Applying FastSLAM to Articulated Rovers

by

Robert Alexander Hewitt, B.Eng

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Applied Science

Ottawa-Carleton Institute for
Mechanical and Aerospace Engineering

Department of Mechanical and Aerospace Engineering
Carleton University
Ottawa, Ontario, Canada

January 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87796-8

Our file Notre référence
ISBN: 978-0-494-87796-8

The undersigned recommend to

the Faculty of Graduate Studies and Research

acceptance of the Thesis

Applying FastSLAM to Articulated Rovers

Submitted by **Robert Alexander Hewitt, B.Eng**

in partial fulfilment of the requirements for the degree of

Master of Applied Science

Dr. Anton de Ruiter, Co-supervisor

Dr. Alex Ellery, Co-supervisor

Dr. Metin Yaras, Department Chair

Carleton University

2012

Abstract

This thesis presents the navigation algorithms designed for use on *Kapvik*, a 30 kg planetary micro-rover built for the Canadian Space Agency; the simulations used to test the algorithm; and novel techniques for terrain classification using *Kapvik's* LIDAR (Light Detection And Ranging) sensor.

Kapvik implements a six-wheeled, skid-steered, rocker-bogie mobility system. This warrants a more complicated kinematic model for navigation than a typical 4-wheel differential drive system. The design of a 3D navigation algorithm is presented that includes nonlinear Kalman filtering and Simultaneous Localization and Mapping (SLAM). A neural network for terrain classification is used to improve navigation performance. Simulation is used to train the neural network and validate the navigation algorithms. Real world tests of the terrain classification algorithm validate the use of simulation for training and the improvement to SLAM through the reduction of extraneous LIDAR measurements in each scan.

Acknowledgments

I would like to first thank my thesis co-supervisors, Dr. Anton de Ruiter and Dr. Alex Ellery. Whenever I needed guidance, or suggestions Dr. de Ruiter was always there to help. He has been flexible enough to allow me to follow opportunities and research interests that have made my time at Carleton University a memorable and valuable experience. His technical abilities and attention to detail were essential to bringing out my best work and he always had a constructive approach. His confidence in me and unwavering patience, particularly in my last semester, allowed me to produce a thesis I am proud of. Dr. Alex Ellery, my co-supervisor, has provided me with a unique opportunity to work on a project that I am fortunate to have been a part of with the Canadian Space Agency. His upbeat attitude, leadership, and enthusiasm forms the backbone of a team that works hard to accomplish everything it sets out to do. Working within his team has brought about lifelong friendships and has challenged me in so many different ways that I can't imagine a better place for me to complete my Master's in.

I would like to thank my close friends and colleagues, Tim Setterfield and Marc Gallant. Completing my Master's with these two has been fun and enlightening. Whether it was arguing about Kalman filter theory, travelling to shuttle launches, or sharing a pitcher of beer, they have been an incredibly positive influence on my work and life over the last two years.

I would like to extend my gratitude to the rest of the *Kapvik* team: Chris Nicol,

Jesse Hiemstra, Matt Cross, Helia Sharif, Ala' Qadi, Cameron Frazier, Brian Lynch, Adam Mack, Javier Romualdez, and Amy Deeb. We all worked extremely hard together to accomplish something we can all be proud of, and along the way have become a tight-knit group. I have always believed that the key to a career or profession that you enjoy is working with people that you consider friends, and this experience has made that much more apparent. This team has such a wide range of knowledge and interests that there has never been a dull moment in the "Lab" for me.

Finally, I would like to thank my family. My mother Leslie, my father Glen, and my brother Brian. Their love, support, guidance, and values, have been an invaluable source of encouragement as I worked to complete my Master's.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

This thesis uses SI units throughout: kg, m, s

Vectors and matrices are in boldface: $\mathbf{v}, \mathbf{M}$

The standard deviation of a variable is denoted σ_x

Estimated variables are denoted $\hat{x}$

The first time derivative is denoted $\dot{x}$

The second time derivative is denoted $\ddot{x}$

Variables

Variable	Variable Meaning
$\mathbf{0}$	Zero vector or zero matrix
D_M	Mahalanobis distance
$\mathbf{C}_R^G$	Rotation matrix from reference frame R to reference frame G
f	Motion model
h	Measurement model
$\mathbf{I}_{m \times m}$	Identity matrix of dimension $m \times m$
$\mathbf{K}$	Kalman Gain

m	Index of current particle
n_x	Number of states
$\mathcal{N}$	Gaussian probability density function
p_0	Likelihood of new feature belonging to a measurement in SLAM
p_n	Likelihood of old feature belonging to a measurement in SLAM
$\mathbf{P}$	State covariance matrix
$\mathbf{P}_{xy}$	Cross-covariance between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\mathbf{P}_{xx}$	Covariance of vector $\boldsymbol{x}$
$\mathbf{Q}$	Process noise covariance
$\mathbf{R}$	Measurement noise covariance
r_w	Wheel radius
s^t	Set of robot poses up to time t
s_t	Robot pose at time t
s_t^a	Augmented state vector
$\mathbf{T}_R^G$	Transformation matrix from reference frame R to reference frame G
T_s	Time step length in seconds
t	Index of current time step
$\boldsymbol{u}$	Control or input vector
$\mathbf{U}$	Rover Configuration vector

$\boldsymbol{v}$	Velocity vector
v	Velocity magnitude
$\boldsymbol{w}$	Normally distributed, discrete, white, zero-mean process noise
$w_t^{[m]}$	weight of particle m at time t
n^t	Data associations at time t
$\boldsymbol{x}$	State vector
$\boldsymbol{x}_f$	Map feature position vector
X_G, Y_G, Z_G	Axes in the global reference frame
$X_{Rover}, Y_{Rover}, Z_{Rover}$	Axes in the rover reference frame
$X_{LIDAR}, Y_{LIDAR}, Z_{LIDAR}$	Axes in the laser reference frame
$\boldsymbol{y}$	Measurement vector
$\boldsymbol{z}$	Measurement vector in SLAM
β	Bogie angle
δ	Terrain-wheel contact angle
ϵ	Wheel slip vector
η	Wheel side slip
γ	Terrain gradient
$\boldsymbol{\Lambda}$	Map feature set
$\boldsymbol{\nu}$	Normally distributed, discrete, white, zero-mean measurement noise

$\mathbf{\Phi}$	Orientation vector
$\boldsymbol{\rho}$	Rocker angle
μ_x	mean of x
ξ	Wheel rolling slip
ζ	Wheel rotational slip
ANN	Artificial Neural Network
CKF	Cubature Kalman Filter
EKF	Extended Kalman Filter
MLPN	Multilayer Perceptron Neural Network
UKF	Unscented Kalman Filter

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Surface exploration of other planets in the solar system began with stationary landers. While these landers provided a glimpse at hereto unknown worlds like Venus and Mars, they also greatly limited the amount of science and data that could be sent back to Earth. The Soviet Union was the first to attempt a mobile solution with the Lunokhod rovers sent to the Moon in 1970 and 1973. These rovers had a large mass (840 kg) and were remote controlled via operators on Earth with no autonomous operation available. Not until 1997 was another mission attempted, this time the Mars Pathfinder mission by the U.S.A. Mars Pathfinder was the first mobile exploration on the surface of Mars and sparked new interest in research on planetary rovers. The Sojourner rover made a number of other firsts; it was the first to implement the rocker-bogie system designed to equilibrate the ground pressure on all six wheels without using springs. It was the first micro-rover (defined in this thesis as a rover with a mass under 40 kg) to operate on another planetary body at only 10.5 kg, and the first rover to implement a navigation system that included a LIDAR, Imaging system, and accelerometers. Much of the Sojourner rover technology went into the next NASA rover mission, the Mars Exploration Rovers (MER) in 2004, a pair of

rovers named Spirit and Opportunity respectively. While the MER rovers had a mission duration of 3 months, the Spirit rover operated on the surface of Mars for a total of 2623 Martian solar days, and the Opportunity rover continues to operate at the time of writing. Both these rovers, and the MSL (Mars Science Laboratory) Rover launched in November 2011 also make use of the rocker-bogie mechanism along with more advanced navigation techniques and sensors which indicate NASA's degree of confidence in the design that Sojourner started for Martian planetary rovers.

One of the major areas of improvement with each new generation of rovers is the degree of autonomy. Long transmission delays between Earth and Mars make any improvement in autonomous operations very desirable. Autonomous robotics is an active area of research, with full 3D guidance, navigation and control continuing to be an open problem [1]. With each new iteration of rovers, more computation power as well as memory are available for more advanced algorithms that have been developed in the intervening years. Navigation, defined here as the estimation of the rover's states such as position, remains one of the biggest problems for autonomous rovers. The Spirit rover ultimately succumbed to an error in traction control [2], which relies on accurate estimation of the rover's slip as it traverses over the terrain. Spirit became stuck in soft sand and was unable to escape, or orient itself in a favourable position, before the winter month's approached and with them, insufficient solar power.

Past rovers have also been sent to relatively flat areas and have avoided traversing difficult areas due to the risk involved. Improving the navigation capabilities of the rover as well as its ability to map and understand its surroundings are of major importance to future missions. One of the key goals for future rover missions to Mars are a sample return as well as mapping and exploring an area for valuable samples to be gathered by astronauts in future missions [3]. Thus, in order to accurately map and traverse a large area there is a need for a rover to autonomously navigate and map its terrain. The preferred attempt at solving this problem has been SLAM

(Simultaneous Localization and Mapping) in the robotic literature [4–7], and to the author's knowledge this has not been implemented in 3D on a rocker-bogie rover.

1.2 Goals

The goal of this thesis is to develop a form of SLAM (Simultaneous Localization and Mapping) for an articulated rover. It must estimate the full 6D pose of the rover as well as estimate a 3D map of its environment.

Recently, the Canadian Space Agency has invested in several rover technology contracts. One such contract has been for *Kapvik*, a 30 kg terrestrial prototype for a planetary micro-rover shown in Figure 1.1. *Kapvik* has been developed collaboratively by MPB Technologies, Carleton University, Ryerson University, University of Toronto, Xiphos Technologies, MDA Space Missions, and the University of Winnipeg. Carleton University has specifically contributed to the design of the rover mobility system, the avionics enclosure, and the navigation software. The design of the navigation system of the rover was assigned to the author, and presented an opportunity to design and build a robotic system that included many different types of sensors for the purpose of guidance, navigation and control. In this thesis, a full 6D SLAM approach is developed for the navigation of *Kapvik*.

In order to validate the algorithms presented in this thesis, a simulation is developed of the rover mobility system, the terrain it drives over, a LIDAR sensor that scans the environment around it, and all other sensors used in *Kapvik's* navigation that include an IMU (Inertial Measurement Unit), Sunsensor, Wheel encoders and Potentiometers. Whenever possible, the techniques presented in this thesis are also tested on rover test platforms such as the Pioneer 3-AT rover and the Husky rover shown in Figure 1.2 and Figure 1.3 respectively. The algorithms presented in this thesis make use of a nonlinear Kalman filter for estimation between LIDAR scans,

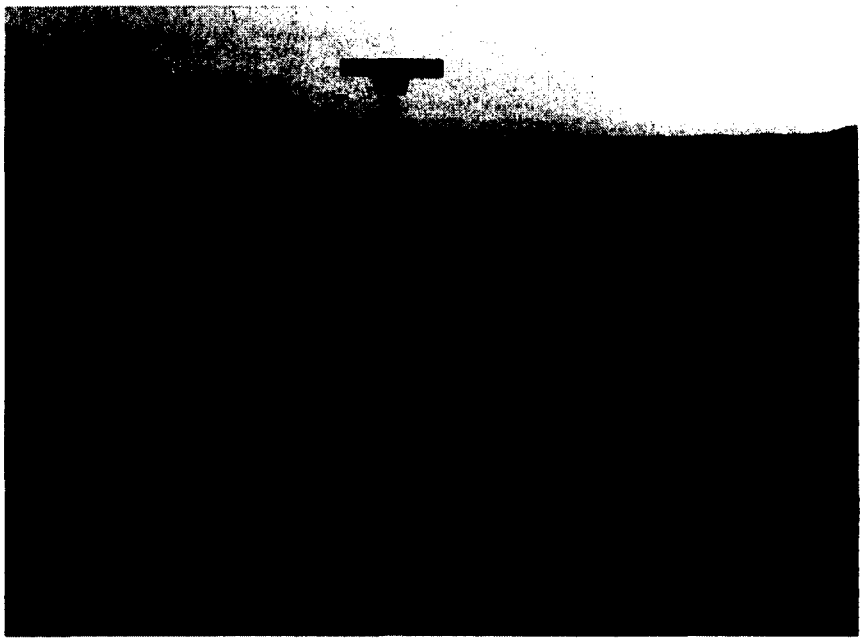


Figure 1.1: A rendering of *Kapvik* micro-rover under development at Carleton University

a methodology for when to take LIDAR scans, classification of LIDAR scans that identifies obstacles, and implementation of localization and mapping using a novel version of FastSLAM, a popular form of SLAM in the literature.

1.3 Scope

The scope of this thesis is divided into three separate areas: (i) developing a kinematic model and measurement model for an articulated rover that can be used for simulation as well as navigation (ii) Parsing and classifying large LIDAR scans in an efficient manner (iii) perform 6D SLAM for the rover and its environment. While it would be ideal to test and validate all of the involved algorithms in a real world environment, this has only been completed for the second area due to time constraints with the project. To address each of the areas mentioned above, the following contributions are made in this thesis:

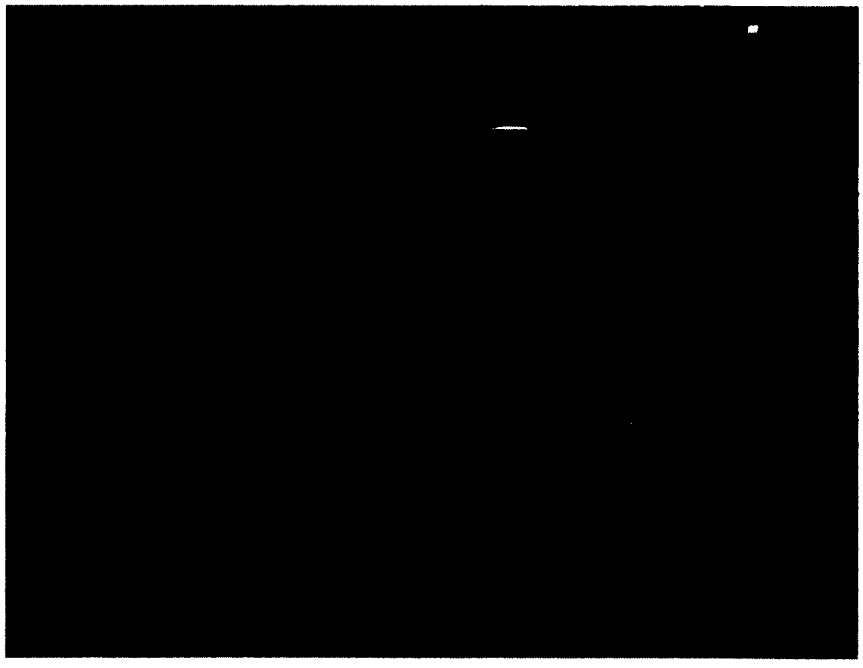


Figure 1.2: Pioneer 3-AT rover used for testing

1. The creation of an efficient LIDAR classification technique using neural networks. This method can process large amounts of data in a short amount of time and can also automatically train itself through use of a simulation. By training the neural network on different classes of terrain in the simulation, it can be easily adapted to multiple types of terrain.

2. The design of a new SLAM algorithm that makes use of an articulated rover motion model and a large suite of sensors including LIDAR. Data association is implemented in a more realistic way that takes into account the whole set of scan data through the use of the Hungarian algorithm. The SLAM algorithm works within the paradigm of a Mars rover mission with limited computation time, doing the majority of computation at "sub-goals" where the rover stops and scans its environment. To improve accuracy and stability a square root Cubature Kalman filter implementation of FastSLAM was developed.

Figure 1.3: Husky rover used for testing

3. The design of a simulation environment that includes a motion model for an articulated rover driving over hilly terrain and a LIDAR sensor model that produces scans of the terrain as well as rocks that are randomly distributed over the surface. This simulation was used for training and validation of the LIDAR classification neural network as well as validation of the SLAM algorithm.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides background information and forms a literature review on the topics that were used for the development of the algorithms presented in this thesis. Detailed explanations are provided for any topic that proved essential to the work done in this thesis as well as references for more information. Topics that were not used directly in this thesis are often mentioned for comparison purposes, with references provided for more rigorous analysis.

Important literature is detailed, focusing on the state of the art for the three areas this thesis seeks to improve upon: articulated rover navigation, FastSLAM, and LIDAR classification. Chapter 3 provides a detailed analysis of the algorithms proposed in this thesis and their integration into a complete navigation system. This includes a derivation of the models used in simulation and the Kalman filter implementation, an explanation of a novel SLAM algorithm, and the development of a new LIDAR classification technique making use of neural networks. Chapter 4 describes the test environments that are used to validate the algorithms proposed in this thesis. This includes both the development of the simulation as well as an overview of the types of terrain used in real world outdoor testing. The results of the algorithms running in simulation as well as real world testing are presented in Chapter 5. The results of the simulated tests include the full implementation of the proposed navigation and LIDAR classification algorithms. The real world tests demonstrate the performance of the LIDAR classification algorithm. Finally, Chapter 6 summarizes the contents of this thesis in addition to a review of the contributions that were made. Possible directions for future work are also detailed.

Chapter 2

Background and Literature Review

Navigation that includes 6D state estimation along with estimation of a map in 3D is considered an unsolved problem in the literature [1, 4, 8–10]. In addition to this, much of the kinematics and estimation of articulated rovers remains an active field of research in planetary exploration [11–13]. This thesis attempts to merge the two areas of research for a Mars micro-rover prototype, *Kapvik*, and use novel techniques to improve the results compared to a simpler approach. Considering the wide range of research that is applicable to this field, an extensive literature review has been done before and during the research conducted for this thesis.

The three main topics that are focused on in this thesis are rover kinematics used in state estimation, 6D SLAM and artificial neural networks for LIDAR classification. While a large number of sources and literature were vital to the completion of this thesis, the following background theory and literature review describes in more detail some of the state of the art concepts that have stood out among the literature and provided the starting point to build upon for the solutions proposed in this thesis.

2.1 State Estimation

In this thesis, state estimation refers to the estimation of *states* in a discrete time system. For the purposes of this thesis, a state refers to variables that provide a description of the status of a system, such as a rover or a map of its surroundings. For example, these states could include the rover's pose, rocker-bogie configuration, wheel contact angles and the position of features in the rover's surroundings, collectively called a map. While the dynamics of a rover driving over uneven terrain are described by a continuous-time differential equation, the input and measurements are sampled by a digital computer at discrete time instants, which this thesis refers to as *time steps*. This is known as a sampled-data system, and is the most frequently encountered system in practical robotics [14].

State estimation is an important problem in all fields of engineering and can be applied whenever a system is mathematically modelled. In the case of aerospace engineering and robotics, it is useful to estimate the state of the system in order to implement a state-feedback controller. For instance, the position and orientation of a rover and the position of non-traversable objects in its surroundings must be estimated in order to control its velocity on its way to a goal while avoiding nearby obstacles. This is the usefulness of simultaneous localization and mapping algorithms, and the focus of this thesis.

