THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL AND NUCLEAR ENGINEERING

# OPTIMIZATION OF ENERGY-AWARE PATH-PLANNING IN GROUND ROBOTICS

VERONICA GRUNING
SPRING 2018

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Dr. Sean Brennan
Professor of Mechanical Engineering
Thesis Supervisor

Dr. Jacqueline O'Connor
Assistant Professor of Mechanical Engineering
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

# Abstract

This thesis presents an optimized approach to planning energy-aware paths for skid-steer vehicles. Specifically, this work expands upon previous models in three ways: (1) this work uses the instantaneous centers of rotation (ICRs) to predict the kinematics and power consumption of a robot's movement; (2) the power model accounts for the changes in ground surface friction; and (3) the power model includes the effect of elevation changes on the energy usage of the robot. The total power needed to travel to a goal location is then combined with the kinematic model to plan energy-aware paths using a Sampling Based Model Predictive Optimization (SBMPO) algorithm. This method is demonstrated using simulated environments that are sampled to create a wide range of testing scenarios representative of real-world usages. The results show situations where a more energy efficient path for skid-steer robots on mixed terrain may occur by increasing the path distance. The results of this project are intended to inform energy consumption models for robotics.

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

I would like to thank my advisors, Dr. Sean Brennan and Dr. Jacqueline OConnor, first. Without your guidance, encouragement, and persistence I do not believe I would be the same engineer or person that I have become. Your investment in my success pushed me to do better, and the advice I have received along the way through my career has been invaluable.

To my supervisors at the Applied Research Laboratory, Dr. Karl Reichard and Dr. Jesse Pentzer, your previous work laid the foundation of my thesis, and your continual mentorship throughout my thesis and internship helped me to get across this finish line. I look forward to the work we may do together in my graduate career.

Thank you to my parents, Stephanie, Vic, and Joyce. My moms may not have given me a singing voice, but they taught me the importance of kindness and honesty. Dad, you pushed me harder than anyone else in this world ever has. This push got me to places I never imagined I would be. You always knew I could be doing better, and so I did.

I thank all the friends I have made along the way from classes, research, and clubs. You put up with the struggles and breakthroughs, and for that, I am grateful.

Finally, to my partner in crime Matt, I look forward to what the future has in store for us. Whatever our path may be and however it may be shaped I am sure it will be optimal.

# Chapter 1

# Introduction

The work on unmanned ground vehicles (UGV) is done primarily for farming, mining, disaster relief, and military purposes. In these applications, UGVs are often used for inspecting an area, to determine what might be out of place, for carrying loads, and even replacing a human to avoid dangerous situations. All of these applications demand a UGV that can autonomously, efficiently, and safely move. The motivation for this research is that the three demands have yet to be met.

Starting with safety, a robot must know when it is reaching its limits. Limitations for rigid bodies lie in a stability problem. Previous work has developed an algorithm to determine the location of the instantaneous centers of rotation of a maneuvering robot; however, it has yet to be applied to finding the current roll, pitch, and yaw stability of the vehicle. This thesis determines the associations between the changes in the Euler angles and the changes in the instantaneous centers of rotation (ICRs). Through this relationship, the UGV can assess its stability in any given situation, particularly those that may result in rollovers. In order to efficiently drive a UGV, the

power estimation algorithm should accurately calculate the energy used in past maneuvers and predict how much energy will be used in potential paths. Currently, algorithms exist that can use the coefficient of friction of a surface to determine how much power is required to travel between locations, but the research explained in the following chapters optimizes this algorithm by including the effects the changes in Euler angles and thus allowing examining of how terrain geometry changes power consumption. These adjustments were made through experimental and simulated procedures.

The simulation of a hill with various coefficients of frictions was created in MATLAB based on the data taken from a hill with mixed terrain types: asphalt, grass, and gravel. By utilizing a lawnmower pattern and taking the accelerometer data, a virtual representation of the terrain can be made. While data acquisition is occurring, the power usage is recorded using power loggers. The power demand is associated with the location in space at that given time, in order to visualize the magnitude of the effect changes in terrain have on power consumption.

Dependent autonomy in robotics requires a reliable path, such that the operator can feel confident in leaving the UGV to perform a given task. Since, the current mode of estimating power lacks consideration for the roll, pitch, and yaw, so does the path planner. By avoiding rougher terrains, the goal is to produce a guidance strategy that minimizes the time that the robot is traveling on high-power-demand paths.

The research in this thesis elaborates on this model and includes the additional changes in terrain when traveling on a slope. In addition to a better representation of power consumption when traveling given path, this updated algorithm allows the operator to turn the time and power dials. This adjustability assigns the robot to either a more time efficient or energy efficient path. The validity of this newer algorithm is confirmed by comparing the results to those from the previous

versions of the algorithm and the experimental data from prior testing.

The remainder of this thesis is organized as follows. Chapter 2 provides a review of relevant research previously done in this field of ground vehicles. Chapter 3 provides the ground work for how to problem is approached, including notation and the basic motion of the vehicle. Chapter 4 outlines the methodology for the original estimation of energy consumed by skid steer robot based on ICR kinematics and surface friction. Chapter 5 introduces the process developed in order to include changes in elevation into the estimator. Chapter 6 presents the results from iterative simulations, and Chapter 7 provides conclusions of this work and future directions are examined.

# Chapter 2

# Literature Review

This chapter presents the relevant work done by others in the field of robotics as it relates to path planning for unmanned ground vehicles. The primary ground vehicle used in this thesis is a skid-steered robot. The kinematics required to model the vehicles behavior and the method used to find these equations of motion is presented. The model allows variable environments to be used in order to observe the effects changes in terrain have on power consumption. Finally, path- planning techniques that minimize surveillance time and energy are then described.

## 2.1 Research on Skid-Steer Robot Kinematics

The work of Martinez et al. [4], provides the kinematic background for tracked mobile robots, specifically skid-steered ground vehicles. Skid-steering robots are controlled by individually adjusting the velocities of the tracks, such that the vehicle heads left when the right track is moving faster. The effect of this slipping contact between the surface and track has on the vehicles motion

and the drivers ability to steer is shown in Figure 2.1 .

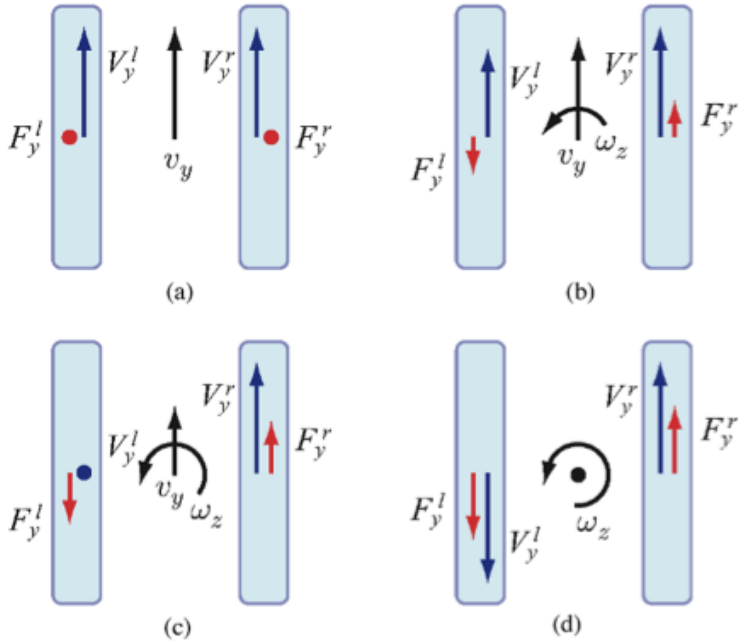


Figure 2.1: Skid-steer motion with variety of track combinations (a) Linear forward motion (b) Nonlinear forward motion with magnitude variance (c) Left turn with left track at rest (d) Left turn with revered velocities. [6]

The work done for this thesis to optimize energy-aware path-planning utilizes these types of robots since they are mechanically simpler and respond faster to user input [14]. However, this variation in wheel velocities creates challenges in describing the motion of the tracks, because standard methods applied to wheeled robots such as assuming slip-free contact do not accurately predict the behavior of tracked robots [4]. Although Martinez et al. performed their research on a planar uniform hard terrain type, and the trials were done a variety of surface coefficients and pitch orientations, the geometric relationships developed can be applied to model the motion of the robot. They considered the vehicle and tracks to be separate rigid bodies. About the instantaneous center of rotation (ICR) there is no translational motion, only rotational. Therefore, the combined motion of the two rigid bodies was calculated to find the velocity at a location on the track. This

model and the algorithms implemented in the authors work is possible under the assumption of Kennedys ICR theorem [11].

The ICR is defined as the position on a rigid body in planar motion, where the relative instantaneous velocity is zero. Knowledge of the ICR location thereby explains how all other locations on the body are moving: they are considered to be strictly rotationally motion around the ICR [13]. Within the context of skid-steer kinematics, the two ICRs of greater significance are those associated with the left and right tracks. These ICRs are situated in between the terrain the vehicle moves across and the tracks or wheels of the robot. The method for calculating wheeled vehicles is applied to tracked robotics and is believed to have the same confidence level [9]. After finding the ICR of one track, the other point is mirrored on a parallel line to vehicles x- axis as illustrated in Figure 2.2 [11]. This assumption comes from Kennedys theorem, also known as the law of three centers, which states that all instantaneous centers of rotation shared by multiple rigid bodies, regardless of whether they are connected to each other, in relative planar motion to another must be located on the same line [9]. In order to use the locations of instantaneous centers as a kinematic model, the track velocities, heading, and orientation must be measurable, and only then can a mapping of the vehicles forward and angular speed be produced [12].

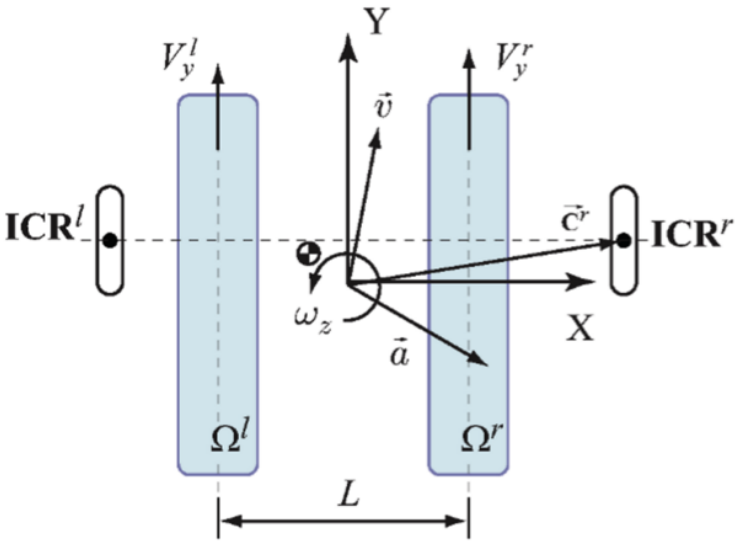


Figure 2.2: ICR locations for a virtual tracked vehicle in planar motion. [6]

## 2.2 Research on Filtering

The method implemented to identify ICR locations is the kinematic Extended Kalman Filter (EKF) developed by Pentzer, Brennan, and Reichard [8]. This technique begins with the kinematic equations of the vehicle, which are based on the Kennedys theorem and include the angular velocity and linear velocities in the x- and y-direction as functions of the ICRs x- and y-coordinates and the left and right track speeds. These three velocities determine the direction of the robot in a north by east mapping coordinate frame. The heading angle is the angular distance from the x-coordinate of the vehicle with respect to true north. Thus the velocities of the vehicle within the north by east plane are functions of the linear x- and y-velocities and the heading angle [12].

It is assumed in the model that the ICR locations stay the same with some small perturbations, much like the small angle approximation. The work by Martinez et al. [4] has proven this a reasonable statement, so long as the robot is moving at a low velocity on a hard surface with

minimal changes in elevation. The ICR locations, under the conditions described, are bound to the same point in the vehicular system notwithstanding the steering style as well [12]. Using the ICR coordinates, north by east coordinates, and heading angle of the state that was propagated and updated in the time step before, as well as the inputs controlling the velocity of each track in the same previous time step, the EKF discretizes the transmitted state for the current moment in time. This new state then becomes the propagated and updated state of the time step before, and the process is iterated until the traversal has concluded. In order for the EKF ICR to converge to the one derived from the mapped path, correction values called the Kalman gain and measurement innovation are used on the updated state. The Kalman gain is the relative coefficient given to the measurements and current state and is altered depending on how much updating needs to occur to achieve a desired performance rate. The measurement innovation is the difference between the measured north-east coordinates and angle heading and estimated locations. This extended filter does not permit the estimated ICR location to change when the robot is not moving, meaning the updated states ICR is held in the memory of the vehicle even when stationary [12].

Pentzer, Reichard, and Brennan [10] utilized ICR estimation when operating skid steered robots on various terrains to find the unknown variables, which are the necessary parameters required to make energy-based cost estimations [6]. The coefficient of friction, learned scaling factor, and coefficient of internal and rolling resistance, which is called the power model coefficient, are the constants found based on the ICR locations using the least-squares method developed by the same research team [9]. The power model coefficient, which is assigned as variable G, can be visualized in Figure 2.3, such that the value changes pigmentation as the vehicle changes the terrain it is traversing, and the results found that the color of the aerial image was highly correlated to the power model coefficient. As seen in Figure 2.4, the power model coefficient was found to increase

as the power map images coloring darkened. The change in pigmentation is due to noise reduction of the map and applying a k-means segmentation. Through segmenting the image, similar pixels are grouped together, which in the work being done generally leads to terrain with similar coefficients of friction being the same shade of grey.The correlation between the filtered map coloring and change in power model coefficient revealed for the skid-steer robot that the most energy is used when traversing the grass. But upon further investigation, another condition consumed an even larger amount of energy but only for short bursts of time: when the skid-steer vehicle was attempting to make a sharp turn. Chuy and Morales [1, 6] have also done work corroborating the observation that skid-steer vehicles making turns use the largest amount of power on a constant surface friction coefficient [5, 9].

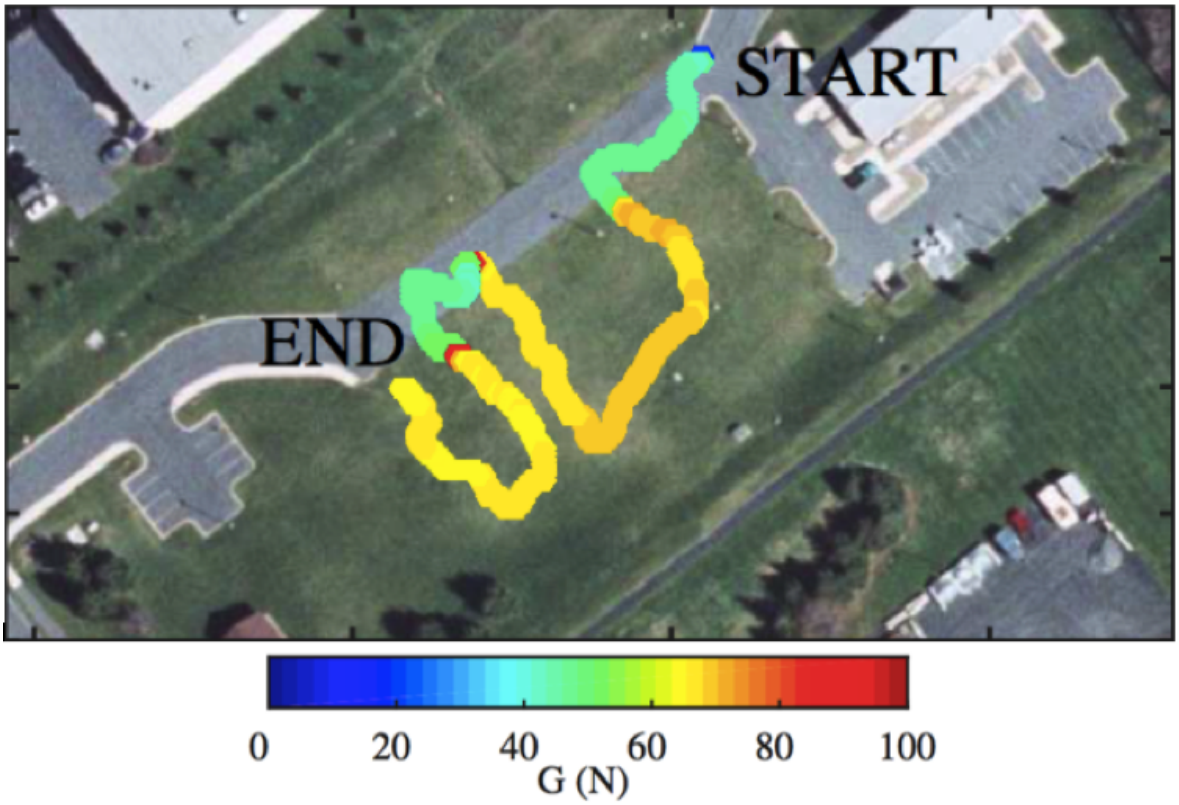


Figure 2.3: Estimation of power model coefficient G as vehicle travels a mixed surface path. [10]

Figure 2.4: Resulting power map from applying k-means segmentation and 9x9 median filter. [10]

## 2.3 Research on Path Planning

The method Pentzer, Reichard, and Brennan used to plan an energy-aware path was Sampling Based Model Predictive Optimization (SBMPO) algorithm, a model that predicts power usage while running on an A-star search algorithm. Path planning such as A*, D*, and potential field are extensions of Dijkstras algorithm from 1959 [2], and they are based in creating a grid over a map and estimating the cost of traveling from one cell to another [8, 10]. One difference from A* that is optimized using SBMPO is a grid where similarly positioned and orientations of nodes are consolidated to one node, and locations that surpass an energy cost are discarded from the possible nodes. The process of SBMPO path planning can be visualized in Figure 2.5, where each new node tree is made up of distinctively different new possible positions, and the node that most efficiently

moves from the start location to goal is taken as the new path. The cost of each node is estimated using the distance traveled from one node to another and the remaining distance the new node is from the goal.

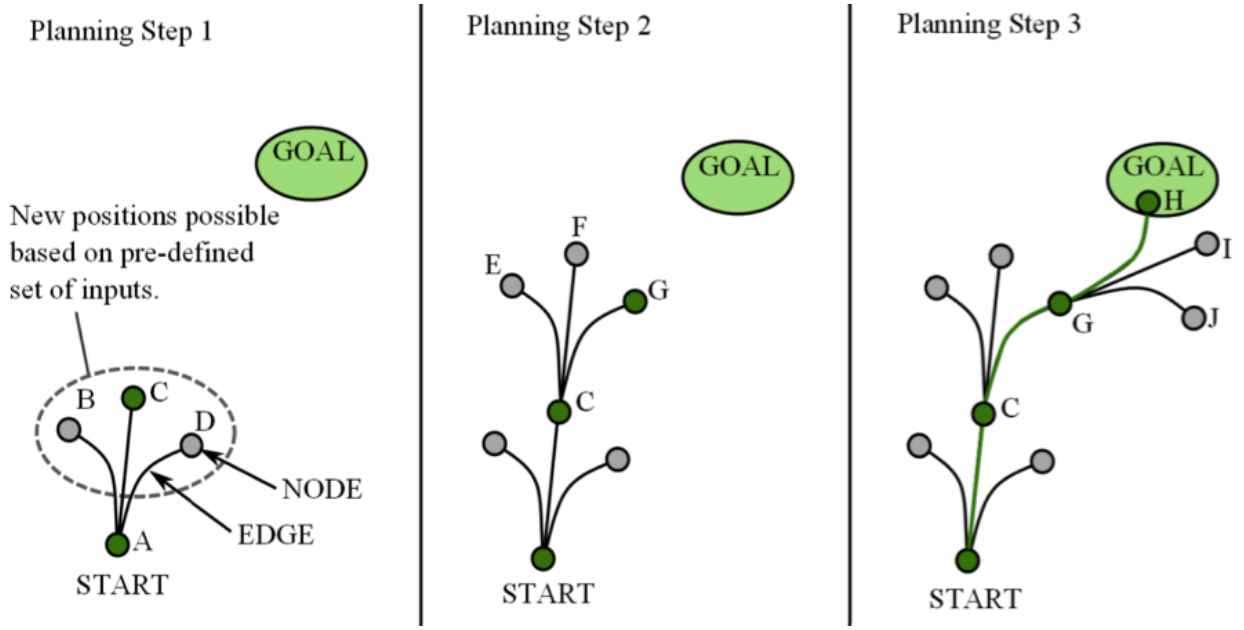


Figure 2.5: Planned path tree from SBMPO to calculate nodes from start to goal. [10]

The energy used to move from one cell to another is found to be a minimum when the vehicle takes the most linear path from the starting cell to the goal location, but obstacle cells are considered to have infinite energy costs associated with them. Hence, in an A* path, a vehicle would simply avoid the obstacle cells by traveling through the adjacent ones, but with skid-steer robots this movement is not always the most energy-efficient method of avoiding obstacles because of the previously discussed excess energy used in making turns. The flaw in this energy cost estimation results in the use of an SBMPO scheme, such that making small adjustments in the path distance conserves energy used in achieving a goal location. The work offered in [6] considers the obstacles not as unconquerable pillars, but costly terrains and are not calculated to be infinite but variable based on the power model coefficient. Changing the cost estimation resulted in paths that

avoided areas with higher coefficients of friction, such as grass, while still trying to maintain the direct aerial path from start to end. This research was done under the assumption that all possible coefficients of friction that may be encountered by the robot are known when planning the path, but with natural and man-made surfaces changing frequently from weather and wear, it is unlikely that a predetermined coefficient would accurately measure the power used at any given time. Their work also lacked the pitch of the robot, which can be of great importance when traversing more organic areas, because pitch mapping has not been prioritized in the research community. This absence of attitude maps is due to the difficulty in establishing them, since most deviations are too small for an aerial vehicle to measure and changes in terrain, such as grass height, are so dynamic [6]. The research conducted by the author is a means of meeting the conditions of variable pitch and coefficients of friction.

Highly dynamic environments are those where either obstacles or the land being traversed changes in time. Even with highly accurate maps describing the elevation gradient and coefficient of friction, the naturally-occurring changes in an outdoors environment happen too rapidly for any surveillance system to keep up with. Also, in time-sensitive situations, such as disaster zones and matters of national security, there may not be the luxury of having current version of a map. In these cases, the vehicle must have a plan in place that allows it to travel unknown terrains without collisions or getting trapped while meeting the goal of the mission. One such method was developed by Tack, Burke, and Sinha [12] and named the planning strategy, which can be seen in Figure 2.6. This tactic outperformed the two other methods previously used in robotics, collision avoidance, where the robot would simply change paths when it encountered an obstacle, and random wandering, where the vehicle moves strategically aimlessly through an environment. The results of their experiments showed that the planning strategy had the best distribution of

position, which was calculated based on the magnitude of entropy experienced in a simulation. Another caliber of excellence was how long it took each method to travel 70% of the environment, and the planning strategy took almost seventeen times less time to achieve this task than collision avoidance. Random wandering failed to reach 70% of the simulation area before the six hours were over.

