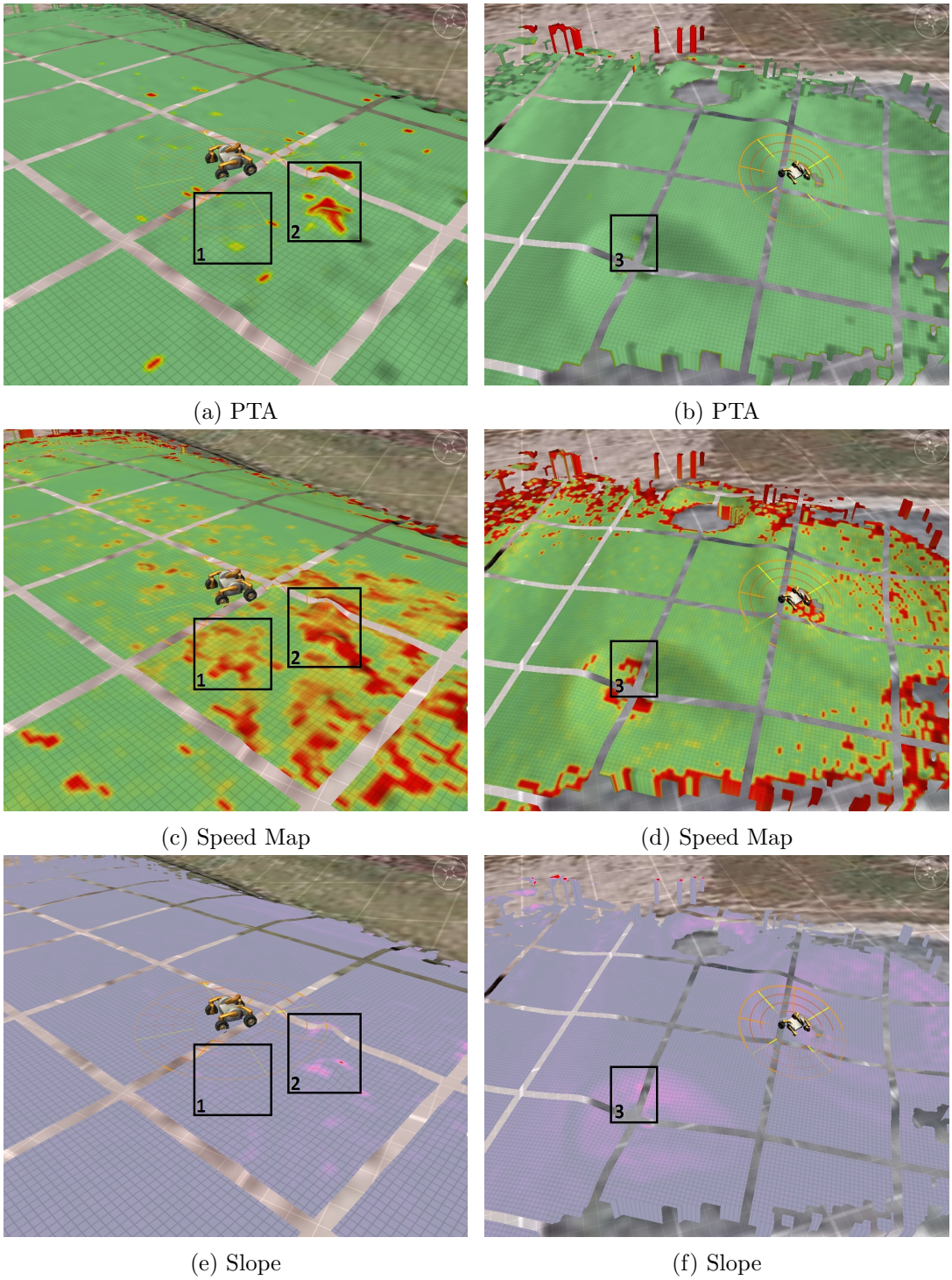
6.4 Method Comparison

In this section we compare our speed map method with PTA and slope calculation. PTA uses an obstacle height threshold of $\delta = 0.3$ within its immediate neighbors, and the slope calculation uses a roll/pitch threshold to determine how traversable the cell is. The speed map method uses the following conditions:

Parameter	Value
Cell Size	$0.3m$
Window Size (K)	2
Alpha Weight (α)	0.8
Obstacle Height (δ)	$0.3m$
Slope Power (η)	4
Roughness Scale Factor (ψ)	2
Roll/Pitch Threshold (ϕ/θ)	60°

Table 6.4: Speed map conditions for method comparison

We use visual inspection to compare the methods since each method uses their own metric to define obstacles and roughnesses. The left hand column of Fig. 6.10 uses JSC's rockyard test site, and the right hand column uses JSC's crater test site to compare methods. The color scheme of PTA is based on the probability the terrain is an obstacle or not given by red and green– (obstacle and non-obstacle). The speed map follows the color gradient of [red→ green] which is equivalent to [no/low speed → max/high speed]. The coloring of the slope follows [purple→ pink] which is based on how close the slope is to the roll/pitch threshold– [0 slope → slope threshold]



(a) PTA (b) PTA

(c) Speed Map (d) Speed Map

(e) Slope (f) Slope

Figure 6.10: Method Comparison PTA vs Speed Map vs Slope using JSC Rockyard (left column) and Crater (right column) data set

PTA is a binary obstacle detection algorithm that only detects changes in height as seen in Fig. 6.10a and Fig. 6.10b. The speed map detects the slightest changes in surfaces (Fig. 6.10c and 6.10d) and reacts to these changes more quickly. Another

advantage of this algorithm is its resilience to terrain changes over time. The differences in methods are seen in boxes 1-3 of Fig. 6.10. PTA only detects several large rocks in box 2 of the JSC rockyard, whereas the speed map detects all the rocks that are considered obstacles and rough. Within the crater data set, PTA detects no obstacles while the slope of the crater wall, in box 3, is detected as steep in Fig. 6.10f. The speed map also detected box 3 as steep; additionally it detected very small changes on the surface as shown by the scattered yellow blotches in Fig. 6.10d whereas the slope method cannot detect these small changes since all the slopes are well below the threshold.

This chapter presented the results of the speed map algorithm using real sensor data from various field tests. We verified the algorithms repeatability and accuracy of terrain roughness by comparing true terrain (DEM) results with the KRex LiDAR system results. Specific terrain formations in Marscape were used to test specific features in the speed map algorithm such as: the dot product surface normal method, rover's obstacle height, and rover's roll/pitch threshold. Additionally, we compared the speed map with PTA and slope methods. These results show that the speed map algorithm adds granularity to roughness and provides better feature detection of the terrain. This information rich method allows for better obstacle/hazard detection and better intuition of allowable speed.