The Kalman filter was developed in the 1960s by Rudolph Kalman [15] and is an optimal state estimation algorithm for linear systems. However, most real systems are nonlinear. In the case of a rover moving over uneven terrain, the system dynamics and measurement dynamics are both nonlinear. Various nonlinear implementations of the Kalman filter have been proposed over time, and in this thesis three variants will be explored and used for the general SLAM problem, at times in combination with one another. The following topics treat each filter individually and build from

the simple Extended Kalman filter to the more involved Cubature Kalman filter [16]. Much of this discussion is adapted from [14].

State estimation is also dependent on the model that is used to describe the progression of the states over time, often called the motion model. In the case of this thesis, a motion model for *Kapvik* is required, which is an articulated rover. A brief overview of previous work in this area by Tarokh *et al.* is presented at the end of this section.

2.1.1 Extended Kalman Filter

The Extended Kalman filter (EKF) is a recursive probabilistic state estimator for nonlinear dynamic systems. The derivation is based on linearizing the system around the state estimate of the Kalman filter, which in turn is an estimate based on the linearized system. The algorithm is popular in the literature, especially in the context of navigation, for its ability to estimate unmeasurable states such as position and orientation from a system model and sensor measurements that both are in the presence of noise. This idea was originally developed by Stanley Schmidt [17] to apply the Kalman filter to spacecraft navigation. The noise in an EKF is assumed to be zero mean, white and Gaussian and is implemented as such in the simulations presented in this thesis. However this assumption is not the case in actual implementations. If needed, the non-Gaussian probability density function can be approximated by the weighted sum of several Gaussian probability density functions or, more simply, the covariance values that define the Gaussian distribution can be tuned by empirical methods. For the purpose of this thesis the latter method is used in actual implementations. The states in a system are also considered to follow a Gaussian distribution and are defined by $\mathcal{N}(\bar{\boldsymbol{x}}, \mathbf{P})$, with a mean of $\bar{\boldsymbol{x}}$ and a covariance of $\mathbf{P}$.

Given these assumptions, consider a discrete-time nonlinear system model

$$\boldsymbol{x}_t = f_{t-1}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{w}_{t-1}), \tag{2.1}$$

$$\boldsymbol{y}_t = h_t(\boldsymbol{x}_t, \boldsymbol{v}_t), \tag{2.2}$$

$$\boldsymbol{w}_t = \mathcal{N}(0, \boldsymbol{Q}_t), \tag{2.3}$$

$$\boldsymbol{v}_t = \mathcal{N}(0, \boldsymbol{R}_t), \tag{2.4}$$

where

$\boldsymbol{x}_t$ = the state vector at time step t;

$f_t(\cdot)$ = nonlinear process model;

$\boldsymbol{u}_{t-1}$ = the control input at time step t;

$\boldsymbol{w}_{t-1}$ = zero mean, white, Gaussian process noise at time step t-1;

$\boldsymbol{y}_t$ = the measurement vector at time step t;

$h_t(\cdot)$ = nonlinear measurement model;

$\boldsymbol{v}_t$ = zero mean, white, Gaussian measurement noise at time step t;

$\boldsymbol{Q}_t$ = process covariance matrix at timestep t;

$\boldsymbol{R}_t$ = measurement covariance matrix at timestep t;

and t is the current time step of the system. In this thesis a superscript of "+" or "−" refers to estimates that take place before and after a measurement has been taken respectively. Kalman filters are usually split into these two steps, where a state is propagated forward through its process model, along with its covariance, and afterwards a correction step, known as the measurement step, updates the state. A Taylor series expansion of the nonlinear process model is taken around $\boldsymbol{x}_{t-1} = \hat{\boldsymbol{x}}^+_{t-1}$ and $\boldsymbol{w}_{t-1} = \boldsymbol{0}$, where $\hat{\boldsymbol{x}}^+_{t-1}$ represents the EKF estimate of the last time step, which

results in the following:

$$\begin{aligned} \boldsymbol{x}_t &= f_{t-1}(\hat{\boldsymbol{x}}_{t-1}^+, \boldsymbol{u}_{t-1}, \boldsymbol{0}) + \left.\frac{\partial f_{t-1}}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{t-1}^+} (\boldsymbol{x}_{t-1} - \hat{\boldsymbol{x}}_{t-1}^+) + \left.\frac{\partial f_{t-1}}{\partial \boldsymbol{w}}\right|_{\hat{\boldsymbol{x}}_{t-1}^+} \boldsymbol{w}_{t-1} \\ &= f_{t-1}(\hat{\boldsymbol{x}}_{t-1}^+, \boldsymbol{u}_{t-1}, \boldsymbol{0}) + \boldsymbol{F}_{t-1}(\boldsymbol{x}_{t-1} - \hat{\boldsymbol{x}}_{t-1}^+) + \boldsymbol{L}_{t-1}\boldsymbol{w}_{t-1}. \end{aligned} \tag{2.5}$$

$\boldsymbol{F}_{t-1}$ and $\boldsymbol{L}_{t-1}$ are defined above. Similarly, the measurement model can be linearized around $\boldsymbol{x}_t = \hat{\boldsymbol{x}}_t^-$ and $\boldsymbol{v}_t = \boldsymbol{0}$ as follows:

$$\begin{aligned} \boldsymbol{y}_t &= h_t(\hat{\boldsymbol{x}}_t^-, \boldsymbol{0}) + \left.\frac{\partial h_t}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_t^-} (\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^-) + \left.\frac{\partial h_t}{\partial \boldsymbol{v}}\right|_{\hat{\boldsymbol{x}}_t^-} \boldsymbol{v}_t, \\ &= h_t(\hat{\boldsymbol{x}}_t^-, \boldsymbol{0}) + \boldsymbol{H}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^-) + \boldsymbol{M}_t\boldsymbol{v}_t. \end{aligned} \tag{2.6}$$

$\boldsymbol{H}_t$ and $\boldsymbol{M}_t$ are defined above. The result is a linear state-space model in (2.5) and a linear measurement model in (2.6). The standard Kalman filter equations [14] are then applied to these models to estimate the state as follows:

$$\begin{aligned} \boldsymbol{P}_t^- &= \boldsymbol{F}_{t-1}\boldsymbol{P}_{t-1}^+\boldsymbol{F}_{t-1}^T + \boldsymbol{L}_{t-1}\boldsymbol{Q}_{t-1}\boldsymbol{L}_{t-1}^T, \\ \boldsymbol{K}_t &= \boldsymbol{P}_t^-\boldsymbol{H}_t^T(\boldsymbol{H}_t\boldsymbol{P}_t^-\boldsymbol{H}_t^T + \boldsymbol{M}_t\boldsymbol{R}_t\boldsymbol{M}_t^T)^{-1}, \\ \hat{\boldsymbol{x}}_t^- &= f_{t-1}(\hat{\boldsymbol{x}}_{t-1}^+, \boldsymbol{u}_{t-1}, \boldsymbol{0}), \\ \hat{\boldsymbol{x}}_t^+ &= \hat{\boldsymbol{x}}_t^- + \boldsymbol{K}_t(\boldsymbol{y}_t - h_t(\hat{\boldsymbol{x}}_t^-, \boldsymbol{0})), \\ \boldsymbol{P}_t^+ &= (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t)\boldsymbol{P}_t^-. \end{aligned} \tag{2.7}$$

The initial state covariance matrix $\boldsymbol{P}_0^+$ is a diagonal matrix of the initial i^{th} state

covariances, $\sigma^2_{x_i}$, as shown below

$$\boldsymbol{P}_0^+ = \begin{bmatrix} \sigma^2_{x_1} & 0 & \cdots & 0 \\ 0 & \sigma^2_{x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma^2_{x_n} \end{bmatrix}. \tag{2.8}$$

If the initial state is known perfectly, then $\boldsymbol{P}_0^+ = diag(0)_{n\times n}$, and if it is not known at all, $\boldsymbol{P}_0^+ = \infty \boldsymbol{I}_{n\times n}$. In practical application this is not useful, so in many cases the initial covariance is set to some large value, indicating a high level of uncertainty in the initial guess. Taking the square root of the terms in $\boldsymbol{P}_t^+$ and multiplying by 3 results in the 3σ error bound values for each state estimate which represents a $\sim 99\%$ confidence interval.

The process and measurement noise, $\boldsymbol{w}_t$ and $\boldsymbol{v}_t$ respectively, are distributed according to $\boldsymbol{Q}$ and $\boldsymbol{R}$ as shown in (2.3) and (2.4). $\boldsymbol{Q}$ and $\boldsymbol{R}$ represent independent process and noise matrices that are constructed as:

$$\boldsymbol{Q}_t = \begin{bmatrix} \sigma^2_{w_{1,t}} & 0 & \cdots & 0 \\ 0 & \sigma^2_{w_{2,t}} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma^2_{w_{n,t}} \end{bmatrix}, \tag{2.9}$$

$$\boldsymbol{R}_t = \begin{bmatrix} \sigma^2_{v_{1,t}} & 0 & \cdots & 0 \\ 0 & \sigma^2_{v_{2,t}} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma^2_{v_{n,t}} \end{bmatrix}, \tag{2.10}$$

where $\sigma^2_{w_{i,t}}$ is the process noise variance of the i^{th} state and $\sigma^2_{v_{i,t}}$ is the measurement noise variance of the i^{th} measurement. The values for the measurement noise variance are related to the associated sensor (in a rover's case this could include an IMU, sun sensor, GPS, LIDAR) which usually have known noise statistics given by the manufacturer. If the sensor error is known the variance can be calculated, making assumptions such as modelling the noise on a Gaussian distribution. For instance if a laser is specified to have a range measurement error of $\pm 1mm$, then this would correspond to $3\sigma = 1mm$, and therefore $\sigma^2 = (\frac{1mm}{3})^2$. The assumption made for R_t is that the measurement errors are uncorrelated, which may not be true in a real world scenario. This would have to be tested and analyzed further and is not done for this thesis.

Given these descriptions, the algorithm for the discrete time EKF can be formulated and is described in Algorithm 2.1.

In the pre-measurement steps, Jacobians are calculated from the process model with respect to the state and noise. This is where linearization of the nonlinear model takes place, and allows the covariance to be transformed to the measurement update step. The states are inputs to the nonlinear process model, f, and the result is the new estimate, $\hat{\boldsymbol{x}}_t^-$, which is simply a function of the previous state and the last control input. This value is often known as the *a priori* estimate, and is also a so called *dead-reckoning* estimate. Due to unknown dynamics in the process model dead-reckoning can quickly lead to errors in estimates if other measurements are not taken.

Algorithm 2.1 discrete time EKF

Require: $t \geq 0$

$t = 1$

$\hat{\boldsymbol{x}}_0^+ = E(\boldsymbol{x}_0)$

$\boldsymbol{P}_0^+ = E[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0^+)^T]$

loop

{Pre-measurement step}

$\boldsymbol{F}_{t-1} = \frac{\partial f_{t-1}}{\partial \boldsymbol{x}}\Big|_{\hat{\boldsymbol{x}}_{t-1}^+}$

$\boldsymbol{L}_{t-1} = \frac{\partial f_{t-1}}{\partial \boldsymbol{w}}\Big|_{\hat{\boldsymbol{x}}_{t-1}^+}$

$\boldsymbol{P}_t^- = \boldsymbol{F}_{t-1}\boldsymbol{P}_{t-1}^+\boldsymbol{F}_{t-1}^T + \boldsymbol{L}_{t-1}\boldsymbol{Q}_{t-1}\boldsymbol{L}_{t-1}^T$

$\hat{\boldsymbol{x}}_t^- = f_{t-1}(\hat{\boldsymbol{x}}_{t-1}^+, \boldsymbol{u}_{t-1}, \boldsymbol{0})$

{Post-measurement step}

$\boldsymbol{H}_t = \frac{\partial h_t}{\partial \boldsymbol{x}}\big|_{\hat{\boldsymbol{x}}_t^-}$

$\boldsymbol{M}_t = \frac{\partial h_t}{\partial \boldsymbol{v}}\big|_{\hat{\boldsymbol{x}}_t^-}$

$\boldsymbol{K}_t = \boldsymbol{P}_t^-\boldsymbol{H}_t^T(\boldsymbol{H}_t\boldsymbol{P}_t^-\boldsymbol{H}_t^T + \boldsymbol{M}_t\boldsymbol{R}_t\boldsymbol{M}_t^T)^{-1}$

$\hat{\boldsymbol{x}}_t^+ = \hat{\boldsymbol{x}}_t^- + \boldsymbol{K}_t(\boldsymbol{y}_t - h_t(\hat{\boldsymbol{x}}_t^-, 0))$

$\boldsymbol{P}_t^+ = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t)\boldsymbol{P}_t^-$

$t \leftarrow t + 1$

end loop

This is where the post-measurement step comes in, where the *a posteriori* estimate is determined using the EKF. Using different measurements, preferably ones that come from absolute measurements (such as a sun sensor, GPS), the measurement model is linearized around the a priori estimate, $\hat{\boldsymbol{x}}_t^-$. The Kalman gain is then calculated, which helps weight the affect of new measurements on the a priori estimate. On line 14 of algorithm 2.1 the a priori estimate is being adjusted by the Kalman gain and the difference between a new measurement and the predicted measurement, $h_k(\hat{\boldsymbol{x}}_k^-, 0)$, which is calculated using the a priori estimate. These three steps are able to accommodate many sensors at once and give the EKF its ability to fuse different sensor measurements together.

2.1.2 Nonlinear Kalman Filters

While the EKF is a "nonlinear" filter in the strictest sense, it achieves nonlinear filtering by linearizing the nonlinear problem about the current state estimate. There are two problems with this: First, large Jacobian matrices must be computed. Second, it is inaccurate to propagate the probability density function of a relatively nonlinear system through a linear approximation. This can lead to stability issues.

There are a variety of nonlinear filters that do not linearize the problem, most notable of which is the UKF (Unscented Kalman Filter) [18]. The UKF is a local filter that fixes the posterior density to take an *a priori* form. It is assumed to be Gaussian and is simple and fast to execute. It does not require the calculation of large Jacobian matrices and lowers the error of propagating probability density functions through nonlinear approximations. The Unscented Kalman filter was introduced by Julier and Uhlmann in [18] and the filter used initially for the research in this thesis closely followed the form of the filter in Dan Simon's book "Optimal State Estimation" [14].

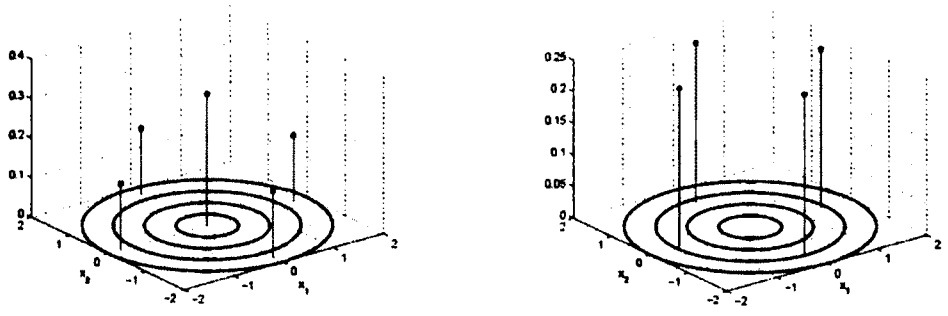
However, nonlinear filters such as the UKF are inherently suboptimal solutions

to the Bayesian filter [16], and currently known nonlinear filters suffer the *curse of dimensionality* [19]. For high dimensional state spaces they often diverge for reasons such as an incomplete or inaccurate motion or measurement model, information loss due to the Gaussian assumption for multi-modal posterior densities, high degrees of nonlinearity and numerical errors [16].

Haykin *et al.* in [16] have developed yet another local nonlinear Kalman filter design called the Cubature Kalman Filter (CKF). The Bayesian filter is solvable when all conditional densities are assumed to be Gaussian, which leads to the computation of multi-dimensional integrals. Highly efficient methods of integration, known as cubature rules, are exploited in the CKF. The CKF is claimed to be more accurate and easily extendable to high-dimensional problems compared to the UKF [20]. The full derivation of the CKF and the square root variant can be found in [16], however the algorithm and its principle differences with the UKF are presented here, along with the full algorithm for the square root variant which is in use for the remainder of this thesis for Kalman filter estimation.

Both the UKF and CKF use the concept of weighted sigma points which are transformed through nonlinear functions, resulting in another set of sigma points. In both cases the sigma points, $\boldsymbol{X}$, approximate the random variable $\boldsymbol{x}$ [14] [18]. They have a mean of $\boldsymbol{\mu_x}$ and a covariance $\boldsymbol{P}$. However, as seen in Figure 2.1, the spread and weights of the sigma-point set differs between the CKF and UKF. The points and their weights are denoted by the location and height of the stems, respectively. This difference is rooted in the third-order cubature rule as explained in [16]. What these differences boil down to are a set of limitations of the sigma point set built into the UKF that are not built into the CKF:

- There are numerical inaccuracies that lead to instability in the UKF when the dimension of $\boldsymbol{x}$, denoted n, extends beyond three.



(a) Sigma point set for the UKF

(b) Third-degree spherical-radial cubature point set for the CKF

Figure 2.1: Two kinds of distributions of point sets in two-dimensional space [16].

- It may not be possible to compute a square-root solution for the UKF that has the same numerical advantages of the CKF [16].

- The UKF computed covariance matrix is not guaranteed to be positive definite due to the negative weights, further limiting the use of a square root solution.

With the advantages of the CKF laid out and references given for a full mathematical derivation, the basic CKF algorithm is presented which is referenced throughout this thesis. Given a nonlinear, time invariant system that is affine in the noise, with n_x states, that is modelled as a discrete time system, it can be described by:

$$\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) + \boldsymbol{w}_{t-1}, \tag{2.11}$$

$$\boldsymbol{y}_t = h(\boldsymbol{x}_t) + \boldsymbol{v}_{t-1}, \tag{2.12}$$

$$\boldsymbol{w}_t \sim \mathcal{N}(0, \boldsymbol{Q}_t), \tag{2.13}$$

$$\boldsymbol{v}_t \sim \mathcal{N}(0, \boldsymbol{R}_t), \tag{2.14}$$

where $\boldsymbol{x}$ is the state vector, f is the discrete-time motion model, $\boldsymbol{u}_{t-1}$ is the input vector, t is the current time step, $\boldsymbol{w}_t$ is the normally distributed process noise with

covariance $\boldsymbol{Q}$ and zero mean, $\boldsymbol{y}$ is the measurement vector, h is the measurement model and $\boldsymbol{v}_t$ is the normally distributed measurement noise with covariance $\boldsymbol{R}$ and zero mean.

The filter begins with a state estimate of $\hat{\boldsymbol{x}}_0^+$ and a state covariance estimate of $\boldsymbol{P}_0^+$. After initialization, the CKF will repeat the following steps indefinitely.