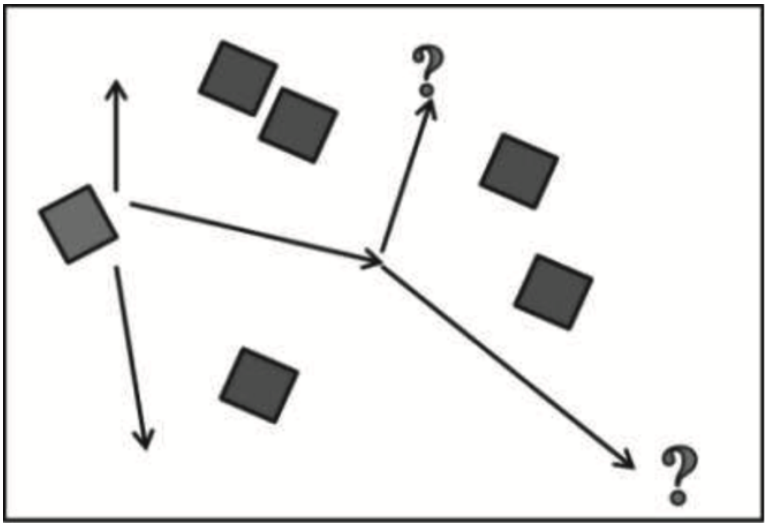


Figure 2.6: Planning strategy expected path in a dynamic environment. [12]

The time to explore the area in its entirety was minimized with the application of the planning strategy, in part due to the fact that it did not spend excess time in hot spots of the simulation. Since there was no map known to the robot or a universal location, the performance of the planning strategy is outstanding. Another contribution to the methods excellence is the algorithms ability to avoid traps and local minima resulting from obstacles. However, this strategy had its issues, the main concern being that there were still cases, like tight corners, which as previously discussed are difficult for skid steer robots to maneuver, that the vehicle could be trapped indefinitely. This same logic can be applied to the work discussed in this thesis, where a slope or coefficient of friction are paired in such a way that it is impossible for a robot to escape or find an alternative path. The

pitfalls and achievements of other researchers in the robotics field discussed in this sections have driven the method of optimization to be discussed in following chapters.

# Chapter 3

# Methodology

This chapter describes how the robot's kinematic behavior is modeled for the rest of the thesis. The majority of the nomenclature and information on skid steer vehicle motion is found in [7, 8, 9, 10]. The chapter sections are organized as follows: (a) notation required to understand the equations used; (b) the coordinate system used to describe the kinematics and direction of the vehicle; (c) the equations that dictate the robot's movements; and (d) the MATLAB code that predicts those movements.

## 3.1 Notation

For the remainder of the thesis, the following notation is employed:

| | | |
|---|---|---|
| $a$ | = | the vector from the geometric center of the UGV to a differential area. |
| $C_{node}$ | = | the cost of a node given in $m$. |
| $child$ | = | the node that is chosen to move to next from the parent node out of all the |

potential children given from the path tree.

$d_{node_1,node_2}$ = the Euclidean distance between two nodes given in $m$.

$D_{node,goal}$ = the Euclidean distance between a given node and the goal location given in $m$.

$\Delta d$ = the set threshold that dictates if the change in location is far enough in distance for a node to qualify as a child node. This value is given in $m$.

$E$ = the required energy for a given trajectory given in $J$.

$E_1$ = the parent East or x value given in $m$.

$E_2$ = the child East or x value given in $m$.

$g$ = the acceleration due to gravity in $\frac{m}{s^2}$.

$G$ = the resistance coefficient given in $N$.

$J_G$ = the constant required to scale the power effects from the power gain model input given in $\frac{m}{s}$.

$J_\mu$ = the constant required to scale the power effects from frictional input given in $W$

$J_\theta$ = the constant required to scale the power effects from the pitch input given in $W$.

$m$ = the mass of the UGV in $kg$.

$N_1$ = the parent North or y value given in $m$.

$N_2$ = the child North or y value given in $m$.

$nodes$ = a point in space where the UGV exists or could potentially exist. Node coordinates include [N, E, U, H].

| | | |
|---|---|---|
| $parent$ | = | the starting node of a potential path tree. |
| $P_x$ | = | the power loss due to x reasoning is given in $W$. |
| $\Delta t$ | = | the time step in $s$. |
| $U_1$ | = | the parent Up or z value given in $m$. |
| $U_2$ | = | the child Up or z value given in $m$. |
| $UGV$ | = | an Unmanned Ground Vehicle. |
| $V_l$ | = | the velocity of the UGV's left track in $\frac{m}{s}$. |
| $V_r$ | = | the velocity of the UGV's right track in $\frac{m}{s}$. |
| $v_x$ | = | the vehicle velocity along the X axis given in $\frac{m}{s}$. |
| $V_x$ | = | the forward velocity of the UGV in $\frac{m}{s}$. |
| $V_x^l$ | = | the velocity of the left track relative to the body of the robot. |
| $V_x^r$ | = | the velocity of the right track relative to the body of the robot. |
| $v_y$ | = | the lateral vehicle velocity along the Y axis given in $\frac{m}{s}$. |
| $V_y$ | = | the sideways velocity of the UGV in $\frac{m}{s}$. |
| $x_{ICRv}$ | = | the X coordinate of the ICR location between the robot chassis and the ground given in $m$. |
| $y_{ICRl}$ | = | the Y coordinate of the ICR location between the left track and the ground given in $m$. |
| $y_{ICRr}$ | = | the Y coordinate of the ICR location between the right track and the ground given in $m$. |
| $y_{ICRv}$ | = | the Y coordinate of the ICR location between the robot chassis and the ground given in $m$. |

| | | |
|---|---|---|
| $\delta_d$ | = | the Euclidean distance between the child and parent node in $m$. |
| $\mu$ | = | the coefficient of friction: the dimensionless value of the force of friction between a UGV and the surface of the terrain it traverses. For the purposes of this thesis, the surface types investigated include asphalt and grass, which are low and high friction respectively. |
| $\psi_1$ | = | the parent Heading value given in degrees. In some locations it may be written as $H_1$. |
| $\psi_2$ | = | the child Heading value given in degrees. In some locations it may be written as $H_2$. |
| $\psi_d$ | = | the heading metric is calculated by the difference in headings between two nodes, given in degrees. |
| $\Delta\psi$ | = | the set threshold that dictates if the change in heading is enough for a node to qualify as a child node. This value is given in degrees. |
| $\theta$ | = | the pitch angle of the terrain between two nodes is given in degrees. |
| $\omega_z$ | = | the angular velocity of the UGV given in $\frac{rad}{s}$. |

## 3.2 Coordinate Definitions

This section presents the coordinate system used to define the kinematics and direction of the UGV's motion. A body-fixed reference frame is used to describe the X and Y axes of the UGV's chassis. The global coordinate system is given as North, East, and Up, called NEU coordinates. Both systems can be seen in Figure 3.1, where the difference between true North and the X-axis

describes $\psi$, the heading angle, of the vehicle.

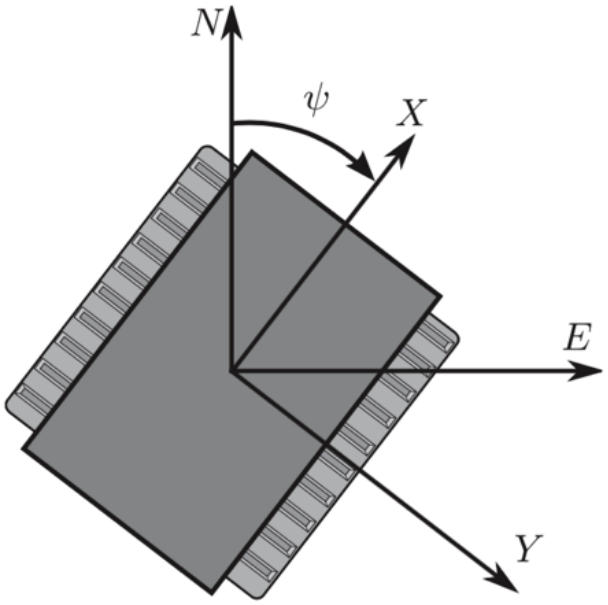


Figure 3.1: Heading angle in relation to the vehicular coordinate system and the rigid north by east coordinate system. [8]

The velocities associated with each body-fixed position are described in Figure 3.2 with the vectors $V_x^l$, $V_x^r$, and $\omega_z$. The figure also includes qualitative locations for the ICR locations. The left and right $x_{ICR}$ values are equal and opposite from the X-axis, and the $y_{ICR}$'s are dependent upon the center of gravity. Therefore, the $y_{ICR}$ runs parallel to the Y-axis, but must not be collinear.

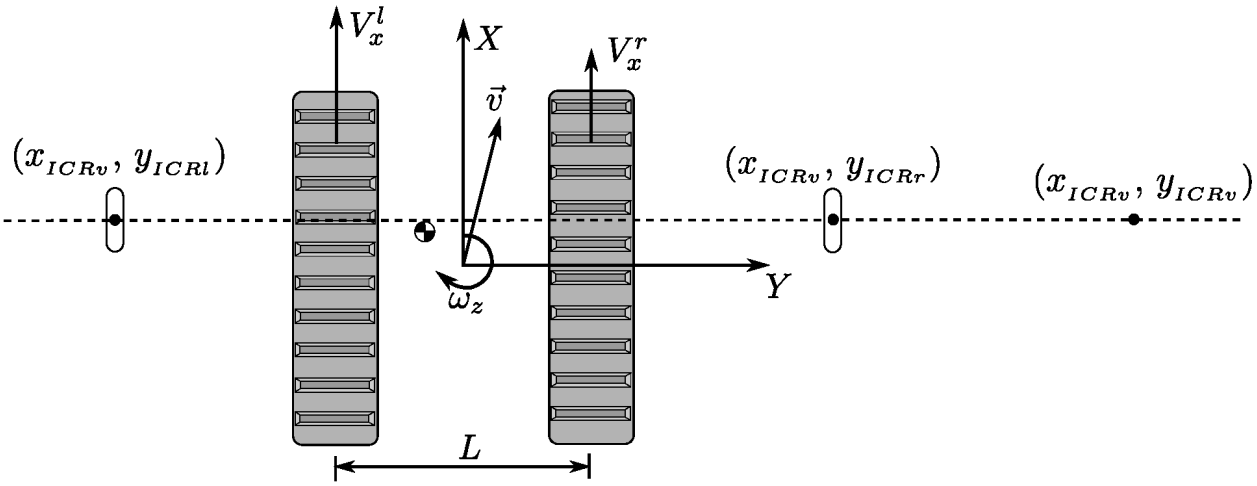


Figure 3.2: Diagram of ICR locations on a skid-steer robot. [7]

For the path planner used in this thesis, the geometrical parameters of the vehicle are modeled after the Tankbot bogie found in Figure 3.3; however, these values could be easily altered within the program in accordance to the geometry of the vehicle being used. The exact measurements are dimensionalized in Figure 3.4. The width and length of the vehicle are .52$m$ and .496$m$ respectively. The ICR location of the left track is -0.5$m$ from the X axis, and the right track's ICR is 0.5$m$. The X coordinate of the ICR locations is closer to the nose of the vehicle at 0.1$m$.

Figure 3.3: Tankbot bogie used to gather experimental data. [7]

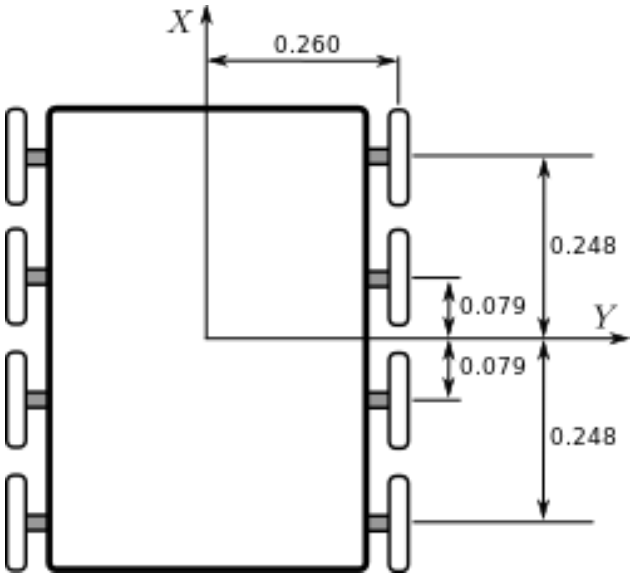


Figure 3.4: Dimensions of Tankbot bogie wheels relative to frame geometric center. Dimensions in meters and Tankbot is symmetrical about the X and Y axes. [7]

In addition to the NEU and body-fixed coordinate systems previously described, an additional coordinate system is used later in the thesis for image processing steps. This coordinate system is generally described using a a row-column format, with the coordinate (1,1) located in the top left of the image, and the x-coordinate indicating rows and y-coordinate indicating columns.

## 3.3 Equations of Robot Kinematics

This section elaborates on the equations used to describe and predict a skid-steer vehicle's movements. See Section 3.1 and 3.2 for clarifications on terminology. Many of the ICR skid-steer kinematics equations presented in this section have been published by Martinez [4].

Beginning with the forward velocity of the vehicle being equivalent to the product of the angular velocity and the magnitude of the vector from the center of gravity to $y_{ICRv}$. This equation is derived from the principle that all points on a vehicle are in rotation about an ICR [13]. In the

circumstances encountered in this thesis, the velocity is known at all times, so (3.1) can provide the $y_{ICRv}$, which is unknown.

$$v_x = y_{ICRv} \times \omega_z \tag{3.1}$$

$$y_{ICRv} = \frac{v_x}{\omega_z} \tag{3.2}$$

For a skid-steer vehicle though, the left and right track velocities are rarely equal and the variation in speed must be accounted for. (3.3) and (3.4) calculate the left and right ICR Y-coordinates by using the velocity of the robot chassis with respect to the track velocity.

$$y_{ICRl} = \frac{v_x - V_x^l}{\omega_z} \tag{3.3}$$

$$y_{ICRr} = \frac{v_x - V_x^r}{\omega_z} \tag{3.4}$$

Similar to (3.1), the $x_{ICRv}$ is found in (3.5) with the lateral velocity relative to the ground and angular velocity with a $-1$ constant. This constant follows through in (3.6) and (3.7) which are the X-coordinate parallel equations to (3.3) and (3.4).

$$x_{ICRv} = -\frac{v_y}{\omega_z}. \tag{3.5}$$

$$x_{ICRl} = -\frac{v_y - V_y^l}{\omega_z}. \tag{3.6}$$

$$x_{ICRr} = -\frac{v_y - V_y^r}{\omega_z}. \tag{3.7}$$

However, the tracks on skid-steer vehicles move parallel to the X-axis, therefore it can be assumed

that there is no lateral velocity. Thus, $V_x^l = V_x^r = 0\frac{m}{s}$ and:

$$x_{ICRr} = x_{ICRl} = x_{ICRv} = -\frac{v_y}{\omega_z}. \tag{3.8}$$

resulting in the all ICR locations lying on a line that is parallel to the Y-axis, as seen in 3.2.

Equations (3.3)-(3.5) are manipulated to become (3.9)-(3.11) in order to express the outputs: longitudinal, lateral, and angular velocities of the robot chassis as a function of the track velocities and ICR coordinates.

$$v_x = \frac{V_x^r y_{ICRl} - V_x^l y_{ICRr}}{y_{ICRl} - y_{ICRr}} \tag{3.9}$$

$$v_y = \frac{(V_x^l - V_x^r) x_{ICRv}}{y_{ICRl} - y_{ICRr}} \tag{3.10}$$

$$w_z = -\frac{V_x^l - V_x^r}{y_{ICRl} - y_{ICRr}} \tag{3.11}$$

Since the $y_{ICRr}$ is left of the X-axis for a stable UGV, absolute value bars must be used to calculate the correct direction and sign of the velocities, resulting in (3.12)-(3.14).

$$v_x = \frac{V_x^r y_{ICRl} - V_x^l y_{ICRr}}{-\left|y_{ICRl} - y_{ICRr}\right|} \tag{3.12}$$

$$v_y = \frac{(V_x^l - V_x^r) x_{ICRv}}{-\left|y_{ICRl} - y_{ICRr}\right|} \tag{3.13}$$

$$w_z = -\frac{V_x^l - V_x^r}{-\left|y_{ICRl} - y_{ICRr}\right|} \tag{3.14}$$

Due to the symmetry about the X-axis, $y_{ICRl} = -y_{ICRr}$. When substituted into (3.12)-(3.14)

$$v_x = \frac{V_x^r y_{ICRl} - V_x^l(-y_{ICRl})}{-|y_{ICRl} - (-y_{ICRl})|} \tag{3.15}$$

$$v_y = \frac{(V_x^l - V_x^r)x_{ICRv}}{-|y_{ICRl} - (-y_{ICRl})|} \tag{3.16}$$

$$w_z = -\frac{V_x^l - V_x^r}{-|y_{ICRl} - (-y_{ICRl})|} \tag{3.17}$$

and simplified, yields

$$v_x = \frac{V_x^r + V_x^l}{2} \tag{3.18}$$

$$v_y = \frac{(V_x^l - V_x^r)x_{ICRv}}{2y_{ICRl}} \tag{3.19}$$

$$w_z = -\frac{V_x^l - V_x^r}{2y_{ICRl}}. \tag{3.20}$$

because, by keeping the velocities in terms of $y_{ICRl}$ exclusively, the absolute values and negative signs are removed.

The velocities of the robot with respect to the global coordinates, North and East, are a function of $v_x$, $v_y$, and $\psi$. For a visual explanation of this transformation, see Figure 3.1.

$$\dot{N} = v_x \cos\psi - v_y \sin\psi \tag{3.21}$$

$$\dot{E} = v_x \sin\psi + v_y \cos\psi \tag{3.22}$$

The equations of motion to describe the UGV are

$$\begin{bmatrix} \dot{N} \\ \dot{E} \\ \omega_z \\ \dot{y}_{ICRr} \\ \dot{y}_{ICRl} \\ \dot{x}_{ICRv} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi + w_N \\ v_x \sin\psi + v_y \cos\psi + w_E \\ -\frac{V_x^l - V_x^r}{y_{ICRl} - y_{ICRr}} + w_\omega \\ w_r \\ w_l \\ w_x \end{bmatrix} \tag{3.23}$$

such that $w_N - w_x$ are additive zero-mean Gaussian process noises. In order to predict the location of the vehicle, the Euler method is applied and the kinematic model is discretized over a time step of size $\Delta t$.

$$\begin{bmatrix} N_k \\ E_k \\ \psi_k \\ y_{ICRr_k} \\ y_{ICRl_k} \\ x_{ICRv_k} \end{bmatrix} = \begin{bmatrix} N_{k-1} + \Delta t V_{N_{k-1}} + \Delta t w_N \\ E_{k-1} + \Delta t V_{E_{k-1}} + \Delta t w_E \\ \psi_{k-1} + \Delta t \omega_{z_{k-1}} + \Delta t w_\omega \\ y_{ICRr_{k-1}} + \Delta t w_r \\ y_{ICRl_{k-1}} + \Delta t w_l \\ x_{ICRv_{k-1}} + \Delta t w_x \end{bmatrix} \tag{3.24}$$

All the equations provided in this section are used to simulate the movement of a skid-steer UGV traversing variable terrain.

## 3.4 Code to Predict Kinematics

The MATLAB code presented in this section is based on the equations derived in Section 3.3 in order predict the kinematics of a skid-steer vehicle undergoing a path that can be easily adapted for a real life scenario.

The first block of code offered below consists of all the essential, measurable constants of a vehicle, which are necessary for all further calculations. The properties listed in lines 4 - 51 are indicative of the Tankbot bogie's geometry and physical characteristics. Whereas lines 54 - 59 and 72 - 78 are set by the user to dictate the grid size for the A-star style grid, maximum velocity for each track, and the parameters for each $tmax$ seconds.,

```
1  %% Properties of vehicle. Approximately equal to Tankbot values.
2
3  % ICR Locations (m)
4  y_r = 0.5; % right track ICR
5  y_l = -0.5; % left track ICR
6  x_v = 0.1; % ICR on the y-axis
7
8  % Vehicle Mass
9  Mass = 80.2858495; % kg
10 Weight = Mass*9.81;  % Weight of the vehicle (N)
11
12 % Width between tracks (m)
13 W = 0.5207;
14
15 % Pressure at each wheel (N)
16 P = Weight/8.0;
17
18 % Distance from geometric center to track center along Y axis
19 Ty = 0.26033;
20 % Vectors from vehicle center to left bogie wheel locations
21 Vl1 = [0.2476, -Ty];
22 Vl2 = [0.0755, -Ty];
23 Vl3 = [-0.0826, -Ty];
24 Vl4 = [-0.2477, -Ty];
25
26 % Vectors from vehicle center to right bogie wheel locations
27 Vr1 = [0.2476, Ty];
28 Vr2 = [0.0755, Ty];
29 Vr3 = [-0.0826, Ty];
```

```
Vr4 = [-0.2477, Ty];

% Pre-Compute sums for power model.
A_l1 = Vl1 - [x_v, y_l];
A_l2 = Vl2 - [x_v, y_l];
A_l3 = Vl3 - [x_v, y_l];
A_l4 = Vl4 - [x_v, y_l];

A_r1 = Vr1 - [x_v, y_r];
A_r2 = Vr2 - [x_v, y_r];
A_r3 = Vr3 - [x_v, y_r];
A_r4 = Vr4 - [x_v, y_r];
Sum1 = P*norm(A_l1);
Sum2 = P*norm(A_l2);
Sum3 = P*norm(A_l3);
Sum4 = P*norm(A_l4);

Sum5 = P*norm(A_r1);
Sum6 = P*norm(A_r2);
Sum7 = P*norm(A_r3);
Sum8 = P*norm(A_r4);
Sum = Sum1+Sum2+Sum3+Sum4+Sum5+Sum6+Sum7+Sum8;

%% Set parameters for each node for the psuedo-grid implementation, ...
    define the minimum distance3D and heading change allowed between two ...
    nodes.
Grid_D = 0.5; %(m)
Grid_H = 20*pi/180; %(rad)

% Maximum track speeds (m/s)
Max_Vl = 1.0; % left track max speed
Max_Vr = 1.0; % right track max speed

% Determine a set of movements for the vehicle. These will be ...
    transformed to expand array points during the path planning stage

%Get a set of inputs sampled from the control space.
V_l = func_halton_set(2,40)*Max_Vl;
V_r = func_halton_set(3,40)*Max_Vr;

%Add control commands allowing the vehicle to drive straight
V_l = [0.5; V_l;];
V_r = [0.5; V_r;];

%Simulate each command for tmax seconds
Dist = 0;
Psi = 0;
Movement = zeros(length(V_l),6);
q = [0; 0; 0; y_r; y_l; x_v];
t = 0;
dt = 0.01;
tmax = 8;
```

The function `func_halton_set` is called upon in order to calculate the initial velocities of each track, which are then scaled by the maximum velocity set in lines 58 and 59. A Halton sequence is a purposefully random set of points, which creates a first set of possible child nodes. The MATLAB code below follows the basic algorithm of a Halton sequence [3].

```
1  function [S] = func_halton_set(Base, Num_Points)
2  %This function computes a Halton sequence with a given base and length
3  %
4  %Inputs:    Base: The base number for the Halton set
5  %           Num_Points: The length of the returned sequence
6  %
7  %Outputs:   S: Computed Halton sequence
8
9  S = zeros(Num_Points,1);
10
11 for k = 1:Num_Points
12
13     pp = Base;
14     kp = k;
15     phi = 0;
16
17     while kp > 0
18         a = mod(kp,Base);
19         phi = phi + a/pp;
20         kp = floor(kp/Base);
21         pp = pp*Base;
22     end
23
24     S(k) = phi;
25 end
```

The next large section of code that describes the kinematics of the UGV is given below. From this block, the user is able to plot the movement trajectories used to expand each row if `Plot_Movements` is set to true. Lines 6 and 7 are pulled directly from equations (3.11) and (3.9) to calculate $\omega_z$ and $v_x$ respectively.

```
1  %% Simulate the movement of the vehicle using ICR kinematics. In this ...
       simulation assume that the ICRs remain constant so this calculation ...
       is only done once at the beginning.
2  for count = 1:length(V_l)
```

```
3
4       % ICR Kinematics to determine speed and angular rate
5       U = [V_r(count), V_l(count)];
6       omega = -(V_l(count) - V_r(count))/(y_l-y_r);
7       Vx = (V_l(count)*y_l - V_r(count)*y_r)/(y_l-y_r);
8
9       % Simulate movement until the max time is reached
10      while(1)
11          [qk] = func_Kinematics(q,U,dt);
12          Dist = sqrt(qk(1)^2 + qk(2)^2);
13          Psi = qk(3);
14          t = t + dt;
15          if t >= tmax
16
17              % The max time has elapsed. Store the state AND power model ...
                    terms.
18              Movement(count,:) = [qk(1:3)', abs(omega)*Sum, ...
                    (abs(U(1))+abs(U(2))), t];
19              break;
20          end
21
22          q = qk;
23
24          % Plot the movement, if desired.
25          if Plot_Movements
26              plot3(q(2),q(1),q(3)*180/pi,'b.');
27          end
28
29      end
30
31      q = [0; 0; 0; y_r; y_l; x_v];
32      t = 0;
33  end
34
35  % Add labels to the movement plot.
36  if Plot_Movements
37      xlabel('East');
38      ylabel('North');
39      zlabel('Heading');
40      grid on;
41  end
```

`func_Kinematics` is required to calculate $qk$, which describes the position of the UGV for each time step. After the $\Delta t$ is reached, the state vector becomes a potential parent node for the next child $qk$. The function currently has the forward velocity and $y_{ICR}$'s as non time varying, but is available for variation. Lines 24 - 26 create a new position for the robot to travel to and is stored in the array, `Movement(count,:)`, which is later parsed through, and the energy optimal position

sequence is selected.

```
function [qk] = func_Kinematics(q,U,dt)
%This function takes in the current state estimate and control input and
%integrates the ICR kinematics for one time step
%
%Inputs:    q: State vector
%           U: Control vector
%
%Outputs:   qk: State vector after time step

%Extract values from state vector so kinematic equations are more readable
N = q(1);
E = q(2);
theta = q(3);
yr = q(4);
yl = q(5);
xv = q(6);

%Extract values from control vector so kinematic equations are more
%readable
Vr = U(1);
Vl = U(2);

%Integrate the kinematic equations using first order Euler integration
qk(1,1) = N + dt*(cos(theta)*(Vr*yl - Vl*yr)/(yl-yr) - sin(theta)*(Vl ...
    -Vr)*xv/(yl - yr));
qk(2,1) = E + dt*(sin(theta)*(Vr*yl - Vl*yr)/(yl-yr) + cos(theta)*(Vl ...
    -Vr)*xv/(yl - yr));
qk(3,1) = theta + dt*((Vr - Vl)/(yl - yr));
qk(4,1) = q(4);
qk(5,1) = q(5);
qk(6,1) = q(6);
```

# Chapter 4

# Friction-and Kinematics-Based Path Planning

Previous work into developing an energy aware path planner was developed by Pentzer [7, 8, 9, 10], followed by the algorithms used to model the equations previously discussed. The code provided in this chapter was the starting point for this thesis and is the basis for what is discussed in Chapter 5.