Chapter 7

Discussion

7.1 Conclusion

The intent of this thesis has been to define a new metric that measures terrain roughness and use that metric to create a speed map for rover navigation. The speed map provides a boost in current average rover speeds over diverse terrain using a variety of information known about the surrounding terrain to improve decision making on how fast to travel. The speed map can also provide a new path optimization; instead of the shortest path from point A to point B, the safest and fastest way can be computed.

The effectiveness of the speed map method is seen in both the simulation and experimental results when compared to other methods. The speed map looks at the terrain surface and is able to quantify roughness through the use of surface normals, projected plane heights, slopes, residuals, and coverage. This approach is able to detect and provide more information about the terrain than current existing methods, and allows the speed map to generate a range of recommended speeds over varying terrain rather than a simple binary speed selection.

The speed map does have its own limitations. It does not take into account the rover's suspension system nor does it include the wheel-to-ground interaction model.

Including them would improve recommended speed predictions using vibration estimates of the rover. However, the speed map replaces both of these with the surface normal (s_N), $residualCost$, and projected height (s_μ) cost functions. This simplifies the allowable speed problem down to a naive model which linearizes the roughness to speed transformation.

One of the benefits of using surface normals is the robustness against time dependent plane height errors induced by inherent pose errors. This method is also adaptable to any four-wheeled systems, only requiring the system to know the wheel size and its roll/pitch threshold to adjust obstacle definitions ($\delta, \theta/\phi$) and window sizing (K). This method has a running time that takes at worst $O(2[(2K+1)^2-1]N)$, which is still in linear time.

In the end, we were able to create a novel method that defines terrain roughness and creates a speed map for rover navigation. The speed map determines an allowable speed over rough terrain that results in increased average rover speeds. The calculated allowable speeds has not been validated experimentally, and therefore more work is required to prove that this method does increase average rover speeds. Overall, the speed map algorithm adds granularity to roughness and provides better feature detection of the terrain. This information rich method allows for better obstacle/hazard detection and better intuition of allowable speed.

7.2 Future Work

There is a great deal of potential in using this research for future work and several ways to improve this new method. At the time of this writing, rover vibration tests have not been performed to identify the magnitude and extent of vibration the system can withstand without damaging on-board electronics. Having this information would allow the recommended speed to be validated via a closed loop system– using the generated speed produced by the map to operate the rover's velocity controller. This is just one of many possible directions.

The algorithm can be improved by optimizing the update and roughness calculation algorithm to make the processing faster for larger window and tile sizes. Currently, a window size of 2 and tile size of 192 have been used for field testing; no delays in computation or visual feedback have been included. Optimizing this algorithm for larger window and tile sizes would allow for larger terrain area to be processed. This could potentially improve the ability of the rover to predict hazards or dangers sooner, as well as construct paths with greater facility. Additionally, further research into finding the best values for the parameters in the speed map equations is required (such as: window size (K), alpha weight (α), slope power (η), roughness scaling factor (ψ), and coverage threshold (ω)).

The current center/neighbor averaging is limited in its simplicity. A Gaussian filter for roughness merging is potentially better. This would be used when including neighboring cells in the dot product phase or the α weight phase of merging center roughness and neighbor roughnesses together. This filter would only apply for window sizes larger than 2. The idea is to weight the closer neighbors more heavily and incrementally decrease the weight of the neighbor the further away it is from the center cell. This concept stems from the fact the farther away terrain A is from terrain B, the less of a relationship the two cells have.

Some future work can also be done in applying machine learning to heuristically convert the roughness metric to rover speed more intelligently, similar to *Stanley* [25] or *Alice* [4]. Both autonomous cars used supervised or unsupervised learning from a training set to establish a base speed map.

The major area this work contributes to is path planning. Combining this work with path planning algorithms provides a new way of determining shorter, faster, and safer routes through terrain, which would improve overall efficiency. This combination would allow rovers to be more autonomous because the speed map provides

detailed information needed to traverse through unknown terrain in real-time.

The speed map is a robust and adaptable real-time method that can be applied outside of planetary rovers. Some applicable uses for the speed map could include: farming, military, transportation, recreational, and exploration. The speed map used in a military setting can be a valuable asset for their ground vehicles and associated personnel. It would aid drivers to detect obstacles/hazards in an unknown or dynamically changing terrain. In a transportation setting, the speed map would provide the automobile drivers situational awareness and safety. The speed map can detect, in real-time, obstacles/hazards on the road such as: road kill, furniture, pot holes, rocks, unseen speed bumps, large debris, and other automobiles. In a recreational setting, the speed map would provide off-roading vehicles a map of the area and assist in navigating extreme terrain. In all these applications, the speed map provides increased safety, efficiency, and situational awareness.

Appendix A

Equations

Statistics stored in matrix form:

$$\underbrace{\begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix}}_{F} \underbrace{\begin{bmatrix} A \\ B \\ C \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \\ \sum z_i \end{bmatrix}}_{y} \tag{A.1}$$

F and y simplified as:

$$F = \begin{bmatrix} S_{xx} & S_{xy} & S_x \\ S_{xy} & S_{yy} & S_y \\ S_x & S_y & n \end{bmatrix} \tag{A.2}$$

$$y = \begin{bmatrix} S_{xz} \\ S_{yz} \\ S_z \end{bmatrix} \tag{A.3}$$

Determinant of 3×3 matrix F:

$$detF = S_{xx}(nS_{yy} - S_yS_y) - S_{xy}(nS_{xy} - S_xS_y) + S_x(S_{xy}S_y - S_{yy}S_x) \tag{A.4}$$

Parameters as a result of matrix inverse $x = F^{-1}y$:

$$A = (S_{xz}(nS_{yy} - S_yS_y) + S_{yz}(S_xS_y - nS_{xy}) + S_z(S_yS_{xy} - S_xS_{yy}))/detF \tag{A.5}$$

$$B = (S_{xz}(S_xS_y - nS_{xy}) + S_{yz}(nS_{xx} - S_xS_x) + S_z(S_xS_{xy} - S_yS_{xx}))/detF \tag{A.6}$$

$$C = (S_{xz}(S_{xy}S_y - S_xS_{yy}) + S_{yz}(S_xS_{xy} - S_yS_{xx}) + S_z(S_{xx}S_{yy} - S_{xy}S_{xy}))/detF \tag{A.7}$$