The previous *a posteriori* estimate $\hat{\boldsymbol{x}}_t^+$ is transformed into a set of cubature points based on the cubature rule, which are denoted by $\hat{\boldsymbol{X}}_t^+$:

$$\hat{\boldsymbol{X}}_{i,t-1}^+ = \hat{\boldsymbol{x}}_{t-1}^+ + \left(\sqrt{n_x \boldsymbol{P}_{t-1}^+}\right)_i^T \qquad i = 1, ..., n_x, \tag{2.15}$$

$$\hat{\boldsymbol{X}}_{n_x+i,t-1}^+ = \hat{\boldsymbol{x}}_{t-1}^+ - \left(\sqrt{n_x \boldsymbol{P}_{t-1}^+}\right)_i^T \qquad i = 1, ..., n_x, \tag{2.16}$$

the cubature points are propagated through the nominal dynamics according to:

$$\hat{\boldsymbol{X}}_{i,t}^- = f\left(\hat{\boldsymbol{X}}_{i,t-1}^+, \boldsymbol{u}_{t-1}\right) \quad i = 1, ..., 2n_x, \tag{2.17}$$

where i represents the i^{th} column of the matrix. The evaluated cubature points, $\hat{\boldsymbol{X}}_{i,t}^-$ are averaged to obtain a new state estimate, $\hat{\boldsymbol{x}}_t^-$ and covariance, $\boldsymbol{P}_t^-$:

$$\hat{\boldsymbol{x}}_t^- = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \hat{\boldsymbol{X}}_{i,t}^-, \tag{2.18}$$

$$\boldsymbol{P}_t^- = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \left(\hat{\boldsymbol{X}}_{i,t}^- - \hat{\boldsymbol{x}}_t^-\right)\left(\hat{\boldsymbol{X}}_{i,t}^- - \hat{\boldsymbol{x}}_t^-\right)^T + \boldsymbol{Q}_t. \tag{2.19}$$

This completes the so called *a priori* update. Next, the state is updated based on the measurements received in the *a posteriori* update. The cubature points are once again generated just as before, but this time are propagated through the measurement model, h :

$$\bar{\boldsymbol{X}}_{i,t}^{-} = \hat{\boldsymbol{x}}_t^{-} + \left(\sqrt{n_x \boldsymbol{P}_t^{-}}\right)_i^T \qquad i = 1, ..., n_x, \tag{2.20}$$

$$\bar{\boldsymbol{X}}_{n_x+i,t}^{-} = \hat{\boldsymbol{x}}_t^{-} - \left(\sqrt{n_x \boldsymbol{P}_t^{-}}\right)_i^T \qquad i = 1, ..., n_x, \tag{2.21}$$

$$\hat{\boldsymbol{Y}}_{i,t} = h\left(\bar{\boldsymbol{X}}_{i,t}^{-}\right) \qquad i = 1, ..., 2n_x. \tag{2.22}$$

The predicted measurement is calculated from the mean of $\hat{\boldsymbol{Y}}_{i,t}$ and the innovation covariance matrix, $\boldsymbol{P}_{yy,t}$, and cross covariance, $\boldsymbol{P}_{xy,t}$, are computed:

$$\hat{\boldsymbol{y}}_t = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \hat{\boldsymbol{Y}}_{i,t}, \tag{2.23}$$

$$\boldsymbol{P}_{yy,t} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \left(\hat{\boldsymbol{Y}}_{i,t} - \hat{\boldsymbol{y}}_t\right)\left(\hat{\boldsymbol{Y}}_{i,t} - \hat{\boldsymbol{y}}_t\right)^T + \boldsymbol{R}_t, \tag{2.24}$$

$$\boldsymbol{P}_{xy,t} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \left(\bar{\boldsymbol{X}}_{i,t}^{-} - \hat{\boldsymbol{x}}_t^{-}\right)\left(\hat{\boldsymbol{Y}}_{i,t} - \hat{\boldsymbol{y}}_t\right)^T. \tag{2.25}$$

Similar to the conventional Kalman filter, the *a posteriori* estimate is obtained as:

$$\boldsymbol{K}_t = \boldsymbol{P}_{xy,t} \boldsymbol{P}_{yy,t}^{-1}, \tag{2.26}$$

$$\hat{\boldsymbol{x}}_t^{+} = \hat{\boldsymbol{x}}_t^{-} + \boldsymbol{K}_t \left(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t\right), \tag{2.27}$$

$$\boldsymbol{P}_t^{+} = \boldsymbol{P}_t^{-} - \boldsymbol{K}_t \boldsymbol{P}_{yy,t} \boldsymbol{K}_t^T, \tag{2.28}$$

where $\boldsymbol{K}_t$ is the CKF gain.

To implement the square root version of this filter, a few changes have to be made to the above algorithm that are outlined in [16]. The square root version of the filter will preserve the symmetry and positive definiteness of the error covariance matrix [16]. In practice these two properties are often lost on digital computers, especially the positive definiteness of the error covariance matrix. This is especially true for

systems where very accurate measurements are utilized in comparison to the process noise such as in SLAM for a rover with an accurate LIDAR. For these reasons a square root version of the CKF is utilized in this thesis. First, a general triangularization algorithm is used to take the "square root" of any matrix. In the case of this thesis QR decomposition is used [21]. The algorithm is denoted as $\boldsymbol{S} = \mathit{Tria}(\boldsymbol{A})$, where $\boldsymbol{S}$ is a lower triangular matrix. The matrices $\boldsymbol{A}$ and $\boldsymbol{S}$ are related through the following: Let $\boldsymbol{R}$ be an upper triangular matrix obtained from QR decomposition on $\boldsymbol{A}^T$:

$$\boldsymbol{A}^T = \boldsymbol{Q}\boldsymbol{R}, \tag{2.29}$$

then $\boldsymbol{S} = \boldsymbol{R}^T$. The symbol / is used to represent the matrix right divide operator. When $\boldsymbol{X} = \boldsymbol{B}/\boldsymbol{A}$ is performed and $\boldsymbol{B}$ is not square it is the solution in the least squares sense to the under or over determined system of equations $\boldsymbol{X}\boldsymbol{A} = \boldsymbol{B}$.

To initialize the filter, Cholesky factorization [21] is done on all of the covariance matrices and produces a set of square root matrices (ie. $\boldsymbol{S}_P = \mathit{chol}(\boldsymbol{P}_0^+)$, $\boldsymbol{S}_{Q,t} = \mathit{chol}(\boldsymbol{Q}_0)$, $\boldsymbol{S}_{R,t} = \mathit{chol}(\boldsymbol{R}_0)$). Because $\boldsymbol{S}_P$ is being used directly now, the equations in (2.15)-(2.16) will be modified to:

$$\hat{\boldsymbol{X}}^+_{i,t-1} = \hat{\boldsymbol{x}}^+_{t-1} + \sqrt{n_x}(\boldsymbol{S}^+_{P,t-1})_i \qquad i = 1, ..., n_x, \tag{2.30}$$

$$\hat{\boldsymbol{X}}^+_{i+n_x,t-1} = \hat{\boldsymbol{x}}^+_{t-1} - \sqrt{n_x}(\boldsymbol{S}^+_{P,t-1})_i \qquad i = 1, ..., n_x, \tag{2.31}$$

The last change to the *a priori* steps is to compute the square root covariance where the equation in (2.19) is replaced with:

$$\boldsymbol{S}^-_{P,t} = \mathit{Tria}\left(\left[\frac{1}{\sqrt{2n_x}}\left[\hat{\boldsymbol{X}}^-_{1,t} - \hat{\boldsymbol{x}}^-_t \quad \hat{\boldsymbol{X}}^-_{2,t} - \hat{\boldsymbol{x}}^-_t ... \hat{\boldsymbol{X}}^-_{2n_x,t} - \hat{\boldsymbol{x}}^-_t\right] \qquad \boldsymbol{S}_{Q,t}\right]\right). \tag{2.32}$$

Moving on to the *a posteriori* steps, the equations in (2.20)-(2.21) are modified in the same as (2.30)-(2.31) and (2.24) is modified to:

$$\boldsymbol{S}_{yy,t} = Tria\left(\left[\frac{1}{\sqrt{2n_x}}\left[\hat{\boldsymbol{Y}}_{1,t} - \hat{\boldsymbol{y}}_t \quad \hat{\boldsymbol{Y}}_{2,t} - \hat{\boldsymbol{y}}_t \ldots \hat{\boldsymbol{Y}}_{2n_x,t} - \hat{\boldsymbol{y}}_t\right] \qquad \boldsymbol{S}_{R,t}\right]\right). \tag{2.33}$$

The Kalman gain, $\boldsymbol{K}_t$, in (2.26) is adjusted to:

$$\boldsymbol{K}_t = \left(\boldsymbol{P}_{xy,t}/\boldsymbol{S}_{yy,t}^T\right)/\boldsymbol{S}_{yy,t}. \tag{2.34}$$

Finally, as shown in [16], the *a posteriori* covariance is changed to:

$$\begin{aligned}\boldsymbol{S}_{P,t}^+ = Tria\Bigg(\Bigg[&\frac{1}{\sqrt{2n_x}}\left[\hat{\boldsymbol{X}}_{1,t}^- - \hat{\boldsymbol{x}}_t^- \quad \hat{\boldsymbol{X}}_{2,t}^- - \hat{\boldsymbol{x}}_t^- \ldots \hat{\boldsymbol{X}}_{2n_x,t}^- - \hat{\boldsymbol{x}}_t^-\right] \\ &- \boldsymbol{K}_t\frac{1}{\sqrt{2n_x}}\left[\hat{\boldsymbol{Y}}_{1,t} - \hat{\boldsymbol{y}}_t \quad \hat{\boldsymbol{Y}}_{2,t} - \hat{\boldsymbol{y}}_t \ldots \hat{\boldsymbol{Y}}_{2n_x,t} - \hat{\boldsymbol{y}}_t\right] \qquad \boldsymbol{K}_t\boldsymbol{S}_{R,t}\Bigg]\Bigg).\end{aligned} \tag{2.35}$$

2.1.3 Kinematics Modeling and Analyses of Articulated Rovers

A general approach to the kinematic modeling and analysis of articulated rovers over uneven terrain was presented by Tarokh and McDermott in [11]. Their model is derived for a full six degree of freedom motion. Individual wheel motions in contact with terrain are derived and the resulting equations are then combined to give a full equation for rover motion. Three types of kinematics are discussed: navigation, actuation and slip kinematics, with equations and applications addressed. A simulation is created for the motion of a Rocky 7 rover over uneven terrain and the results are provided.

The first section of the paper is a derivation of equations of motion that relate the motion of the rover wheels to the rover reference frame. The rover used for this analysis is the Rocky 7 rover used at JPL as a prototype for research on articulated rovers. The Rocky 7 has a familiar rocker-bogie arrangement, much like the

micro-rover *Kapvik* presented in this thesis. However, the Rocky 7 has the additional property of steerable front wheels, which adds a degree of freedom to each front wheel. The rover is divided up into sections that start at the wheel contact reference frame and form a kinematic chain up to the rover reference frame. The transformations from wheel contact reference frame to rover reference frame are presented as a series of Denavit-Hartenberg (D-H) transformations. The paper assumes that some measurement of the configuration of the rover is given, including the rocker and bogie angles, the rover's pitch and roll, and wheel encoders.

The slip of the rover's wheels is also characterized into three types: wheel rolling slip along the x-axis, wheel side slip along the y-axis and turn slip about the z-axis. Using the measurements of the rover configuration and estimating the slip and wheel contact angles with the terrain, an equation of motion is derived for each of the 6 wheels and these are combined to form the rover's motion. The Jacobian matrix that describes these equations is made explicit for each of the rover's wheels.

The next part of the paper describes a series of manipulations that can be made to the equations of motion to separate known and unknown quantities from each other so that it is possible to solve for the unknown quantities. The method is applied to navigation kinematics, slip kinematics and actuation kinematics. These methods assume perfect measurements for the "known" quantities such as pitch and roll. A test for whether a unique solution exists for the unknown quantities is explained by testing the rank of the matrix that is associated with the unknown quantities. If there is a unique solution, a least-squares method is used to solve for the unknown quantities vector. The Actuation kinematics discussed in the paper solves a similar set of equations, this time setting up some of the quantities as "desired" values, and the unknown quantities as the control inputs that are necessary to achieve the desired values. For instance given the sensed rover quantities such as pitch, roll and rocker-bogie configuration rates, what are the necessary wheel rates to achieve the desired

heading rate.

Finally, a simulation is described that was designed in Matlab. The simulation handles the transformation from rover reference frame to global reference frame by calculating the wheel contact angles that constrain the rover wheels to remain in contact with the surface it traverses holding the rover's heading and X and Y position constant. The technique is then run through the simulation over various types of terrain and different maneuvers. The same tests are also run with noise to gauge the affect of sensor noise on the model. By the end of the paper a thorough derivation of articulated rover kinematics has been presented, along with applications to various types of navigation and control and a simulation to demonstrate its effectiveness.

One of the limitations of the work is the assumption of a single point of contact for the wheels on the terrain. In reality, wheels will have multiple points of contact over rugged terrain and often sudden, discontinuous changes in the point of contact due to traversing over uneven obstacles such as rocks. The terrain may also be deformable which makes the problem considerably more complex. However, the final result provides a comprehensive way to test basic rover kinematics and a building block for further research in navigation and control of articulated rovers.

2.2 Simultaneous Localization and Mapping

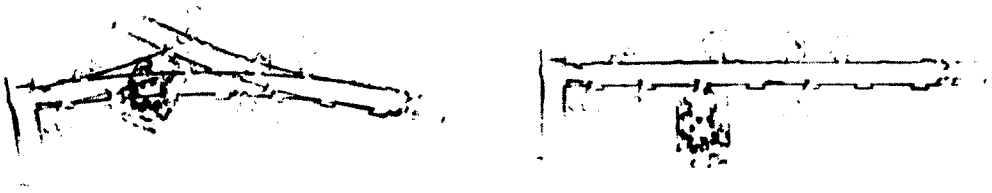
Simultaneous Localization and Mapping (SLAM) is a large topic of discussion in the robotics literature. What SLAM does is address the problem of a robot moving through an environment with no a priori "map". This is common for robots that work in environments that change as people interact with them, outdoor environments, and particularly completely unknown environments, like the surface of Mars, that a rover such as *Kapvik* is exploring. The goal for SLAM is for a robot to both estimate its location and its environment accurately, over an extended distance and time period.

However, the problem of SLAM remains to be solved and is often considered a prerequisite to true autonomous robots [22]. It would be straightforward to estimate a map if the rover's path was known. Likewise, if the map was known a priori, estimating the rover's pose would be greatly simplified. However, in the case of the applications mentioned above, neither of these aspects is known a priori. The robot in question must simultaneously estimate its pose and the map to successfully navigate and map its surroundings using the sensors available to it.

The relationship between localization and mapping comes from errors in the localization of the rover propagating into errors in the sensor measurements. As error in the path builds, the measurements of the map are corrupted because they are made on the pretence that the estimate of the rover's pose is correct. This results in error from both the localization and the sensor itself being inserted into the map. As Montemerlo and Thrun put it [22], *error in the robot's path correlates errors in the map*. This identification of the relationship between localization and mapping was first made by Smith and Cheeseman [23] in 1986.

An example of this error accumulating in a map can be seen in Figure 2.2. The robot that was used to generate this map had only wheel encoders to generate its path estimate. This is a form of dead-reckoning estimation, and as the robot drives the error in its path becomes greater and greater. Figure 2.2a shows the resulting map, which is plotted based on the estimated position of the rover at the time of the scan. It is clear that as the rover moves, the map becomes increasingly inaccurate. Figure 2.2b demonstrates the LIDAR readings plotted by a SLAM algorithm.

Just as in the Kalman framework described in Section 2.1 the SLAM framework is made up of two types of information, the *prediction* information also known as a priori information, and the *measurement* information, also known as a posteriori. In the case of a rover like *Kapvik*, the prediction step consists of a vehicle model that takes in inputs from internal measurements such as encoders, potentiometers, and



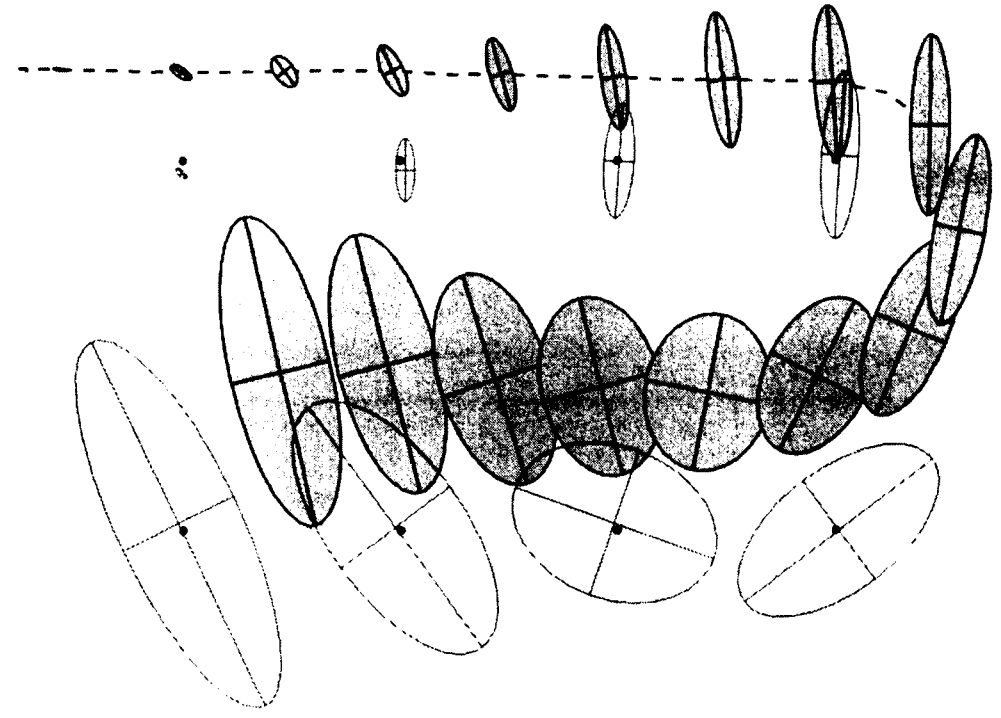
(a) Map built without correcting robot path (b) Map built using SLAM

Figure 2.2: Correlation between robot path error and map error [5].

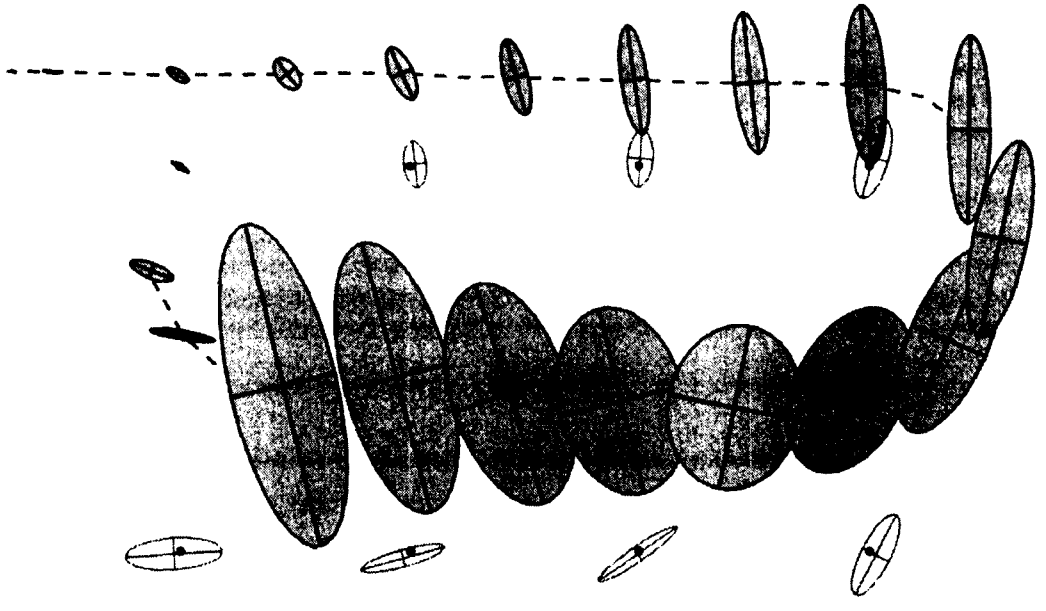
gyros and outputs the predicted pose of the rover. The measurement step consists of external measurements that are compared to the predicted measurements based on a measurement model and the predicted pose and this difference is used to update the pose and uncertainty estimates for the rover. These measurements are also known as observations.