## 4.1 Equations for Power Estimation

This section discusses the equations used by Pentzer et al. [9] to describe and predict the power consumption of a UGV traveling on mixed terrain. The three main components that account for energy use are internal resistances, friction of the surface, and the changes in elevation. The amount of friction within the drive train is linearly dependent upon the velocities of the tracks and

a resistance coefficient that was found experimentally [9].

$$P_I = G\left(|V_r| + |V_l|\right) \tag{4.1}$$

$G$ was found to be dependent on the type of surface on which the vehicle travels. The next power loss component is based on the surface friction of the terrain the UGV is traveling upon: asphalt or grass. The power loss due to surface friction, $P_S$, is modeled in equation 4.2, where $A^{l,r}$ is the area of the left and right tractive surfaces, the velocity is relative between the $A^{l,r}$ and the point of interest, and $\vec{F}$ is the force of friction that is resisting slip at any given point. All relevant vectors can be found in Figure 4.1.

$$P_S^{l,r} = \int_{A^{l,r}} \vec{F}(a) \cdot \vec{v}(a) dA \tag{4.2}$$

The force of friction at a point of interest is given by:

$$\vec{F}_a = -\mu p(a) \frac{\vec{v}_a}{\|\vec{v}_a\|} \tag{4.3}$$

after the application of a Coulomb friction model. The dynamic friction of the surface, $\mu$, pressure at point a, $p(a)$, and the corrective factor that directs the force in the opposite direction of the velocity, $\frac{\vec{v}_a}{\|\vec{v}_a\|}$, all correlate with the force of friction. Similar to how $G$ was experimentally found, the dynamic friction of asphalt and grass were found. The average values for $G$ and $\mu$ are

Table 4.1: Mean values for the resistance coefficient and coefficient of dynamic friction for asphalt and grass

| Terrain Type | $G$ | $\mu$ |
|---|---|---|
| Asphalt | 94.26 | 0.59 |
| Grass | 134.17 | 1.92 |

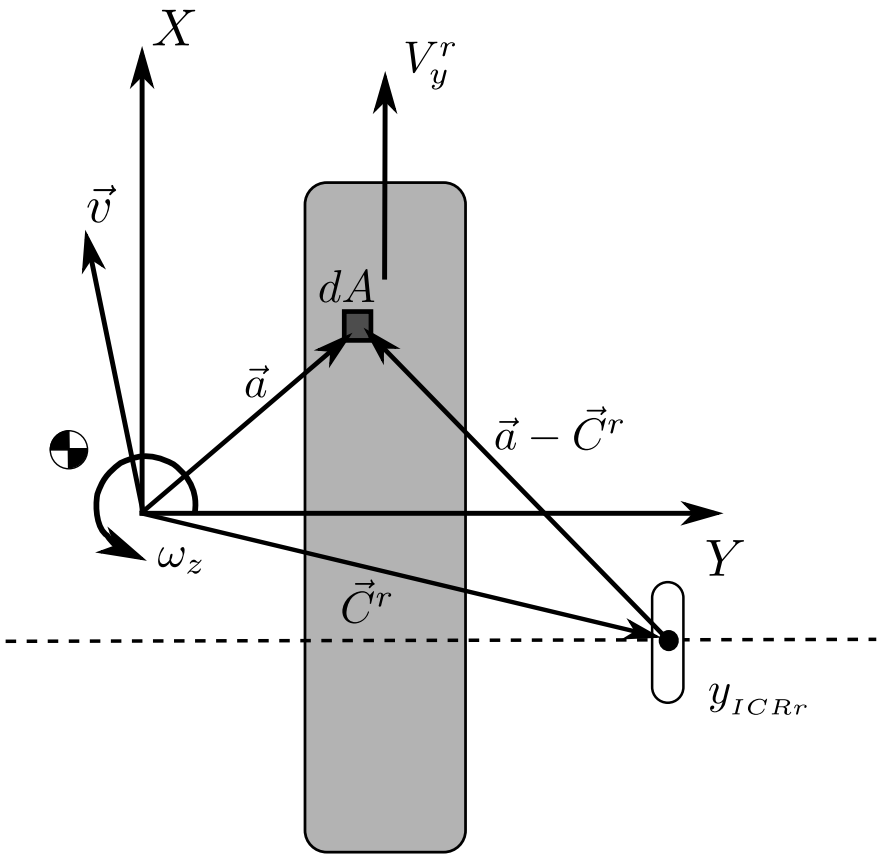


Figure 4.1: Diagram of right side track on skid-steer vehicle showing differential element of track contact area relative to ICR location. [7]

In order to calculate the relative velocity between a differential area and a specified location in space, $\vec{v}(a)$ is found to be the cross product of the UGV's angular velocity and the distance between the track's ICR location and $dA$. The vector $\vec{a} - \vec{C}_{r,l}$ is with respect to the geometric center of the UGV.

$$\vec{v}(a) = \vec{\omega}_z \times (\vec{a} - \vec{C}_{r,l}) \tag{4.4}$$

When equations 4.3 and 4.4 are substituted into 4.2, equations 4.5 - 4.7 are acquired, such that $\mu$ is assumed to remain constant over the tractive surface at the point of interest.

$$P_S^{l,r} = \int_{A^{l,r}} -\mu p(a) \frac{\vec{v}_a}{\|\vec{v}_a\|} \vec{v}_a dA \tag{4.5}$$

$$= \int_{A^{l,r}} \mu p(a) \|\vec{v}_a\| dA \tag{4.6}$$

$$= \mu |\omega_z| \int_{A^{l,r}} p(a) \|\vec{a} - \vec{C}_{r,l}\| dA \tag{4.7}$$

For the purpose of this thesis, the ground pressure is modeled after the Tankbot bogie. Due to the skids of the Tankbot, there are eight wheels, four on either side, that remain in contact with the ground. Therefore the weight of the vehicle is distributed amongst the eight wheels and

$$p = \frac{mg}{N} \tag{4.8}$$

where $N$ is the number of contact points of the vehicle. This dissemination of weight can be see in Figure 4.2. As a result, the pressure is considered constant along the body, so $P_S$ is constant on either side, assuming both tracks are on the same terrain type and equation 4.7 simplifies to equation 4.9.

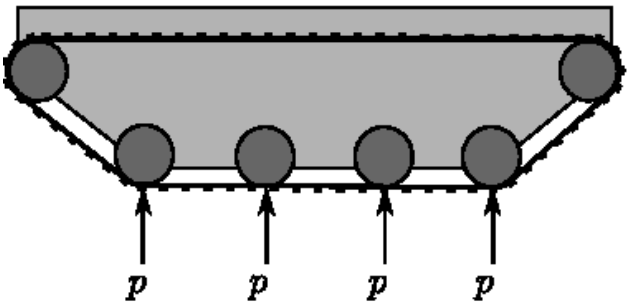


Figure 4.2: Estimated locations of normal forces acting on Tankbot bogie. Modified from [7]

$$P_S = \mu|\omega|p\sum_{n=1}^{N}\|\vec{a} - \vec{C}_{r,l}\| \tag{4.9}$$

The third and final factor in estimating the power loss is the power required to overcome the pitch changes. Equations 4.10 - 4.13 were derived in by Pentzer and are pulled directly from [7] and [10]. They were used to calculate the power and energy consumed in order to traverse a specific path. Chapter 5 elaborates on how these equations were fully accounted for, including the scaling factor $\beta$ that Pentzer used to compensate for the changes in elevation.

$$P_P = \beta m g V_x \sin\theta \tag{4.10}$$

$$P = \mu J_{mu} + G J_G + \beta J_\beta \tag{4.11}$$

$$P = \mu|\omega|p\sum_{n=1}^{N}\|\vec{a} - \vec{C}_{r,l}\| + G\left(|V_r| + |V_l|\right) + \beta m g V_x \sin\theta. \tag{4.12}$$

$$E = \Delta t(\mu J_\mu + G J_G + \beta J_\beta) \tag{4.13}$$

The total power loss, found most directly through 4.12 is the summation of 4.1, 4.9, and 4.10. The total energy required to achieve a path is the time it takes to travel a course multiplied by $P$. However, when Pentzer created code to simulate these traversals, he did not account for the changes in elevation; therefore, the UGV was only penalized for the terrain type it traversed. So,

the equations used in MATLAB are

$$P = \mu J_{mu} + G J_G \tag{4.14}$$

$$P = \mu|\omega|p \sum_{n=1}^{N} \|\vec{a} - \vec{C}_{r,l}\| + G\left(|V_r| + |V_l|\right) \tag{4.15}$$

$$E = \Delta t(\mu J_\mu + G J_G) \tag{4.16}$$

## 4.2 Equations for Path Planning

For the purposes of planning the UGV's path an SMBPO method is used. As a result of specifying the `Planning_Mode` to `Energy`, it is necessary to calculate the energy potentially required to arrive at the child node of each $i^{th}$ branch. The $\Delta_t$ found in equation 4.17 is the time it would take to arrive at each node, which can be estimated as $\frac{\delta_d}{V_x}$.

$$E_i = \Delta_t(\mu J_{mu,i} + G J_{G,i}) \tag{4.17}$$

In a two dimensional approach to the path planner the distance between the child and parent nodes would be calculated as:

$$\delta_d = \sqrt{(E_1 - E_2)^2 + (N_1 - N_2)^2} \tag{4.18}$$

and is seen in Figure 4.3 as the Euclidean distance between the coordinate points, [$E_1$, $N_1$] and [$E_2$, $N_2$].

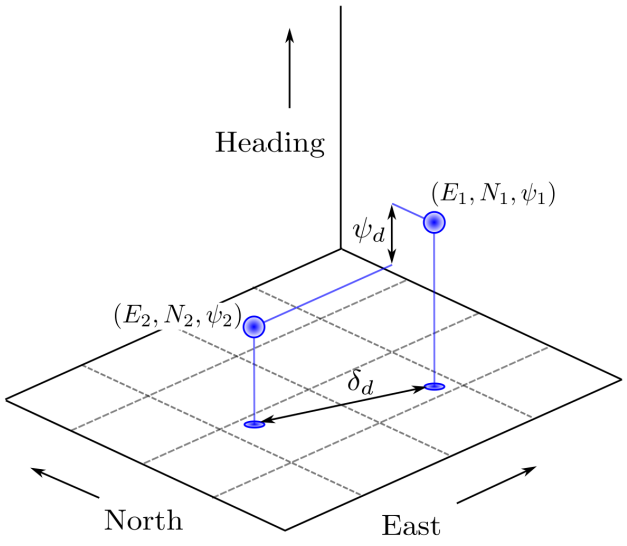


Figure 4.3: A child and parent node with pseudo-grid dimensions. [7]

Another coordinate used for path planning is the heading, $\phi$, such that the change is found in 4.19.

$$\psi_d = |\psi_1 - \psi_2|. \tag{4.19}$$

The power and energy equations derived in the previous section do not impact the path planner if the `Planning_Mode` is set to `Distance`. If this is the case, then the cost of moving from one node to the other would be calculated as:

$$C_C = d_{A,C} + D_{C,GOAL} \tag{4.20}$$

where the theoretical cost is how far the UGV has to move to arrive at the next node and then the distance from the new node to the goal. In application, this results in a linear two dimensional path between the start and target locations.

## 4.3 Code for Power Estimation

Prior to any power calculations or path predictions, image mapping programs must run to filter satellite, colored images into categorical images that describe low energy, high energy, and inaccessible areas. Figure 4.5 is the filtered version of Figure 4.4 such that the low energy, asphalt is light gray, the high energy grass is dark gray, and the infinite obstacles are black.


Figure 4.4: Sample satellite imaging of a location in State College, PA. [7]

Figure 4.5: Power map image after k-means segmentation. The color variation represents the traversability of the area. [7]

The MATLAB file found below, calculates the $\mu$ and $G$ for the UGV at the location [East, North] and the minimum $\mu$ and $G$ for the entire map, based on `POW_Map.Image`, where the East and North values correspond to a pixel in an image with a respective column and row. From that pixel, the algorithm determines the coloring indicating surface type.

```
function [in_flag, mu, G, min_mu, min_G] = func_Power_Gain(East, North)

%This function takes in a position set and outputs the proper power model
%parameters for that position from the power map.
%
%Inputs:    East: East position (m)
%           North: North position (m)
%
%Outputs:   in_flag: True if position lies within power map, false if not
%           mu: Mu power model parameter for input position
%           G: G power model parameter for input position
%           min_mu: Minimum mu power model parameter within the map.
%           min_G: Minimum G power model parameter within the map.

global POW_Map;

%Determine if the point lies within the positions covered by the power map
if East > POW_Map.ImProp.E_min_max(1) + 2 && East < ...
    POW_Map.ImProp.E_min_max(2) - 2 && North > ...
    POW_Map.ImProp.N_min_max(1) + 2 && North < ...
    POW_Map.ImProp.N_min_max(2) - 2

    %Calculate the row/col in the image equivalent to the N/E position
    [row, col] = func_NE_to_image(North, East, POW_Map.ImProp);

    %Get the pixel color for the current position
    Pixel = POW_Map.Image(row, col);

    %Find which row in the colors array corresponds to this pixel color
    Index = find(POW_Map.Colors(:,1) == Pixel);

    %Extract the power model parameters from the colors array
    mu = POW_Map.Colors(Index,2);
    G = POW_Map.Colors(Index,3);

    %Determine the minimum mu and G values within the power map
    min_mu = min(POW_Map.Colors(:,2));
    min_G = min(POW_Map.Colors(:,3));

    %Set the bounds flag
    in_flag = 1;

```

```
40 else %Position lies outside of area covered by map. Set variables ...
       accordingly
41      in_flag = 0;
42      mu = 0;
43      G = 0;
44      min_mu = 0;
45      min_G = 0;
46 end
```

The final component of this code allows the planner to attempt locations beyond the size of `POW_Map.Image`, by simply setting the coefficients of friction and resistance to zero. The issue with this assumption is that it makes locations outside of the known map zero energy nodes, which would make them desirable paths. Instead, this program was altered such that locations outside of the boundaries are energy obstacles and thereby undesirable as children nodes.

## 4.4 Code for Path Planning

The following file creates an `exp_array` when the `expand_array` function is called. The output array is in the format of [`s_N`, `s_E`, `s_H`, `hn`, `gn`, `fn`, `Energy`], such that `s_N`, `s_E`, and `s_H` are the coordinates of a new node state. The value of `hn` becomes the previous cost of traveling from the start to current position added to the cost of traveling from the current node to the new node state.

To ensure that the new node is far enough away or different enough in direction, lines 53 - 67 checks that the distance between the nodes is greater than the prescribed grid size. If the distance is farther than the grid size, the potential node is passed along to find its cost efficiency. However, when a potential child is within the grid, it passes through another set of logic, which verifies that the next position would have the vehicle facing a significantly different heading than before. Lines 57 - 62 constrain possible headings to a range of [-$\pi$, $\pi$]. If the heading is greater than the set

heading differential, the potential child is evaluated. If the node fails to fulfill either parameter it is assigned a flag equal to zero and is no longer considered.

Line 73 calls the `func_Power_Gain` function described in the previous section, in order to calculate the energy it would require for the vehicle to travel to a new node, regardless of the planning mode. The energy consumption is found in lines 84 and 85 and evaluates the lower cost nodes to the expanded child nodes. The result of which is stored in `exp_array(:,7)`, so total energy used in Joules is stored as a variable which allows comparisons between paths.

```
1  function exp_array=expand_array(node_N,node_E,node_h,hn,NTarget, ...
       ETarget,CLOSED,Movement,Grid_D,Grid_H,Planning_Mode)
2      %Function to return an expanded array.
3      %
4      %This function takes a node and returns the expanded list of ...
           successors,with the
5      %calculated fn values. The criteria being none of the successors ...
           are on the CLOSED list.
6      %
7      %Inputs:    node_N: North position (m) of lowest cost node
8      %           node_E: East position (m) of lowest cost node
9      %           node_h: Heading (rad) of lowest cost node
10     %           hn: Cost to reach current node from start position
11     %           NTarget: Target north coordinate (m)
12     %           ETarget: Target east coordinate (m)
13     %           CLOSED: List of point coordinates that have been expanded
14     %           Movement: An array of movement trajectories for the ...
           expansion
15     %           Grid_D: Minimum distance (m) between expanded nodes
16     %           Grid_H: Minimum heading difference (rad) between nodes
17     %           Planning_Mode: String choice between "Energy" and ...
           "Distance"
18     %
19     %Outputs:   exp_array: array of nodes expanded from current lowest ...
           cost position
20
21     %Expand current node position with movement trajectories.
22     %Trajectories are rotated using current node heading.
23     New_Positions = zeros(size(Movement,1),3);
24     for count = 1:size(Movement,1)
25         R = [cos(node_h), -sin(node_h); sin(node_h), cos(node_h)];
26         New_Positions(count,1:2) = (R*Movement(count,1:2)')' + [node_N, ...
               node_E];
27         New_Head = node_h + Movement(count,3);
28
```

```
29          %Constrain heading value to lie between -pi and +pi
30          while New_Head > pi
31              New_Head = New_Head - 2*pi;
32          end
33          while New_Head < -pi
34              New_Head = New_Head + 2*pi;
35          end
36          New_Positions(count,3) = New_Head;
37      end
38
39      %Create expanded array. Each new point is compared with previous
40      %nodes to ensure that it does not violate the pseudo-grid rules.
41      exp_array=[];
42      exp_count=1;
43      c2=size(CLOSED,1);%Number of elements in CLOSED including the zeros
44
45      for count = 1:length(New_Positions)
46          %Get a new node state
47          s_N = New_Positions(count,1);
48          s_E = New_Positions(count,2);
49          s_H = New_Positions(count,3);
50          flag=1;
51
52          %Make sure this expanded node isn't too close to a closed node
53          for c1=1:c2
54              Dist = distance(s_N,s_E,CLOSED(c1,1),CLOSED(c1,2));
55              if( Dist < Grid_D)
56                  Head_Diff = s_H - CLOSED(c1,3);
57                  while Head_Diff > pi
58                      Head_Diff = Head_Diff - 2*pi;
59                  end
60                  while Head_Diff < -pi
61                      Head_Diff = Head_Diff + 2*pi;
62                  end
63                  if abs(Head_Diff) < Grid_H
64                      flag=0;
65                  end
66              end;
67          end;%End of for loop to check if a successor is on closed list.
68
69          %If the node is alright, calculate the costs and add it to the ...
               array
70          if (flag == 1)
71
72              %Calculate the power model coefficients from the power map
73              [in_flag, mu, G, min_mu, min_G] = func_Power_Gain(node_E, ...
                   node_N);
74
75              %If in_flag is true, then the point lies within the power map
76              if in_flag
77
78                  %Add position
79                  exp_array(exp_count,1) = s_N;
80                  exp_array(exp_count,2) = s_E;
```

```
81                  exp_array(exp_count,3) = s_H;
82
83                  %Calculate the energy cost or the distance cost to move ...
                        from the current low-cost node to this expanded child
84                  Energy = Movement(count,4:5)*[mu; G];
85                  Energy = Energy*Movement(count,6);
86                  if strcmp(Planning_Mode, 'Energy')
87                      New_hn = Energy;
88                  elseif strcmp(Planning_Mode, 'Distance')
89                      New_hn = distance(node_N,node_E,s_N,s_E);
90                  end
91                  exp_array(exp_count,4) = hn+New_hn;%cost of travelling ...
                        to node
92
93                  %Calculate the energy cost or the distance cost to move ...
                        from
94                  %the current low-cost node to the target point.
95                  Dist_Goal = distance(NTarget,ETarget,s_N,s_E);
96                  if strcmp(Planning_Mode, 'Energy')
97                      Head_Goal = atan2(ETarget-s_E,NTarget-s_N);
98                      Head_Diff = Head_Goal - s_H;
99                      while Head_Diff > pi
100                         Head_Diff = Head_Diff - 2*pi;
101                     end
102                     while Head_Diff < -pi
103                         Head_Diff = Head_Diff + 2*pi;
104                     end
105                     Time_Turn = Head_Diff/0.1745;
106                     Ener_Turn = 0.1745*min_mu*Time_Turn;
107                     Time_Goal = Dist_Goal/0.5;
108                     Ener_Goal = 1.0*min_G*Time_Goal;
109                     New_gn = Ener_Goal+Ener_Turn;
110                 elseif strcmp(Planning_Mode, 'Distance')
111                     New_gn = Dist_Goal;
112                 end
113
114                 %Finish adding information to expanded array
115                 exp_array(exp_count,5) = New_gn;%distance or energy ...
                        cost between node and goal
116                 exp_array(exp_count,6) = ...
                        exp_array(exp_count,4)+exp_array(exp_count,5);%fn
117                 exp_array(exp_count,7) = Energy;
118                 exp_count=exp_count+1;
119             end
120         end
121     end
```

On line 89, the cost of traveling to the new node is based on a distance optimizer and directly comes from equation 4.20 where `New_hn` $= d_{A,C}$, such that the only cost of a node is dictated by how far it puts the UGV from the target node. Depending on the which optimizer is being utilized,

`exp_array(:,4)` or $hn$, the cost of the node, is either energy or distance dependent. The term cost describes what is lost when traveling to one node over another.

The next major section of the algorithm is looking towards what distance is left between the expanded child node and the target node. The $gn$ or `exp_array(:,5)` depends on the planning mode, but stores either the distance or energy cost of traveling from the current node to the target node. In the case of using a distance optimizer, line 111 calls back to equation 4.20 such that `New_gn` $= D_{C,GOAL}$. For the energy optimizer, the `New_gn` is equal to the energy needed to get from the current node to the target in addition to the energy it would take to turn the UGV towards goal node.