Appendix B

Simulation Results

Input	Window Size (K)	Alpha (α)	Roughness Left	Roughness Center	Roughness Right	Speed (m/s) Center
step 0.00m	1	1	0	0	0	3
step 0.05m	1	1	0.1008	0.2404	0.0647	2.2788
step 0.10m	1	1	0.2162	0.4778	0.1527	1.5666
step 0.15m	1	1	0.3277	0.6761	0.2482	0.9717
step 0.20m	1	1	0.4195	0.8367	0.3363	0.4899
step 0.25m	1	1	0.4804	0.9443	0.4055	0.1671
step 0.30m	1	1	0.5235	0.9891	0.4663	0.0327
step 1m	1	1	0.4004	1	0.4004	0
step 0.05m	2	1	0.122	0.4079	0.1116	1.7763
step 0.15m	2	1	0.3634	0.8852	0.3474	0.3444
step 0.20m	2	1	0.4652	0.9398	0.4544	0.1806
step 1m	2	1	0.5744	1	0.6175	0
step 0.05m	2	0.8	0.1272	0.3575	0.1183	1.9275
step 0.15m	2	0.8	0.3735	0.818	0.3594	0.546
step 0.20m	2	0.8	0.4702	0.8842	0.4604	0.3474
step 1m	2	0.8	0.5807	0.9844	0.6154	0.0468
step 0.05m	2	0.5	0.1349	0.278	0.1281	2.166
step 0.15m	2	0.5	0.3883	0.6881	0.3774	0.9357
step 0.20m	2	0.5	0.4776	0.7671	0.4693	0.6987
step 1m	2	0.5	0.59	0.9022	0.6122	0.2934
step 0.05m	3	1	0.1536	0.5405	0.1476	1.3785
step 0.15m	3	1	0.4386	0.9337	0.4315	0.1989
step 0.20m	3	1	0.5564	0.9647	0.5545	0.1059
step 1m	3	1	0.7161	1	0.7525	0

Table B.1: Step simulation results with various window sizes and alpha values

Input	Window Size (K)	Alpha (α)	Angle ($\Theta°$)	Roughness	Speed (m/s)	Slope ($tan(\Theta)$)
ramp 0	1	1	0	0	3	0
ramp 10	1	1	10	0.0004	2.9988	0.1763
ramp 20	1	1	20	0.0078	2.9766	0.3640
ramp 30	1	1	30	0.0275	2.9175	0.6080
ramp 40	1	1	40	0.2028	2.3916	0.8391
ramp 50	1	1	50	0.6376	1.0872	1.1918
ramp 60	1	1	60	1	0	1.7321
ramp 70	1	1	70	1	0	2.7475

Table B.2: Ramp simulation results

Appendix C

Matlab Code

C.1 occupancy_grid.m

```
%function ocgrid = occupancy_grid()
clear all;

save_occ = true;
fpat = 'rover_data';
folder = 'basalt'; %folder for data saving
dataname = 'all_terrain';
ext = '.txt';
data = load([fpat filesep folder filesep dataname ext]);

disp(['Load ' dataname ' Complete']);

[r_data,c_data] = size(data);
cm = 30;
cell_dim = cm/100; %20cm

[cmin i] = min(data);
[cmax i] = max(data);

%% Initialization
global occ_grid cell_size window gfilter max_vel vel_table alpha

max_vel = 3; %m/s
%window size for roughness calc 2*window + 1 = side
window = 1; %window = 2 => 1 meter x 1 meter frame
%gaussian_filter();
alpha = 1.0;

cell_size = ceil((cmax-cmin)/cell_dim); %calc grid size (xaxis, ...
    yaxis, zaxis)

%occ_grid = cell(cell_size(1),cell_size(2));
cell_grid = struct('first',1,'x',0,'y',0,'z',0,...
    'xx',0,'yy',0,'zz',0,'xy',0,'xz',0,'yz',0,'count',0,...
    'coeffs',zeros(3,1),'normal',zeros(3,1),...
    'roughnessCenter',0,'roughness',0,'vel',0);

%initializing occ_grid to default values
for i = 1:cell_size(1)
```

```
39      for j = 1:cell_size(2)
40          occ_grid{i,j} = cell_grid;
41      end
42  end
43
44  %positively align coordinates
45  adjs = [0 0 0];
46  if cmin(1) < 0 %x axis
47      adjs(1) = abs(cmin(1));
48  end
49
50  if cmin(2) < 0 %x axis
51      adjs(2) = abs(cmin(2));
52  end
53
54  count_points = 0;
55
56  %initialize vel table
57  vel_table = ones(cell_size(1),cell_size(2))*NaN;
58
59  disp('Initialization Complete');
60
61  %% Calculating Speed Map
62  %ct = 0;
63  % aviobj = avifile('basalthills_sample_all.avi','compression','None');
64
65  for i=1:r_data
66      data_adjs = adjs + data(i,:);
67      index = ceil(data_adjs/cell_dim);
68
69      row = index(1); col = index(2);
70      if row == 0
71          row = 1;
72      end
73      if col == 0
74          col = 1;
75      end
76
77      add_point(row,col,data_adjs);
78      count_points = count_points + 1;
79
80  %         if mod(i,500)== 0 || i==r_data
81  %             %ct = ct + 1;
82  %             h1 = figure(1); clf(1);
83  %             scatter(x(:),y(:),2,vel_table(:));
84  %             title('Speed Map');
85  %             colorbar
86  %
87  %             F = getframe(h1);
88  %             aviobj = addframe(aviobj,F);
89  %         end
90      %pause;
91  end
92  disp('PCA Done');
93  r = cell_size(1); %# rows = length of y axis
94  c = cell_size(2); %# cols = length of x axis
95  for i=1:r
96      for j=1:c
97          if occ_grid{i,j}.first == 0
98              occ_grid{i,j}.roughnessCenter = calc_roughness(i,j,0);
```

```
99          end
100     end
101 end
102 for i=1:r
103     for j=1:c
104         if occ_grid{i,j}.first == 0
105             occ_grid{i,j}.roughness = calc_roughness(i,j,1);
106             occ_grid{i,j}.vel = occ_grid{i,j}.roughness*max_vel;
107             vel_table(i,j) = occ_grid{i,j}.vel;
108         end
109     end
110 end
111 disp('Roughness Calc Done');
112
113 if save_occ == true
114     occname = sprintf('occ_grid_%s_%0.2dcm.mat',dataname,cm);
115     save([fpat filesep folder filesep occname],'occ_grid');
116     disp('Save Complete');
117 end
118
119 disp('Speed Map Complete');
120 %return;
121 % aviobj = close(aviobj);
122
123 minV = (min(vel_table))/max_vel
124 maxV = (max(vel_table))/max_vel
125
126 %% Speed Map Results
127 h1 = figure(1); clf(1);
128 %scatter3(x(:),y(:),vel_table(:),3,vel_table(:))
129
130 r = cell_size(2); %# rows = length of y axis
131 c = cell_size(1); %# cols = length of x axis
132 [x y] = meshgrid(1:r,1:c);
133
134 % scatter(x(:),y(:),1,vel_table(:),'.');
135 % mesh(1:r,1:c,vel_table); %use for small data sets
136 scatter(x(:),y(:),3,vel_table(:),'filled');
137 %scatter3(x(:),y(:),1-(vel_table(:)/3),7,vel_table(:));
138 t = sprintf('Speed Map [cell:%0.2dcm, win:%d]',cm,window);
139 title(t);
140 contourcmap([0:0.03:3],'jet');
141 colorbar
142 % axis([140 350 125 310]); %for marscape only
143 view(0,90); %for small data sets
144 xlabel('x-axis');
145 ylabel('y-axis');
146 ylabel(colorbar,'Velocity [m/s]')
147
148 h2 = figure(2); clf(2);
149 %scatter3(x(:),y(:),1-(vel_table(:)/3),7,vel_table(:));
150 % mesh(1:r,1:c,1-(vel_table/3));
151 scatter(x(:),y(:),3,1-(vel_table(:)/3),'filled'); %use this for ...
      larger data sets
152 t = sprintf('Roughness Map [cell:%0.2dcm, win:%d]',cm,window);
153 title(t);
154 contourcmap([0:0.01:1],'jet');
155 colorbar
156 % axis([140 350 125 310]); %for marscape only
157 view(0,90);
```

```
158 xlabel('x-axis');
159 ylabel('y-axis');
160 ylabel(colorbar,'Roughness')
161
162 fpat = 'results';
163 filename = ...
        sprintf('%s_speed_map_cell_%0.2dcm_win_%d',dataname,cm,window);
164 print(h1,'-depsc2', [fpat filesep folder,filesep,filename]);
165
166 filename = ...
        sprintf('%s_roughness_map_cell_%0.2dcm_win_%d',dataname,cm,window);
167 print(h2,'-depsc2', [fpat filesep folder,filesep,filename]);
168
169 return;
170
171 %% Elevation Map
172 ele_map = ones(cell_size(1),cell_size(2))*NaN;
173 for i = 1:cell_size(1)
174     for j = 1:cell_size(2)
175         if occ_grid{i,j}.first == 0
176             xN = (i)*cellSize-cellSize/2;
177             yN = (j)*cellSize-cellSize/2;
178             mA = occ_grid{i,j}.coeffs(1);
179             mB = occ_grid{i,j}.coeffs(2);
180             mC = occ_grid{i,j}.coeffs(3);
181             mHeight = mA*xN + mB*yN + mC;
182 %             mHeight = occ_grid{i,j}.z/occ_grid{i,j}.count;
183             ele_map(i,j) = mHeight;
184         end
185     end
186 end
187
188 h3 = figure(3); clf(3);
189 scatter(x(:),y(:),3,ele_map(:),'filled');
190 % view(90,270);
191 title('Elevation Map');
192 % contourcmap([-4:0.1:2],'jet');
193 colorbar
194 ylabel(colorbar,'Elevation[meters]');
195 xlabel('Longitude[meters]');
196 ylabel('Latitude[meters]');
197
198 filename = sprintf('%s_elevation_map',dataname);
199 print(h3,'-depsc2', [fpat filesep folder,filesep,filename]);
```