In the same way, SLAM has these two sets of information. The control or input information combined with a realistic noise model should constrain the rover's pose relative to the previous estimate. The external measurements constrain the relative pose of the environment and the rover. Unlike the use of a Kalman filter for localization, which includes a sun sensor for an external heading measurement and accelerometers for an external pitch and roll measurement, the external measurements in SLAM are obtained from the rover's environment, in this case a LIDAR scan.

When the rover first starts driving, these constraints are not very restrictive, however as the rover observes parts of its environment repeatedly, its path and environment estimates become more and more constrained. This is shown visually by Montemerlo and Thrun in Figure 2.3. As the rover begins its drive, it continually observes new features and gains uncertainty in its pose as depicted in Figure 2.3a. The grey ellipses in the diagram show the increasing uncertainty in the rover's pose, and a corresponding increase in uncertainty is seen in the position of new features as they are observed along the path, represented by the transparent ellipses. But as



(a) Pose and map estimates before closing loop. Shaded ellipses represent pose uncertainty, unshaded ellipses represent map feature uncertainty.



(b) Pose and map estimates after closing loop. After revisiting a map feature the uncertainty of all previous map features decrease.

Figure 2.3: Motion error correlates map errors [5].

the rover begins to close a loop and measure features that are already in its map, the rove pose and map uncertainty become more and more constrained as shown in Figure 2.3b. In this diagram, the rover re-observes the first feature it observed along its path, and the uncertainty in its pose as well as the uncertainty of the features along the path decrease accordingly. It has been shown that as the number of observations and controls approaches infinity the position of all features in a map are fully correlated [5].

Solutions to SLAM that are run in real-time (also known as online solutions) typically attempt to estimate the posterior probability distribution over all possible maps, Λ_t, and robot poses s_t, given the full set of controls or inputs $\boldsymbol{u}^t$ and observations $\boldsymbol{z}^t$ up to time t. The robot's map is a set of point features defined by an x, y, and z position. This set of N features is written as $\Lambda_t = \{\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, ..., \boldsymbol{\lambda}_N\}$. The robot path is a set of robot poses up to time t, written as

$$\boldsymbol{s}^t = \{\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_t\}. \tag{2.36}$$

As a robot traverses the environment, it is measuring information that relates to its motion. In the case of *Kapvik* this comes from the potentiometers that measure the rocker bogie angles, the wheel encoders, the IMU, and the sun sensor. These sensors form an input vector that is described in more detail in Section 3.2.2 but for now will be denoted $\boldsymbol{u}$. Following the notation presented for the pose, the input at time t is denoted $\boldsymbol{u}_t$ and the set of all inputs is written as

$$\boldsymbol{u}^t = \{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_t\}. \tag{2.37}$$

Similarly, the set of all external measurements, in *Kapvik's* case these are LIDAR

measurements, are written as

$$\boldsymbol{z}^t = \{\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_t\}\,. \tag{2.38}$$

Together, this information forms the posterior distribution which is written as:

$$p(\boldsymbol{s}_t, \boldsymbol{\Lambda}_t | \boldsymbol{z}^t, \boldsymbol{u}^t), \tag{2.39}$$

which represents the probability density function of $\boldsymbol{s}_t, \boldsymbol{\Lambda}_t$ given $\boldsymbol{z}^t, \boldsymbol{u}^t$. There are two key aspects of estimating the pose of the rover and position of features in the environment. The first is the simple fact that considering a distribution of possibilities makes the algorithms involved more robust. By considering multiple solutions, bad measurements will less likely lead to bad estimates. Secondly, by considering the uncertainty in different measurements in the problem they can be weighted with respect to each other. For instance, the uncertainty in *Kapvik's* gyroscopes versus the uncertainty in its potentiometers will result in them being weighted differently in a Kalman filter that estimates the orientation of the rover (done through the calculation and application of the Kalman gain, $\boldsymbol{K}$, first shown in Section 2.1.1). Another example is the map, where some areas are more uncertain compared to well known parts of the map. These are the key aspects that must be considered when comparing other 6D solutions to SLAM that have been proposed. Many of the solutions that have been presented for online operation do not estimate the posterior distribution, and simply estimate the most likely solution using methods such as Iterative Closest Point (ICP) [4, 10, 24–27] to compare the changes in subsequent scans. These methods may seem to work, but they are not robust to sensor errors because they do not take into account the posterior distribution and uncertainty in the solution. Based on these reasons, this thesis implements a posterior distribution

estimation version of SLAM.

How the map, Λ_t, is described is also of importance and will be fully discussed in Section 2.3. The SLAM posterior described in (2.39) assumes no known correspondence between the measurements z^t and the map Λ_t. This correspondence is also known as data association, which refers to the rover's ability to discern which measurements belong to the current map, and which should be added to the map as new features. This thesis describes in Section 2.4 the problem of data association and methods in which it can be solved, but before this, as a baseline for the SLAM algorithm presented in Section 3.3.2, the algorithm assumes a posterior distribution of known data associations, written as:

$$p(s_t, \Lambda_t | z^t, u^t, n^t), \tag{2.40}$$

where n_t represents a single feature identity measured at time t for each of the measurements, z^t, and n^t represents all feature identities measured up to time t. The Bayes Filter for SLAM [5] can be used to compute the distribution in (2.40) which is written as:

$$p(s_t, \Lambda_t | z^t, u^t, n^t) = \eta p(z_t | s_t, \Lambda_t, n_t) \int p(s_t | s_{t-1}, u_t) p(s_{t-1}, \Lambda_t | z^{t-1}, u^{t-1}, n^{t-1}) \mathrm{d}s_{t-1}, \tag{2.41}$$

where η is the denominator in Bayes rule that acts as a normalizing constant, in this case $\eta = p(z^t, u^t, n^t)$. The result is a recursive formula for calculating the posterior at time t given by the motion model, measurement model, and posterior at time t - 1. The integral in (2.41) cannot usually be evaluated in closed form, but by assuming a form of the posterior, techniques like the Kalman filter and particle filter can be used as approximations to the Bayes filter [5].

The EKF method for SLAM will be presented in Section 2.2.1 and the FastSLAM

algorithm which makes use of a particle filter technique will be presented in Section 2.2.2. Building on the FastSLAM algorithm one of the contributions of this thesis is using a nonlinear Kalman filter such as the Cubature Kalman Filter (CKF) within the FastSLAM algorithm which will be discussed in Section 3.3.2.

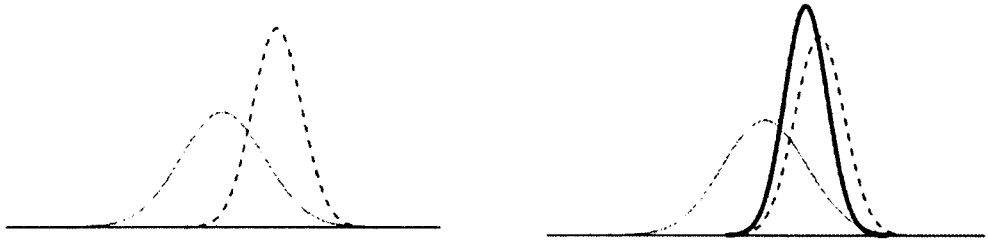
2.2.1 EKF-SLAM

Smith and Cheeseman introduced the SLAM problem to the literature in [23] and presented one of the most widely used methods to solve it, an EKF version of SLAM. In this version of SLAM an EKF is used to estimate the posterior over the robot's pose and map. This simple EKF-SLAM approximates the posterior with a high-dimensional Gaussian that includes the robot's pose and the position of all features in the map as the states. This provides a covariance matrix that includes on its off-diagonal terms the correlations between all pairs of states. This is important because the SLAM problem is characterized by correlated errors.

To describe the SLAM posterior as described in (2.41) using the EKF outlined above as an approximation of the Bayes filter, the posterior is represented as a high dimensional, multivariate Gaussian with a mean of, $\boldsymbol{\mu}_t$ and a covariance matrix $\boldsymbol{P}_t$. The mean describes the likely robot pose and map feature positions and the covariance matrix represents the correlations between pairs of these state variables. Given this, the left hand side of (2.41) is rewritten as:

$$p(\boldsymbol{s}_t, \boldsymbol{\Lambda}_t | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t) = \mathcal{N}(\boldsymbol{s}_t, \boldsymbol{\Lambda}_t; \boldsymbol{\mu}_t, \boldsymbol{P}_t), \tag{2.42}$$

$$\boldsymbol{\mu}_t = \left[\boldsymbol{\mu}_{\boldsymbol{s}_t}^T, \boldsymbol{\mu}_{\lambda_{1,t}}^T, ..., \boldsymbol{\mu}_{\lambda_{N,t}}^T\right]^T, \tag{2.43}$$



(a) Prior estimate (solid line) and new measurement (dashed line).

(b) Estimate after combining with new measurement (thick line)

Figure 2.4: Kalman Filter [5].

$$\boldsymbol{P}_t = \begin{bmatrix} \boldsymbol{P}_{s_t,t} & \boldsymbol{P}_{s_t\lambda_1,t} & \cdots & \boldsymbol{P}_{s_t\lambda_N,t} \\ \boldsymbol{P}_{\lambda_1 s_t,t} & \boldsymbol{P}_{\lambda_1,t} & \cdots & \boldsymbol{P}_{\lambda_1\lambda_N,t} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{P}_{\lambda_N s_t,t} & \boldsymbol{P}_{\lambda_N\lambda_1,t} & \cdots & \boldsymbol{P}_{\lambda_N,t} \end{bmatrix}. \tag{2.44}$$

Figure 2.4 is a visual depiction of how a Kalman filter estimates a single dimension of a feature's position in the robot's map. Figure 2.4a shows the estimate at time t -1 which is represented by a solid line, and a new, noisy measurement of that feature. What the Kalman filter does is describe the optimal method of combining these two Gaussian distributions to form the new estimate at time step t, as shown in Figure 2.4b. While the Kalman filter is the optimal estimator for linear systems [14], the measurement and motion models are rarely linear in real life, so an EKF is used to approximate the motion and measurement models with linearizations taken around the mean of the system at time step t, μ_t. Using Algorithm 2.1, the EKF update

equations for SLAM can be written as:

$$\begin{aligned}
\boldsymbol{P}_t^- &= \boldsymbol{F}_{t-1}\boldsymbol{P}_{t-1}^+\boldsymbol{F}_{t-1}^T + \boldsymbol{L}_{t-1}\boldsymbol{Q}_{t-1}\boldsymbol{L}_{t-1}^T, \\
\boldsymbol{K}_t &= \boldsymbol{P}_t^-\boldsymbol{H}_t^T(\boldsymbol{H}_t\boldsymbol{P}_t^-\boldsymbol{H}_t^T + \boldsymbol{M}_t\boldsymbol{R}_t\boldsymbol{M}_t^T)^{-1}, \\
\hat{\boldsymbol{\mu}}_t^- &= f_{t-1}(\hat{\boldsymbol{\mu}}_{t-1}^+, \boldsymbol{u}_{t-1}, \boldsymbol{0}), \\
\hat{\boldsymbol{\mu}}_t^+ &= \hat{\boldsymbol{\mu}}_t^- + \boldsymbol{K}_t(\boldsymbol{y}_t - h_t(\hat{\boldsymbol{\mu}}_t^-, n_t, 0)), \\
\boldsymbol{P}_t^+ &= (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t)\boldsymbol{P}_t^-,
\end{aligned} \tag{2.45}$$

where f and h represent the nonlinear motion and measurement models.

The obvious problem that is encountered with this approach is the covariance matrix and state vector grow larger as more features are added to a map. In large, outdoor environments, such as *Kapvik* will be exploring, this will result in both memory usage and computation time to scale quadratically with the number of features in a map [28]. The quadratic scaling is due to the covariance matrix and the operations on that matrix in the Kalman filter. In a simple, 2D world, the state vector will include $2N + 3$ elements for the X and Y position of each feature and the pose of the rover. The covariance matrix will then have $(2N + 3)^2$ elements that encode the correlations between each state pair. Most applications of EKF-SLAM do not actually set up the problem in this way, and instead build up a series of sub-maps, limiting the number of correlations between features in the overall map. This can be done because the measurements and controls that a robot has accumulated at time step t constrain only a small part of the map. For example, two features that are very far apart will usually be weakly correlated. There are a number of approaches in the literature that take advantage of these properties of EKF-SLAM that allow the whole map to be constructed out of smaller sub-maps.

The less obvious problem with these methods is related to data association. Data association is the ability of a robot to assign measurements of its environment to

features already in the map, or identify measurements as unique and add them to the map. Methods of Data association are discussed in Section 2.4, however the problem of data association is difficult, and is itself based on statistical probability and includes uncertainty. There is always a chance that a wrong data association will be made, where a new measurement is associated to a part of the map when in actuality it was generated by a new feature, and vice versa. Because the EKF in EKF-SLAM does not encode the uncertainties in data association, if an error is made, it will not be discarded. The filter will diverge if a large number of wrong data associations are made, and is a well known cause of failure for EKF-SLAM [29]. In the case of diverse, outdoor environments such as the one *Kapvik* will encounter, wrong data associations are likely and using an EKF-SLAM method would not be appropriate. One solution that has been proposed is multiple copies of an EKF, containing multiple probable data association hypotheses. This solution could work, but it requires that much more computational and memory capabilities. A more elegant solution was presented by Montemerlo and Thrun in [22,30] where the robot path and map are decoupled using a particle filter. This approach forms the basis of the solution this thesis presents for the SLAM method used on *Kapvik*, and is presented in the following section.

2.2.2 FastSLAM

While Kalman filters represent probability distributions using a multivariate Gaussian, particle filters represent them with a finite number of sample states, known as "particles". Regions of high probability contain a large number of particles, and regions of low probability contain few or none. With enough particles, the particle filter can approximate complex, multi-modal distributions. As the number of particles approaches infinity, the probability distribution can be reconstructed completely [31]. Using a particle filter, the Bayes filter update equation is used with a simple particle sampling method.

(a) Pose is very uncertain with few measurements.

(b) Pose estimate converges on an estimate after several measurements.

Figure 2.5: Particle filter for pose estimation [5].

Particle filters have been commonly used in localization problems. One common method, called Monte Carlo Localization [32], is used to represent the distribution of robot poses in a known map. No information is given to the robot about its pose, and when localization begins the particles are distributed with uniform probability as shown inFigure 2.5a across the entire map. After including a number of inputs and measurements from the robot's environment, the particle filter's estimated posterior converges to a unimodal distribution shown in Figure 2.5b.

The ability of particle filters to estimate multi-modal distributions of nonlinear systems makes it a very robust alternative to non-linear Kalman filters. However, the number of particles needed can in some cases scale exponentially with the number of states being estimated. In most cases this limits particle filters to low dimensional or offline problems. This would suggest that particle filters would be particularly ill-suited to the online SLAM problem, which has potentially millions of dimensions as the map grows with new measurements.

Nevertheless, FastSLAM makes use of the particle filter in an ingenious way that

avoids this potential pitfall while at the same time harnessing the robust features of the particle filter. By factoring the SLAM problem into a set of independent map feature estimation problems that are conditioned on an estimate of the robot's path, FastSLAM's particle filter operates only on the rover's path, which is of low dimensionality.

Most approaches to SLAM follow the approach outlined in the beginning of this section. The posterior is estimated over the map and the robot pose.

$$p(\boldsymbol{s}_t, \boldsymbol{\Lambda}_t | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t), \tag{2.46}$$

whereas FastSLAM estimates the posterior over the map and robot path.

$$p(\boldsymbol{s}^t, \boldsymbol{\Lambda}_t | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t). \tag{2.47}$$

This allows the SLAM posterior to be factored into a product of simpler terms. This is important because as was discussed at the beginning of this section, if the true path of the robot is known, then the position of each feature in the map is conditionally independent of any other feature. In other words, a measurement of the position of one feature provides no new information about the position of another. This means that the SLAM problem can be factored in the case of a known robot path into simpler terms, and is written as:

$$p(\boldsymbol{s}^t, \boldsymbol{\Lambda} | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t) = \underbrace{p(\boldsymbol{s}^t | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t)}_{\text{path posterior}} \underbrace{\prod_{n=1}^{N} p(\boldsymbol{\lambda}_n | \boldsymbol{s}^t, \boldsymbol{z}^t, \boldsymbol{u}^t, n^t)}_{\text{feature estimators}}. \tag{2.48}$$

This factorization states that the posterior can be separated into a product of the robot's path posterior and N landmark posteriors conditioned on the path. The factorization is derived from the SLAM posterior by first rewriting the posterior as:

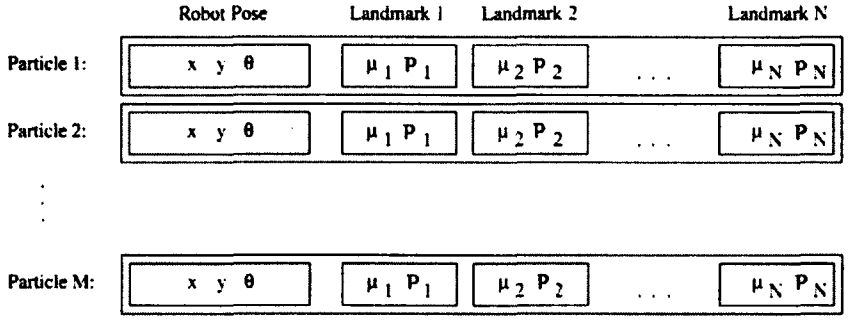


Figure 2.6: For M particles there are N independent EKFs [5].

$$p(\boldsymbol{s}^t, \boldsymbol{\Lambda} | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t) = p(\boldsymbol{s}^t | \boldsymbol{z}^t, \boldsymbol{u}^t, n^t) p(\Lambda | \boldsymbol{s}^t, \boldsymbol{z}^t, \boldsymbol{u}^t, n^t). \tag{2.49}$$

To derive the equation shown in (2.48), the right most term in (2.49) can be shown to follow:

$$p(\boldsymbol{\Lambda} | \boldsymbol{s}^t, \boldsymbol{z}^t, \boldsymbol{u}^t, n^t) = \prod_{n=1}^{N} p(\boldsymbol{\lambda}_n | \boldsymbol{s}^t, \boldsymbol{z}^t, \boldsymbol{u}^t, n^t). \tag{2.50}$$

FastSLAM makes use of this factored representation by maintaining $N + 1$ filters, each of which estimates a term in (2.48). The first term, representing the path posterior, is estimated using a particle filter. The rest of the feature estimators are estimated using EKFs, with each EKF tracking a single feature position which is low dimensional and does not grow as the map gets larger. The feature EKFs are conditioned on the robot paths generated by the particle filter, with each particle having its own set of EKFs. In total there are $N \times M$ EKFs where M is the number of particles as depicted in Figure 2.6.