Once `exp_array(:,4)` and `exp_array(:,5)` are found, $fn$ is merely the sum of the two columns and is saved as `exp_array(:,6)`. The value of $fn$ is equivalent to $C_C$ from equation 4.20 for the distance optimizer, but would be equivalent to

$$C_C = E_{A,C} + E_{C,GOAL} \tag{4.21}$$

where $E_{A,C}$ is the energy required for the UGV to travel from the start node to the current node and $E_{C,GOAL}$ is the energy estimated to be used to arrive at the goal.

In another function, the minimum $fn$ value is found from the entire `OPEN` array of possible paths. This acquisition of the min fn node is done through the use of the min function in MATLAB. Once, the optimal node is found, all previous options in `OPEN` go to the `CLOSED` array and are discarded from being considered again. Then, if the most current expansion is still not close enough to the target node, the algorithm above is repeated. All nodes corresponding to the min fn values are then considered a node along the optimal path, which guides the UGV from the start to

the final node.

# Chapter 5

# Friction-, Elevation-, and Kinematics-Based Path Planning

As stated in Section 4.1, the previous models of path planning did not account for changes in elevation when calculating the energy consumed or creating an optimal path. This chapter presents and explains the equations necessary for estimating the energy consumed, the adjustments made to previous models, and what a user might experience as expected outputs when running the algorithm.

## 5.1 Equations for Power Estimation of Elevation Changes

The following equation combines and reiterates the power component specific to pitch from equations 4.10 and 4.11, where $\beta$ was a constant Pentzer used to compensate the amount of power

consumed in order to overcome elevation changes.

$$P_P = \beta J_\beta = \beta m g V_x \sin\theta \tag{5.1}$$

Beginning with the force induced on a stationary UGV due to gravity, the weight force is equivalent to

$$F_P = mg \sin\theta \tag{5.2}$$

if $\theta$ is equal to the angle demonstrated in Figure 5.1.

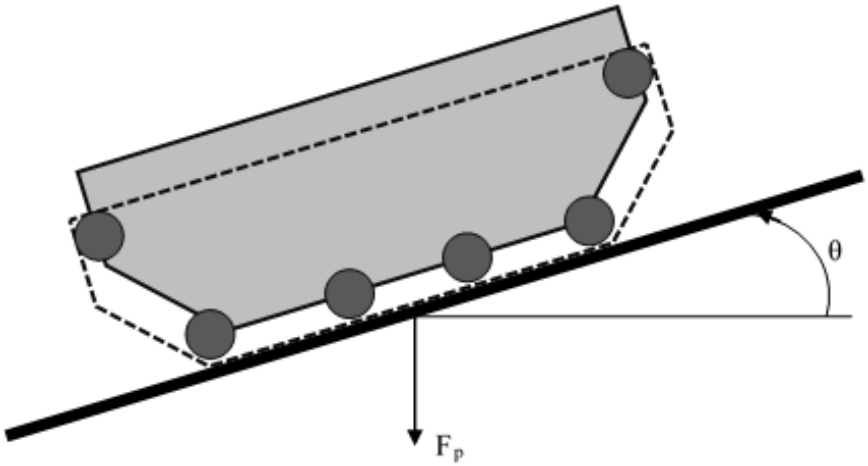


Figure 5.1: Free body diagram of a Tankbot bogie on an incline.

In order to predict the power required to travel from one node to another, the gravity force of the vehicle must be multiplied by the velocity of the UGV with respect to the surface, yielding equation 5.3.

$$P_P = m g V_x \sin\theta \tag{5.3}$$

The above equation predicts power, yet the optimizer is energy-based, so $P_P$ must be multiplied

by a time step. Therefore, the energy between two nodes that would take a delta time step to travel is given by:

$$E_P = P_P \Delta t = mgV_x \sin\theta \Delta t \tag{5.4}$$

where $\Delta t$ is equivalent to:

$$\Delta t = \frac{\delta_d}{V_x} \tag{5.5}$$

such that $\delta_d$ is transformed from equation 4.18 to include changes in the vertical direction to become:

$$\delta_d = \sqrt{(E_1 - E_2)^2 + (N_1 - N_2)^2 + (U_1 - U_2)^2} \tag{5.6}$$

After substituting equation 5.5 into the energy lost to changes in elevation equation, the term $E_P$ simplifies to:

$$E_P = mgV_x \sin\theta \frac{\delta_d}{V_x} = mg \sin\theta \delta_d \tag{5.7}$$

Since mass and gravity are constant, the energy consumed based on elevation changes is solely a function of the slope of the hill and the Euclidean distance traveled from one point to the other.

To simplify the application of these equations for algorithm implementation, the slope is assumed to to be linear between any two nodes. Therefore,

$$\theta = \tan^{-1} \frac{U_2 - U_1}{\sqrt{(E_1 - E_2)^2 + (N_1 - N_2)^2}} \tag{5.8}$$

## 5.2 Optimization of Previous Algorithms

To implement the changes given by the equations above, the code implementations of the path-planning algorithm written by Pentzer were adjusted in multiple locations and across a variety of functions in order to accommodate adding the additional dimension of elevation. This section describes these changes specifically and is organized by the key implementation changes for the path planner and the code segments where the energy use calculations were needed.

### 5.2.1 Code for Planner with Vertical Component

First, to create data representing a vertical change in elevation, a simulated hill is created using peaks function from MATLAB. To implement this, the following section of code is used from the overall path planning algorithm; it creates a virtual hill for the planner to simulate the UGV environment. When this work is applied to real-life scenarios, the hill information would be comprised of the vertical GPS data. For the purpose of this thesis though, all results are from this simulated hill equation.

The hill definition within the algorithm is stored within a mesh data structure that uses an image-like format. Additionally, the image format used for planning has more columns than rows in pixels. There are two steps to creating the hill. Lines 6 - 11 establish the left side of the image format with a dynamic incline that fills all of the rows. The variables `r` and `p` are arrays that range from -1 to 1 in intervals of 0.00562, so that the mesh grid created in line 72 fills the size of all rows of the image and the columns 1 through the number of rows. However, this method leaves some columns left empty, so lines 80 - 88 fill the rest of the columns by slowly decreasing the height by

.01 meter such that the hill extends to the end of the array.

```
%% Adjust the POW map with the hill and altitude changes
% Create a struct to save the power map results for use in other ...
    algorithms.
% Create a sub struct that is a hill created by RPQ for [0:356, 0:356] this
% is an entirely simulated hill, and with measured elevations, this
% component would be removed
r = -1:0.00562:1; % .00562 = (1-(-1))/size(POW_Map.Image(:,1))
p = -1:0.00562:1; % .00562 = (1-(-1))/size(POW_Map.Image(:,1))
[R,P] = meshgrid(r,p);
Q =  3*(1-R).^2.*exp(-(R.^2) - (P+1).^2) ...
    + 10*exp(-R.^2-P.^2)-1.3533528; % creates the desired hill
POW_Map.Altitude = Q;

%Create a sub struct that decreasing the rest of the hill down to zero
%altitude for [357:609, 0:356], because the Q can only create a square
%matrix of data
for k = 357:609
    for i = 1:356
        if POW_Map.Altitude(i,k-1) ≤ .1
            POW_Map.Altitude(i,k) = 0;
        else
            POW_Map.Altitude(i,k) = POW_Map.Altitude(i,k-1) - .01;
        end
    end
end
```

The results of these data structures are saved to a file that contains the altitude for every pixel, `POW_Map.Altitude`, which is the same size as `POW_Map.Image`, a data structure which contains the friction information of the area in question. Visually, the hill that is investigated in this thesis can be seen in Figures 5.2 and 5.3 in three and two dimensional plots.

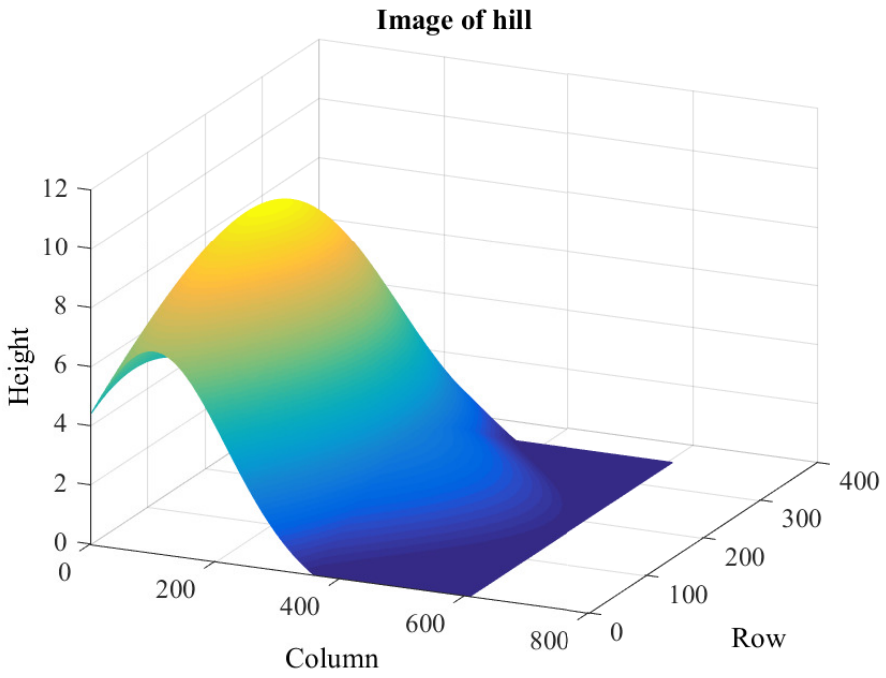


Figure 5.2: Mesh plot of a man made and simulated hill within a image coordinate system of row and column.

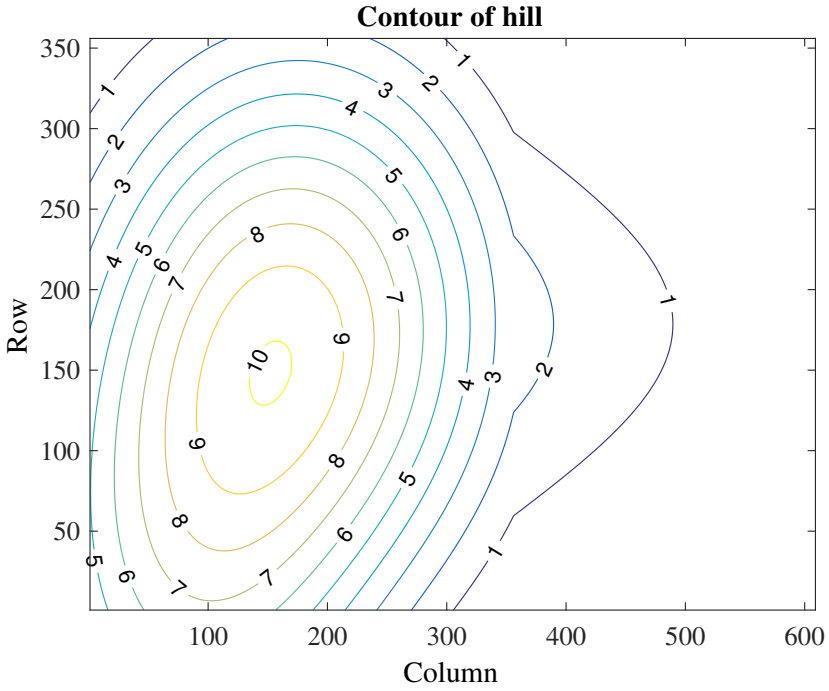


Figure 5.3: Contour plot of a man made and simulated hill within a image coordinate system of row and column.

After creating the hill, the rows are inverted about the center row using the `flipud` or flip up-down function. This new array is then saved as `POW_Map.AltitudeInteractive`, named as such because the only purpose for this structure is for visualization purposes; this inversion of rows and colums allows an image such that, when superimposed with the image data, the hill is in the anticipated location.

```
% Create a sub struct that has a converted pixels to the same ...
    orientation as POW_MAP.Image
for ii = 1:length(POW_Map.Altitude)
    POW_Map.AltitudeInteractive(:,ii) = flipud(POW_Map.Altitude(:,ii));
end

save('POW_Map_Hill.mat','POW_Map');
```

### 5.2.2 Code for Open Array with Vertical Component

The final subsection elaborates on how the expansion process accounts for changes in pitch when path planning and energy calculations. To do this, a variable representing the OPEN list is used throughout the algorithm. This OPEN list is an array of all possible expansion nodes from the current node, i.e. possible path locations that the robot can visit from its current location. This OPEN list is then filtered through to determine viable paths from the current location, and the rejected nodes are sent to the CLOSED list. In order to include the Up values, two additional columns are added as `zval` which is column of potential Up values, and the `parent_zval` represents the parent Up value for each tree. Due to this extension of the OPEN list, many indeces are shifted by one or two columns from the original path planning algorithm developed by Pentzer.

```
function new_row = insert_open_NEU(xval,yval,zval,hval,parent_xval, ...
    parent_yval,parent_zval,parent_hval,hn,gn,fn,energy)
```

```
2  %Function to Populate the OPEN LIST
3  %OPEN LIST FORMAT
4  %-------------------------------------------------------------------------
5  %IS ON LIST 1/0 |N val |E val |U val|Heading |Parent N val |Parent E ...
       val |Parent U val |Parent Heading |h(n) |g(n)|f(n)|
6  %-------------------------------------------------------------------------
7  %
8  %   Copyright 2009-2010 The MathWorks, Inc.
9  %   Adjustments made by Veronica Gruning in March 2018
10 new_row=[1,13];
11 new_row(1,1)=1;
12 new_row(1,2)=xval;
13 new_row(1,3)=yval;
14 new_row(1,4)=zval;
15 new_row(1,5)=hval;
16 new_row(1,6)=parent_xval;
17 new_row(1,7)=parent_yval;
18 new_row(1,8)=parent_zval;
19 new_row(1,9)=parent_hval;
20 new_row(1,10)=hn;
21 new_row(1,11)=gn;
22 new_row(1,12)=fn;
23 new_row(1,13)=energy;
24 end
```

### 5.2.3 Code for Expansion Array with Vertical Component

In searching out new paths, one has to ensure that the robot does not leave the map. To implement this, Lines 27 - 32 and 45 are used to keep the potential expansion nodes within the boundaries of the number of rows and columns that are the dimensions of `POW_Map.Altitude`. If the new node computed in lines 25 and 26 lies beyond the size of `POW_Map.Altitude` in either direction it is categorized as not a number. Line 45 removes any nodes that are non numbers from the OPEN list.

In order to account for the energy lost due to the slope of the terrain, lines 56, 66, 103, 143 merely add the z-component to a vector describing the location of the UGV. Another change made to the previous compilation of programs is that the distance function is now called `distance3D_squared`, because finding the square root of the three dimensional distance between two nodes becomes com-

putationally exhaustive. Hence lines 58 - 59, 73, 110, 157 differ from the code found in Section 4.4, such that the square root to calculate the `Dist_Goal_Vector` is only taken once per instance that `expand_array_NEU` function is called.

```
1  function exp_array = expand_array_NEU(node_N, node_E, node_U, node_h, ...
       hn, NTarget, ETarget, UTarget, CLOSED, Movement, Grid_D_Squared, ...
       Grid_H, Planning_Mode, ImProp, Altitude, w, v)
2  %Function to return an expanded array.
3  %
4  %This function takes a node and returns the expanded list of ...
       successors,with the
5  %calculated fn values. The criteria being none of the successors are on ...
       the CLOSED list.
6  %
7  %Inputs:    node_N: North position (m) of lowest cost node
8  %           node_E: East position (m) of lowest cost node
9  %           node_h: Heading (rad) of lowest cost node
10 %           hn: Cost to reach current node from start position
11 %           NTarget: Target north coordinate (m)
12 %           ETarget: Target east coordinate (m)
13 %           CLOSED: List of point coordinates that have been expanded
14 %           Movement: An array of movement trajectories for the expansion
15 %           Grid_D: Minimum distance (m) between expanded nodes
16 %           Grid_H: Minimum heading difference (rad) between nodes
17 %           Planning_Mode: String choice between "Energy" and "Distance"
18 %
19 %Outputs:   exp_array: array of nodes expanded from current lowest cost ...
       position
20
21 %Expand current node position with movement trajectories.
22 %Trajectories are rotated using current node heading.
23 New_Positions = zeros(size(Movement,1),4);
24 R = [cos(node_h), -sin(node_h); sin(node_h), cos(node_h)];
25 for count = 1:size(Movement,1)
26     New_Positions(count,1:2) = (R*Movement(count,1:2)')' + [node_N, ...
           node_E];
27     [row,col] = func_NE_to_image(New_Positions(count,1), ...
           New_Positions(count,2),ImProp);
28     if isreal((length(Altitude(:,1)) - round(row) + 1)) && ...
           (length(Altitude(:,1)) - round(row) + 1) > 0 && isreal(col) && ...
           col ≤ size(Altitude,2)
29         New_Positions(count,3) = Altitude(length(Altitude(:,1)) - ...
               round(row) + 1,round(col));
30     else
31        New_Positions(count,3) = NaN;
32     end
33     New_Head = node_h + Movement(count,4);
34
35     %Constrain heading value to lie between -pi and +pi
```

```
36      while New_Head > pi
37          New_Head = New_Head - 2*pi;
38      end
39      while New_Head < -pi
40          New_Head = New_Head + 2*pi;
41      end
42      New_Positions(count,4) = New_Head;
43  end
44
45  New_Positions(isnan(New_Positions(:,3)),:) = [];
46
47
48  %Create expanded array. Each new point is compared with previous
49  %nodes to ensure that it does not violate the pseudo-grid rules.
50  exp_array=[];
51  exp_count=1;
52  c2=size(CLOSED,1);%Number of elements in CLOSED including the zeros
53
54  s_N_vector = New_Positions(:,1);
55  s_E_vector = New_Positions(:,2);
56  s_U_vector = New_Positions(:,3);
57  s_H_vector = New_Positions(:,4);
58  Dist_Goal_Squared_Vector = distance3D_squared(NTarget,ETarget,UTarget, ...
        s_N_vector,s_E_vector,s_U_vector);
59  Dist_Goal_Vector = Dist_Goal_Squared_Vector.^0.5;
60  Dist_Goal2D_Vector = distance(NTarget,ETarget,s_N_vector,s_E_vector);
61
62  for count = 1:length(New_Positions)
63      %Get a new node state
64      s_N = New_Positions(count,1);
65      s_E = New_Positions(count,2);
66      s_U = New_Positions(count,3);
67      s_H = New_Positions(count,4);
68      flag=1;
69
70      %Make sure this expanded node isn't too close to a closed node
71      Dist_squared = ...
            distance3D_squared(s_N,s_E,s_U,CLOSED(:,1),CLOSED(:,2),CLOSED(:,3));
72      for c1=1:c2
73          if(Dist_squared(c1,1) < Grid_D_Squared)
74              Head_Diff = s_H - CLOSED(c1,4);
75              while Head_Diff > pi
76                  Head_Diff = Head_Diff - 2*pi;
77              end
78              while Head_Diff < -pi
79                  Head_Diff = Head_Diff + 2*pi;
80              end
81              if abs(Head_Diff) < Grid_H
82                  flag=0;
83              end
84          end
85      end %End of for loop to check if a successor is on closed list.
86
87      %If the node is alright, calculate the costs and add it to the array
```

```
88      if (flag == 1)
89
90          %Calculate the power model coefficients from the power map
91          [in_flag, mu, G, min_mu, min_G, ¬] = func_Power_Gain(node_E, ...
                node_N);
92
93          %See if the expanded node is not in an obstacle
94          [not_obs, ¬, ¬, ¬, ¬, ¬] = func_Power_Gain(s_E, s_N);
95
96          %If in_flag is true, then the point lies within the power map
97          %and is not in an obstacle
98          if in_flag && not_obs
99
100             %Add position
101             exp_array(exp_count,1) = s_N;
102             exp_array(exp_count,2) = s_E;
103             exp_array(exp_count,3) = s_U;
104             exp_array(exp_count,4) = s_H;
105
106             %Calculate the energy cost or the distance cost to move from
107             %The current low-cost node to this expanded child
108             Thetaseg = atan((s_U - node_U))/distance(node_E,...
109                 node_N, s_E, s_N);
110             dist = ...
                    sqrt(distance3D_squared(node_N,node_E,node_U,s_N,s_E,s_U));
111             Ener_Thet = w*dist*sin(Thetaseg);
112             if Ener_Thet ≤ 0
113                 Ener_Theta = 0;
114             else
115                 Ener_Theta = Ener_Thet;
116             end
117             Power = Movement(count,4:5)*[mu; G];
118             Energy = Power*Movement(count,6)+Ener_Theta; ...
                    %Movement(count,6) is the time to complete the motion ...
                    trajectory
119             if strcmp(Planning_Mode, 'Energy')
120                 New_hn = Energy;
121             elseif strcmp(Planning_Mode, 'Distance')
122                 New_hn = distance(node_N,node_E,s_N,s_E); %2-D distance ...
                        planning
123             end
124             exp_array(exp_count,5) = hn+New_hn;%cost of traveling to node
125
126             %Calculate the energy cost or the distance cost to move ...
                    from the current low-cost node to the target point.
127             Dist_Goal = Dist_Goal_Vector(count,1);
128             Dist_Goal2D = Dist_Goal2D_Vector(count,1);
129             if strcmp(Planning_Mode, 'Energy')
130                 Head_Goal = atan2(ETarget-s_E,NTarget-s_N);
131                 Head_Diff = Head_Goal - s_H;
132                 while Head_Diff > pi
133                     Head_Diff = Head_Diff - 2*pi;
134                 end
135                 while Head_Diff < -pi
```

```
136                     Head_Diff = Head_Diff + 2*pi;
137                 end
138                 Time_Turn = Head_Diff/0.1745;  %Estimate of turn rate ...
                       for Tankbot to align with goal heading in rad/s
139                 Ener_Turn = abs(0.1745*min_mu*Time_Turn);
140
141                 NTrack = linspace(s_N, NTarget, floor(Dist_Goal));
142                 ETrack = linspace(s_E, ETarget, floor(Dist_Goal));
143                 UTrack = linspace(s_U, UTarget, floor(Dist_Goal));
144                 SegLength = Dist_Goal/(floor(Dist_Goal));
145                 SegTime = SegLength/0.5; %0.5 is speed of robot driving ...
                       straight to goal
146                 Ener_Goal = 0;
147                 for SegNum = 2:length(NTrack)
148                     [obs_flag, ¬, G, ¬, ¬, maxG] = ...
                           func_Power_Gain(ETrack(SegNum), NTrack(SegNum));
149                     Gseg = G;
150                     Thetaseg = atan((UTrack(SegNum) - UTrack(SegNum - ...
                           1)))/distance(ETrack(SegNum - 1),...
151                         NTrack(SegNum - 1), ETrack(SegNum), ...
                               NTrack(SegNum));
152
153                     if obs_flag == 0
154                         Gseg = maxG;
155                     end
156
157                     dist = sqrt(distance3D_squared(NTrack(SegNum-1), ...
                           ETrack(SegNum-1), UTrack(SegNum-1), ...
                           NTrack(SegNum), ETrack(SegNum), UTrack(SegNum)));
158                     Ener_Thet = w*dist*sin(Thetaseg);
159                     if Ener_Thet < 0
160                         Ener_Theta = 0;
161                     else
162                         Ener_Theta = Ener_Thet;
163                     end
164                     Ener_Seg = 1.0*Gseg*SegTime;  %Equation is ...
                           |Vr|+|Vl|. Track velocities are 0.5, so add to 1.
165                     Ener_Goal = Ener_Goal + Ener_Seg + Ener_Theta;
166                 end
167                 New_gn = Ener_Goal+Ener_Turn;
168             elseif strcmp(Planning_Mode, 'Distance')
169                 New_gn = Dist_Goal2D;
170             end
171
172             %Finish adding information to expanded array
173             exp_array(exp_count,6) = New_gn;%distance between node and goal
174             exp_array(exp_count,7) = ...
                    exp_array(exp_count,5)+exp_array(exp_count,6);%fn
175             exp_array(exp_count,8) = Energy;
176             exp_count=exp_count+1;
177         end
178     end%Populate the exp_array list
179 end%End of if node is not its own successor loop
```

In the above code, the specific alterations made in regard to the energy lost due to slope are in lines 108 - 116, 118, 150, 158 - 163, 165. Lines 108 and 150 calculate the angle between two nodes. Lines 111 and 158 find the energy lost due to the angle calculated in the previous line. The logic found in lines 112 - 116 and 159 - 163 determines if the energy consumed by approaching a slope is negative or positive. If negative, the `Ener_Theta` is set to zero, because it would be inaccurate to calculate that the UGV gains energy by going down a hill, as the robot is incapable of regenerating energy. If `Ener_Thet` is positive, then the values is passed through as the energy it would take to overcome a change in global height.