C.2 calc_roughness.m

```
1 %% calc_roughness(row,col)
2 function roughness = calc_roughness(row,col,getNbr)
3
4 global occ_grid cell_size window alpha gfilter
5 %       c-1  c  c+1
6 %     |---|---|---|
7 % r-1 |   |   |   |
8 %     |---|---|---|
9 % r   |   | * |   |
```

```
%     |---|---|---|
% r+1 |   |   |   |
%     |---|---|---|
% boundary cond: 1-> max
%if row-1 < 1 %no neighbor below boundary
    cellSize = 0.3;

    center = occ_grid{row,col}.normal;
    center_mean = occ_grid{row,col}.z/occ_grid{row,col}.count;
    p = occ_grid{row,col}.coeffs; %plane equation
    mA = p(1);
    mB = p(2);
    %mC = -mA*((row-1)*cellSize) - mB*((col-1)*cellSize) - p(3) + ...
        center_mean;
    mC = p(3); %center_mean;%mA*((row-1)*cellSize) + ...
        mB*((col-1)*cellSize) + center_mean;

    roughness_slope = 0; roughness_DPNV = 0; ...
        roughness_MeanHeightCost = 0; roughness_Nbr = 0;
    ResidualCost = 0; CovHeightCost = 0;

    obstacle = 0.3;

    count = 0;
    weight = 0;
    dot_prod = zeros(2*window+1);

    roughness_slope = getSlope(row,col);

    centerResidualCost = ...
        log10(1/abs(getResidual(row,col)))/log10(1/(0.05^2));
    if centerResidualCost > 1
        centerResidualCost = 1;
    elseif centerResidualCost < 0
        centerResidualCost = 0;
    end
%     centerResidualCost = 1;

    for i=-window:1:window %row
        for j=-window:1:window %column
            if (row+i ≥ 1 && row+i ≤ cell_size(1) &&...
                col+j ≥ 1 && col+j ≤ cell_size(2) &&...
               ¬(i==0 && j==0))
                if occ_grid{row+i,col+j}.first == 0
                    residual = abs(getResidual(row+i,col+j));
                    if residual==0
                        %disp('Residual is 0');
                        ResidualCost = ...
                            1;%log10(1/1e-6);%10000;%1000000000;
                        %max residual of 1e-6 is having a max ...
                            error of 1mm
                    else
%                         ResidualCost = log10(1/residual);
                        ResidualCost = ...
                            log10(1/(residual))/log10(1/(0.05^2));
%                         ResidualCost = 1/residual;
                        if ResidualCost > 1
                            ResidualCost = 1;
                        elseif ResidualCost < 0
                            ResidualCost = 0;
```

```
63                          end
64
65                      end
66  %                     ResidualCost = 1;
67  %                     ResidualCost = ResidualCost*6;
68                      cov = computeCov(row+i,col+j);
69
70                      count = count + 1; %count how many used
71                      nb = occ_grid{row+i,col+j}.normal;
72
73                      roughness_DPNV = abs(dot(nb,center));
74
75                      xN = (row+i)*cellSize-cellSize/2;
76                      yN = (col+j)*cellSize-cellSize/2;
77
78  %                     xN = (i)*cellSize+cellSize/2; %testing
79  %                     yN = (j)*cellSize+cellSize/2; %testing
80
81                      mA_n = occ_grid{row+i,col+j}.coeffs(1);
82                      mB_n = occ_grid{row+i,col+j}.coeffs(2);
83                      mC_n = occ_grid{row+i,col+j}.coeffs(3);
84                      nHeight = mA_n*xN + mB_n*yN + mC_n;
85  %                     nHeight = mA_n*0.15 + mB_n*0.15 + mC_n; %testing
86
87  %                     roughness_MeanHeightCost = ...
    exp(abs(center_mean-mean_z)/-obstacle);
88                      %proj_height = mA*((i*cellSize)) + ...
                            mB*((j*cellSize)) + mC;% + center_mean;
89
90                      proj_height = mA*xN + mB*yN + mC;
91
92  %                     mean_z = ...
    occ_grid{row+i,col+j}.z/occ_grid{row+i,col+j}.count; %testing
93  %                     r_N = abs(mean_z-center_mean)/obstacle; %testing
94
95                      r_N = abs(proj_height-nHeight)/obstacle;
96                      if r_N > 1
97                          r_N = 1;
98                      end
99                      roughness_MeanHeightCost = 1-r_N;
100
101                     r_slope = getSlope(row+i,col+j);
102
103                     dot_prod(i+window+1,j+window+1) = ...
                            roughness_DPNV*roughness_MeanHeightCost*r_slope...
104                         *occ_grid{row+i,col+j}.count*ResidualCost;
105 %                         *occ_grid{row+i,col+j}.count;
106
107                     weight = weight + ...
                            occ_grid{row+i,col+j}.count*ResidualCost;
108 %                     weight = weight + occ_grid{row+i,col+j}.count;
109
110                     if getNbr == 1
111                         roughness_Nbr = roughness_Nbr + ...
                                occ_grid{row+i,col+j}.roughnessCenter;
112                     end
113                 end
114             end %boundary checker
115         end %j
116     end %i
```

```
117     %dot_prod
118     %roughness: 0 = very rough; 1 = same plane
119     %roughness = sum(sum(dot_prod.*gfilter))
120     if count == 0
121         roughness = count;
122     else
123         roughness = ...
                (sum(sum(dot_prod))/weight)*roughness_slope*centerResidualCost;
124         if getNbr == 1
125             roughness = ...
                    power(alpha*occ_grid{row,col}.roughnessCenter + ...
                    (1-alpha)*roughness_Nbr/count,2);
126         end
127     end
128
129 function s = getSlope(row,col)
130     global occ_grid;
131
132     slopeThresh = tan(60*pi/180);
133     slopex = abs(occ_grid{row,col}.coeffs(1))/slopeThresh;
134     slopey = abs(occ_grid{row,col}.coeffs(2))/slopeThresh;
135     if slopex > slopey
136         slope = slopex;
137     else
138         slope = slopey;
139     end
140
141     if slope<1
142         s=1-(slope^4);
143     else
144         s=0;
145     end
146
147 function cov = computeCov(row,col)
148 global occ_grid
149 cov(1,1) = occ_grid{row,col}.xx/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.x/occ_grid{row,col}.count)^2;
150 cov(1,2) = occ_grid{row,col}.xy/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.x*occ_grid{row,col}.y)/(occ_grid{row,col}.count^2);
151 cov(1,3) = occ_grid{row,col}.xz/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.x*occ_grid{row,col}.z)/(occ_grid{row,col}.count^2);
152 cov(2,1) = cov(1,2);
153 cov(2,2) = occ_grid{row,col}.yy/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.y/occ_grid{row,col}.count)^2;
154 cov(2,3) = occ_grid{row,col}.yz/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.y*occ_grid{row,col}.z)/(occ_grid{row,col}.count^2);
155 cov(3,1) = cov(1,3);
156 cov(3,2) = cov(2,3);
157 cov(3,3) = occ_grid{row,col}.zz/occ_grid{row,col}.count - ...
        (occ_grid{row,col}.z/occ_grid{row,col}.count)^2;
```