There are two versions of the FastSLAM algorithm that have been presented by the creators, Montemerlo and Thrun. The FastSLAM 1.0 algorithm forms the basis of both algorithms, with FastSLAM 2.0 handling the environment measurements in a slightly different manner that allows for a more accurate estimate of the posterior

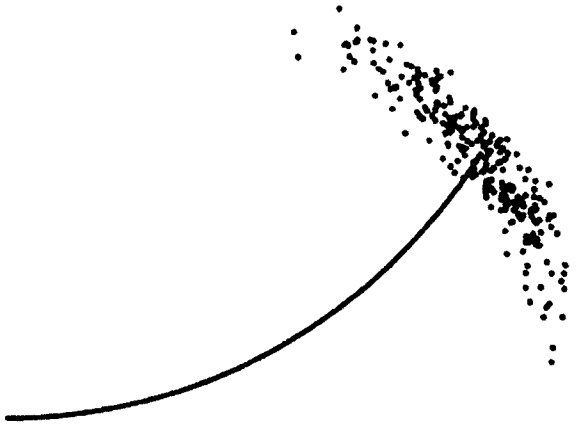


Figure 2.7: Samples from the probabilistic motion model [5].

distribution. The rest of this section includes an explanation of both algorithms with the FastSLAM 2.0 discussion building off of the first.

FastSLAM 1.0 [22]

With an understanding of the basic SLAM problem, a derivation of the FastSLAM factorization, and knowledge of a particle filter, the FastSLAM 1.0 algorithm can be outlined using the presentation from [22].

At each new time step, t, the posterior particle set is calculated from the one at time t-1 by generating a new particle set. This new particle set includes the last control, $\boldsymbol{u}_t$, and a measurement $\boldsymbol{z}_t$. With each measurement there is a corresponding data association, n_t. To perform this update there are four basic steps which are outlined as follows:

Step 1: Sample a new robot pose

Each of the particle sets being calculated is found from the previous set of particles, and a new observation and control. Therefore, the full posterior distribution is not available to sample a new set of particles from. Instead a proposal distribution is used,

and the difference between this and the true distribution is corrected in the particle weighting step. The "guesses" that are being sampled are from the probabilistic motion model, an example of which is shown in Figure 2.7, and is described by:

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}), \tag{2.51}$$

where $s_t^{[m]}$ is the m-th particle's pose and the $\sim$ symbol represents proportionality. A particle is drawn from a distribution through sampling from the distribution, and the terms are used interchangeably throughout this thesis. This is added to a temporary set of particles along with the path, $s^{t-1,[m]}$. These new particles are drawn from the proposal distribution:

$$p(s^t|z^{t-1}, u^t, n^{t-1}). \tag{2.52}$$

This step has a constant computation cost for every particle, as it does not depend on the size of the map. It is also dependent on only the last pose estimate, and is therefore independent of time index t which allows all other previous pose estimates to be disregarded.

Step 2: Feature location estimation

Assuming that the data associations are known, the features that have been measured are updated using an EKF. Because the estimator is conditioned on a fixed robot path, the only estimate will be of the feature's position. The measurement noise is assumed to be Gaussian with covariance $\mathbf{R}_L$ of zero-mean. Because the features are assumed to be not moving, the process model is static and there is no process noise. A prediction of the feature position is obtained through the measurement model:

$$\hat{z} = h(s_t^{[m]}, \mu_{n_t,t-1}), \tag{2.53}$$

where $\mu_{n_t,t-1}$ represents the mean of the Gaussian representing the n^{th} feature's location. To form the measurement update functions of the EKF the Jacobian of the measurement model is taken to linearize the motion model:

$$G_{\lambda,n_t} = \left.\frac{\partial h(s_t, \lambda_{n_t})}{\partial \lambda_{n_t}}\right|_{s_t^{[m]}, \lambda_{n_t}=\mu_{n_t,t-1}^{[m]}}. \tag{2.54}$$

The Kalman gain is then calculated as

$$K_t = P_{n_t,t-1}^{[m]} G_{\lambda,n_t}^T (G_{\lambda,n_t} P_{n_t,t-1}^{[m]} G_{\lambda,n_t}^T + R_L)^{-1}. \tag{2.55}$$

Finally, the feature positions are updated as

$$\mu_{n_t,t}^{[m]} = \mu_{n_t,t-1}^{[m]} + K_t(z_t - \hat{z}_t), \tag{2.56}$$

$$P_{n_t,t}^{[m]} = (I - K_t G_{\lambda,n_t} P_{n_t,t-1}^{[m]}). \tag{2.57}$$

Updating each feature filter is a constant-time operation due to the constant size of the feature state vector and covariance. If the measurement has been determined through data association to come from a new feature, a feature is added to the map through an inversion of the measurement model

$$\mu_{n_t,t}^{[m]} = h^{-1}\left(s_t^{[m]}, z_t\right), \tag{2.58}$$

$$P_{n_t,t}^{[m]} = \left(G_{\lambda,n_t}^T R_L^{-1} G_{\lambda,n_t}\right)^{-1}. \tag{2.59}$$

Step 3: Calculate Importance Weights

The particles that are represented by $s_t^{[m]}$ are from the *proposal distribution.* This is according to the previous time step's measurement of the robot's environment, the corresponding data associations, and the latest input. The true posterior distribution

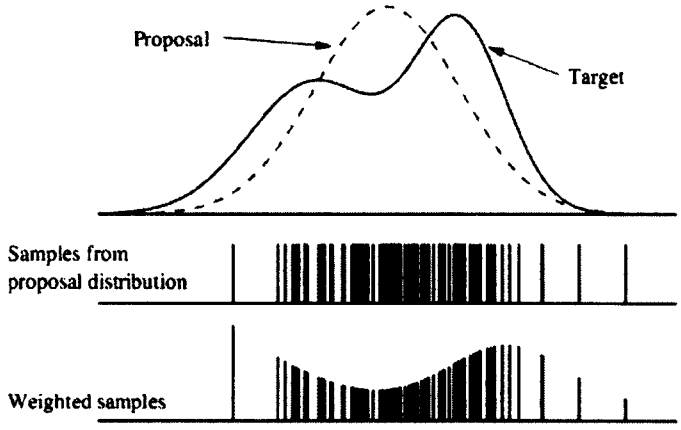


Figure 2.8: Samples can not be drawn from the target distribution (solid line) so they are sampled from the proposal distribution (dotted line). These samples are drawn and weighted proportional to their importance weights [5].

incorporates the latest measurement from the current time step, t, and therefore the proposal distribution does not match the desired posterior distribution. To correct for this difference something called importance sampling is done as depicted in Figure 2.8.

Because the sampling is done on a proposal distribution (dotted line in Figure 2.8) the particles must be weighted relative to the difference with the true posterior distribution (solid line). This means particles in a region where the true distribution is greater than the proposal distribution will be weighted higher whereas particles in regions where the true distribution is lower will be weighted lower. In other words, particles in the former region will be chosen more and vice versa. For an infinite number of particles, this produces particles distributed according to the true posterior distribution exactly.

To calculate the weights, the ratio of the target and the proposal distribution is computed

$$
\begin{aligned}
w_t^{[m]} &= \frac{target\ distribution}{proposal\ distribution}, \\
&= \frac{p(\boldsymbol{s}^{t,[m]}|\boldsymbol{z}^t, \boldsymbol{u}^t, n^t)}{p(\boldsymbol{s}^{t,[m]}|\boldsymbol{z}^{t-1}, \boldsymbol{u}^t, n^{t-1})}.
\end{aligned}
\tag{2.60}
$$

Which through use of Bayes Rule [5] gives:

$$
w_t^{[m]} = \frac{1}{\sqrt{|2\pi \boldsymbol{Z}_{n_t,t}|}} \exp\{-\frac{1}{2}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{n_t,t})^T \boldsymbol{Z}_{n_t,t}^{-1}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{n_t,t})\}, \tag{2.61}
$$

where $\boldsymbol{Z}_{n_t,t}$ is the innovation covariance defined by,

$$
\boldsymbol{Z}_{n_t,t} = \boldsymbol{G}_{\lambda,n_t} \boldsymbol{P}_{n_t,t-1}^{[m]} \boldsymbol{G}_{\lambda,n_t}^T + \boldsymbol{R}_L. \tag{2.62}
$$

Once again this calculation is constant-time operation, depending only on the dimensionality of the measurement which in *Kapvik's* case is three (range, bearing and tilt of the LIDAR).

Step 4: Resample the particles

With the temporary particle set that has been assigned weights, a new set is sampled with probabilities in proportion to the weights. There are many different types of resampling techniques, a common one [14] visually depicted in Figure 2.9. The basic concept is a random number uniformly generated between 0 and 1 is sampled, and the weights for the temporary particle set are normalized to unity. For each new particle, starting at index 1, the weights are then added up, starting at the temporary set's particle 1. When the cumulative sum of the weights is greater than the random number generated, the temporary particle that contributed to the cumulative sum one step before is resampled. For instance, if the first particle in the resampled set is being determined, and the cumulative sum is greater than the random number at particle m, particle $m-1$ is resampled as the first particle in the resampled set.

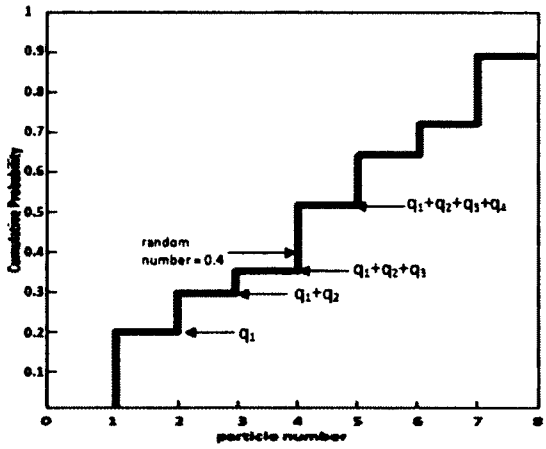


Figure 2.9: Visual representation of one type of particle filter resampling.

FastSLAM 2.0

Sampling over the robot's path has been shown to lead to good scaling and robust data association properties compared to EKF-SLAM. However, there are some drawbacks. For instance, if the measurement sensor is *too* accurate (compared to the motion noise of the robot), the algorithm may become unstable due to a poor match between the proposal and posterior distributions. To account for this, FastSLAM 2.0 was developed by Montermerlo and Thrun [30]. FastSLAM 2.0 incorporates the most recent measurement of the robot's environment into the proposal distribution, and not just the importance weights. This allows a better match between the proposal and posterior distributions and the resulting algorithm is superior to FastSLAM 1.0 in most ways, with the main disadvantage being the complexity that FastSLAM 2.0 adds to the algorithm.

One of the aspects FastSLAM 2.0 helps improve is in the resampling of the particle filter. In particle filtering, there are two key steps: sampling from the proposal distribution and importance resampling. The proposal distribution generates new path trajectories, and resampling eliminates unlikely trajectories allowing a constant number of particles. Particle filters that maintain a diverse set of trajectories will in

Figure 2.10: Mismatch between the proposal and posterior distributions [5].

general lead to better estimation accuracy. The challenge comes in the resampling process, where it is important to eliminate the bad trajectories but not to eliminate in such a way that the diversity of the particle set becomes very low. As many measurements and inputs are sent through the filter the particles will inevitably share a common ancestor and hence, a common history that precedes it. One performance indicator of a particle filter is how far back this common ancestor is.

The more diverse the particle set, the more FastSLAM is able to revise the path of the robot when a new measurement is taken. The farther back in time the common ancestor exists, the farther back FastSLAM can re-weight past hypotheses. This is a crucial performance indicator of FastSLAM as it determines how large a loop can be closed.

The relationship between the noise of the robot motion and the robot measurements on the performance of the particle filter is visually depicted in Figure 2.10. The motion model propagates the particles over a large region, but when a measurement is made the particles are constrained by a much smaller area depicted by the ellipse. Only a small number of particles receive significant weights when this measurement is incorporated in the resampling step. In the next time step, only a small part of the original particle set survives and diversity has been reduced. As more measurements

are taken, fewer unique samples will survive and at some point the particle filter will diverge. This problem is known as sample impoverishment.

FastSLAM 2.0 incorporates the latest measurement into the proposal distribution through linearizing the measurement model in the same way as the EKF-SLAM solutions. The significant accuracy increases from FastSLAM 2.0 are important as most real world systems, like *Kapvik*, have large input noises due to phenomenon such as slip, but relatively accurate sensors such as LIDAR, which is the root cause of sample impoverishment.

At its most basic level, FastSLAM 2.0 is different from FastSLAM 1.0 due to the distribution it samples $s_t^{[m]}$ from. FastSLAM 1.0 samples a pose as shown in (2.51) but FastSLAM 2.0 instead samples from a distribution that includes the latest measurements of the robot's environment, as well as the data associations with those measurements. It is written as:

$$\boldsymbol{s}_t^{[m]} \sim p(\boldsymbol{s}_t | \boldsymbol{u}_t, \boldsymbol{s}_{t-1}^{[m]}, \boldsymbol{z}^t, n^t). \tag{2.63}$$

The key change is through linearizations of the motion and measurement model similar to those used in EKF-SLAM, the measurements can be incorporated to update the proposal distribution. To begin, the particle set is propagated through the motion model:

$$\hat{\boldsymbol{s}}_t = f(\boldsymbol{s}_{t-1}^{[m]}, \boldsymbol{u}_t). \tag{2.64}$$

Next, the linearized measurement model calculations are done, written as

$$\begin{aligned} \hat{\boldsymbol{z}}_{t,n} &= h(\hat{\boldsymbol{s}}_t, \boldsymbol{\mu}_{n,t-1}^{[m]}), \\ \boldsymbol{G}_{s,n} &= \nabla_{\boldsymbol{s}_t} h(\boldsymbol{s}_t, \boldsymbol{\lambda}_n)|_{\boldsymbol{s}_t = \hat{\boldsymbol{s}}_t; \boldsymbol{\lambda}_n = \boldsymbol{\mu}_{n,t-1}^{[m]}}. \end{aligned} \tag{2.65}$$

Where $\hat{\boldsymbol{z}}_{t,n}$ is the predicted measurement of the n^{th} feature in the map at $t-1$ and $\boldsymbol{G}_{s,n}$ is the Jacobian of the measurement model with respect to $\boldsymbol{s}_t$. Just as in FastSLAM 1.0, the innovation covariance matrix is written as

$$\boldsymbol{Z}_{t,n} = \boldsymbol{R}_L + \boldsymbol{G}_{\lambda,n}\boldsymbol{P}^{[m]}_{n,t-1}\boldsymbol{G}^T_{\lambda,n}. \tag{2.66}$$

Using the equations from (2.64)-(2.66) and the latest measurement, $\boldsymbol{z}_t$, the proposal distribution's covariance matrix and mean are computed as

$$\begin{aligned} \boldsymbol{P}_{s_t,n} &= \left[\boldsymbol{G}^T_{s,n}\boldsymbol{Z}^{-1}_{t,n}\boldsymbol{G}_{s,n} + \boldsymbol{Q}^{-1}\right]^{-1}, \\ \boldsymbol{\mu}_{s_t,n} &= \hat{\boldsymbol{s}}_t + \boldsymbol{P}^{[m]}_{s_t,n}\boldsymbol{G}^T_{s,n}\boldsymbol{Z}^{-1}_{t,n}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,n}). \end{aligned} \tag{2.67}$$

A new set of poses is then sampled from this proposal distribution, written as

$$\boldsymbol{s}^{[m]}_{t,n} \sim \mathcal{N}\left(\boldsymbol{s}_t; \boldsymbol{\mu}_{s_t,n}, \boldsymbol{P}_{s_t,n}\right). \tag{2.68}$$

With the innovation covariance matrix, $\boldsymbol{Z}_{t,n}$ and the measurement model h, the likelihood of a measurement being attributed to a feature in the map can be calculated. This likelihood can then be used to determine which feature, if any, the measurement should be associated to. The set of calculations in (2.64)-(2.68) are repeated for each feature in the map, with a final calculation of the likelihood:

$$p^{[m]}_n = |2\pi\boldsymbol{Z}_{t,n}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{z}_t - h(\boldsymbol{s}^{[m]}_{t,n}, \boldsymbol{\mu}_{n,t-1}))^T\boldsymbol{Z}^{-1}_{t,n}(\boldsymbol{z}_t - h(\boldsymbol{s}^{[m]}_{t,n}, \boldsymbol{\mu}_{n,t-1}))\right\}. \tag{2.69}$$

The pose, $\boldsymbol{s}^{[m]}_{t,n}$, is then carried forward to be used in the next time step. When there are multiple measurements, indexed with j, each must be compared to the last map estimate. The likelihood for each feature in the old map belonging to each measurement in the latest set of measurements is computed. First, $\hat{\boldsymbol{s}}_t$ is used

to calculate the predicted measurement, innovation covariance and pose uncertainty from (2.65)-(2.67). After this is done, for each of the J new measurements, a unique $\boldsymbol{\mu}_{s_t,n,j}$ is calculated from (2.67). Using this mean a new pose is sampled for each measurement using (2.68) and a likelihood is computed from (2.69). The end result is a $N \times J$ matrix, $p_{n,j}$, that contains the likelihood of each measurement belong to each feature in the map at $t-1$. It will be shown in Section 2.4 how this information can be used to associate the measurements, but for now the assumption is the measurements have all been associated correctly using this information.

Due to this new proposal distribution, the importance weights, w_t, for each particle must also be calculated differently. Just as in FastSLAM 1.0 they are calculated using Bayes Rule and are defined by

$$w_t^{[m]} = \left|2\pi \boldsymbol{L}_t^{[m]}\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,\hat{n}_t})^T \boldsymbol{L}_t^{-1}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,\hat{n}_t})\right\}, \tag{2.70}$$

where

$$\boldsymbol{L}_t^{[m]} = \boldsymbol{G}_{s,\hat{n}_t} \boldsymbol{Q} \boldsymbol{G}_{s,\hat{n}_t}^T + \boldsymbol{G}_{\lambda,\hat{n}_t} \boldsymbol{P}_{n_t,t-1}^{[m]} \boldsymbol{G}_{\lambda,\hat{n}_t}^T + \boldsymbol{R}_L. \tag{2.71}$$

An overall summary of FastSLAM 2.0 is provided in Algorithm 2.2.

2.2.3 Unscented FastSLAM

An algorithm based on applying the unscented transformation to FastSLAM was presented by Kim *et al.* in [33] which they deemed UFastSLAM. Important drawbacks of the original FastSLAM algorithm include the computation of Jacobian matrices and the linearization of nonlinear functions. They argue that depending on the complexity of the application, these limitations can lead to an inconsistent filter. Their solution is to use the unscented transform in two respects and are presented as follows.