## 5.3 User Process of Optimized Algorithm

When running the path planner program, the user has the option to hard code the start and end points, or to manually select these locations from an overhead view of the map. This section explains the steps to select the nodes manually and shows examples of the resulting figures. See Appendix B on lines 250 - 320 on how the user interface was programed.

Immediately after running the code, the user is presented with a pop-up window showing an image similar to that of Figure 5.4, which contains a contour plot and variation in color to signify friction. The user then clicks on the point of the map they wish to be the goal, and then clicks another time to assign a location to the start node. After this initial input of information, the rest of the algorithm runs until there is an optimized path solution.

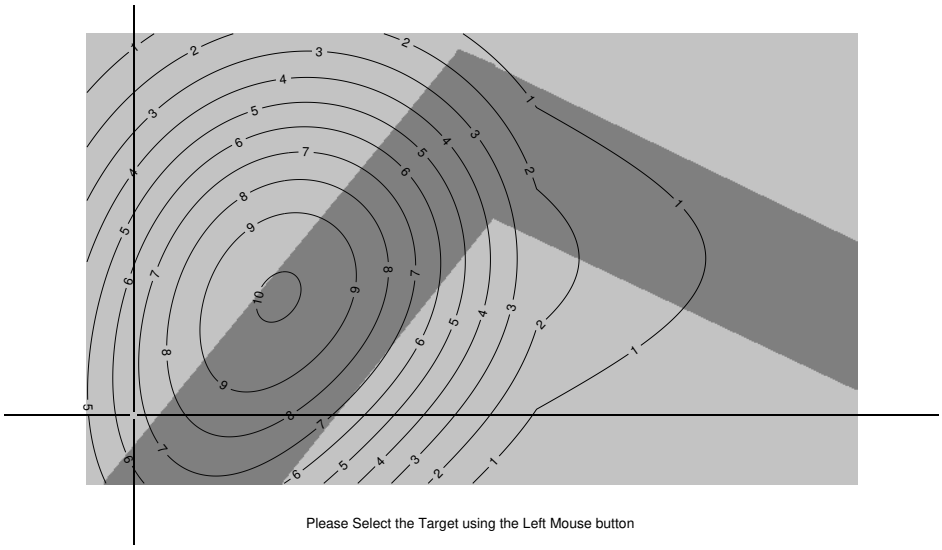


Figure 5.4: Initial user interface for path-planner.

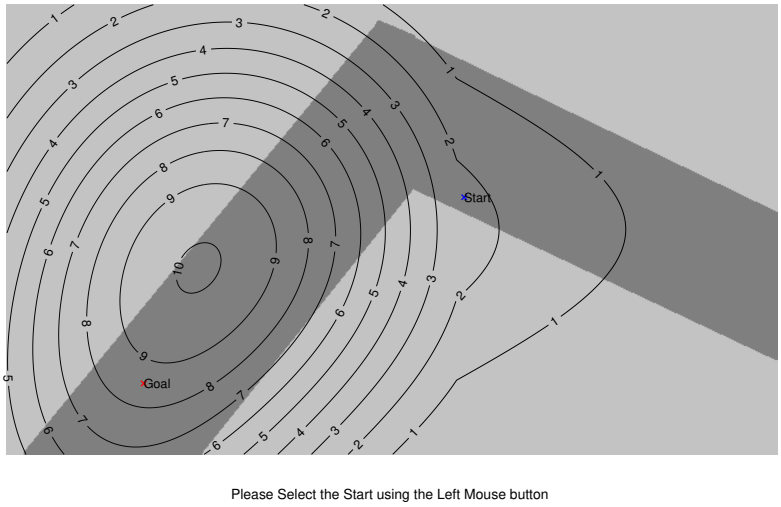


Figure 5.5: User interface of path-planner after selecting the Start and Goal node.

Depending on the planning mode selected, whether it is based on distance or energy, the final results look like Figures 5.7 or 5.8 respectively. In the upper subplot the starting point is given a black circle and the end point is designated with a star. In the lower subplot the start and end points are the red circle and red diamond respectively. The green dots in both plots are the nodes selected for the optimal path, and the surrounding "haze" represents potential paths that came

from the `exp_array` but were not selected as the optimal expansion. By enclarging an area within this haze, found in Figure 5.6, the blue empty circles show the nodes that are now stored in the CLOSED list, and the red lines are the branches that would have connected the path nodes if they had been chosen for the final path solution.

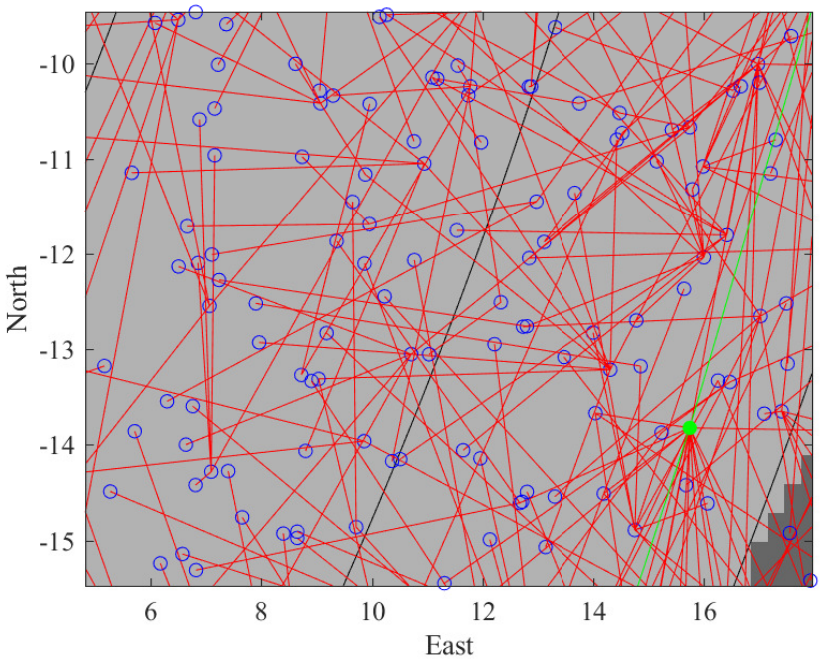


Figure 5.6: Enlarged area of the path-planning options showing the optimizer densitiy of choices.

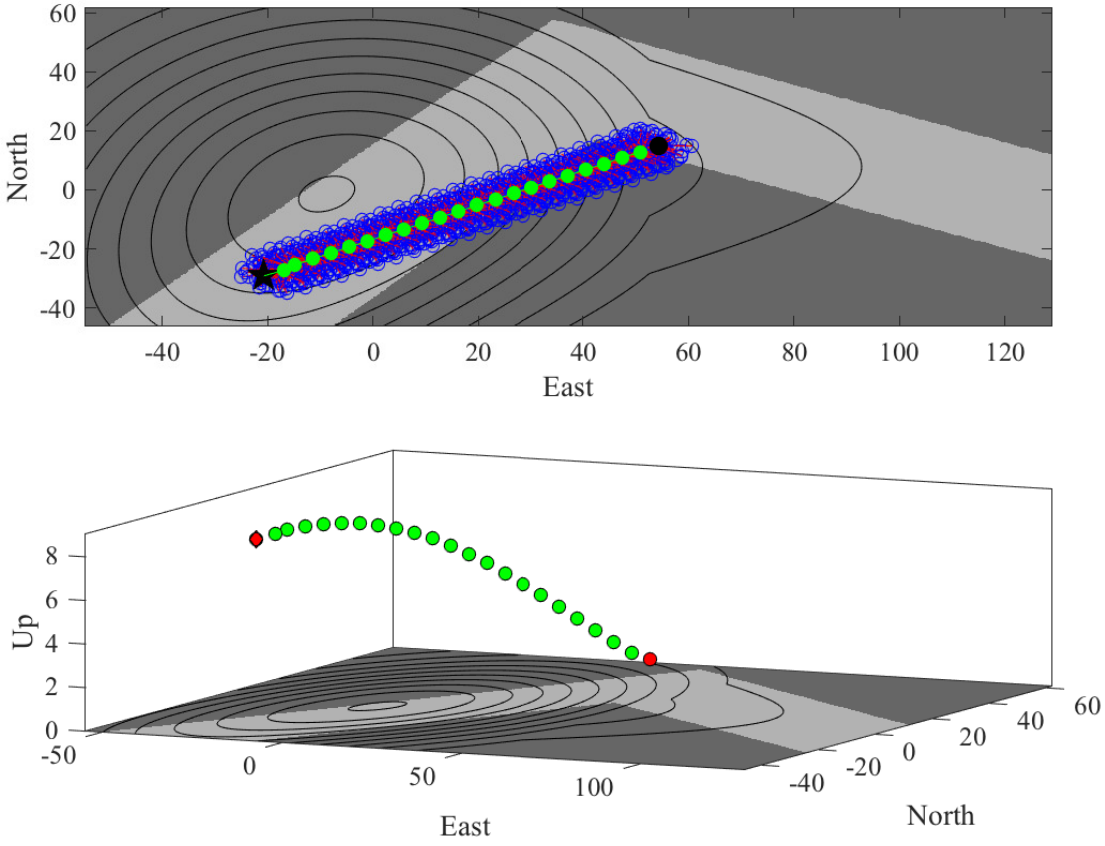


Figure 5.7: Results from using a Distance based optimizer.

The distance-optimized path is based on a birds-eye view approach, so the nodes of the optimal path create a straight line between the start and target as seen in Figure 5.7. Although the distance and time the simulated UGV takes to arrive at the goal point is only NUMm, the energy required is NUMJ. In contrast, the energy-based optimizer can be found in 5.8, where the optimal path not only stays within the low-friction zone but also skirts around the hill in order to maintain the same elevation as the goal. As a result, the energy cost is NUMJ with a distance of NUMm.

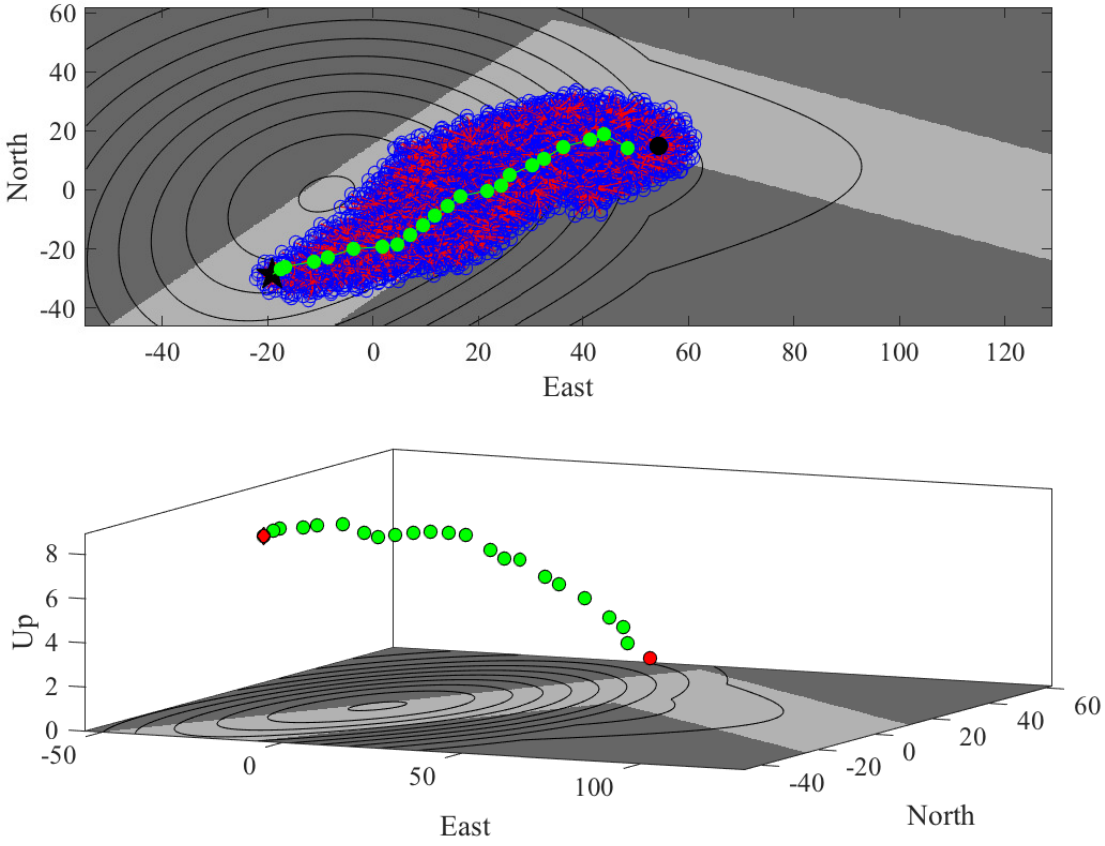


Figure 5.8: Results from using an Energy based optimizer.

# Chapter 6

# Results and Analysis

This chapter reports and explains the results of repeated simulation runs using the algorithm in consideration of vertical terrain geometry. Ten start and end nodes are used throughout this chapter, in order to compare the effects of using an energy based optimizer over traveling purely the Euclidean distance. Tables A.1 and A.2 describe the start and end nodes for the ten runs in Appendix A.

## 6.1 Frictionless Path Planning with Simulated Hill

First, the algorithm is run without considering variable surface friction types. Therefore, the components that are attributed to energy consumed while traversing an area are only the internal resistances and the changes in elevation. This method answers the question of how the the vertical component affects planning based on energy estimations.

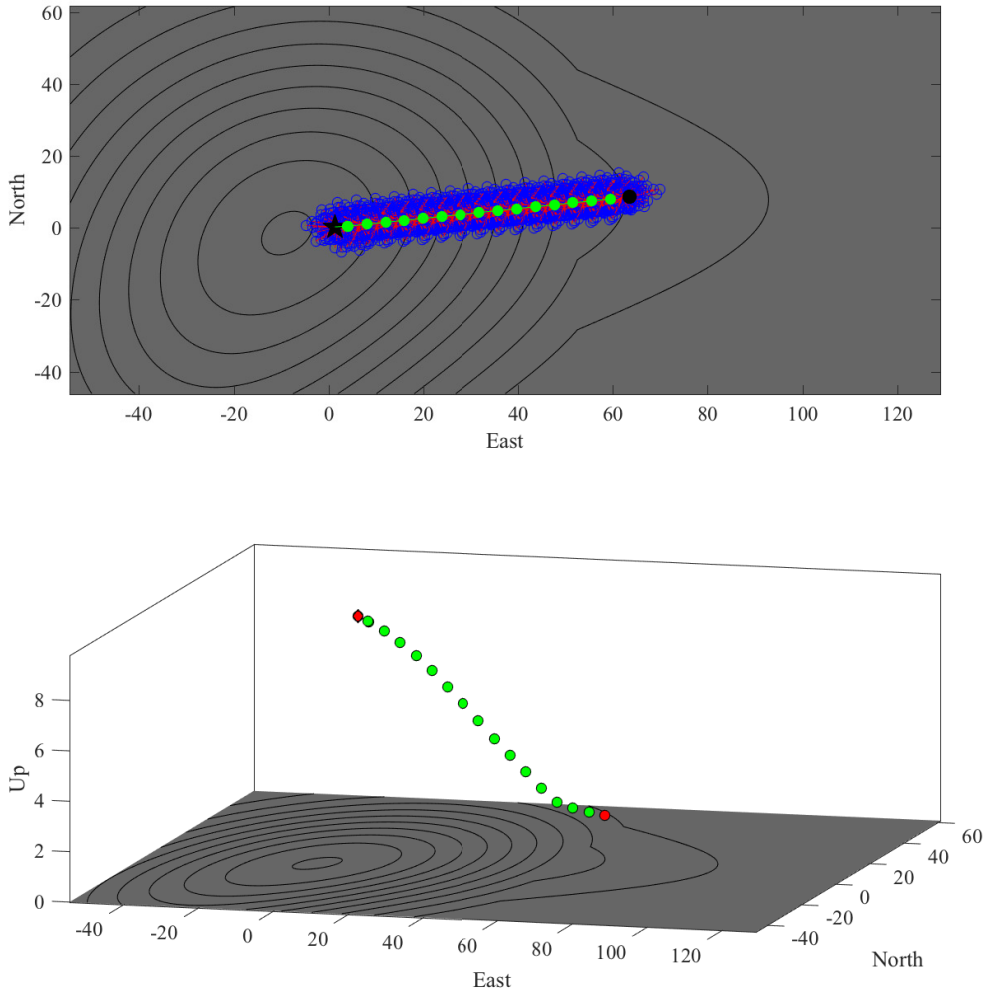


Figure 6.1: Solution path for distance-based optimizer.

Figures 6.1 and 6.2 exhibit the path solution for both the distance- and energy-based optimizer respectively. This particular run resulted in a reduction of energy consumed by 1921.68J, which represents a 9.01% decrease from the distance optimizer. However, the tradeoff is that the UGV would have to travel an additional 93.69m to achieve the goal node. This 149% increase in distance may require additional time, which may not be acceptable in occasions of crisis. But for routine transportation of nonperishable supplies, the sacrifice in timeliness may be worth the energy sav-

ings.

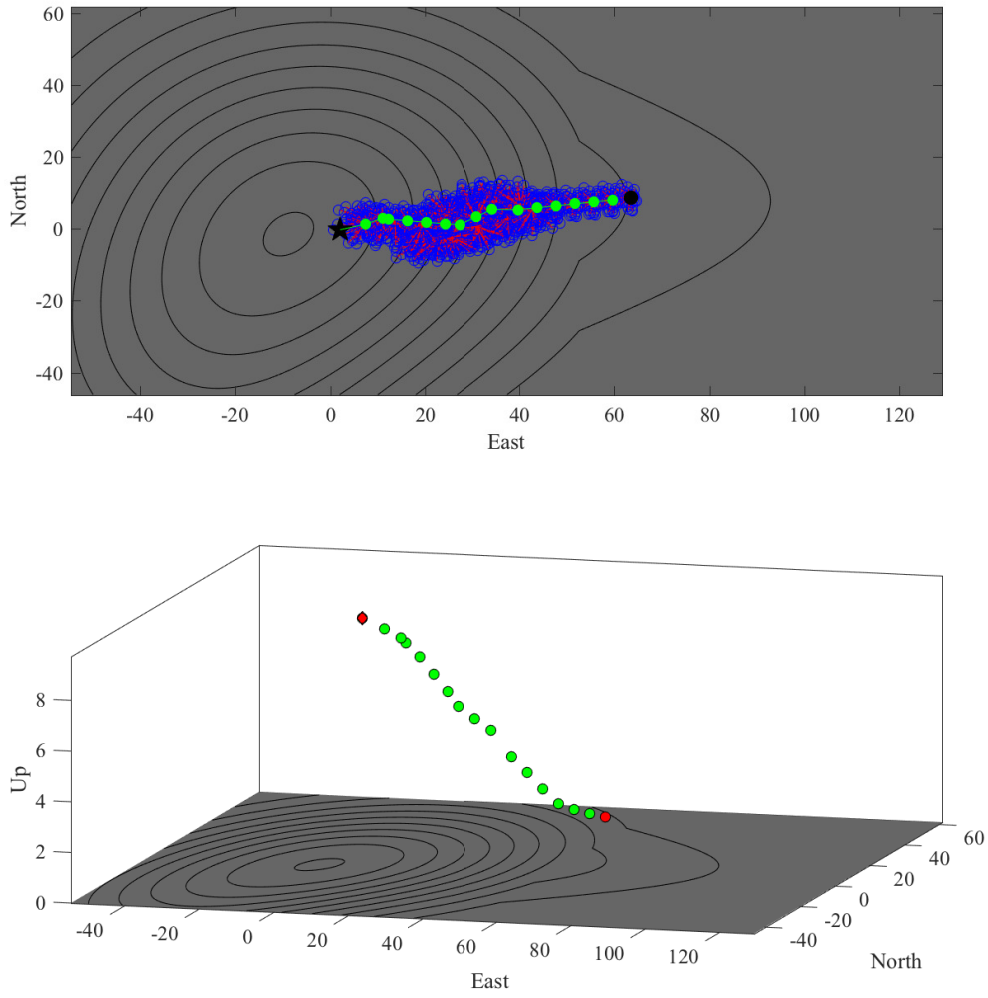


Figure 6.2: Solution path for energy based optimizer.

Between the ten simulations, the average decrease in energy used was 10.113%, and the average increase in distance was 97.817%. The standard deviations are listed in Table 6.1, and the data used to solve for the values below are given in Tables A.3 and A.4.

Table 6.1: Mean value and standard deviation of the change rate from distance to energy optimizer at original slope

| | Mean (%) | Standard Deviation (%) |
|---|---|---|
| Distance (m) | 97.817 | 83.048 |
| Energy (J) | -10.113 | 4.962 |

## 6.2 Path Planner Considering All Energy Components

This section contains the results of running the planner for ten node sets for energy- and distance-based optimization now including all factors: the friction, slope, and internal resistances considered together. By including all three factors, this thesis marries the novel work with that of Pentzer [7]. After running ten simulated runs over the same map displayed in Figure 5.8, the average rate of change in distance and energy and one standard deviation from those values were calculated and are listed in Table 6.2. Tables 6.3 and A.6 contain a comprehensive list of results from the simulations.

Table 6.2: Mean value and standard deviation of the change rate from distance- to energy-based optimizer at original slope with friction

| | Mean (%) | Standard Deviation (%) |
|---|---|---|
| Distance (m) | 26.744 | 37.453 |
| Energy (J) | -17.352 | 18.932 |

The real-world scenario this simulated map is mimicking is of a hill with a road leading from one side to the other and the corner of the road occurs at the crux of the hill. An example of how the added friction and elevation affects the path planner is seen in Figures 6.3 and 6.4. In Figure 6.3 the distance-based optimizer is utilized and results in a path that runs through the high friction

zone in the dark gray and directly up the hill. The cost of the path is 3.51kJ and takes 106.04m to arrive at the target node. But by adding an additional 5.13m, an energy-optimized path would have saved 2.40kJ. The optimal path found in Figure 6.4 avoids the majority of the high friction zone and does not directly climb the hill. These small changes are what make the major difference in energy savings.