C.3 add_point.m

```
1 %% add_point(row,col,point)
2 function add_point(row,col,point)
3 global occ_grid max_vel vel_table window cell_size
```

```
4  x = point(1); y = point(2); z = point(3);
5
6  occ_grid{row,col}.first = 0;
7
8  %LSQ fit plane
9  %f(x,y,z) = a*x + b*y + c*z + d = 0
10 %rewritten: z = Mx+Ny+b
11 %J(M,N,b) = SUM [(Mx+Ny+b) - z]^2
12 %gradient(J) = 2*SUM [(Mx+Ny+b) - z][x,y,1]
13 occ_grid{row,col}.x = occ_grid{row,col}.x + x;
14 occ_grid{row,col}.y = occ_grid{row,col}.y + y;
15 occ_grid{row,col}.z = occ_grid{row,col}.z + z;
16 occ_grid{row,col}.xx = occ_grid{row,col}.xx + x^2;
17 occ_grid{row,col}.yy = occ_grid{row,col}.yy + y^2;
18 occ_grid{row,col}.zz = occ_grid{row,col}.zz + z^2;
19 occ_grid{row,col}.xy = occ_grid{row,col}.xy + x*y;
20 occ_grid{row,col}.xz = occ_grid{row,col}.xz + x*z;
21 occ_grid{row,col}.yz = occ_grid{row,col}.yz + y*z;
22 occ_grid{row,col}.count = occ_grid{row,col}.count + 1;
23
24 % Sx = occ_grid{row,col}.x;
25 % Sy = occ_grid{row,col}.y;
26 % Sz = occ_grid{row,col}.z;
27 % Sxx = occ_grid{row,col}.xx;
28 % Syy = occ_grid{row,col}.yy;
29 % Szz = occ_grid{row,col}.zz;
30 % Sxy = occ_grid{row,col}.xy;
31 % Sxz = occ_grid{row,col}.xz;
32 % Syz = occ_grid{row,col}.yz;
33 % count = occ_grid{row,col}.count;
34
35 A = [occ_grid{row,col}.xx occ_grid{row,col}.xy occ_grid{row,col}.x;
36     occ_grid{row,col}.xy occ_grid{row,col}.yy occ_grid{row,col}.y;
37     occ_grid{row,col}.x occ_grid{row,col}.y occ_grid{row,col}.count];
38
39 Y = [occ_grid{row,col}.xz occ_grid{row,col}.yz occ_grid{row,col}.z]';
40
41 occ_grid{row,col}.coeffs = pinv(A) * Y;
42 % d = ...
       occ_grid{row,col}.x*(-(occ_grid{row,col}.yy*occ_grid{row,col}.x) ...
       + occ_grid{row,col}.xy * occ_grid{row,col}.y) +...
43 %     occ_grid{row,col}.xy*( ...
       (occ_grid{row,col}.y*occ_grid{row,col}.x) - ...
       occ_grid{row,col}.xy*occ_grid{row,col}.count) +...
44 %     ...
       occ_grid{row,col}.xx*(-(occ_grid{row,col}.y*occ_grid{row,col}.y) ...
       + occ_grid{row,col}.yy*occ_grid{row,col}.count);
45 %
46 % A = (Sz * (-(Sx*Syy) + Sxy*Sy) +...
47 %     Syz * ( (Sx*Sy) - Sxy*count) +...
48 %     Sxz * (-(Sy*Sy) + Syy*count))/d;
49 %
50 % B = (Sz * ( (Sx*Sxy) - Sxx*Sy) +...
51 %     Syz * (-(Sx*Sx) + Sxx*count) +...
52 %     Sxz * ( (Sy*Sx) - Sxy*count))/d;
53 %
54 % C = (Sz * (-(Sxy*Sxy) + Sxx*Syy) +...
55 %     Syz * ( (Sxy*Sx) - Sxx*Sy) +...
56 %     Sxz * (-(Syy*Sx) + Sxy*Sy))/d;
57 %
```

```
% occ_grid{row,col}.coeffs = [A B C]';

%Normal = gradient(f) = [a b c]';
occ_grid{row,col}.normal = [-occ_grid{row,col}.coeffs(1) ...
    -occ_grid{row,col}.coeffs(2) 1]'; %[-M -N 1]
occ_grid{row,col}.normal = ...
    occ_grid{row,col}.normal/norm(occ_grid{row,col}.normal);
```