Algorithm 2.2 FastSLAM 2.0

for $m = 1 \rightarrow M$ **do** {Loop over all particles}
  $\boldsymbol{s}_{t-1}^{[m]}, \boldsymbol{P}_{s_t,t-1}^{[m]}, \left\{\mu_{1,t-1}^{k}, P_{1,t-1}^{k}, ..., \mu_{N_{t-1}^{k},t-1}^{k}, P_{N_{t-1}^{k},t-1}^{k}\right\}$ from $\boldsymbol{S}_{t-1}$
  $\hat{\boldsymbol{s}}_t = f\left(\boldsymbol{s}_{t-1}^{[m]}, \boldsymbol{u}_t\right)$
  for $n = 1 \rightarrow N_{t-1}^{k}$ **do** {Compute likelihood of data associations}
    $\hat{\boldsymbol{z}}_{t,n} = h\left(\hat{\boldsymbol{s}}_t, \boldsymbol{\mu}_{n,t-1}^{[m]}\right)$
    $\boldsymbol{G}_{s,n} = \nabla_{\boldsymbol{s}_t} h\left(\boldsymbol{s}_t, \boldsymbol{\lambda}_n\right)\big|_{s_t=\hat{s}_t;\lambda_n=\mu_{n,t-1}^{[m]}}$
    $\boldsymbol{G}_{\lambda,n} = \nabla_{\lambda_n} h\left(\boldsymbol{s}_t, \boldsymbol{\lambda}_n\right)\big|_{s_t=\hat{s}_t;\lambda_n=\mu_{n,t-1}^{[m]}}$
    $\boldsymbol{Z}_{t,n} = \boldsymbol{R}_t + \boldsymbol{G}_{\lambda,n}\boldsymbol{P}_{n,t-1}^{[m]}\boldsymbol{G}_{\lambda,n}^{T}$
    $\boldsymbol{P}_{s_t,n} = \left[\boldsymbol{G}_{s,n}^{T}\boldsymbol{Z}_{t,n}^{-1}\boldsymbol{G}_{s,n} + \boldsymbol{Q}_t^{-1}\right]^{-1}$
    $\boldsymbol{\mu}_{s_t,n} = \hat{\boldsymbol{s}}_t + \boldsymbol{P}_{s_t,n}^{[m]}\boldsymbol{G}_{s,n}^{T}\boldsymbol{Z}_{t,n}^{-1}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,n})$
    $\boldsymbol{s}_{t,n}^{[m]} \sim \mathcal{N}\left(\boldsymbol{s}_t; \boldsymbol{\mu}_{s_t,n}, \boldsymbol{P}_{s_t,n}\right)$
    $p_n^{[m]} = |2\pi\boldsymbol{Z}_{t,n}|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{z}_t - h(\boldsymbol{s}_{t,n}^{[m]}, \boldsymbol{\mu}_{n,t-1}))^{T}\boldsymbol{Z}_{t,n}^{-1}(\boldsymbol{z}_t - h(\boldsymbol{s}_{t,n}^{[m]}, \boldsymbol{\mu}_{n,t-1}))\right\}$
  end for
  $p_{N_{t-1}^{[m]}+1}^{[m]} = p_0$
  $\hat{n}_t$ data association chosen from maximum value of p_n
  if $\hat{n}_t = N_{t-1}^{[m]} + 1$ **then**
    $N_t^{[m]} = N_{t-1}^{[m]} + 1$
    $\boldsymbol{G}_{\lambda,\hat{n}_t} = \nabla_{\lambda_n} h\left(\boldsymbol{s}_t^{[m]}, \boldsymbol{\lambda}_n\right)\Big|_{s_t=s_t^{[m]};\lambda_n=\mu_{n,t-1}^{[m]}}$
    $\boldsymbol{\mu}_{\hat{n}_t,t}^{[m]} = h^{-1}\left(\boldsymbol{s}_t^{[m]}, \boldsymbol{z}_t\right) \qquad \boldsymbol{P}_{\hat{n}_t,t}^{[m]} = \left(\boldsymbol{G}_{\lambda,\hat{n}_t}^{T}\boldsymbol{R}_t^{-1}\boldsymbol{G}_{\lambda,\hat{n}_t}\right)^{-1}$
    $w_t^{[m]} = p_0$
  else
    $\boldsymbol{s}_t^{[m]} = \boldsymbol{s}_{t,\hat{n}_t}^{[m]} \qquad N_t^{[m]} = N_{t-1}^{[m]}$
    $\boldsymbol{K}_{\hat{n}_t,t} = \boldsymbol{P}_{\hat{n}_t,t-1}\boldsymbol{G}_{\lambda,\hat{n}_t}^{T}\boldsymbol{Z}_{\hat{n}_t,t}^{-1}$
    $\boldsymbol{\mu}_{\hat{n}_t,t-1}^{[m]} + \boldsymbol{K}_{\hat{n}_t,t}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{\hat{n}_t}) \qquad \boldsymbol{P}_{\hat{n}_t,t}^{[m]} = (\boldsymbol{I} - \boldsymbol{K}_{\hat{n}_t,t}\boldsymbol{G}_{\lambda,\hat{n}_t})\boldsymbol{P}_{\hat{n}_t,t-1}^{[m]}$
    $\boldsymbol{L}_t = \boldsymbol{G}_{s,\hat{n}}\boldsymbol{Q}_t\boldsymbol{G}_{s,\hat{n}}^{T} + \boldsymbol{G}_{\lambda,\hat{n}_t}\boldsymbol{P}_{\hat{n}_t,t}^{[m]}\boldsymbol{G}_{\lambda,\hat{n}_t}^{T} + \boldsymbol{R}_t$
    $w_t^{[m]} = |2\pi\boldsymbol{L}_t|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,\hat{n}_t}\right)^{T}\boldsymbol{L}_t^{-1}\left(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{t,\hat{n}_t}\right)\right\}$
  end if
  for $n = 1 \rightarrow N_t^{[m]}$ **do**
    if $n \neq \hat{n}_t$ **then**
      $P_{\lambda_n,t}^{[m]} = P_{\lambda_n,t-1}^{[m]}$
    end if
  end for
end for
$\boldsymbol{S}_t^{-} = \left\langle \boldsymbol{s}_t^{[m]}, N_t^{[m]}, \boldsymbol{\mu}_{1,t}^{[m]}, \boldsymbol{P}_{1,t}^{[m]}, ..., \boldsymbol{\mu}_{N_t^{[m]},t}^{[m]}, \boldsymbol{P}_{N_t^{[m]},t}^{[m]} \right\rangle$
for $m = 1 \rightarrow M$ **do**
  draw particle from $\boldsymbol{S}_t^{-}$ with probability $\propto w_t^{[m]}$
  add new particle to $\boldsymbol{S}_t$
end for
return $\boldsymbol{S}_t$

1. Unlike FastSLAM, UFastSLAM computes the proposal distribution by measurement updates of the unscented filter in the sampling step of the particle filter. This avoids linearized transformations of the vehicle uncertainty and removes the need to derive the potentially complex Jacobian matrices.

2. Each feature state is updated by a UKF which avoids accumulation of a linear approximation error and another computation of a complex Jacobian matrix.

They claim the result is a FastSLAM algorithm that is significantly more accurate than FastSLAM 2.0, even with fewer particles. The paper goes on to detail each step of the FastSLAM algorithm and what will be changed to apply the unscented transform. It also suggests and compares two approaches to implementing measurement noise into the algorithm.

The algorithm is then tested on a SLAM simulator developed in Matlab by Tim Bailey [34] and compared to the provided FastSLAM 2.0 algorithm. The tests were run with only one of the two improvements (ie. unscented particle filter and unscented feature update) as well as both in conjunction. The results show a significant reduction in the MSE from FastSLAM 2.0 by just making one of the improvements, and better still when combining the two. The authors also go on to show that with one fifth of the particles they obtain similar results in magnitude and consistency of MSE comparing UFastSLAM to FastSLAM 2.0.

With these results the authors applied the same UFastSLAM algorithm to the Victoria Park dataset which uses a LIDAR sensor and an indoor environment using sonar detection. The results show a significant improvement in accuracy over FastSLAM 2.0 with only a small increase in computational cost.

2.2.4 Other Developments in Vehicle Localization

Besides these additions to SLAM and FastSLAM there have been significant achievements in the literature related to SLAM and vehicle localization. A collection of these are the focus of this section, discussing the strengths and weaknesses in each approach and the ways in which this thesis improves or draws upon these previous results in the context of the *Kapvik* microrover.

DARPA Challenge Robots

The DARPA Challenge provided the basis for much of the recent robot navigation literature. The results of various teams, especially the winning team, are often cited. Therefore some of the work that was done for this challenge will be reviewed and how it can or can't be applied to *Kapvik*.

A very high profile paper was presented by Thrun et al. in [35] which is an overview of Stanley, the robot that won the DARPA Grand Challenge in 2005. This robot was designed for autonomous, high-speed, desert driving. The vehicle made use of 5 SICK LIDARs housed along it's roof to provide different tilt angle views, along with radar and cameras to provide environmental views. For localization, the robot used GPS and an IMU.

The external sensors form a map with the purpose of determining where the road lies. This LIDAR map is classified based on a height difference between close points in a LIDAR point cloud and computer vision techniques are used to find the location of the road in the camera's field of view. Stanley makes use of the GPS while a signal is available, as well as an IMU, and wheel encoders for vehicle state estimation which includes pose and velocity. A UKF is used to estimate these quantities.

The disadvantage to the Stanley system is its power requirements (500 W), computational power (six Pentium M computers) and reliance on a reliable GPS signal.

The system can make use of the IMU for GPS outages of up to 2 minutes, but beyond that the errors in estimation become too large for safe operation. Many of these disadvantages make the Stanley system unsuitable for a Mars rover navigation system. *Kapvik* has a low power budget (approximately 40 W), only one LIDAR, and relatively small computional power. *Kapvik* also has no access to GPS sensors for navigation.

Stanley did not make use of SLAM for this reason, and it's generated maps were largely used for simple path planning that involved staying on a road. This is much different behaviour than what will be expected of a Mars rover. For these reasons, this thesis uses SLAM as an alternative to the GPS based system of Stanley. A method of terrain classification based on the stop and go approach that is computationally efficient, versus a moving scan as in Stanley, is also desirable for SLAM operation and environmental mapping. This avoids introducing systematic errors that a scan generated as the robot is moving will produce. While the Stanley system introduces a probabilistic approach to solve this problem, it adds additional computation to the navigation system which is undesirable for the *Kapvik* system that has limited computational power.

Another vehicle in the challenge, TerraMax, was one of the 5 participants to complete the challenge, albeit slower than Stanley. In [36], Broggi *et al.* demonstrate a single frame method of obstacle detection using a stereo camera that classifies obstacles and non-obstacles efficiently using a set of filters applied sequentially. This was of interest as a single LIDAR "frame" was to be used for *Kapvik* scans and the resulting obstacles were similar. Their result was a generated map that included only obstacles in a timely manner, and demonstrates the improvements to path planning when efficient obstacle detection can be achieved. While the difference between performing LIDAR classification and stereo image classification is large, the resulting goal of clear, reliable terrain classification is the same and has been demonstrated for

stereo cameras.

FastSLAM using cameras and LIDAR for localization in planetary environments

Barfoot *et al.* have presented a variety of papers in the area of planetary exploration using SLAM. Their approach makes use of cameras as well as LIDAR for SLAM and utilizes a variety of methods for localization of a rover. In [37] a method of visual motion estimation using SLAM is developed with a camera sensor using SIFT features. A UKF version of FastSLAM is also successfully used, further verifying its use as was done in [33]. The method of using SIFT features for estimation in FastSLAM is promising, as it means SLAM can be used to accurately localize the rover in outdoor environments. This thesis attempts to use LIDAR for SLAM, and while the approach of SIFT features works well for vehicle motion estimation, it is seemingly limited to camera systems. By creating a sparse map, much like the SIFT features do, LIDAR could be used in a similar fashion, but retain information to form a global map. It also provides more accurate measurements of feature locations than a stereo camera based system. However, LIDAR is more susceptible to data association errors than SIFT features, due mostly to occlusions as the rover views the same features from different angles.

To address some of these problems, in [38] Barfoot *et al.* design a system that makes use of batch processing techniques that align LIDAR point clouds based on the visual motion's estimation of the rover pose. While this method provides a global map and rover pose, it does not provide an accurate estimate of the map's uncertainty, and LIDAR data in the map does not improve upon the rover's pose. The LIDAR information could potentially be used to improve the system. This thesis attempts to use a similar system that incorporates only LIDAR scans and creates a sparse feature map using LIDAR classification.

Finally, Barfoot *et al.* show in [39] a technique to use only LIDAR for SLAM, but like the camera system, provides only the methods in which to apply SIFT techniques to LIDAR data by incorporating intensity data. In this case only one sensor is used, and can be used in a variety of lighting conditions, which provides an advantage over using a camera system. However, it does not provide a global map at the end of the traverse.

SLAM Consistency

Bailey *et al.* have questioned the consistency of both EKF SLAM and FastSLAM in [40] and [41] respectively. In [40], an analysis is done that shows the EKF-SLAM algorithm produces very optimistic estimates after a certain uncertainty threshold has been met in the vehicle orientation. This is due primarily due to the state estimate being a linearizion of the true state mean and variance and may not match as well as the true probability distribution being non-Gaussian. It is also proposed that this failure cannot be detected without comparing to ground-truth and that conventional solutions do not improve the situation. The conclusion of the paper is that EKF-SLAM can be consistent if orientation uncertainty is kept small, and can be improved using sub-map methods where a large global map is represented by many smaller maps. However, uncertainty will always increase over time. Even when closing the loop, while uncertainty will dramatically drop at that point, it will never be as low as when the rover first observed that area of the map. Therefore they show that the consistency of EKF-SLAM is dependent on both time and the orientation uncertainty, and in many cases the orientation uncertainty is the key problem. They propose that EKF SLAM consistency cannot be improved using more nonlinear filters such as the Unscented Kalman Filter and only sub-maps will improve EKF-SLAM.

For this reason, a potential alternative may be FastSLAM, which is built on a different foundation to that of EKF-SLAM. However, in [41], the authors show that

FastSLAM presents another set of problems that leads to the algorithm degenerating with time. It is shown that in the resampling step of FastSLAM, the number of samples representing past poses is depleted which erodes the statistics of the map estimate. A 2D simulation of FastSLAM is used as an experiment where the number of particles that represent a chosen feature is tallied immediately after the feature exits the rover's field of view. It is shown that the particle diversity at this point decreases exponentially, even for a dense particle set. Increasing the density of the map leads to faster depletion. The conclusion is that in it's current state FastSLAM provides an accurate and practical solution in the short term, but that it is not consistent in the long term. Without a method of improving the resampling in the filter, it is suggested that FastSLAM only be used for a front-end SLAM component that processes current data and a short term local map, with a later process merging the short term local maps to a global EKF-SLAM map.

While this provides a good analysis of the shortcomings of both algorithms, the conclusion of the paper, especially in the FastSLAM case, points to the algorithm working well in practical examples at accurately estimating the mean and variance of the state. For the purposes of this thesis and *Kapvik's* initial navigation algorithms, FastSLAM seems the most practical approach to begin with, even despite its shortcomings presented by Bailey *et al.*

Resampling improvements to FastSLAM

Resampling is an important step in particle filters due to the problem of particle depletion. When low likelihood particles are too easily discarded, the diversity of the particle set will decrease and be more susceptible to data association errors. While these resampling techniques were not used due to the extra layer of complexity needed to implement them, in most cases the algorithm presented in this thesis can be adapted to implement these improvements which appear in the literature and are

presented here as potential future work.

In [42], Cugliari and Martinellli show that an adaptive selective resampling technique can improve the performance of FastSLAM. The choice of when to resample is most often done by taking the inverse summation of the weights of each particle and deeming that the *effective* number of particles. When this number reaches below some threshold, resampling occurs. However, as discussed in the paper, this does not take into account the evolution of the effective number of particles over time. They suggest that if the effective number is low, but is generally oscillating about a constant value, it may still correspond to a good posterior approximation and should not indicate that a resampling should occur (which can lead to particle impoverishment or depletion as shown by Bailey *et al.* in [41]. Therefore the authors present a dynamic threshold for the effective number of particles that is based on the past history of the effective number of particles. To validate that this approach improves the performance of the particle filter, they compare the ratio of the absolute pose error after resampling to that of the pose error before. The comparison is done between the same particle filter with a dynamic threshold and a more conventional static one. The result is that the average ratio is almost always less than one for the dynamic threshold, and in most cases greater than one for the static threshold. This means that on average the resampling step is much less likely to decrease error using the dynamic threshold.

As a side note, this paper also introduces a UKF version of FastSLAM that is very similar to that proposed in both [33] and [37] with similar improvements to the accuracy of the algorithm. The improvement of nonlinear filters such as the UKF that don't linearize the motion or measurement models appears several times in the literature, with these three papers representing a sample of this. This was a motivating factor in selecting a nonlinear filter for the SLAM algorithm used on *Kapvik*.

Another improvement to FastSLAM resampling was presented by Kim *et al.* in [43]. By using geometric relations between particles in the particle set represented by a KD-tree, in addition to the weights, an estimation error potential is determined for a group of particles. A group of particles is selected from the KD-tree and the estimation error potential is determined, if the value satisfies the conditions that must be met, this group is resampled. The result compared to using a conventional resampling threshold on all particles is a decrease in root mean square pose error and uncertainty of more than 30% for a small increase in computational requirements.

Digital Elevation Maps Included in Motion Model

LAAS (Laboratoire d'Analyse et d'Architecture des Systemes) in France has conducted research that uses a 6-wheeled Mars rover prototype known as a Marsokhod. While the rover's wheels are articulated, they are not the same form as the NASA rocker-bogie model. However, in their work they have implemented terrain models into the motion model of the rover to improve the rover's state estimate. While this is not implemented in this thesis due to added complexity, the work in this thesis could be extended to include some of the techniques described in [44]. In the paper they make use of a Bayesian classification method to define the terrain using stereo cameras and use a cell grid to represent the map much like what is adopted for this thesis. As points are added, a digital elevation map is generated, with a method presented of fusing new measurements into the digital elevation map to make up the values that each cell represents to the motion model (elevation, precision, and confidence). This information not only provides a global map of the area, the information of the terrain slope is added into the motion model. The rover's pose can be calculated more accurately given the proposed forward motion that odometry predicts and the shape of the terrain underneath that position and heading.

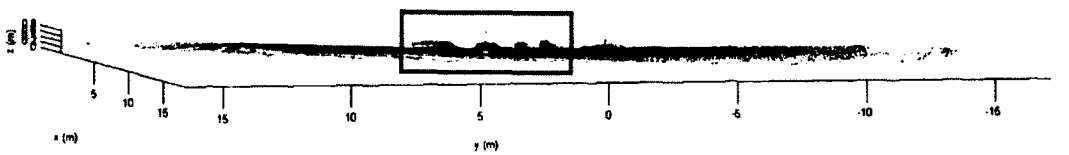
2.3 Map Representation

One of the important decisions in mapping an environment is how to represent such a map. Questions of complexity, memory usage, and accuracy must be asked when choosing how the map will be represented. The map will be accessed by many different parts of an autonomous rover's guidance, navigation and control software, and the map must be compatible with all of these algorithms. In this thesis "feature" or "point" (terms used interchangeably) based maps are chosen for their relative simplicity and the wide range of literature that supports this method of representation.

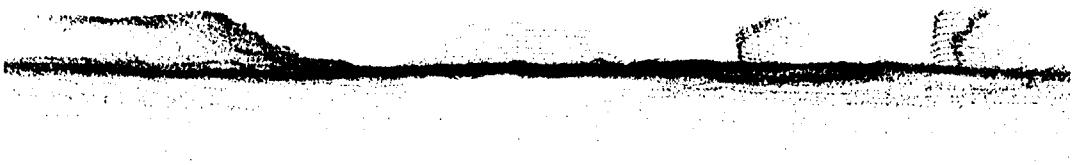
The path planning algorithm that *Kapvik* uses (Theta* path planning) represents its own map with a grid cell format [45], with a resolution defined by its own constraints. The same holds true for the neural network grid that is proposed for classifying scans in this thesis. Taking all of these factors into consideration it was necessary to start out using a feature based map, but by no means does it rule out other forms of map representation for future development. In the interest of better understanding the differences, the three main map representations that appear in the literature are presented below.