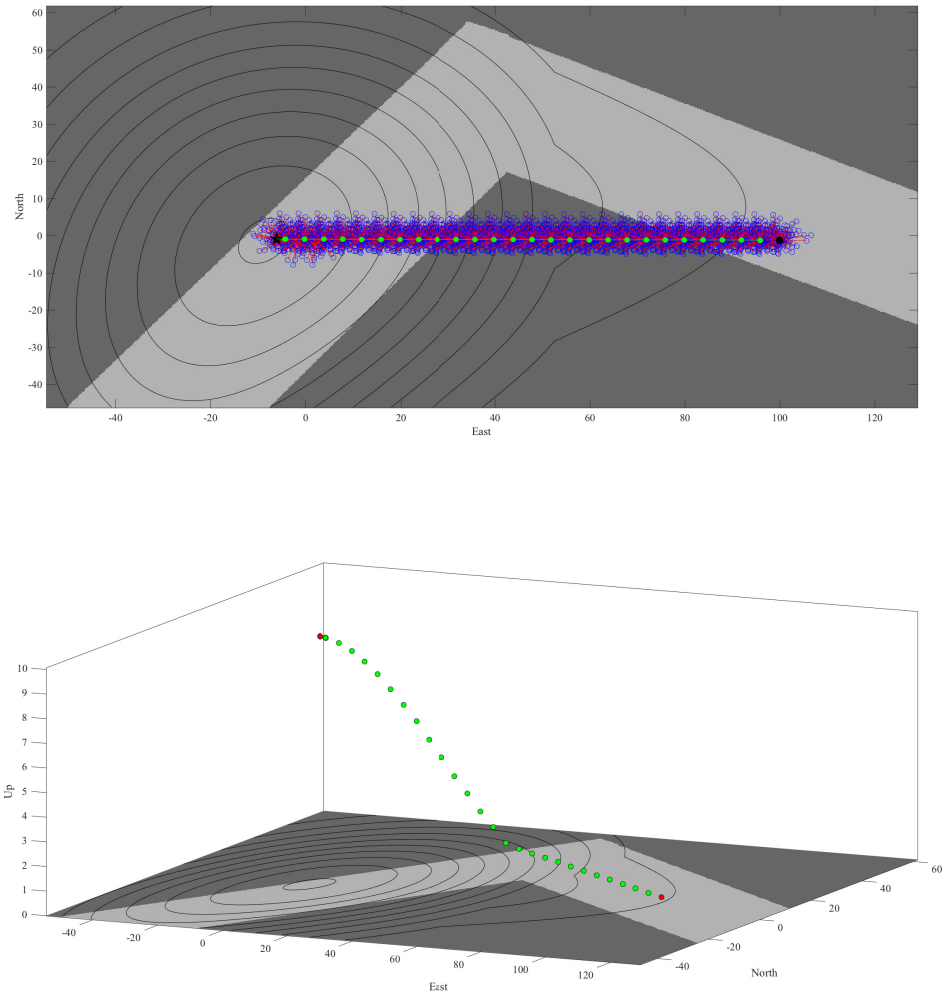


Figure 6.3: Solution path for distance-based optimizer with friction.

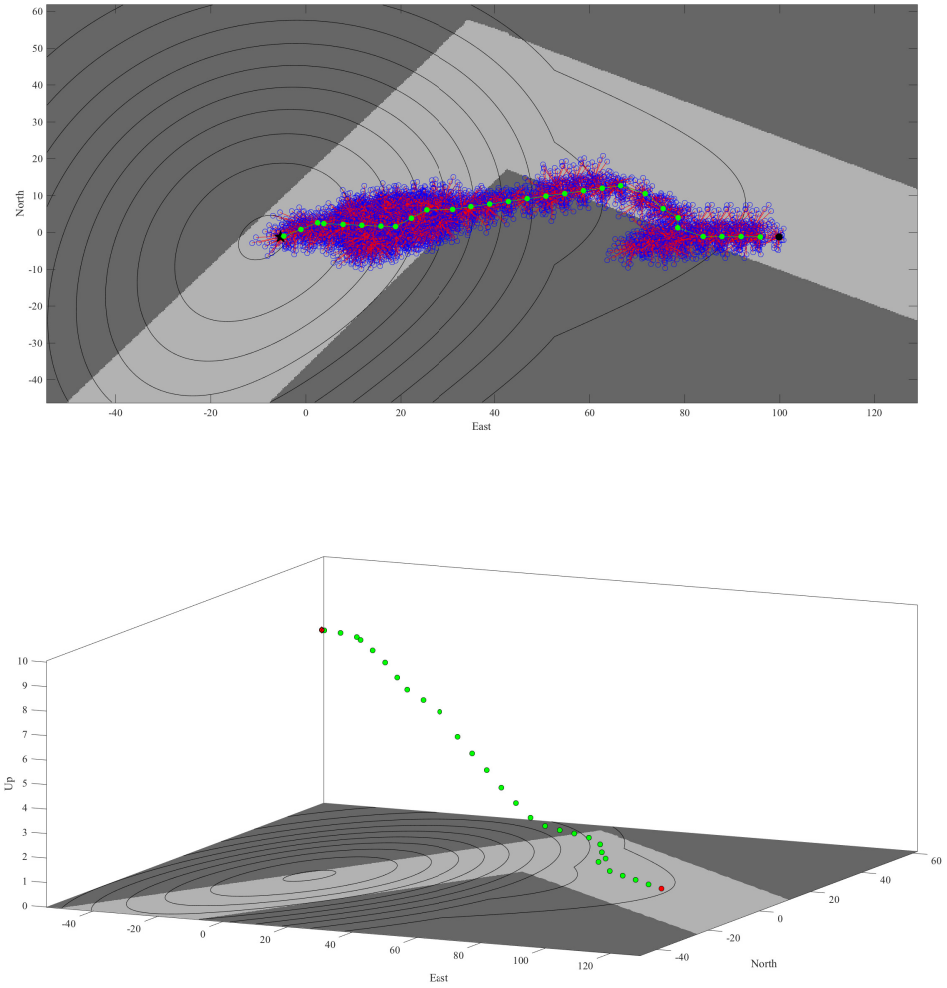


Figure 6.4: Solution path for energy-based optimizer with friction.

## 6.3 The Effects of Increasing the Slope on Planner Results

The prior discussions investigated how a particular hill affects optimal path created by the algorithm, but the slope of that prior terrain is not particularly steep. This section delves into the results of making the slope of an area steeper. On the right side of Figures 6.6 and 6.7, there is an

example of the resulting path after increasing the angle of the hill to become Figure 6.5.

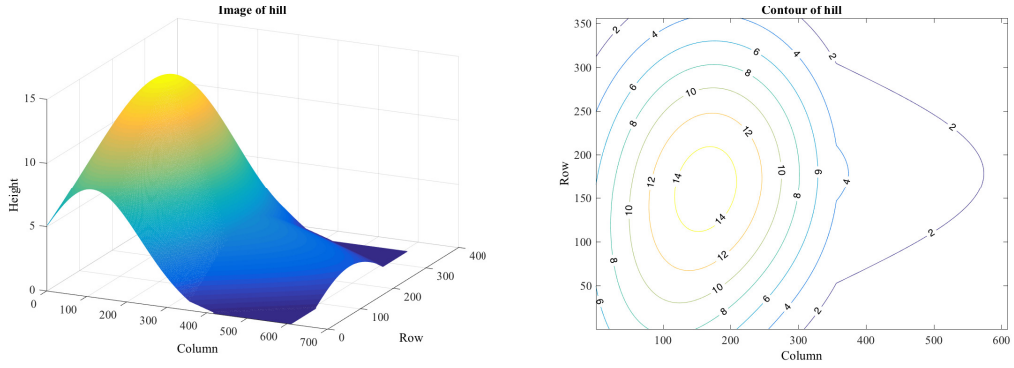


Figure 6.5: Adjusted simulated hill in mesh (left) and contour (right) plots.

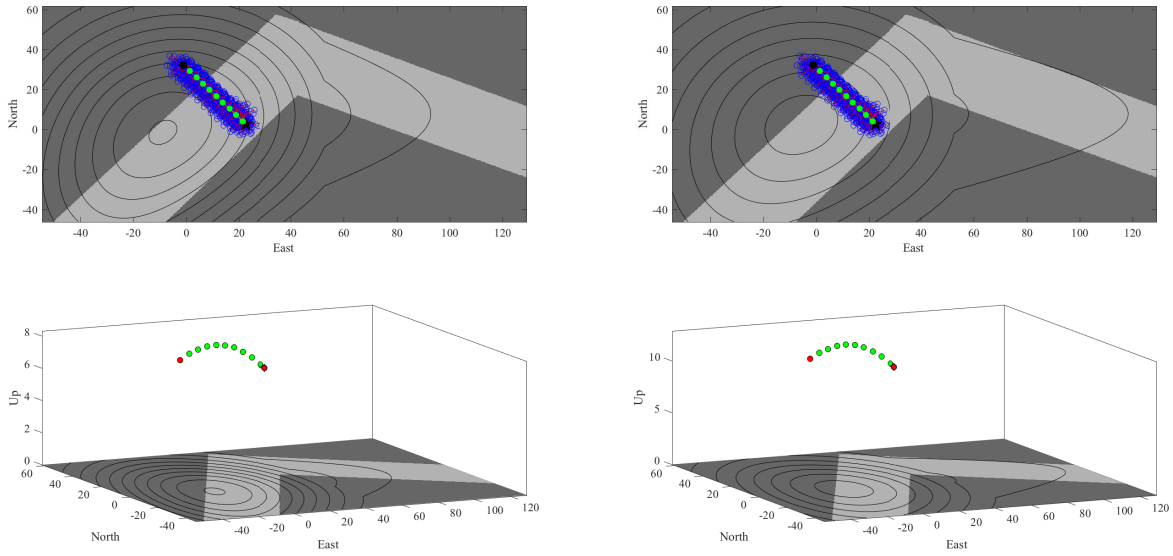


Figure 6.6: Distance-based optimizer path result with original (left) vs heightened (right) slope.

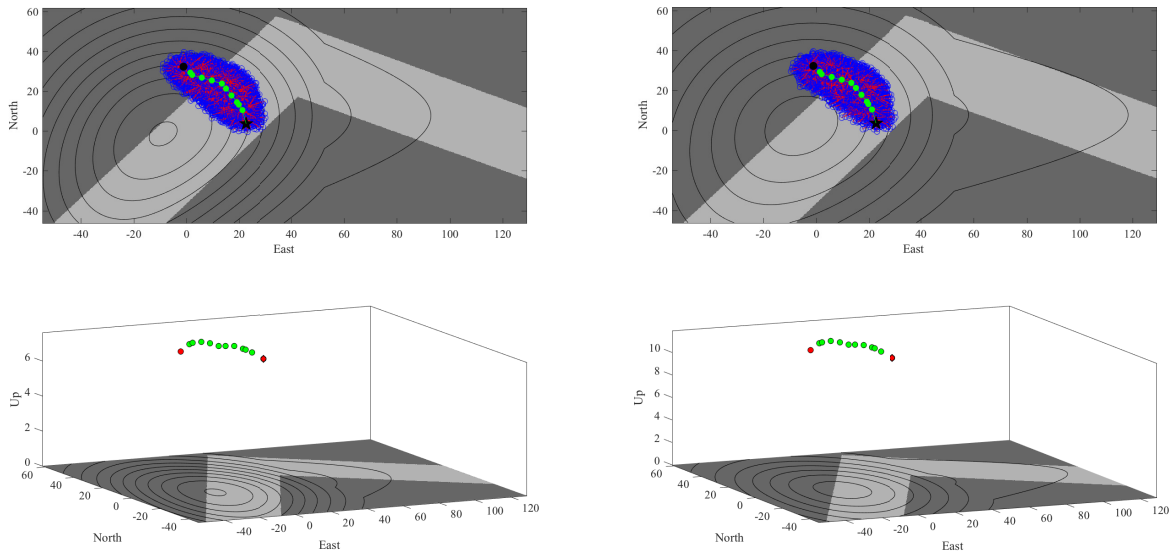


Figure 6.7: Energy-based optimizer path result with original (left) vs heightened (right) slope.

From the comparisons shown in these figures, the energy-optimized paths for both hills are similar in shape. This similarity is due largely in part to the fact that the Up coordinate of the Target node is only .375m higher for the original hill and .142m higher for the altered hill than the Up coordinate of the Start node. Since the vertical components are close to equal in both cases, the UGV seeks to avoid energy-intensive hill-climbing and descents, and instead follows a path along the contour line rather than run up and back down the hill in a Euclidean path. As the pitch map is increased, the effectiveness of the energy optimizer increases, as seen in Tables 6.3 and 6.2.

Table 6.3: The energy cost for simulated run number six using the energy and distance optimizer for an original hill and one with an increased slope gradient.

| Slope Type | $EnergyCost_{Energy}$ (J) | $EnergyCost_{Dist}$ (J) |
|---|---|---|
| Original | 10193.90 | 12105.46 |
| Altered | 10557.48 | 12666.69 |

Table 6.4: The energy cost for simulated run number six using the energy and distance optimizer for an original hill and one with an increased slope gradient.

| Slope Type | $\Delta EnerCost$ (J) | $ChangeRate_{Energy}$ (%) |
|---|---|---|
| Original | -1911.564 | -15.791 |
| Altered | -2109.214 | -16.652 |

These results agree with intuition that, if a robot has to arrive on the other side of a mountain, it would be more energy effective to increase the energy consumed due to internal resistances by driving around the base of the mountain than to climb to the peak and back down to the base, thereby raising the energy necessary to fulfill the pitch-induced power. The energy optimizer continually weighs the cost benefit of avoiding an energy inefficient zone over heading straight towards the target, so applied the algorithm never excludes the distance based path. Figure 6.4 is an ideal example of when avoiding a high energy area no longer saves the overall energy consumed, which results in the UGV traveling through the high friction patch. Prior to that point though, the vehicle skirted around the edge of the road.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

This thesis presents the modifications made to an established energy-aware path-planning method, modeling the distance, surface-friction, and gravitational energy needed for UGV's to travel from a given start node to a goal location. Beginning with a brief history of path-planning approaches, the body of this work focuses on the modivications necessary for energy-based implemention of an SMBPO model. The first component of calculating energy usage requires methods to describe robot vehicle kinematics and the significance in ICR locations for skid-steer vehicle behavior. The next layer added to the environment is a friction map. The thesis results illustrate how overlaying a pitch map might impact the amount of energy consumed.

The results show that the vertical component of terrain has an important effect on the energy required to travel, and this thesis demonstrates that although, computationally expensive, the energy cost allows more effective paths to be determined in complex terrain than a simple straight-line

motion. The research has yielded an algorithm which could be applied to a variety of robotic platforms and establish a more energy-efficient path in their use.

## 7.2 Future Work

The next steps to this research would be to investigate potential topics encountered in the research, including:

- **Improving the Algorithm Run Time**: Currently, the energy-based optimizer can take up to several hours to solve for an optimal path. There are time inefficient operations that could be manipulated into more effective functions.

- **Experimental Data**: After collecting data of a traversable area for $G$, $\mu$, and pitch maps, it would be insightful to validate the algorithm with experimental data. The algorithm would have to be run first with the experimental maps and thereby dictate the start and end locations, and the speed of the vehicle. The UGV would be given the optimal path and drive at a constant speed across it. Power loggers would gather the energy consumption of the vehicle over the given path, so it could then be compared to the estimated energy cost from the algorithm.

# Bibliography


[1] CHUY JR., O., COLLINS JR., E., YU, W., AND ORDONEZ, C. Power modeling of a skid steered wheeled robotic ground vehicle. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation* (May 2009), pp. 4118–4123.

[2] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[3] HALTON, J. H. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2* (1960), 84–90.

[4] MARTÍNEZ, J. L., MANDOW, A., MORALES, J., PEDRAZA, S., AND GARCÍA-CEREZO, A. J. Approximating kinematics for tracked mobile robots. *The International Journal of Robotics Research 24*, 10 (Oct. 2005), 867–878.

[5] MATA, A. S., TORRAS, A. B., CARRILLO, J. A. C., JUANCO, F. E., FERNÁNDEZ, A. J. G., MARTÍNEZ, F. N., AND FERNÁNDEZ, A. O. *Fundamentals of machine theory and mechanisms*, vol. 40. Springer, 2016.

[6] MORALES, J., MARTÍNEZ, J. L., MANDOW, A., GARCÍA-CEREZO, A. J., AND PEDRAZA, S. Power consumption modeling of skid-steer tracked mobile robots on rigid terrain. *IEEE Transactions on Robotics 25*, 5 (2009), 1098–1108.

[7] PENTZER, J. *Utilization of ICR Kinematics in Estimation, Control, and Energy-Aware Mission Planning for Skid-Steer Vehicles*. PhD thesis, The Pennsylvania State University, 2014.

[8] PENTZER, J., BRENNAN, S., AND REICHARD, K. Model-based prediction of skid-steer robot kinematics using on-line estimation of track instantaneous centers of rotation. *Journal of Field Robotics 31*, 3 (2014), 455–476.

[9] PENTZER, J., BRENNAN, S., AND REICHARD, K. On-line estimation of vehicle motion and power model parameters for skid-steer robot energy use prediction. In *Proceedings of the American Control Conference* (2014).

[10] PENTZER, J., REICHARD, K., AND BRENNAN, S. Energy-based path planning for skid-steer vehicles operating in areas with mixed surface types. In *American Control Conference (ACC), 2016* (2016), IEEE, pp. 2110–2115.

[11] SHIGLEY, J., AND UICKER, J. Theory of machines and mechanisms, 1980.

[12] TACK, S., BURKE, M., AND SINHA, S. Local exploration strategies for a mobile robot in a highly dynamic environment. In *Robotics and Mechatronics Conference of South Africa (ROBOMECH), 2012 5th* (2012), IEEE, pp. 1–6.

[13] UICKER, JR, J. J., PENNOCK, G. R., AND SHIGLEY, J. E. *Theory of Machines and Mechanisms*, third ed. Oxford University Press, New York, 2003.

[14] WONG, J. *Theory of Ground Vehicles*, 4 ed. John Wiley and Sons, 2008.

# Appendix A

This appendix includes the results of ten simulations on a sloped terrain without changes in friction and ten simulations, using the same start and end nodes, on a sloped terrain with changes in friction. Table A.1 lists the starting North, East, and Up values in meters for each run and Table A.2 provides the target node coordinates in a mirrored column format. The rest of this chapter is divided into a section that focuses on the results of frictionless terrain and another on the results when all components of energy consumption are considered and equally important.

Table A.1: Starting node locations for ten simulated runs

| Run | North Start (m) | East Start (m) | Up Start (m) |
|---|---|---|---|
| 1 | 54.6393 | 14.8580 | 2.1965 |
| 2 | 70.1260 | 15.7522 | 1.6588 |
| 3 | 54.3415 | 14.8580 | 2.2065 |
| 4 | 86.2082 | 7.7043 | 1.2054 |
| 5 | 99.9079 | -1.2379 | 0.6442 |
| 6 | -1.0529 | 32.4442 | 7.0236 |
| 7 | 63.5739 | 8.5985 | 1.9640 |
| 8 | -8.4984 | -2.4302 | 10.0906 |
| 9 | 45.1091 | 17.5407 | 3.2778 |
| 10 | 124.0313 | -20.0164 | 0 |

Table A.2: Target node locations for ten simulated runs.

| Run | North Target (m) | East Target (m) | Up Target (m) |
|---|---|---|---|
| 1 | -21.3047 | -28.3624 | 8.5436 |
| 2 | -15.3483 | -32.5354 | 8.1449 |
| 3 | -21.0069 | -28.9585 | 8.5005 |
| 4 | -18.0287 | -39.6891 | 7.3985 |
| 5 | -7.0093 | -0.9398 | 10.0896 |
| 6 | 23.0705 | 2.0409 | 7.1332 |
| 7 | 0.1383 | -0.0456 | 9.8601 |
| 8 | 126.1161 | 13.9638 | 0 |
| 9 | -2.8399 | -35.2181 | 7.0072 |
| 10 | 33.1963 | 8.8966 | 5.4386 |

# Results from Frictionless and Elevated Image

## Data Recorded from MATLAB

Table A.3: The distance and energy cost for ten simulated runs using the energy and distance optimizer without friction.

| Run | $Dist_{Energy}$ (m) | $EnergyCost_{Energy}$ (J) | $Dist_{Dist}$ (m) | $EnergyCost_{Dist}$ (J) |
|---|---|---|---|---|
| 1 | 158.5297195 | 22577.59934 | 86.01865751 | 25033.0781 |
| 2 | 157.5050779 | 24593.17693 | 96.3162292 | 26236.50016 |
| 3 | 158.2703995 | 22523.26751 | 86.01865751 | 24978.74626 |
| 4 | 157.4296688 | 27363.71851 | 114.0536575 | 29956.47061 |
| 5 | 152.1347355 | 29235.86515 | 106.0436575 | 30985.06982 |
| 6 | 153.8522602 | 8323.711232 | 37.95865751 | 10826.64955 |
| 7 | 156.5434165 | 19397.0581 | 62.85783503 | 21318.73955 |
| 8 | 155.5107148 | 25516.10984 | 134.0786575 | 27729.9672 |
| 9 | 157.9797406 | 17055.58393 | 69.99865751 | 19511.06268 |
| 10 | 158.735216 | 22983.8888 | 94.02865751 | 25043.80999 |

## Calculations Made from MATLAB Data

Table A.4: The difference between distance and energy cost and the rate of change between energy and distance optimizer results for ten simulated runs without friction.

| Run | $\Delta EnerCost$ (J) | $\Delta Dist$ (m) | $ChangeRate_{Energy}$ (%) | $ChangeRate_{Dist}$ (%) |
|---|---|---|---|---|
| 1 | -2455.478756 | 72.51106196 | -9.808936584 | 84.29690029 |
| 2 | -1643.323228 | 61.1888487 | -6.263500156 | 63.52911571 |
| 3 | -2455.478756 | 72.25174201 | -9.830272223 | 83.99543087 |
| 4 | -2592.752102 | 43.37601131 | -8.655065332 | 38.03123219 |
| 5 | -1749.204671 | 46.09107798 | -5.645314602 | 43.46424771 |
| 6 | -2502.938322 | 115.8936027 | -23.11830922 | 305.3153359 |
| 7 | -1921.681454 | 93.68558151 | -9.014048179 | 149.0436021 |
| 8 | -2213.857362 | 21.43205724 | -7.983627771 | 15.98468961 |
| 9 | -2455.478756 | 87.9810831 | -12.58505903 | 125.6896721 |
| 10 | -2059.921192 | 64.70655853 | -8.225270807 | 68.81578472 |

# Results from Friction and Elevated Image

## Data Recorded from MATLAB

Table A.5: The distance and energy cost for ten simulated runs using the energy and distance optimizer.

| Run | $Dist_{Energy}$ (m) | $EnergyCost_{Energy}$ (J) | $Dist_{Dist}$ (m) | $EnergyCost_{Dist}$ (J) |
|---|---|---|---|---|
| 1 | 109.687142 | 24977.15652 | 86.01865751 | 26631.5912 |
| 2 | 96.68559824 | 28223.22786 | 96.3162292 | 29753.22898 |
| 3 | 89.24381402 | 24695.37498 | 86.01865751 | 26577.25937 |
| 4 | 113.20248302 | 33272.89380 | 114.0536575 | 35711.11778 |
| 5 | 111.17049564 | 11117.04956 | 106.0436575 | 35141.20388 |
| 6 | 84.33354761 | 10193.89641 | 37.95865751 | 12105.46004 |
| 7 | 91.973617 | 21633.41687 | 62.85783503 | 23556.6579 |
| 8 | 154.7728588 | 28055.04355 | 134.0786575 | 34503.35856 |
| 9 | 98.11883939 | 18148.96241 | 69.99865751 | 21109.57578 |
| 10 | 101.1743911 | 27127.71185 | 94.02865751 | 35333.93018 |

## Calculations Made from MATLAB Data

Table A.6: The difference between distance and energy cost and the rate of change between energy and distance optimizer results for ten simulated runs.

| Run | $\Delta EnerCost$ (J) | $\Delta Dist$ (m) | $ChangeRate_{Energy}$ (%) | $ChangeRate_{Dist}$ (%) |
|---|---|---|---|---|
| 1 | -1654.434684 | 23.66848445 | -6.21230129 | 27.51552411 |
| 2 | -1,530.0011249 | 0.3693690467 | -5.1423028 | 0.3834961665 |
| 3 | -1881.884386 | 3.225156507 | -7.080806789 | 3.749368568 |
| 4 | -2438.2239795 | -0.8511745 | -6.8276328 | -0.7462930 |
| 5 | -24024.1543207 | 5.1268381 | -68.3646309 | 4.8346485 |
| 6 | -1911.563626 | 46.37489009 | -15.79092096 | 122.1721028 |
| 7 | -1923.24103 | 29.11578197 | -8.164320417 | 46.32005215 |
| 8 | -6448.315013 | 20.69420126 | -18.68894879 | 15.43437385 |
| 9 | -2960.613375 | 28.12018188 | -14.02497807 | 40.17245884 |
| 10 | -8206.218325 | 7.145733569 | -23.22475389 | 7.599527376 |