C.4 getResidual.m

```
function r = getResidual(row,col)
global occ_grid
%1/n sum( (z-height(x,y))^2)
coeffs = occ_grid{row,col}.coeffs;
A = coeffs(1); B = coeffs(2); C = coeffs(3);
r = occ_grid{row,col}.zz + ...
    [occ_grid{row,col}.xx,occ_grid{row,col}.yy,occ_grid{row,col}.count] ...
    * occ_grid{row,col}.coeffs.^2 + ...
    2*(A*(C*occ_grid{row,col}.x + B*occ_grid{row,col}.xy - ...
        occ_grid{row,col}.xz) + ...
       B*(C*occ_grid{row,col}.y - occ_grid{row,col}.yz) - ...
       C*occ_grid{row,col}.z);
r = r/occ_grid{row,col}.count;
```

Appendix D

Source Code

D.1 RoughnessMap.h

```
1  #ifndef _ROVERNAV_SPEEDMAP_H_
2  #define _ROVERNAV_SPEEDMAP_H_
3
4  #include "RoughnessMapCell.h"
5  #include "rovernav/mapping/roughness/RoughnessMapParameters.h"
6
7  #include "rovernav/core/MapView.h"
8  #include "rovernav/core/Matrix.h"
9  #include "rovernav/math/LinearAlgebra.h"
10
11 #include "rovernav/travmap/TravMap.h"
12 #include "rovernav/travmap/TerrainMapAccumulator.h"
13
14 namespace rovernav {
15   template <class Indexer = StandardMapIndexer >
16   class RoughnessMap : public MapView<RoughnessMapCell, Indexer> { ...
         //inheritance of MapView class
17   public:
18     typedef RoughnessMapCell pixel_type;
19     typedef RoughnessMapParameters Parameters;
20
21     RoughnessMap(const MapReference& mapRef,
22                  const Parameters& params = Parameters()) : ...
                        //constructor
23       MapView<RoughnessMapCell, Indexer>(mapRef),
24       m_params(params),
25       m_cellSize(mapRef.cellLength().x()),
26       m_winLen(2*m_params.winSize+1),
27       m_certAlpha(-m_winLen),
28       m_maxCells((m_winLen*m_winLen)-1),
29       m_terrainMapAccumulator(mapRef) {
30     }
31
32     Parameters const& parameters() const { return m_params; }
33
34     /* importTerrainMap
35     *  imports TerrainMapAccumulator from MapMaker for reference ...
           to Normals and other needed data.
36     */
```

```
37      void importTerrainMap(TerrainMapAccumulator<Indexer>& map) {
38        m_terrainMapAccumulator = map;
39      }
40
41      /* caclRoughness(x,y)
42      *  Calculates the roughness map using a specified window size ...
            and linearly associates speed with roughness to creates ...
            speed limit.
43      *  - Implicitly updates speedMap
44      */
45      void calcRoughness(int x, int y, bool getNeighbors) {
46        int count = 0, max_x = this->sizeX(), max_y = this->sizeY();
47        typename TerrainMapAccumulator<Indexer>::pixel_type *tmCell ...
              = &( m_terrainMapAccumulator(x,y) ); //TerrainMap center ...
              cell
48        typename TerrainMapAccumulator<Indexer>::pixel_type* nCell; ...
              //TerrainMap neighbor cell
49        typename RoughnessMap<Indexer>::pixel_type *cCell = ...
              &(*this)(x,y); //Speed map center cell
50        typename RoughnessMap<Indexer>::pixel_type *nSMCell; //Speed ...
              map neighbor cell
51
52        double r_tot = 0, weight = 0, residual = 0;
53        double ResidualCost = 0;
54        unsigned PlanarityCost = 0, cPlanarity = 0;
55        double roughness_DPNV = 0, roughness_MeanHeightCost = 0, ...
              roughness_Neighbors = 0;
56        double proj_height = 0, nbr_height = 0;
57
58        Matrix3x3 cov;
59        Matrix2 cov2; //covariance 2x2 matrix for eigenvalue computation
60        Vector2 d; //eigenvalues
61
62        //Get center cell's covariance matrix
63        tmCell->planeFitMoments().computeCovariance(cov);
64        cov2(0,0) = cov(0,0);
65        cov2(0,1) = cov(0,1);
66        cov2(1,0) = cov(1,0);
67        cov2(1,1) = cov(1,1);
68
69        d = eigenval2(cov2); //eigenvalue calculation
70        (d(0)==0 || d(1)==0) ? PlanarityCost = 0 : PlanarityCost = ...
              1; //Any zero eigenvalue fails planarity
71        cPlanarity = PlanarityCost;
72
73        if( tmCell->numPoints() ≥ m_params.minCellPts && ...
              cPlanarity!=0) { //check if sufficient numPts and pass ...
              planarity test
74          for(int i = -m_params.winSize; i≤ (int)m_params.winSize; ...
                i++) {
75            for(int j=-m_params.winSize; j≤ (int)m_params.winSize; ...
                  j++) {
76              if( x+i ≥ 0 && x+i ≤ max_x &&
77                  y+j ≥ 0 && y+j ≤ max_y &&
78                  !(i==0 && j ==0) ) {
79
80                nCell = &(m_terrainMapAccumulator)(x+i,y+j);
81
82                //Cost function for confidence: planarity test
83                nCell->planeFitMoments().computeCovariance(cov);
```

```
84                  if( cov(0,0) ≤ 0 || cov(1,1) ≤ 0 ) {
85                    std::cout << "******************Cov(2,2) == ...
                          0***************\n";
86                  }
87                  //PlanarityCost = (cov(0,0)+cov(1,1))/2; //cost ...
                        factor range [0:infiniti]
88                  cov2(0,0) = cov(0,0);
89                  cov2(0,1) = cov(0,1);
90                  cov2(1,0) = cov(1,0);
91                  cov2(1,1) = cov(1,1);
92
93                  d = eigenval2(cov2); //eigenvalue calculation
94                  (d(0)==0 || d(1)==0) ? PlanarityCost = 0 : ...
                        PlanarityCost = 1;
95
96                  if( nCell->numPoints() ≥ m_params.minCellPts && ...
                        PlanarityCost!=0 ) { //checks if there is ...
                        sufficient # points in cell
97
98                    if( !getNeighbors ) {
99                      //Cost function for confidence: covariance of Z ...
                            error, plane fit in Z direction
100                     residual = ...
                            nCell->planeFitMoments().getResidual(nCell->getPlane());
101
102                     if( residual == 0 ) {
103                       std::cout << "******************Resdiual == ...
                              0****************\n";
104                     }
105                     //ResidualCost = 1/residual; //cost factor range ...
                            [0:infiniti]
106                     ResidualCost = log10(1/residual)/6; //cost ...