2.3.1 Point Clouds

Point clouds are the most simple form of map representation as the measurements are not manipulated beyond an initial transformation to the global reference frame. An example of a point cloud scan taken with the LMS 111 mounted on the Husky rover is shown in Figure 2.11. There are many advantages to using a point cloud based map representation. The first is a relatively simple measurement model that allows all points to be simply transformed to the global Cartesian coordinate system. The advantage of a simple measurement model is apparent in the FastSLAM algorithm where it is extensively used to update and add features to the map. Point cloud maps



(a) Point cloud taken from Husky with LMS 111 LIDAR on sandy terrain with rocks.



(b) Close up view of Figure 2.11a within the rectangle which includes the slopes and rocks.

Figure 2.11: Point cloud example

are also easily extended to 3D environments, and in fact see much of their use in the literature in this context [4,10]. These maps are also easily converted or partitioned to cellular grids for use in other algorithms that process the map in different ways. This is very advantageous for the purposes of *Kapvik*, which involved various guidance, navigation and control algorithms being developed in tandem. Point cloud maps also provided a generic type of map that could be generated by a wide range of sensors including sonar, cameras and LIDAR.

While it is relatively simple to convert measurements from the rover's LIDAR sensor to a feature in the map, the maps will also grow without bound, constrained only by the precision of the computer being used. While these maps allow for more

accurate mapping, they also become much too large when simply converting measurements to raw point clouds for real time usage. Many LIDAR will include up to hundreds of thousands of points in a single scan and the computational complexity needed to process these scans in SLAM along with the memory requirements to store the maps must be addressed.

2.3.2 Occupancy Grids

Occupancy Grids represent a map by dividing up the map into a cell grid as shown in Figure 2.12. Usually the grid is divided uniformly along the dimensions of the reference system. If a single measurement is attributed to a position within one of these cells, the grid is considered "occupied". At first glance there are a few things to note about this approach. First, it is obviously very dependent on how high the resolution of the grid is. As the grid's resolution approaches infinity, it approaches the representation of a point cloud based map. This allows an occupancy grid map to scale with the memory constraints of the rover it is being used on. For the same reasons of resolution, its accuracy is also limited, and it is not immediately clear how the statistical methods used to update features in the map explained in 2.2.2 would be applied to this type of representation. However it has been shown [46, 47] that a statistical approach is possible for occupancy grids and can be directly applied to the FastSLAM problem.

The disadvantages of occupancy grids come with their limited use in the literature and their ability to be used in other algorithms efficiently. In the literature occupancy grids have mostly been applied to 2D problems and in largely indoor, cyclical environments [46, 47]. They are well suited to these problems where walls and other well defined obstacles make up most of the grid cells that are "occupied" with unoccupied cells generally representing traversable, flat areas. In an outdoor environment the environment is much more complex. The surface the rover is traversing is often not

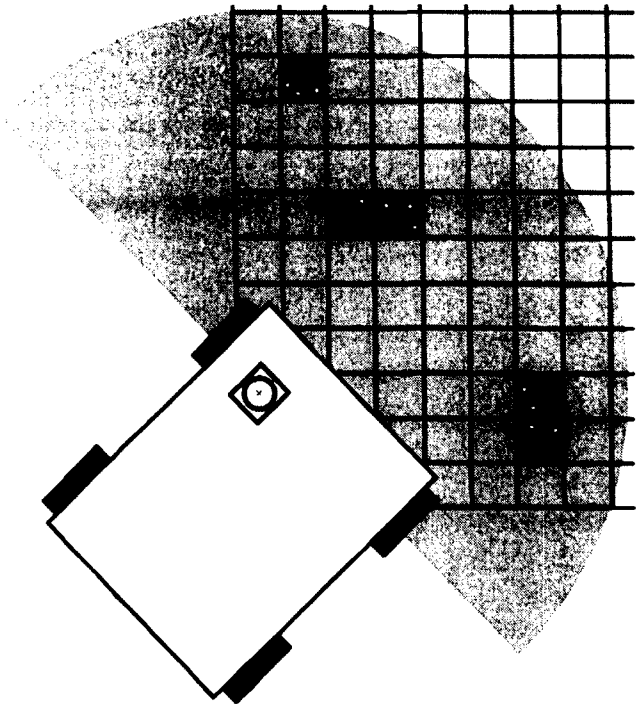


Figure 2.12: Occupancy grid based on LIDAR measurements. Black cells are occupied, with white dots representing the LIDAR measurements.

flat and has many different textures and shapes that will be detected by the rover. Outside of the surface the rover travels over, there are other objects such as rocks that the laser will detect but may or may not be traversable. The methods used for matching these scans are complex, and represent another area of research that is ongoing.

Another aspect of occupancy grids is their ability to be easily interchanged with other algorithms that are running on *Kapvik*. Compared to a feature map that is defined simply by the global Cartesian coordinate system, an occupancy grid may have a different resolution compared to the maps used by other algorithms such as the path planning algorithm that *Kapvik* uses. By providing a global map that is estimated for other algorithms to make use of, any number of methods can be used to convert it to a form that is appropriate to the algorithm at hand. In the context

of a rover being used for active research, it makes sense to use the most general form possible for map representation.

2.3.3 Lines and more complicated shapes

Both feature based and occupancy grid cells in their basic form represent measurements in discrete points, either as a point or a cell. However, it is possible to extend this representation into the form of lines and more complicated shapes such as polygons. The advantage of doing this can be seen when considering points that belong to a wall, or other surface that may generate measurements that are very close to one another. The problem that these closely bunched points represent is demonstrated in Figure 2.13. While a strictly point based map would include many points in the example shown, by recognizing points that belong to a line or corner, the number of "features" in the map can be reduced dramatically. Not only this, but in the problem of data association, it is easier to determine which scans belong to a feature that was observed in previous scans, and which belong to new features by considering the geometric relationships between these shapes. This helps reduce the dependency of data association on the pose of the rover when it observes the same feature in a subsequent scan. This will improve both memory requirements and accuracy in the map.

Data association methods such as Joint Compatibility Branch and Bound work on this premise, dividing up scan points into specific shapes for features but have largely been presented in the context of 2D environments in the literature [48]. While it is straightforward to build a measurement model for these shapes and even to extend the process into 3D environments, the ray tracing operations become prohibitively expensive for 3D polyhedrons when generating predicted measurements. These types of maps also are not immediately transferable to an occupancy grid map and would require an extra layer of software to do so for the path planning and neural network architectures and to be compatible with other sensors such as stereo vision.

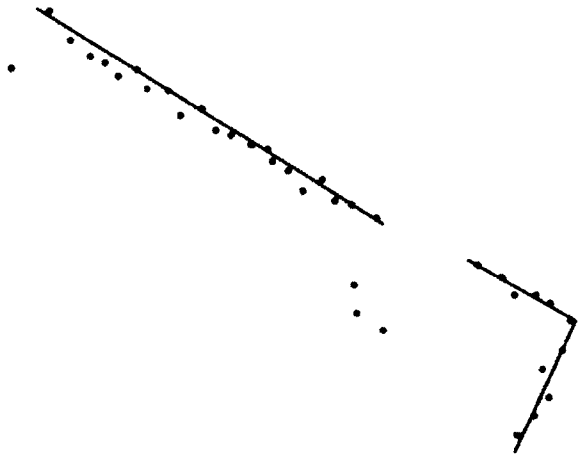


Figure 2.13: Lines and corners can be extracted from point data to limit the ambiguity between closely spaced points

2.4 Data Association

Data association is the process of identifying features in the rover's stored map that appear in new measurements. In the case of LIDAR it will involve comparing the latest point cloud of measurements to any features in the rover's stored map that are within the rover's field of view. As explained by Montemerlo and Thrun in [5], the common solution to the data association problem is to maximize the likelihood of a single sensor measurement $\boldsymbol{z}_t$ given all of the features in the current map. This is described as:

$$\hat{n} = \arg\max_{n_t} p\left(\boldsymbol{z}_t \mid n_t, \hat{n}^{t-1}, \boldsymbol{s}^t, \boldsymbol{z}^{t-1}, \boldsymbol{u}^t\right). \tag{2.72}$$

The term on the right hand side of (2.72) is the likelihood that measurement $\boldsymbol{z}_t$ is the map feature n_t from the data associations $\hat{n}^{t-1}$ that have been made, given the rover path, s^t, and the set of inputs, u^t. By choosing the feature that gives the maximum likelihood, including a threshold value, p_0 that represents a new feature in the map, all measurements can be associated.

It is important to note that the advantage that FastSLAM will bring to data

association is the concept of multiple data association hypotheses. Regardless of which data association method is used in the end, EKF-based SLAM approaches are brittle to data association errors. If even one data association error is made, large errors can appear in the map. By keeping a set of particles that each contain its own set of data associations dependent on s^t, FastSLAM reduces the effect of wrong data associations. The primary cause of data association errors is due to uncertainty in the SLAM posterior that cause ambiguities in the data association problem. There are two types of uncertainty that specifically lead to this.

2.4.1 Data Association Ambiguity

The first type of uncertainty is measurement uncertainty. As the noise in a measurement increases, the distributions of features in the map become larger, and it becomes more likely that these distributions will overlap with features that are close in the map as seen in Figure 2.14. However the effect of a wrong data association on the rover's estimated pose and map will be small in this case, as the measurement could have been generated by either feature in the map.

The second type of uncertainty is motion uncertainty, which can have much larger effects on estimation. If the rover's pose is uncertain, using techniques such as nearest neighbour approach can lead to drastically different data associations hypothesis. Due to the rover generally moving over a horizontal surface, the effects of rotational uncertainty is the largest. An example of the effect a pose uncertainty can have in data association results is depicted in Figure 2.15.

While measurement uncertainty adds error to the map, motion uncertainty can quickly cause a traditional filter to diverge. However, because the particles in FastSLAM are initially propagated according to the distribution of the rover's pose estimate, the particles that contain correct data associations will be weighted higher than ones that choose wrong data associations and eventually these wrong particles will

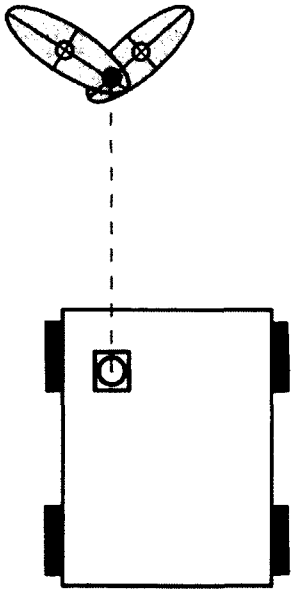


Figure 2.14: Demonstration of data association ambiguity through measurement uncertainty.

be removed through particle re sampling. This provides an implicit way for SLAM to remove bad data associations from the map.

With the advantages that FastSLAM can provide to any data association method, the next step is to choose a form of data association. The nearest neighbour approach which has already been mentioned may work, however there are more complex types of data association that take into account the fact that no two measurements can belong to the same feature in a single laser scan, which is not addressed in the nearest neighbour approach without using heuristics. Some of these more complex methods are the Hungarian algorithm [49,50], Joint Compatibility Branch and Bound, and Combined and Constrained Data Association [48] which were researched for this thesis.

Due to the ease of implementation and speed in which it can be run in real time for 3D applications, the Hungarian algorithm is used in this thesis. The algorithm will be explained in the rest of this section.

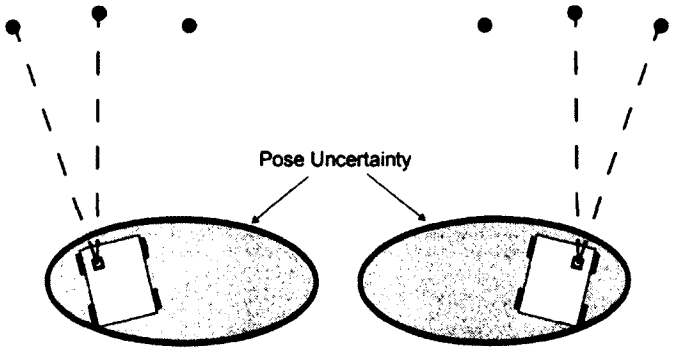


Figure 2.15: Demonstration of data association ambiguity through pose uncertainty. Significantly different data associations can be made for different pose estimates.

2.4.2 Mahalanobis Distance

Euclidean distance can be very sensitive to the scales of the variables involved with respect to one another, and is not able to account for correlated variables whereas Mahalanobis distance does. The Mahalanobis distance of a multivariate vector, x, from a known vector y with a covariance of P is given as:

$$D_M(x) = \sqrt{(x-y)^T \boldsymbol{P}^{-1}(x-y)}. \tag{2.73}$$

This can also be thought of as a measure of dissimilarity between two vectors of the same distribution with a covariance of $\boldsymbol{P}$. In the case of SLAM, data associations are chosen such that they maximize the likelihood of the sensor measurements, z, given all available data as described in (2.72). This term is the likelihood, and the data association approach which attempts to maximize this is called a maximum likelihood estimator or nearest neighbour data association, where the negative log likelihood is described as a distance function. For Gaussians, the negative log likelihood is the Mahalanobis distance [5]. The estimator used in this thesis performs data associations by minimizing the Mahalanobis distance.

Each of the feature estimators are Kalman filters, so the distance can be calculated using the innovations, $(\boldsymbol{z}_t - \hat{\boldsymbol{z}}_{n_t})$, in the measurement update step. The Mahalanobis

distance is defined in this problem as:

$$D_M(z_t) = \sqrt{(z_t - \hat{z}_{n_t})^T Z_{n,t}^{-1}(z_t - \hat{z}_{n_t})}, \tag{2.74}$$

where Z is the feature covariance for position in terms of measurements. For computational simplicity the squared Mahalanobis distance is calculated. The squared Mahalanobis distance is already calculated in the FastSLAM algorithm in (2.69) that will be presented in Chapter 3 for the importance weight in the importance sampling step of the particle filter.

A higher distance represents a lower likelihood that a measurement can be attributed to an already established feature estimate in the current map. Each measurement must be associated with a point in the map that has the highest likelihood of causing the measurement, above some threshold. If the point with the highest likelihood is below this threshold, then the measurement is recorded as belonging to a new feature, and added to the map. Otherwise the new measurement is used as an update to the current estimate of the feature it has been attributed to with the feature update step in SLAM. The threshold is calculated from the inverse cumulative distribution function for a multivariate normal distribution, which is the chi-square inverse for a given probability [51].

2.4.3 The Assignment Problem

In the end, the sum of the assignments' costs (Mahalanobis distances) should be minimized. If each measurement was simply assigned to the feature that had the lowest Mahalanobis distance, there is the possibility that multiple measurements could be assigned to the same feature which is impossible.

Instead of simply searching through all of the Mahalanobis distances for each measurement with each point in the map and choosing the minimum distance (and

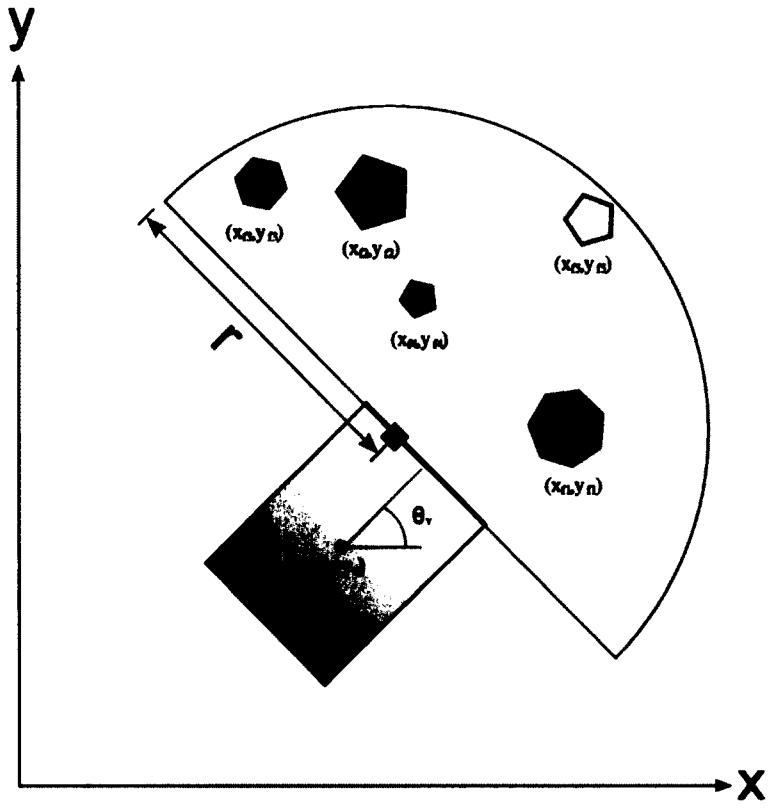


Figure 2.16: A rover performing data association

therefore highest likelihood), there are assignment techniques such as the Hungarian algorithm which speed up the process and are able to solve the assignment problem in polynomial time. The Hungarian algorithm was first published by Harold Kuhn [49] in 1955, and was later reviewed by James Munkres [50] and is also known as the Munkres assignment problem. The Hungarian algorithm approach was first applied to FastSLAM by Gamallo et al. [52] and their method was adapted to the nonlinear FastSLAM approach that is proposed in Chapter 3 along with a cost metric defined by the Mahalanobis distance. An example of the algorithm can be shown for a $n \times m$ matrix as follows:

If it is supposed that a rover has built a preliminary map in the beginning of its traverse and it has detected 4 features (represented by a position estimate and

an associated covariance) as shown in Figure 2.16. The blue figures represent features already measured and the yellow one has not been measured previously. After traversing a distance, it stops and takes a new measurement, this time detecting 5 features. To represent these two sets of data a matrix is constructed, the measurement represented by rows and the known features represented by columns:

$$\begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \\ c_1 & c_2 & c_3 & c_4 & c_5 \\ d_1 & d_2 & d_3 & d_4 & d_5 \\ e_1 & e_2 & e_3 & e_4 & e_5. \end{matrix} \tag{2.75}$$

In the above matrix, a through e rows represent the 5 new measurements and 1 through 4 columns represent known features. The 5th column represents the cost of assigning a measurement to a new feature, which is the threshold mentioned earlier. Therefore, a_1 represents the cost of assigning measurement a to feature 1, which in this case is the Mahalanobis distance between the measurement and the predicted measurement for that feature. The algorithm can be divided up into steps as follows:

Steps

1. The first step is to subtract the lowest element in each row from each other element in the row. This will leave at least one zero in each row. The next step is to attempt to assign measurements to features so that each measurement is assigned to only one feature, and the cost to do so is zero:

$$\begin{matrix} 0' & a'_2 & a'_3 & a'_4 & a'_5 \\ b'_1 & 0' & b'_3 & b'_4 & b'_5 \\ c'_1 & c'_2 & 0' & c'_4 & c'_5 \\ d'_1 & d'_2 & d'_3 & 0' & d'_5 \\ e'_1 & e'_2 & e'_3 & e'_4 & 0'. \end{matrix} \tag{2.76}$$

Where each 0' is the assignment made and an apostrophe represents a subtraction that has been made from that element.