# Appendix B

This chapter contains the main file required to run the optimizer. It is broken into multiple sections that tell a story with a beginning, middle, and end. The program starts by flags for plots, loading initial data files and constants, simulating initial kinematics, selecting start and end points, and initializing the list needed to run the algorithm. The meat of the code starts the algorithm and finds the optimal path. The file finishes by plotting the resulting path and saving data results. In the beginning commented section, the other files needed to use this program are listed and can be found in Appendix C.

```
%This is the main script in the kinematically constrained A* type path
%planning algorithm for skid-steer robots. This algorithm is based on the
%SBMPO papers cited in Pentzer's and Gruning theses. This program ...
    allows you to select
%a start and end point on a Power Map and then plans a path between the two
%points. The user can choose to use either a distance or energy-based cost
%function. This planner includes the slope and friction of the terrain to
%predict how it affects the energy costwalker
%
% Inputs: Start and End Points, ICR locations of vehicle
%
% Outputs: Energy or Distance optimal path
%
% Other m-files required: func_Kinematics, func_image_to_NEU, distance3D,
% expand_array_NEU, func_NE_to_image, func_Power_Gain, insert_open_NEU,
% min_fn, node_index, func_halton_set
%
% Author: Jesse Pentzer & Veronica Gruning
% email: vag5076@psu.edu
% May, 2018

close all;
%clear all;
```

```
clc;

global POW_Map;

%Set flag to 1 to plot figures 1 & 2 showing hills
flag_plot_hills = 1;

%Set true to plot progress of planning algorithm. Useful for debugging, but
%very slow.
Plot_During_Planning = true;

%Set true to plot the movement trajectories used to expand each row.
Plot_Movements = false;

%Choose one planning mode, either Energy or distance3D by uncommenting ...
    a line.
Planning_Mode = 'Energy';
% Planning_Mode = 'Distance';

% PLOT_LIMIT is the number of iterations before plotting
PLOT_LIMIT = 200;

%This function is used to convert (N,E) position into pixel coordinates so
%the proper power model parameters can be looked up.

%Load a power map of the operational area. Load a junk variable with
%something that should be there in the workspace. If that load fails,
%reload the workspace.
try
    temp = POW_Map.AltitudeInteractive(1,1);
catch
    load('POW_Map_Test.mat');

    %The power map is a structure with the following fields.
    %   Image: A grayscale image of the test area. There should be as many
    %          gray levels as there are unique surfaces in the map.
    %   Colors: An array that relates the gray levels to power model ...
        parameters.
    %          Format: [Gray Level, Alpha, G, Beta]
    %   CornersLL: An array containing the [Latitude, Longitude] of the ...
        four corners
    %          of the image. They are listed by row in order of ...
        Northwest, Northeast,
    %          Southeast, and Southwest.
    %   ImProp: Another structure with information about the image.
    %   Altitude: The height at any point in the image
    %   AltitudeInteractive: The flipud(Altitude) for an accurate visual
    %   representation of the hill
    %% Properties of vehicle. Approximately equal to Tankbot values.

    % ICR Locations (m)
    y_r = 0.5; % right track ICR
    y_l = -0.5; % left track ICR
    x_v = 0.1; % ICR on the y-axis
```

```
73
74      % Vehicle Mass
75      Mass = 100; %80.2858495 mass of tankbot kg
76      Weight = Mass*9.81;  % Weight of the vehicle (N)
77
78      % Width between tracks (m)
79      W = 0.5207;
80
81      % Pressure at each wheel (N)
82      P = Weight/8.0;
83
84      % Distance from geometric center to track center along Y axis
85      Ty = 0.26033;
86
87      % Vectors from vehicle center to left bogie wheel locations
88      Vl1 = [0.2476, -Ty];
89      Vl2 = [0.0755, -Ty];
90      Vl3 = [-0.0826, -Ty];
91      Vl4 = [-0.2477, -Ty];
92
93      % Vectors from vehicle center to right bogie wheel locations
94      Vr1 = [0.2476, Ty];
95      Vr2 = [0.0755, Ty];
96      Vr3 = [-0.0826, Ty];
97      Vr4 = [-0.2477, Ty];
98
99      % Pre-Compute sums for power model.
100     A_l1 = Vl1 - [x_v, y_l];
101     A_l2 = Vl2 - [x_v, y_l];
102     A_l3 = Vl3 - [x_v, y_l];
103     A_l4 = Vl4 - [x_v, y_l];
104
105     A_r1 = Vr1 - [x_v, y_r];
106     A_r2 = Vr2 - [x_v, y_r];
107     A_r3 = Vr3 - [x_v, y_r];
108     A_r4 = Vr4 - [x_v, y_r];
109     Sum1 = P*norm(A_l1);
110     Sum2 = P*norm(A_l2);
111     Sum3 = P*norm(A_l3);
112     Sum4 = P*norm(A_l4);
113
114     Sum5 = P*norm(A_r1);
115     Sum6 = P*norm(A_r2);
116     Sum7 = P*norm(A_r3);
117     Sum8 = P*norm(A_r4);
118     Sum = Sum1+Sum2+Sum3+Sum4+Sum5+Sum6+Sum7+Sum8;
119
120     %% Set parameters for each node for the psuedo-grid implementation, ...
          define the minimum distance3D and heading change allowed between ...
          two nodes.
121     Grid_D = .5; %(m)
122     Grid_D_Squared = Grid_D^2;
123     Grid_H = 20*pi/180; %(rad)
124
```

```
125     % Maximum track speeds (m/s)
126     Max_Vl = 1.0; % left track max speed
127     Max_Vr = 1.0; % right track max speed
128
129     % Determine a set of movements for the vehicle. These will be ...
            transformed to expand array points during the path planning stage
130
131     %Get a set of inputs sampled from the control space.
132     V_l = func_halton_set(2,40)*Max_Vl;
133     V_r = func_halton_set(3,40)*Max_Vr;
134
135     %Add control commands allowing the vehicle to drive straight
136     V_l = [0.5; V_l;];
137     V_r = [0.5; V_r;];
138
139     %Simulate each command for tmax seconds
140     Dist = 0;
141     Psi = 0;
142     Movement = zeros(length(V_l),6);
143     q = [0; 0; 0; y_r; y_l; x_v];
144     t = 0;
145     dt = 0.01;
146     tmax = 8;
147
148     %% Adjust the POW map with the hill and altitude changes
149     % Create a struct to save the power map results for use in other ...
            algorithms.
150     % POW_Map.Image(1:356,1:609) = 127; %same low friction for all ...
            terrain area
151
152     % Create a sub struct that is a hill created by RPQ for [0:356, ...
            0:356] this
153     % is an entirely simulated hill, and with measured elevations, this
154     % component would be removed
155     r = -1:.00563:1;
156     p = -1:.00563:1;
157     [R,P] = meshgrid(r,p);
158     Q =  3*(1-R).^2.*exp(-(R.^2) - (P+1).^2) + ...
            15*exp(-R.^2-P.^2)-1.3533528; %function that creates the desired ...
            hill
159     POW_Map.Altitude = Q;
160
161     %Create a sub struct that decreasing the rest of the hill down to zero
162     %altitude for [357:609, 0:356], because the Q can only create a square
163     %matrix of data
164     for k = 357:609
165         for i = 1:356
166             if POW_Map.Altitude(i,k-1) ≤ .1
167                 POW_Map.Altitude(i,k) = 0;
168             else
169                 POW_Map.Altitude(i,k) = POW_Map.Altitude(i,k-1) - .01;
170             end
171         end
172     end
```

```
173
174     %Create a sub struct that has a converted pixel to NEU altitude
175     for ii = 1:length(POW_Map.Altitude)
176         POW_Map.AltitudeInteractive(:,ii) = flipud(POW_Map.Altitude(:,ii));
177     end
178
179     %Save the Altitude and AltitudeInteractive sub structs in addition ...
            to the sub structs already existent in POW_Map_Test.mat
180     save('POW_Map_Hill.mat','POW_Map');
181 end  % Ends try/catch statement to see if the stuff is loaded yet
182
183 % graphs that more explicitly show the terrain's slope and friction
184 if 1==flag_plot_hills
185     % plot of the hill
186     subplot(1,2,1);
187     mesh(POW_Map.Altitude);
188     % colormap(gray);
189     xlabel('Column','Color','black');
190     ylabel('Row','Color','black');
191     zlabel('Height','Color','black');
192     title('Image of hill');
193     view(25,25)
194     set(gca,'fontname','times')
195     set(gca,'fontsize', 12)
196
197     % contour plot of the hill
198     subplot(1,2,2);
199     %hold on
200     [C,h] = contour(POW_Map.Altitude);
201     %colormap(gray);
202     xlabel('Column','Color','black');
203     ylabel('Row','Color','black');
204     title('Contour of hill');
205     clabel(C,h,'FontSize','default')
206     set(gca,'fontname','times')
207     set(gca,'fontsize', 12)
208     hold off
209 end
210
211 %% Simulate the movement of the vehicle using ICR kinematics. In this
212 %simulation I assume that the ICRs remain constant so I can do this
213 %once at the beginning. It could be easily adapted to re-calculate these
214 %trajectories if the ICRs vary.
215
216 for count = 1:length(V_l)
217     %ICR Kinematics to determine speed and angular rate
218     U = [V_r(count), V_l(count)];
219     omega = -(V_l(count) - V_r(count))/(y_l-y_r);
220     Vx = (V_l(count)*y_l - V_r(count)*y_r)/(y_l-y_r);
221     %Simulate movement until the max time is reached
222     while(1)
223         [qk] = func_Kinematics(q,U,dt);
224         Dist = sqrt(qk(1)^2 + qk(2)^2);
225         Psi = qk(3);
```

```
226         t = t + dt;
227         if t ≥ tmax
228             %The max time has elapsed. Store the state AND power model ...
                   terms.
229             Movement(count,:) = [qk(1:3)', abs(omega)*Sum, ...
                    (abs(U(1))+abs(U(2))), t];
230             break;
231         end
232         q = qk;
233         %Plot the movement, if desired.
234         if Plot_Movements
235             plot3(q(2),q(1),q(3)*180/pi,'b.');
236         end
237     end
238     q = [0; 0; 0; y_r; y_l; x_v];
239     t = 0;
240 end
241
242 %Add labels to the movement plot.
243 if Plot_Movements
244     xlabel('East');
245     ylabel('North');
246     zlabel('Heading');
247     grid on;
248 end
249
250 %% Start an interactive session to choose the start and end points from ...
       the power map image.
251 % Show a figure for the user to see
252 imshow(POW_Map.Image);
253 hold on
254 [C,h] = contour(POW_Map.AltitudeInteractive);
255 clabel(C,h,'FontSize','default')
256 xlabel('Please Select the Target using the Left Mouse ...
       button','Color','black');
257 %hold off
258
259 % Wait for the user to click
260 but=0;
261 while (but ≠ 1) %Repeat until the Left button is not clicked
262     [xval,yval,but]=ginput(1);
263 end
264
265 % Show where the point hit on the map
266 plot(xval,yval,'xr');
267 text(xval,yval,'Goal');
268
269 %Round the value to the nearest pixel
270 % Now flip the y-pixel because image is shown in flipped-y coordinates
271 row = round(size(POW_Map.Image,1) - yval);
272 col = round(xval);
273
274 %convert from image coordinates to global
275 [N_user, E_user, U_user] = func_image_to_NEU(row, col, POW_Map.ImProp, ...
```

```
    POW_Map.Altitude);
ETarget = E_user;%X Coordinate of the Target
NTarget = N_user;%Y Coordinate of the Target
UTarget = U_user; %Z Coordinate of the Target


% Ask the user to select the start point
xlabel('Please Select the Start using the Left Mouse ...
    button','Color','black');
but=0;
while (but ~= 1) %Repeat until the Left button is not clicked
    [xval,yval,but]=ginput(1);
end

% Show where the point hit on the map
plot(xval,yval,'xb');
text(xval,yval,'Start');

% Force the image to front, and wait 3 seconds for human to see!
figure(3); pause(3);

%Round the value to the nearest pixel
% Now flip the y-pixel because image is shown in flipped-y coordinates
row = round(size(POW_Map.Image,1) - yval);
col = round(xval);

%convert from image coordinates to global
[N_user, E_user, U_user] = func_image_to_NEU(row, col, POW_Map.ImProp, ...
    POW_Map.Altitude);
EStart=E_user;%Starting Position
NStart=N_user;%Starting Position
UStart = U_user; %Z Coordinate of the Target

%If desired, the start and end points can be hard coded in here and the ...
    graphical inputs are
%ignored.
kk=1; %dictates the row of data saved at the end of code
points = [23.0705 2.0409  -1.0529  32.4442 ];
ETarget = points(1,1);
NTarget = points(1,2);
[row,col] = func_NE_to_image(NTarget,ETarget,POW_Map.ImProp);
UTarget = POW_Map.Altitude(row,col);
EStart = points(1,3);
NStart =  points(1,4);
[row,col] = func_NE_to_image(NStart,EStart,POW_Map.ImProp);
UStart =  POW_Map.Altitude(length(POW_Map.Altitude(:,1)) - row + 1,col);

temp = [ETarget, NTarget, UTarget, EStart, NStart, UStart];

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%LISTS USED FOR ALGORITHM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%OPEN LIST STRUCTURE
%-------------------------------------------------------------------------
```

```
326 %IS ON LIST 1/0 |N val |E val |U val |Heading |Parent N val |Parent E ...
       val |Parent U val |Parent
327 %Heading |h(n) |g(n) |f(n) | Energy
328 %--------------------------------------------------------------------------
329 OPEN=zeros(5000,11);
330 OPEN(:,1) = -1;
331
332 %CLOSED LIST STRUCTURE
333 %---------------------------------
334 %N val | E val | U val | Heading
335 %---------------------------------
336 CLOSED=[];
337 CLOSED_COUNT=size(CLOSED,1);
338
339 %set the starting node as the first node
340 ENode = EStart;
341 NNode = NStart;
342 UNode = UStart;
343 HNode = atan2(ETarget-ENode,NTarget-NNode);
344
345 %Initialize counter
346 OPEN_COUNT=1;
347
348 %First node, no cost to arrive here
349 path_cost=0;
350
351 %Keep track of how much energy we have used. Needed to add this to track
352 %energy during distance3D planning.
353 ener_cost = 0;
354
355 %Calculate the euclidean distance3D to the goal
356 goal_distance=distance3D(NNode,ENode,UNode,NTarget,ETarget,UTarget);
357
358 %Insert this node into the open list.
359 OPEN(OPEN_COUNT,1:13)=insert_open_NEU(NNode,ENode,UNode,HNode,NNode, ...
       ENode,UNode,HNode,path_cost,goal_distance,goal_distance,ener_cost);
360
361 %Close this node because we will expand it first thing
362 OPEN(OPEN_COUNT,1)=0;
363 CLOSED_COUNT=CLOSED_COUNT+1;
364 CLOSED(CLOSED_COUNT,1)=NNode;
365 CLOSED(CLOSED_COUNT,2)=ENode;
366 CLOSED(CLOSED_COUNT,3)=UNode;
367 CLOSED(CLOSED_COUNT,4)=HNode;
368 NoPath=1;
369 Goal_Dist_Squared = 100;
370
371 %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
372 % START ALGORITHM
373 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
374 if(Plot_During_Planning)  % Open and label a figure
375     h_fig = figure(345667);
376     set(h_fig,'Name','Progress');
377     hold on;
```

```
    plot(EStart,NStart,'bx');
    text(EStart,NStart,'Start')
    axis equal;
    xlabel('East');
    ylabel('North');
end
plot_count = 0;
exp_count = 0;
while((Goal_Dist_Squared > 4.0) && NoPath == 1)
    fprintf(1,'Plot count: %d \t OPEN_COUNT: %d \t exp_count %d \t ...
       \n',plot_count,OPEN_COUNT, exp_count);

    %Expand the node that currently has the lowest cost.
    exp_array=expand_array_NEU(NNode,ENode,UNode,HNode,path_cost, ...
       NTarget,ETarget,UTarget,CLOSED,Movement,Grid_D_Squared,Grid_H, ...
       Planning_Mode,POW_Map.ImProp, POW_Map.Altitude,Weight);
    exp_count=size(exp_array,1);

    %UPDATE LIST OPEN WITH THE SUCCESSOR NODES
    %OPEN LIST FORMAT
    %--------------------------------------------------------------------
    %IS ON LIST 1/0 |N val |E val |Heading |Parent N val |Parent E val ...
       |Parent
    %Heading |h(n) |g(n) |f(n) | Energy
    %--------------------------------------------------------------------
    %EXPANDED ARRAY FORMAT
    %--------------------------------
    %|X val |Y val |Z val |h(n) |g(n)|f(n)| Energy
    %--------------------------------
    %Verify that the new nodes are not too similar to nodes already ...
       within the
    %open list
    for i=1:exp_count
        flag=0;
        for j=1:OPEN_COUNT
            Dist = distance3D(exp_array(i,1),exp_array(i,2), ...
               exp_array(i,3),OPEN(j,2),OPEN(j,3),OPEN(j,4));
            if( Dist < Grid_D_Squared)
                Head_Diff = exp_array(i,4) - OPEN(j,5);
                while Head_Diff > pi
                    Head_Diff = Head_Diff - 2*pi;
                end
                while Head_Diff < -pi
                    Head_Diff = Head_Diff + 2*pi;
                end
                if abs(Head_Diff) < Grid_H
                    flag=1;
                end
            end
        end %End of j for
        if flag == 0
            OPEN_COUNT = OPEN_COUNT+1;
            OPEN(OPEN_COUNT,:)=insert_open_NEU(exp_array(i,1), ...
               exp_array(i,2),exp_array(i,3),exp_array(i,4),NNode,ENode, ...
```

```
                    UNode,HNode,exp_array(i,5),exp_array(i,6),exp_array(i,7), ...
                    exp_array(i,8)+ener_cost);
425         end %End of insert new element into the OPEN list
426     end %End of i for
427
428     %% Find the cost minimal point
429
430     %Find out the node with the smallest fn
431     index_min_node = min_fn(OPEN,OPEN_COUNT,ETarget,NTarget,UTarget);
432     if (index_min_node ≠ -1)
433         %Set ENode and NNode to the node with minimum fn
434         NNode=OPEN(index_min_node,2);
435         ENode=OPEN(index_min_node,3);
436         UNode=OPEN(index_min_node,4);
437         HNode=OPEN(index_min_node,5);
438         path_cost=OPEN(index_min_node,10);%Update the cost of reaching ...
                the parent node
439         ener_cost = OPEN(index_min_node,13);
440         %Move the Node to list CLOSED
441         CLOSED_COUNT=CLOSED_COUNT+1;
442         CLOSED(CLOSED_COUNT,1)=NNode;
443         CLOSED(CLOSED_COUNT,2)=ENode;
444         CLOSED(CLOSED_COUNT,3)=UNode;
445         CLOSED(CLOSED_COUNT,4)=HNode;
446
447         OPEN(index_min_node,1)=0;
448     else
449         %No path exists to the Target!!
450         NoPath=0;%Exits the loop!
451     end %End of index_min_node check
452
453     %If desired, plot the progress of the planning algorithm every 50
454     %iterations
455     plot_count = plot_count + 1;
456     if(Plot_During_Planning)
457         if plot_count == PLOT_LIMIT
458             figure(345667)
459             Index = find(OPEN(:,1)==1);
460             plot(OPEN(Index,3),OPEN(Index,2),'ro');
461             hold on;
462             plot(CLOSED(:,2),CLOSED(:,1),'go');
463             plot(ETarget,NTarget,'kd');
464             plot(EStart,NStart,'bo');
465             hold off;
466             drawnow;
467             plot_count = 0;
468         end
469     end
470
471     %Calculate the goal distance3D to determine if goal is reached
472     Goal_Dist_Squared = (ENode - ETarget)^2 + (NNode - NTarget)^2 + ...
            (UNode - UTarget)^2;
473
474 end %End of While Loop
```

```
475
476 %% Once algorithm has run The optimal path is generated by starting of ...
       at the
477 %last node(if it is the target node) and then identifying its parent node
478 %until it reaches the start node. This is the optimal path
479
480 %Traverse OPEN and determine the parent nodes
481 parent_E = ENode;
482 parent_N = NNode;
483 parent_U = UNode;
484 parent_H = HNode;
485 Optimal_path = [];
486 i = 1;
487 while( parent_E ≠ EStart || parent_N ≠ NStart || parent_U ≠ UStart)
488     Optimal_path(i,1) = parent_N;
489     Optimal_path(i,2) = parent_E;
490     Optimal_path(i,3) = parent_U;
491     Optimal_path(i,4) = parent_H;
492
493     %Get the grandparents:-)
494     inode=node_index(OPEN,parent_N,parent_E,parent_U,parent_H);
495     parent_N=OPEN(inode,6);%node_index returns the index of the node
496     parent_E=OPEN(inode,7);
497     parent_U=OPEN(inode,8);
498     parent_H=OPEN(inode,9);
499     i=i+1;
500 end
501
502 %% Plot the optimal path on top of the power model coefficient map
503 subplot(2,1,1);
504
505 N = linspace(POW_Map.ImProp.N_min_max(1,1), ...
       POW_Map.ImProp.N_min_max(1,2),size(POW_Map.Altitude,1));
506 E = linspace(POW_Map.ImProp.E_min_max(1,1), ...
       POW_Map.ImProp.E_min_max(1,2),size(POW_Map.Altitude,2));
507 imagesc([POW_Map.ImProp.E_min_max(1) ...
       POW_Map.ImProp.E_min_max(2)],[POW_Map.ImProp.N_min_max(1) ...
       POW_Map.ImProp.N_min_max(2)],flipud(POW_Map.Image));
508 cmap = [0, 0, 0
509     0.7 0.7 0.7 %//light gray
510     0.4 0.4 0.4]; %//white
511 colormap(cmap);
512 hold on;
513 contour(E,N,POW_Map.Altitude);
514 for count = 2:OPEN_COUNT
515     if OPEN(count,1)≠-1
516         plot([OPEN(count,3), OPEN(count,7)],[OPEN(count,2), ...
               OPEN(count,6)],'r-');
517         plot(OPEN(count,3),OPEN(count,2),'b-o');
518     end
519 end
520 plot(EStart,NStart,'ko','MarkerSize',8,'MarkerFaceColor','k');
521 plot(ENode,NNode,'kp','MarkerSize',15,'MarkerFaceColor','k');
522 xlabel('East');
```

```
523 ylabel('North');
524 set(gca,'ydir','normal');
525 for count = 2:size(Optimal_path,1)
526     plot([Optimal_path(count,2), ...
          Optimal_path(count-1,2)],[Optimal_path(count,1), ...
          Optimal_path(count-1,1)],'g-');
527     plot(Optimal_path(count,2),Optimal_path(count,1),'g-o', ...
          'MarkerFaceColor','g');
528 end
529 set(gca,'fontname','times')
530 set(gca,'fontsize', 12)
531 hold off
532
533 % Make a figure of the optimal path, in 3D
534 subplot(2,1,2);
535
536 imagesc([POW_Map.ImProp.E_min_max(1) POW_Map.ImProp.E_min_max(2)], ...
       [POW_Map.ImProp.N_min_max(1) POW_Map.ImProp.N_min_max(2)], ...
       flipud(POW_Map.Image));
537 cmap = [0, 0, 0
538     0.7 0.7 0.7
539     0.4 0.4 0.4];
540 colormap(cmap);
541 hold on;
542
543 xlabel('East');
544 ylabel('North');
545 zlabel('Up');
546 view(15,25)
547 set(gca,'ydir','normal');
548
549 scatter3(Optimal_path(:,2),Optimal_path(:,1),Optimal_path(:,3),'ko', ...
       'MarkerFaceColor','g');
550 scatter3(EStart,NStart,UStart,'ko','MarkerFaceColor','r');
551 scatter3(ENode,NNode,UNode,'kd','MarkerFaceColor','r');
552 [m,n] = size(POW_Map.Image);
553 contour(E,N,POW_Map.Altitude);
554 set(gca,'fontname','times')
555 set(gca,'fontsize', 12)
556 hold off
557
558 %% calculate the total distance along the optimal path
559 for ii = 2:length(Optimal_path)
560     tot_dist(ii-1,1) = ...
          sqrt(distance3D(Optimal_path(ii-1,2),Optimal_path(ii-1,1), ...
          Optimal_path(ii-1,3),Optimal_path(ii,2),Optimal_path(ii,1), ...
          Optimal_path(ii,3)));
561 end
562 total_distance = sum(tot_dist);
563
564 % save the energy cost and distance depending on kk and planning mode
565 if strcmp(Planning_Mode, 'Energy')
566     thesisdata(kk,1) = total_distance;
567     thesisdata(kk,2)= ener_cost;
```

```
568 elseif strcmp(Planning_Mode, 'Distance')
569     thesisdata(kk,3) = path_cost;
570     thesisdata(kk,4)= ener_cost;
571 end
572 save('thesis_data.mat','thesisdata');
```

# Appendix C

This Appendix includes all of the .m files needed to supplement the program found in order of how they appear in Appendix B. Many of these files come directly from Pentzer [7] and any adjustments made were explained in Section 5.2.