                            factor range [0:1] best=1e-6 (1mm); worst=1 (1m)
107                     (ResidualCost > 1 ? ResidualCost = 1 : ...
                            (ResidualCost < 0 ? ResidualCost = 0 : 1));
108
109                     //Cost function for roughness: dot product ...
                            normal vectors
110                     roughness_DPNV = ...
                            fabs(dot_prod(tmCell->terrainMapCell.normal, ...
                            nCell->terrainMapCell.normal)); //cost ...
                            factor range [0:1]
111
112                     //Cost function for roughness: Δ proj mean height
113                     proj_height = ...
                            tmCell->getPlane().height(i*m_cellSize,j*m_cellSize);
114                     nbr_height = nCell->getPlane().height(0,0);
115
116                     roughness_MeanHeightCost = fabs(proj_height - ...
                            nbr_height)/m_params.Δ; //Δ from roughnessParams
117                     (roughness_MeanHeightCost > 1) ? ...
                            roughness_MeanHeightCost = 0 : ...
                            roughness_MeanHeightCost = ...
                            1-roughness_MeanHeightCost; //cost factor ...
                            range [0:1]
118
119                     //sum of products
120                     r_tot += ...
                            (roughness_DPNV*roughness_MeanHeightCost) * ...
                            (nCell->numPoints()*ResidualCost);
```

```
121                   weight += nCell->numPoints()*ResidualCost;
122                   count++;
123                 }
124                 else {
125                   nSMCell = &(*this)(x+i,y+j); //Speed map center cell
126                   roughness_Neighbors += nSMCell->roughnessCenter; ...
                          //Accumlate neighbors center roughnesses
127                 }// end getNeighbors check
128               }// end numPoints cell check
129
130             }// end range check
131           }// end j
132         }// end i
133       }// end numPoints check for center cell
134
135       if( !getNeighbors ) {
136         cCell->numCells = count; //# cells used to calculate roughness
137
138         if( count==0 ) {
139           cCell->roughnessCenter = cCell->roughness = count;
140         }
141         else {
142           cCell->roughnessCenter = (r_tot/weight);
143         }
144       }
145       else {
146         if( cCell->numCells!=0 ){
147           roughness_Neighbors /= cCell->numCells; //average of ...
                  neighbor roughness: Should filter with Gaussian ...
                  distribution?
148         }
149         cCell->roughness = ...
              1-pow(m_params.rAlpha*cCell->roughnessCenter + ...
              (1-m_params.rAlpha)*roughness_Neighbors,2); ...
              //non-linear transformation
150         cCell->speed = cCell->roughness*m_params.maxVel;
151       }
152     }// calcRoughness
153
154     void update() {
155       //Calculate center cell roughness
156       for(int i=0;i<(int)(this->sizeX());i++) {
157         for(int j=0;j<(int)(this->sizeY());j++) {
158           calcRoughness(i,j,false);
159         }
160       }
161       //Get neighbor roughness
162       for(int i=0;i<(int)(this->sizeX());i++) {
163         for(int j=0;j<(int)(this->sizeY());j++) {
164           calcRoughness(i,j,true);
165         }
166       }
167     }//update
168
169     void exportRoughnessMap(TravMap<StandardMapIndexer>& map) const{
170       map.setMapReference(this->mapRef());
171       Vector2i pixel;
172
173       for(pixel.x() = 0; pixel.x() < map.size().x(); pixel.x()++) {
```

```
174         for(pixel.y() = 0; pixel.y() < map.size().y(); ...
                pixel.y()++) {
175
176           typename TravMap<StandardMapIndexer>::pixel_type *to = ...
                  &map(pixel);
177           const typename RoughnessMap<Indexer>::pixel_type *fromSM ...
                  = &(*this)(pixel);
178           const typename ...
                  TerrainMapAccumulator<Indexer>::pixel_type *fromTM = ...
                  &( m_terrainMapAccumulator(pixel) ); //reference to ...
                  TerrainMap
179
180           to->traversability = 1-fromSM->roughness;
181           //to->certainty = 1 - ...
                  exp(m_certAlpha*(float)(fromSM->numCells)/m_maxCells); ...
                  //could be similar to morphin: 1-exp(alpha*numPtsInCell)
182           to->certainty = (float)(fromSM->numCells)/m_maxCells;
183           to->height = fromTM->getPlane().height(0,0);
184           to->time = fromTM->time();
185
186         } // for x
187       } // for y
188     } // export(TravMap<StandardMapIndexer>
189
190   protected:
191     Parameters m_params;
192     float m_cellSize;
193     unsigned m_winLen;
194     int m_certAlpha;
195     unsigned m_maxCells;
196
197     TerrainMapAccumulator<Indexer> m_terrainMapAccumulator;
198   }; // class RoughnessMap
199 } // namespace rovernav
200 #endif // _ROVERNAV_SPEEDMAP_H_
```

D.2 RoughnessMapCell.h

```
 1 #ifndef rnRoughnessMapCell_h
 2 #define rnRoughnessMapCell_h
 3
 4 namespace rovernav
 5 {
 6   // Structure for Speed Maps
 7   struct RoughnessMapCell
 8   {
 9     double roughness;       // Adjusted Roughness
10     double roughnessCenter; // Roughness based around center
11     double speed;
12     unsigned numCells;      // Number of cells used in roughness ...
           calculation for certainty
13
14     RoughnessMapCell() :
15       roughness(0),
16       roughnessCenter(0),
17       speed(0),
```

```
18        numCells(0)
19      {}
20
21      void clear()
22      {
23        roughness = 0.;
24        roughnessCenter = 0;
25        speed = 0.;
26        numCells = 0;
27      }
28    }; // RoughnessMapCell
29  } // namespace rovernav
30  #endif // rnRoughnessMapCell_h
```

References

[1] Max Bajracharya, Mark W. Maimone, and Daniel Helmick. Autonomy for mars rovers: Past, present, and future. *Computer*, 41:44–50, 2008. ISSN 0018-9162. doi: http://doi.ieeecomputersociety.org/10.1109/MC.2008.515.