**func_Kinematics**

```
function [qk] = func_Kinematics(q,U,dt)
%This function takes in the current state estimate and control input and
%integrates the ICR kinematics for one time step
%
%Inputs:    q: State vector
%           U: Control vector
%
%Outputs:   qk: State vector after time step

%Extract values from state vector so kinematic equations are more readable
N = q(1);
E = q(2);
theta = q(3);
yr = q(4);
yl = q(5);
xv = q(6);

%Extract values from control vector so kinematic equations are more
%readable
Vr = U(1);
Vl = U(2);

%Integrate the kinematic equations using first order Euler integration
qk(1,1) = N + ...
    dt*(cos(theta)*(Vr*yl-Vl*yr)/(yl-yr)-sin(theta)*(Vl-Vr)*xv/(yl-yr));
qk(2,1) = E + ...
    dt*(sin(theta)*(Vr*yl-Vl*yr)/(yl-yr)+cos(theta)*(Vl-Vr)*xv/(yl-yr));
```

```
qk(3,1) = theta + dt*((Vr-Vl)/(yl-yr));
qk(4,1) = q(4);
qk(5,1) = q(5);
qk(6,1) = q(6);
```

**func_image_to_NEU**

```
%func_image_to_NEU - A function to convert from row/pixel coordinates ...
    in the
%overhead image to North/East in the local coordinate system.
%
% Syntax:  [N, E, U] = func_image_to_NEU(row, col, ImProp, Altitude)
%
% Inputs:
%    row - Pixel row position
%    col - Pixel col position
%    ImProp - The image properties structure
%
% Outputs:
%    N - Pixel north position in meters
%    E - Pixel east position in meters
%
% Author: Veronica Gruning
% email: vag5076@psu.edu
% February, 2018

function [N, E, U] = func_image_to_NEU(row, col, ImProp, Altitude)

%Determine the distance from the pixel to the image center
N_Local = (row - ImProp.row_center)*ImProp.N_p_Pixel;
E_Local = (col - ImProp.col_center)*ImProp.E_p_Pixel;
U_Local = Altitude(row, col);

%Add the position of the image center to get the position with respect
%to the local coordinate frame origin.
N = N_Local + ImProp.N_Center;
E = E_Local + ImProp.E_Center;
U = U_Local;
```

**distance3D**

```
function dist = distance3D(x1,y1,z1,x2,y2,z2)
%This function calculates the distance between any two cartesian
%coordinates.
%   Copyright 2009-2010 The MathWorks, Inc.
dist=(x1-x2).*(x1-x2) + (y1-y2).*(y1-y2) + (z1-z2).*(z1-z2);
```

**expand_array_NEU**

```
function exp_array=expand_array_NEU(node_N,node_E,node_U,node_h,hn, ...
    NTarget,ETarget,UTarget,CLOSED,Movement,Grid_D_Squared,Grid_H, ...
    Planning_Mode,ImProp, Altitude,w)
%Function to return an expanded array.
%
%This function takes a node and returns the expanded list of ...
    successors,with the
%calculated fn values. The criteria being none of the successors are on ...
    the CLOSED list.
%
%Inputs:    node_N: North position (m) of lowest cost node
%           node_E: East position (m) of lowest cost node
%           node_h: Heading (rad) of lowest cost node
%           hn: Cost to reach current node from start position
%           NTarget: Target north coordinate (m)
%           ETarget: Target east coordinate (m)
%           CLOSED: List of point coordinates that have been expanded
%           Movement: An array of movement trajectories for the expansion
%           Grid_D: Minimum distance (m) between expanded nodes
%           Grid_H: Minimum heading difference (rad) between nodes
%           Planning_Mode: String choice between "Energy" and "Distance"
%           ImProp: Another structure with information about the image.
%           Altitude: The height at any point in the image
%           w: weight of the vehicle (N)
%
%Outputs:   exp_array: array of nodes expanded from current lowest cost ...
    position

%Expand current node position with movement trajectories.
%Trajectories are rotated using current node heading.
New_Positions = zeros(size(Movement,1),4);
R = [cos(node_h), -sin(node_h); sin(node_h), cos(node_h)];
for count = 1:size(Movement,1)
    New_Positions(count,1:2) = (R*Movement(count,1:2)')' + [node_N, ...
        node_E];
    [row,col] = func_NE_to_image(New_Positions(count,1), ...
        New_Positions(count,2),ImProp);
    if isreal((length(Altitude(:,1)) - round(row) + 1)) && ...
        (length(Altitude(:,1)) - round(row) + 1) > 0 && isreal(col) && ...
        col ≤ size(Altitude,2)
         New_Positions(count,3) = Altitude(length(Altitude(:,1)) - ...
            round(row) + 1,round(col));
    else
       New_Positions(count,3) = NaN;
    end
    New_Head = node_h + Movement(count,4);

    %Constrain heading value to lie between -pi and +pi
    while New_Head > pi
        New_Head = New_Head - 2*pi;
```

```
41      end
42      while New_Head < -pi
43          New_Head = New_Head + 2*pi;
44      end
45      New_Positions(count,4) = New_Head;
46  end
47
48  New_Positions(isnan(New_Positions(:,3)),:) = [];
49
50  %Create expanded array. Each new point is compared with previous
51  %nodes to ensure that it does not violate the pseudo-grid rules.
52  exp_array=[];
53  exp_count=1;
54  c2=size(CLOSED,1);%Number of elements in CLOSED including the zeros
55
56  s_N_vector = New_Positions(:,1);
57  s_E_vector = New_Positions(:,2);
58  s_U_vector = New_Positions(:,3);
59  s_H_vector = New_Positions(:,4);
60  Dist_Goal_Squared_Vector = ...
        distance3D(NTarget,ETarget,UTarget,s_N_vector,s_E_vector,s_U_vector);
61  Dist_Goal_Vector = Dist_Goal_Squared_Vector.^0.5;
62  Dist_Goal2D_Vector = distance(NTarget,ETarget,s_N_vector,s_E_vector);
63
64  for count = 1:length(New_Positions)
65      %Get a new node state
66      s_N = New_Positions(count,1);
67      s_E = New_Positions(count,2);
68      s_U = New_Positions(count,3);
69      s_H = New_Positions(count,4);
70      flag=1;
71
72      %Make sure this expanded node isn't too close to a closed node
73      Dist_squared = ...
            distance3D(s_N,s_E,s_U,CLOSED(:,1),CLOSED(:,2),CLOSED(:,3));
74      for c1=1:c2
75          if(Dist_squared(c1,1) < Grid_D_Squared)
76              Head_Diff = s_H - CLOSED(c1,4);
77              while Head_Diff > pi
78                  Head_Diff = Head_Diff - 2*pi;
79              end
80              while Head_Diff < -pi
81                  Head_Diff = Head_Diff + 2*pi;
82              end
83              if abs(Head_Diff) < Grid_H
84                  flag=0;
85              end
86          end
87      end %End of for loop to check if a successor is on closed list.
88
89      %If the node is alright, calculate the costs and add it to the array
90      if (flag == 1)
91
92          %Calculate the power model coefficients from the power map
```

```
93          [in_flag, mu, G, min_mu, min_G, ¬] = func_Power_Gain(node_E, ...
               node_N);
94
95          %See if the expanded node is not in an obstacle
96          [not_obs, ¬, ¬, ¬, ¬, ¬] = func_Power_Gain(s_E, s_N);
97
98          %If in_flag is true, then the point lies within the power map
99          %and is not in an obstacle
100         if in_flag && not_obs
101
102             %Add position
103             exp_array(exp_count,1) = s_N;
104             exp_array(exp_count,2) = s_E;
105             exp_array(exp_count,3) = s_U;
106             exp_array(exp_count,4) = s_H;
107
108             %Calculate the energy cost or the distance cost to move from
109             %The current low-cost node to this expanded child
110             Thetaseg = atan((s_U - node_U))/distance(node_E,node_N, ...
                   s_E, s_N);
111             dist = sqrt(distance3D(node_N,node_E,node_U,s_N,s_E,s_U));
112             Ener_Thet = w*dist*sin(Thetaseg);
113             if Ener_Thet ≤ 0
114                 Ener_Theta = 0;
115             else
116                 Ener_Theta = Ener_Thet;
117             end
118             Power = Movement(count,4:5)*[mu; G];
119             Energy = Power*Movement(count,6)+Ener_Theta; ...
                   %Movement(count,6) is the time to complete the motion ...
                   trajectory
120             if strcmp(Planning_Mode, 'Energy')
121                 New_hn = Energy;
122             elseif strcmp(Planning_Mode, 'Distance')
123                 New_hn = distance(node_N,node_E,s_N,s_E); %2-D distance ...
                       planning
124             end
125             exp_array(exp_count,5) = hn+New_hn;%cost of travelling to node
126
127             %Calculate the energy cost or the distance cost to move from
128             %the current low-cost node to the target point.
129             Dist_Goal = Dist_Goal_Vector(count,1);
130             Dist_Goal2D = Dist_Goal2D_Vector(count,1);
131             if strcmp(Planning_Mode, 'Energy')
132                 Head_Goal = atan2(ETarget-s_E,NTarget-s_N);
133                 Head_Diff = Head_Goal - s_H;
134                 while Head_Diff > pi
135                     Head_Diff = Head_Diff - 2*pi;
136                 end
137                 while Head_Diff < -pi
138                     Head_Diff = Head_Diff + 2*pi;
139                 end
140                 Time_Turn = Head_Diff/0.1745;  %Estimate of turn rate ...
                       for Tankbot to align with goal heading in rad/s
```

```
141                 Ener_Turn = abs(0.1745*min_mu*Time_Turn);
142
143                 NTrack = linspace(s_N, NTarget, floor(Dist_Goal));
144                 ETrack = linspace(s_E, ETarget, floor(Dist_Goal));
145                 UTrack = linspace(s_U, UTarget, floor(Dist_Goal));
146                 SegLength = Dist_Goal/(floor(Dist_Goal));
147                 SegTime = SegLength/0.5; %0.5 is speed of robot driving ...
                        straight to goal
148                 Ener_Goal = 0;
149                 for SegNum = 2:length(NTrack)
150                     [obs_flag, ¬, G, ¬, ¬, maxG] = ...
                            func_Power_Gain(ETrack(SegNum), NTrack(SegNum));
151                     Gseg = G;
152                     Thetaseg = atan((UTrack(SegNum) - UTrack(SegNum - ...
                            1)))/distance(ETrack(SegNum - 1),...
153                         NTrack(SegNum - 1), ETrack(SegNum), ...
                                NTrack(SegNum));
154
155                     if obs_flag == 0
156                         Gseg = maxG;
157                     end
158
159                     dist = sqrt(distance3D(NTrack(SegNum-1), ...
                            ETrack(SegNum-1),UTrack(SegNum-1),NTrack(SegNum), ...
                            ETrack(SegNum),UTrack(SegNum)));
160                     Ener_Thet = w*dist*sin(Thetaseg);
161                     if Ener_Thet < 0
162                         Ener_Theta = 0;
163                     else
164                         Ener_Theta = Ener_Thet;
165                     end
166                     Ener_Seg = 1.0*Gseg*SegTime;  %Equation is ...
                            |Vr|+|Vl|. Track velocities are 0.5, so add to 1.
167                     Ener_Goal = Ener_Goal + Ener_Seg + Ener_Theta;
168                 end
169                 New_gn = Ener_Goal+Ener_Turn;
170             elseif strcmp(Planning_Mode, 'Distance')
171                 New_gn = Dist_Goal2D;
172             end
173
174             %Finish adding information to expanded array
175             exp_array(exp_count,6) = New_gn;%distance between node and goal
176             exp_array(exp_count,7) = ...
                    exp_array(exp_count,5)+exp_array(exp_count,6);%fn
177             exp_array(exp_count,8) = Energy;
178             exp_count=exp_count+1;
179         end
180     end %Populate the exp_array list!!!
181 end %End of if node is not its own successor loop
```

**func_NE_to_image**

```
%func_NE_to_image - A function to convert from North/East position in the
%local frame to pixel position in the image
%
% Syntax:  [row, col] = func_NE_to_image(N, E, ImProp)
%
% Inputs:
%     N - The north position of interest in meters
%     E - The east position of interest in meters
%     ImProp - The image properties structure
%
% Outputs:
%    row - Row position of pixel
%    col - Column position of pixel
%
% Author: Jesse Pentzer
% email: jlp5573@psu.edu
% September, 2014
function [row, col] = func_NE_to_image(N, E, ImProp)

%Subtract the location of the image center
N_Local = N - ImProp.N_Center;
E_Local = E - ImProp.E_Center;

%Convert from meters to image coordinates
row = round(-N_Local*ImProp.Pixel_p_N) + ImProp.row_center;
col = round(E_Local*ImProp.Pixel_p_E) + ImProp.col_center;
```

**func_Power_Gain**

```
function [in_flag, mu, G, min_mu, min_G, max_G] = func_Power_Gain(East, ...
    North)

%This function takes in a position set and outputs the proper power model
%parameters for that position from the power map.
%
%Inputs:    East: East position (m)
%           North: North position (m)
%
%Outputs:   in_flag: True if position lies within power map, false if not
%           mu: Mu power model parameter for input position
%           G: G power model parameter for input position
%           min_mu: Minimum mu power model parameter within the map.
%           min_G: Minimum G power model parameter within the map.


global POW_Map;
```

```
17
18 %Determine if the point lies within the positions covered by the power map
19 if East > POW_Map.ImProp.E_min_max(1)+2 && East < ...
       POW_Map.ImProp.E_min_max(2)-2 ...
20         && North > POW_Map.ImProp.N_min_max(1)+2 && North < ...
               POW_Map.ImProp.N_min_max(2)-2
21
22     %Calculate the row/col in the image equivalent to the N/E position
23     [row, col] = func_NE_to_image(North, East, POW_Map.ImProp);
24
25     %Get the pixel color for the current position
26     Pixel = POW_Map.Image(row, col);
27
28     %Handle Obstacles
29     if Pixel == 0
30         in_flag = 0;
31         mu = 0;
32         G = 0;
33         min_mu = min(POW_Map.Colors(:,2));
34         min_G = min(POW_Map.Colors(:,3));
35         max_G = max(POW_Map.Colors(:,3));
36         return;
37     end
38
39     %Find which row in the colors array corresponds to this pixel color
40     Index = find(POW_Map.Colors(:,1) == Pixel);
41
42     %Extract the power model parameters from the colors array
43     mu = POW_Map.Colors(Index,2);
44     G = POW_Map.Colors(Index,3);
45
46     %Determine the minimum mu and G values within the power map
47     min_mu = min(POW_Map.Colors(:,2));
48     min_G = min(POW_Map.Colors(:,3));
49     max_G = max(POW_Map.Colors(:,3));
50
51     %Set the bounds flag
52     in_flag = 1;
53
54 else %Position lies outside of area covered by map. Set variables ...
       accordingly
55     in_flag = 0;
56     mu = 0;
57     G = 0;
58     min_mu = 0;
59     min_G = 0;
60     max_G = 0;
61 end
```

**insert_open_NEU**

```
function new_row = insert_open_NEU(xval,yval,zval,hval,parent_xval, ...
    parent_yval,parent_zval,parent_hval,hn,gn,fn,energy)
%Function to Populate the OPEN LIST
%OPEN LIST FORMAT
%-------------------------------------------------------------------------
%IS ON LIST 1/0 |N val |E val |U val|Heading |Parent N val |Parent E ...
    val |Parent U val |Parent Heading |h(n) |g(n)|f(n)|
%-------------------------------------------------------------------------
%
%   Copyright 2009-2010 The MathWorks, Inc.
new_row=[1,13];
new_row(1,1)=1;
new_row(1,2)=xval;
new_row(1,3)=yval;
new_row(1,4)=zval;
new_row(1,5)=hval;
new_row(1,6)=parent_xval;
new_row(1,7)=parent_yval;
new_row(1,8)=parent_zval;
new_row(1,9)=parent_hval;
new_row(1,10)=hn;
new_row(1,11)=gn;
new_row(1,12)=fn;
new_row(1,13)=energy;
end
```

**min_fn**

```
function i_min = min_fn(OPEN,OPEN_COUNT,xTarget,yTarget,zTarget)
% Function to return the Node with minimum fn
% This function takes the list OPEN as its input and returns the index ...
    of the
% node that has the least cost
%
%   Copyright 2009-2010 The MathWorks, Inc.

 temp_array=[];
 k=1;
 for j=1:OPEN_COUNT
     if (OPEN(j,1)==1)
         temp_array(k,:)=[OPEN(j,:) j]; %#ok<*AGROW>
         k=k+1;
     end
 end %Get all nodes that are on the list open

 %Send the index of the smallest node
 if size(temp_array ~= 0)
  [min_fn,temp_min]=min(temp_array(:,12));%Index of the smallest node ...
      in temp array
```

```
20    i_min=temp_array(temp_min,14);%Index of the smallest node in the OPEN ...
          array
21   else
22       i_min=-1;%The temp_array is empty i.e No more paths are available.
23   end
```

**node_index**

```
1  function n_index = node_index(OPEN,Nval,Eval,Uval,Hval)
2      %This function returns the index of the location of a node in the list
3      %OPEN
4      %
5      %   Copyright 2009-2010 The MathWorks, Inc.
6      i=1;
7      while(OPEN(i,2) ≠ Nval || OPEN(i,3) ≠ Eval ||OPEN(i,4) ≠ Uval || ...
           OPEN(i,5) ≠ Hval )
8          i=i+1;
9      end
10     n_index=i;
11 end
```

**func_halton_set**

```
1  function [S] = func_halton_set(Base, Num_Points)
2  %This function computes a Halton sequence with a given base and length
3  %
4  %Inputs:    Base: The base number for the Halton set
5  %           Num_Points: The length of the returned sequence
6  %
7  %Outputs:   S: Computed Halton sequence
8
9  S = zeros(Num_Points,1);
10
11 for k = 1:Num_Points
12
13     pp = Base;
14     kp = k;
15     phi = 0;
16
17     while kp > 0
18         a = mod(kp,Base);
19         phi = phi + a/pp;
20         kp = floor(kp/Base);
21         pp = pp*Base;
22     end
23     S(k) = phi;
```

```
24 end
```

# ACADEMIC VITA


## VERONICA GRUNING

532 White Oak Dr ◇ Virginia Beach, VA, 23462

757 635 7355 ◇ veronicagruning@gmail.com


## EDUCATION

**The Pennsylvania State University** *Aug 2014 - May 2018*
Schreyer Honors College
Bachelor of Science in Mechanical Engineering

## HONORS

Deans List (6 semesters)
Student Engagement Network, Grant Recipient (2017)
Research Experience for Undergraduates at Oregon State University Funded by the National Science Foundation, Grant Recipient (2016)
Research Experience for Undergraduates at Penn State Funded by the National Science Foundation, Grant Recipient (2015)
Wiedhahn Academic Excellence Scholarship (8 semesters)
University Park Provost Scholarship (8 semesters)
Alumni Memorial Scholarship (8 semesters)
Jason & Martha Lee Stone Scholarship (6 semesters)
A & W Colliflower Scholarship (4 semesters)
Corey A. Wincek-Charotte Newcombe Scholarship (2 semesters)
Pre-Eminence in Honors Education Scholarship (2 semesters)
Morreale Family Scholarship (2 semesters)

## WORK EXPERIENCE

**Applied Research Lab at Penn State** May 2017 Present
*Undergraduate Research* *State College, PA*

- Wrote a program that identified the type of terrain a robot was traversing across based on the magnitude of change in the roll, pitch, and yaw angles.
- Studied how a robots center of mass relates to the locations of the instantaneous centers of rotation.
- Tested an unmanned ground vehicle to find its mechanical and electrical limits on an obstacle coarse approved by National Institute of Standards and Technology.
- Upgraded and repaired an unmanned ground vehicle that sat in a state of disuse for nearly three years.

**East Halls at Penn State** Aug 2016 - May 2018
*Resident Assistant* *University Park, PA*

- Directly supervised forty residents and co-supervised 400 residents through their first year transition and advised them on academic decisions for their following semesters.
- Organized educational and social events to establish a sense of community on a coed floor.
- Performed administrative duties including inspection of safety equipment and budget management.

**Robotic Decision Making Lab at Oregon State** Jun 2016 - Aug 2016
*Research Assistant* *Corvallis, OR*

- Analyzed data structures and identified relevant variables in order to reconstruct 2D SONAR images into a 3D map of underwater environments.

· Applied learned mathematical concepts of mechatronics, including number theory and data theory, to write algorithms that aided in the extraction of pertinent variables from large data pools.
· Facilitated the testing of an unmanned underwater vehicle in a prescribed swimming pool setup.

**Intelligent Vehicles and Systems Group at Penn State** Jan 2015 - May 2017
*Research Assistant* *University Park, PA*

· Designed a wiring schematic that removed ground fault loops from a mapping vehicle.
· Completed all aspects of the wiring diagram, including measuring necessary wire lengths, soldering, and fabrication of laser cut acrylic mounts.
· Collected visual mapping data to quantitatively support graduate theses on horizon feature detection.

**Women in Engineering Program at Penn State** Aug 2015 - May 2016
*Academic Facilitator* *University Park, PA*

· Mentored students in WEP to increase the retention of female students.
· Planned and facilitated weekly sessions reviewing the topics of calculus II and differential equations.
· Developed strong teaching abilities by clarifying material from class to ensure student understanding.

## SOFTWARE & SKILLS

| | | | | | | |
|---|---|---|---|---|---|---|
| ROS | Python | Simulink | Arduino | LaTeX | MATLAB | Spanish (proficient) |
| SolidWorks | Linux | Soldering | Machining | Microsoft Excel | C++ | |

## LEADERSHIP

**Out in Science, Technology, Engineering, and Mathematics** Aug 2015 - Present
*Treasurer*

· Acquired a sponsorship and a mentorship program from Ford Motor Company and Leidos
· Gained insight into the process of how academia receives external funding for research and professional development.
· Provided a safe space for people of the LGBTQA community, and saw at the national conference that many major companies celebrate the marketable trait of being diverse

**Prism Benefitting THON** Dec 2016 - April 2018
*Founder and President*

· Created a philanthropic group that actively includes Penn State students of the LGBTQA community to feel welcomed.
· Formulated an independent thought into fruition in terms of an end goal, demonstrating drive when starting a new venture.

**American Institute of Aeronautics and Astronautics** Jan 2017 - May 2018
*Member*

· Explored a new field of study that is not covered in the standard curriculum, to learn topics about an area that is not within comfort level.
· Exhibited initiative to obtain knowledge and insight into a field necessary to achieving an end goal.

**Penn State Crew Team** Aug 2014 - Jan 2017
*Varsity Player*

· Scheduled out a semester in advance knowing that twenty hours per week would be dedicated to rowing practice and five weekends per semester would be away from Penn State at regattas, while maintaining coursework and managing a research timeline.

- Developed the skills to delegate time to specific tasks in order to accomplish the objectives of all projects.