[2] Joseph Carsten, Arturo Rankin, David Ferguson, and Anthony (Tony) Stentz. Global path planning on-board the mars exploration rovers. In *IEEE Aerospace Conference*, 2007.

[3] Mattia Castelnovi, Ronald Arkin, and Thomas R. Collins. Reactive speed control system based on terrain roughness detection. *International Conference on Robotics and Automation*, 2005.

[4] Lars B. Cremean, Tully Foote, Jeremy H. Gillula, George H. Hines, Dmitriy Kogan, Kristopher L. Kriechbaum, Jeffrey C. Lamb, Jeremy Leibs, Laura Lindzey, Christopher E. Rasmussen, Alexander D. Stewart, Joel W. Burdick, and Richard M. Murray. Alice: An information-rich autonomous vehicle for high-speed desert navigation: Field reports. *J. Robot. Syst.*, 23(9), June 2006.

[5] Matthew C. Deans, Terrence Fong, Mark Allan, Xavier Bouyssounouse, Maria Bualat, Lorenzo Flueckiger, Linda Kobayashi, Susan Lee, David Lees, Eric Park, Estrellina Pacis, Liam Pedersen, Debbie Schreckenghost, Trey Smith, Vinh To, and Hans Utz. Robotic scouting for human exploration. In *Proc. AIAA Space*, 2009.

[6] A.S. El-Kabbany and A. Ramirez-Serrano. Terrain roughness assessment for human assisted ugv navigation within heterogeneous terrains. *International Conference of Robotics and Biomimetics*, 2009.

[7] Terrence Fong, Mark Allan, Xavier Bouyssounouse, Maria G. Bualat, Matthew C. Deans, Laurence Edwards, Lorenzo Flckiger, Leslie Keely, Susan Y. Lee, David Lees, Vinh To, and Hans Utz. Robotic site survey at haughton crater. *iSAIRAS*, 2008.

[8] Donald B. Gennery. Traversability analysis and path planning for a planetary rover. *Auton. Robots*, 6(2):131–146, April 1999. ISSN 0929-5593. doi: 10.1023/A:1008831426966. URL `http://dx.doi.org/10.1023/A:1008831426966`.

[9] Thomas D. Gillespie. *Fundamentals of Vehicle Dynamics*. 1992.

[10] Steven Goldberg, Mark Maimone, and Larry Mattheis. Stereo vision and rover navigation software for planetary exploration. *IEEE Aerospace Conference Proceedings*, March 2002.

[11] Grant H. Heiken, David T. Vaniman, and Bevan M. French. *Lunar Sourcebook - A User's Guide to the Moon*. Cambridge University Press, Cambridge, England, 1991.

[12] Regis Hoffman and Eric Krotkov. Terrain roughness measurement from elevation maps, 1989.

[13] A. Howard and H Seraji. Vision-based terrain characterization and traversability assessment. *J. Robotic Syst.*, 18(10):577–587, 2001.

[14] Karl Iagnemma. Mobile robot rough-terrain control (rtc) for planetary exploration. In *Proceedings of the 26th ASME Biennial Mechanisms and Robotics Conference, DETC*, pages 10–13, 2000.

[15] Karl Iagnemma and Steven Dubowsky. *Mobile Robots in Rough Terrain*. 2004.

[16] Gang-Gyoo Jin, Yun-Hyung Lee, Hyun-Sik Lee, and Myung-Ok So. Traversability analysis for navigation of unmanned robots. *SICE Annual Conference*, 2008.

[17] C. Leger, A. Trebi-Ollennu, J. Wright, S. Maxwell, R. Bonitz, J. Biesiadecki, F. Hartman, B. Cooper, E. Baumgartner, and M. Maimone. Mars exploration rover surface operations: Driving spirit at gusev crater. In *In Proceedings of the 2005 IEEE Conference on Systems, Man, and Cybernetics*, October 2005.

[18] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, Matthew Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. A perception-driven autonomous urban vehicle. *J. Field Robot.*, 25(10):727–774, October 2008. ISSN 1556-4959. doi: 10.1002/rob.v25:10. URL `http://dx.doi.org/10.1002/rob.v25:10`.

[19] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers: Field reports. *J. Field Robot.*, 24(3):169–186, March 2007. ISSN 1556-4959. doi: 10.1002/rob.v24:3. URL `http://dx.doi.org/10.1002/rob.v24:3`.

[20] NASA. Mars exploration rover landings, January 2004. URL `http://marsrovers.jpl.nasa.gov/newsroom/merlandings.pdf`.

[21] NASA. Mars science laboratory landing, July 2012. URL `http://mars.jpl.nasa.gov/msl/news/pdfs/MSLLanding.pdf`.

[22] Liam Pedersen, Mark Allan, Hans Utz, Matthew Deans, Xavier Bouyssounouse, Yoonhyuk Choi, Lorenzo Flückiger, Susan Y. Lee, Vinh To, Jonathan Loh, William Bluethmann, Robert R. Burridge, Jodi Graf, and Kimberly Hambüchen. Tele-operated lunar rover navigation using lidar. 2012.

[23] Homayoun Seraji. Traversability index: A new concept for planetary rovers. *International Conference on Robotics and Automation*, 1999.

[24] Matthew Spenko, Yoji Kuroda, Steven Dubowsky, and Karl Iagnemma. Hazard avoidance for high-speed mobile robots in rough terrain. *J. Field Robotics*, 23 (5):311–331, 2006.

[25] David Stavens and Sebastian Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. *Conference on Uncertainty in AI*, 2006.

[26] David Stavens, Gabriel Hoffmann, and Sebastian Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. *International Joint Conference on Artificial Intelligence*, 2007.

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

[28] Sebastian Thrun, Mike Montemerlo, and Andrei Aron. Probabilistic terrain analysis for high speed. *Robotics Science and Systems II*, August 2006.

[29] Christopher Urmson, Joshua Anhalt, Daniel Bartz, Michael Clark, Tugrul Galatali, Alexander Gutierrez, Sam Harbaugh, Joshua Johnston, Hiroki Kato, Phillip L Koon, William Messner, Nick Miller, Aaron Mosher, Kevin Peterson, Charlie Ragusa, David Ray, Bryon K Smith, Jarrod M Snider, Spencer Spiker, Joshua C Struble, Jason Ziglar, and William (Red) L. Whittaker. A robust approach to high-speed navigation for unrehearsed desert terrain. *Journal of Field Robotics*, 23(8):467–508, August 2006.

[30] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker , Robert E Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Ragunathan Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M Snider, Joshua C Struble, Anthony (Tony) Stentz, Michael Taylor , William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466, June 2008.

[31] Velodyne. Velodyne lidar hdl-32e data sheet. URL `http://velodynelidar.com/lidar/hdldownloads/97-0038c%20HDL-32E%20datasheet_APR2012.pdf`.