2. It may be that after this first step it is impossible to use the matrix for assigning. For instance if after step 1, the matrix looked like:

$$\begin{matrix} 0 & 0 & a'_3 & a'_4 & a'_5 \\ b'_1 & b'_2 & 0' & b'_4 & b'_5 \\ c'_1 & c'_2 & c'_3 & c'_4 & 0 \\ d'_1 & d'_2 & 0 & d'_4 & d'_5 \\ e'_1 & 0 & e'_3 & e'_4 & e'_5. \end{matrix} \tag{2.77}$$

Measurement a and e could be assigned to feature 2, and feature 4 cannot be assigned at all. The next step is to repeat the procedure in step 1 but for each column (instead of each row) and a check is done for assignments that can be made.

3. If after this step, there are still assignments that cannot be made, step 3 must be performed. First, assign as many measurements as possible, then mark each

measurement that has no assignment. Next mark all features that have zero cost for that measurement. Finally mark each row that has an assignment in that column. A line is drawn through each column and row that have been marked to visually depict this.

4. From the values that remain, the smallest value is subtracted from the marked rows, and added to the marked columns. Steps 1-4 are repeated until assignment is possible. Eventually assignment will always be possible.

This assignment method greatly reduces the computational time required to perform data association in FastSLAM and perhaps more importantly makes an assignment exclusively for each feature and measurement pair, but there are a few more ways time can be saved. Because the rover at any given point in its path is constrained by the range of its sensors, only a portion of the map is possibly measured. If this is taken into account at each point where SLAM is performed, the number of features that are compared to new measurements is drastically reduced.

Instead of comparing each feature in the map to each new measurement, only the points within range of the sensors on the robot are compared, given the robot's current estimate of its pose and map and the sensor's associated uncertainties (ie range and bearing error). This also reduces the chances of incorrect data associations being made. The trade-off comes if the uncertainty is too conservative or not conservative enough, in which case data association will take longer than needed or could miss valid features.

2.4.4 Hungarian algorithm for SLAM

In this paper Gamallo *et al.* demonstrate FastSLAM using Omnivision [52], and a unique data association technique that uses the Hungarian algorithm which was of particular interest for the work done in this thesis. The author's frame the data

association process in terms of a cost matrix. This cost matrix is $N_{z_t} \times \left(N_{t-1}^k + N_{z_t}\right)$ in size where N_{z_t} is the number of new measurements and N_{t-1}^k is the number of features in the map at time $t-1$. Each element in the matrix, $(\phi_{l,j})$ with $j \leq N_{t-1}^k$ represent the probability that measurement l originated from feature j and elements with $j > N_{t-1}^k$ represent the probability that measurement l came from a new feature. With this cost matrix the best data association is chosen with the Hungarian algorithm.

The author's explain that the Hungarian method is a combinatorial optimization algorithm which solves the assignment problem in polynomial time. The algorithm operates on the cost matrix already computed, with the result being a matrix that has elements that take on either a value of 1 or 0, with 1 representing a measurement l being associated with a feature j. The resulting matrix satisfies a condition of having only a single 1 for each row and column. What this means is that a measurement should only be assigned to a single feature, and likewise any feature should only be associated with one measurement. This is the reason for adding N_{z_t} columns to the cost matrix, in the case of all new measurements being attributed to new features.

One of the aspects of this paper that separates it from most is it details the implementation of the process of data association in the SLAM algorithm. Much of the literature on FastSLAM assumes known data association, or does not address directly the implementation of data association in the algorithm. In the case of FastSLAM 2.0, the measurements are introduced into the proposal distribution in the first part of the algorithm, as detailed in Section 2.2.2. This depends on each measurement being associated with a feature in the map. The authors suggest that for each feature in the map, the updated proposal is calculated for each new measurement. The likelihood is stored in this step, along with a sampled pose. After all features and measurements have been looped through, and all likelihoods and sampled proposals stored, the Hungarian algorithm is run on the cost matrix generated from the likelihoods. Given the result, the sampled proposals from all the chosen data associations are accessed, and

a weighted average is done based on the associated likelihoods. The majority of the FastSLAM 2.0 algorithm follows as described in the original publication.

The authors also make use of another metric to determine when to remove features from the map. An initial value of 1 is given to each measurement that has been associated to a new feature. Based on the field of view of the rover, if the rover expects to see the same feature again and either does or does not, it will subtract or add 1 point to the score. When the score reaches a value of 0, the feature is removed from the map. This allows the rover to disregard features in its map that are due to spurious measurements.

The authors applied their algorithm to a Pioneer 3 rover using an omnidirectional digital camera and using ceiling lights as features. Their tests showed successful navigation of the rover using the Hungarian algorithm for data association.

2.5 Neural Networks

Neural networks are at their most basic, mathematical constructs that have similar characteristics to biological neural networks. These are based on a few assumptions which are stated by Fausett in [53].

- Information processing that occurs at many simple elements deemed "neurons".
- Values are passed between neurons through connections between them.
- Each of these connections has an associated weight which usually is multiplied by the value that is passing through it.
- Each neuron processes the sum of weighted values that pass through it with an activation function which is usually nonlinear.

A neural network is therefore defined by the architecture of these connections, the way each connection is weighted, and the activation functions. These three aspects

are addressed in the design of a neural network for pattern recognition in Chapter 3 of this thesis.

2.5.1 Neural Networks for Pattern Classification

The specific use of a neural network in this thesis relates to pattern classification. By classifying LIDAR scan data, SLAM and path planning can be completed more efficiently. Neural networks have been used in recognition of handwritten characters, for example in [54] a multi layer perceptron neural network like the one used in this thesis was used to recognize handwritten zip codes.

More recently, Hata *et al.* in the paper *Terrain mapping and classification using neural networks* [55] demonstrate a neural network that is applied to the problem of classifying LIDAR scans. In particular, they describe a 3D terrain mapping and classification technique for robots operating in outdoor environments. Their approach is to use a multi-layer neural network for this task and classify data according to navigable, partially navigable and non-navigable. The scans are taken from a pioneer AT robot with a SICK LIDAR tilted at 10 degrees to the rover's horizontal which requires robot motion to create a 3D map and relies on the estimate of the rover's motion from wheel odometry.

After the scan has been taken, the data is segmented into a cellular grid such that multiple scan data points are included in many of the cells. Each of these cells is characterized by two values, the absolute height of the grid cell as well as the relative height between neighbouring cells. These characteristics serve as the inputs to a multilayer neural network that has three outputs, for each of the classification types. If only one of the cells is "activated" and outputs a value of one, the corresponding classification is given. Otherwise if no cells are activated or multiple cells are activated, the classification is left unknown.

The authors used approximately 700 scans that were taken from their rover platform to perform supervised training of the neural network. They compared the results of a neural network with just 4 hidden neurons, as well as 8, 16 and 32 hidden neurons, in all cases using 1 hidden layer. After training the neural network it was tested on approximately 630 new LIDAR scans that were not seen during training. The result was correct classification 99.36% of the time with a similar result in MSE comparing the 4 different sized networks. For this reason, the network with 4 hidden neurons was chosen for computational efficiency.

Chapter 3

Algorithm Development

This chapter provides an in-depth explanation of the algorithms that have been developed to meet the goals laid out in Section 1.2. To begin, Section 3.1 describes the rover chassis and the various sensors that contribute to the motion and measurement models used in the SLAM algorithm presented in this thesis. The sections that follow provide a more detailed explanation for each system presented in the overview. In Section 3.2, the kinematic model that is used to represent both a standard skid-steer four wheel rover and the six wheel *Kapvik* rover in the SLAM framework is presented. The hybrid FastSLAM and Kalman filter framework to take into account the limited availability of scan opportunities is presented in Section 3.3 where both estimation between, and directly after scans, is explained along with a detailed measurement model presented for the LIDAR range finder. An innovative variant of nonlinear FastSLAM which is a contribution of this thesis is discussed in Section 3.3.2. Moving from the basic theory of the SLAM algorithm into data association, a novel LIDAR classification technique is introduced in Section 3.4 where the scans are parsed in such a way that only interesting features are passed on for data association. Finally in Section 3.5, a high level description of the overall algorithm that has been put in place for the execution of SLAM is presented.

3.1 *Kapvik* micro-rover

The *Kapvik* micro-rover was the basis of the research done for this thesis. It contains many sensors that can be used for navigation and the goal was to use them as efficiently as possible. The rover includes a camera and LIDAR for sensing its external environment along with a suite of inertial sensors that are used for navigation between scans. While the IMU and sun sensor can provide an estimate of orientation during a traverse, the rover is limited to dead reckoning when estimating it's position between LIDAR scans. Thus, over time the position error grows unbounded until a new scan is taken. While this section details the algorithms to accomplish this, it is a good idea to explore the rover itself, and describe the sensors that will be used in the remainder of this chapter.

3.1.1 Rocker-Bogie Chassis

The *Kapvik* micro-rover employs a rocker-bogie mobility system. This mobility system was created by National Aeronautics and Space Administration (NASA) and the Jet Propulsion Laboratory (JPL) and has been used on Sojourner, Mars Exploration Rovers and Mars Science Laboratory. The mobility system is made up of a series of linkages with no springs used. This mobility system is designed such that the rover can traverse obstacles up to one wheel diameter in height [56]. The *Kapvik* chassis is 0.782 m wide and 0.850 m long and is shown in Figure 3.1. The left and right sides of the chassis are connected to two rocker links that are constrained to $\pm 16°$ with hard stops. The design of a differential mechanism ensures that the angles of the left and right joint are equal and opposite. At the front of the rover, the end of each rocker joint is connected to the front wheel. On the other side of the rocker link another link is connected that rotates about the end. This is the bogie joint, and it in turn is limited by $\pm 30°$ with hard stops. On the ends of each bogie link are the

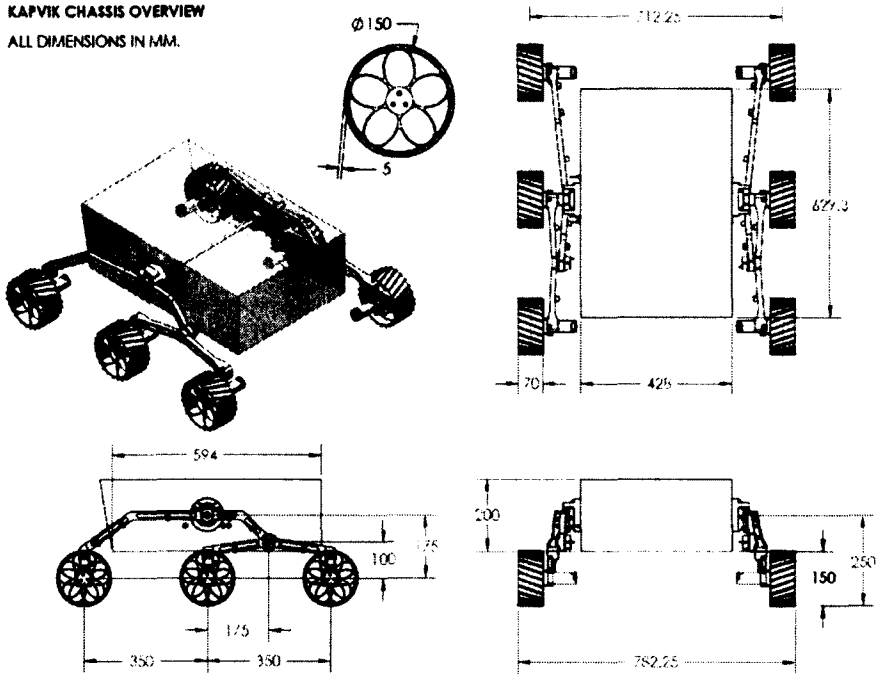
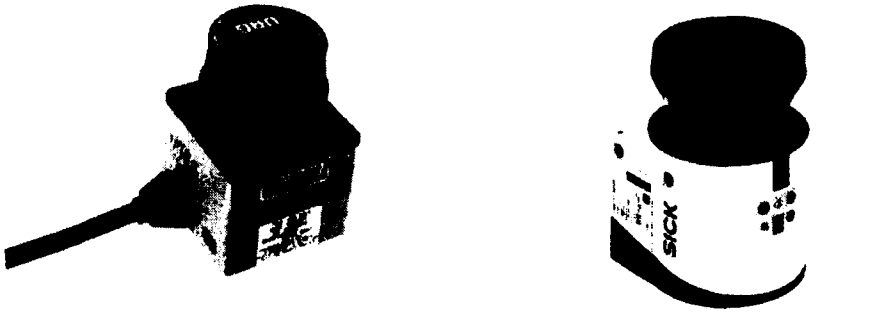


Figure 3.1: *Kapvik* Chassis dimensions, drawing provided by Tim Setterfield.

remaining two wheels for each side of the rover. The links are sized in such a way that the wheels are evenly spaced and aligned along the rover's x-axis.

Kapvik makes use of skid steering instead of the more common Ackermann steering used on NASA's rovers. This was done to reduce mass and complexity, and involves sending different commands to the left and right sets of wheels to induce turning motion. As a side benefit, this is the same type of steering commonly employed on robot platforms for research tasks such as the Pioneer and Husky platforms used in this thesis. This allowed the algorithms developed in this thesis to be tested on these platforms before they were tested on simulations and actual hardware for *Kapvik* in many cases.



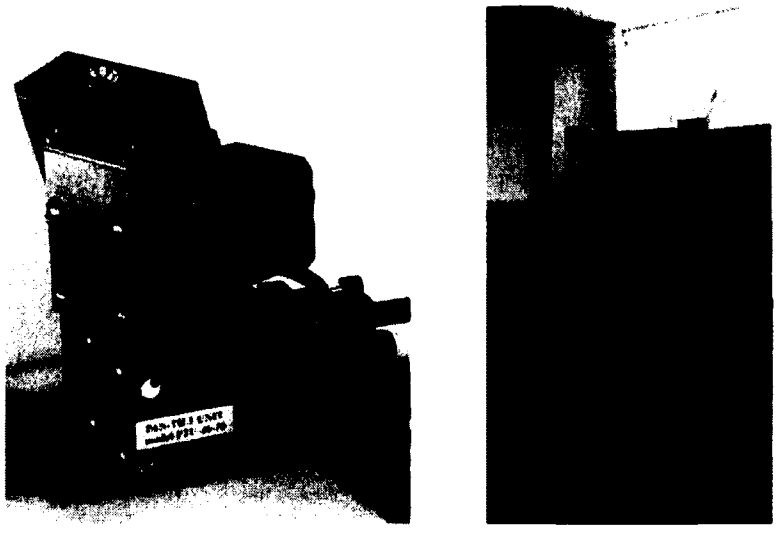
(a) Hokuyo URG-04LX range finder (b) SICK LMS 111 range finder

Figure 3.2: Two LIDAR units used for this thesis

3.1.2 LIDAR Tilt Unit

In this thesis, the LIDAR (Light Detection and Ranging) is the single most important sensor in the suite of sensors on *Kapvik*. To execute SLAM, a sensor that perceives the rover's external environment must be used. In addition to LIDAR other commonly used alternatives are cameras and sonar sensors. While *Kapvik* has a stereoscopic camera, this is not implemented in the work done in this thesis. In the future it will be possible to combine point clouds generated from a stereoscopic camera with ones generated from the LIDAR, and operate both in the confines of the SLAM algorithms presented in this thesis. However point clouds require additional data processing of camera images that was not completed in time for this thesis, and LIDAR was the only sensor that was ready from the beginning to test with.

The LIDAR sensor implemented on *Kapvik* is the Hokuyo URG-04LX rangefinder shown in Figure 3.2a. This LIDAR was chosen because of its fairly high accuracy (approximately 1 % of the range measurement), high angular resolution (0.36°), field of view (240°) low power consumption ($2.5W$) and mass ($0.16kg$). The latter two specifications were especially crucial in the case of *Kapvik* which has a very limited power and weight budget and competing LIDAR sensors tend to use much more power and mass. The sacrifice for low power consumption is the distance the LIDAR



(a) PTU-D46 Pan-Tilt unit.

(b) Kapvik custom LIDAR tilt unit.

Figure 3.3: Two LIDAR tilt units used for this thesis.

can measure, which is approximately 5 m. This was deemed adequate for the type of exploration *Kapvik* was envisioned for, but it does put some limits on the loop closing ability of *Kapvik* when performing SLAM.

The Hokuyo LIDAR was mounted on a pan-tilt unit to allow it to take 3D scans of its environment by tilting along the rover's y-axis. This produces "layers" of data, and by increasing the resolution of the tilt, a more accurate 3D representation is obtained. For testing purposes, a PTU-D46 shown in Figure 3.3a was used on the Pioneer and Husky platforms to accomplish this task. For *Kapvik* a custom tilt unit was designed and built at Carleton that met the power and weight constraints of the project and is shown in Figure 3.3b. In either case, the technique is the same, and requires a simple adjustment of the position of the tilt unit in the rover reference frame as well as the distance the LIDAR is mounted from the center of rotation in the algorithms that will be presented in the rest of this chapter.

In addition to the Hokuyo LIDAR, a SICK LMS 111 pictured in Figure 3.2b was

Figure 3.4: Inscale GL60 hollow-shaft potentiometer.

used for additional testing on the Husky rover platform. The LMS 111 boasts a much larger range (approximately 20 m) higher angular resolution (0.25° intervals) and a quoted range accuracy of 20 mm which is more or less than the Hukoyo depending on the magnitude of the range measurement. The weight (1.1 kg) and power consumption (12 W) exclude it from being used on *Kapvik*, but these specifications were fine for use on the Husky rover platform.

3.1.3 Wheel Encoders

Wheel angle measurements, θ, are obtained using wheel encoders. Wheel encoders measure the number of "ticks" that the wheel passes as it rotates. For *Kapvik*, Maxon magneto-resistant encoders were used, with 2000 ticks per motor revolution. Given the gear ratio on *Kapvik* the resolution of a single count is 1.286×10^{-4}°. The noise in encoder position is approximated by a normal distribution, with the quantization error 3σ bounds given by $\pm\frac{1}{2}$ resolution. This results in a wheel encoder standard deviation of $\sigma_\theta = 2.143 \times 10^{-5}$°

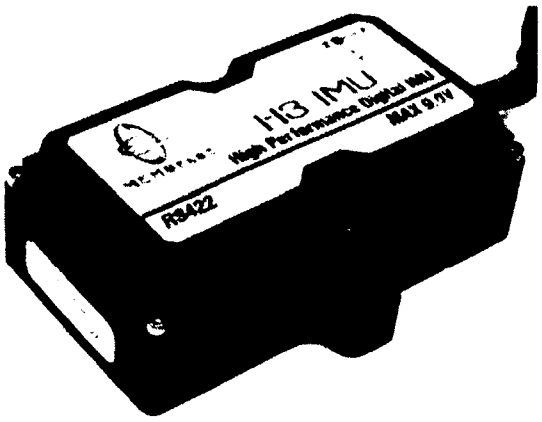


Figure 3.5: Memsense H3-IMU

3.1.4 Potentiometers

To measure the rocker and bogie angles, potentiometers were used for their ability to provide an absolute measurement of the quantities. On *Kapvik*, Inscale GL60 hollow-shaft potentiometers were used shown in Figure 3.4. The repeatability of this sensor is 0.1° [57].

3.1.5 Inertial Measurement Unit

Measurements of the rover's pitch, roll and heading can be obtained using an inertial measurement unit (IMU). The IMU used on *Kapvik* is the Memsense H3-IMU HP02-0300 depicted in Figure 3.5. It includes three orthogonal accelerometers and three orthogonal gyroscopes. The accelerometers have a range of $\pm 2g$ and a standard deviation of $\sigma_{acc} = 1.27 \times 10^{-4} g$ [58]. Each gyroscope has a standard deviation of $\sigma_{gyro} = 0.56°/s$ [58].

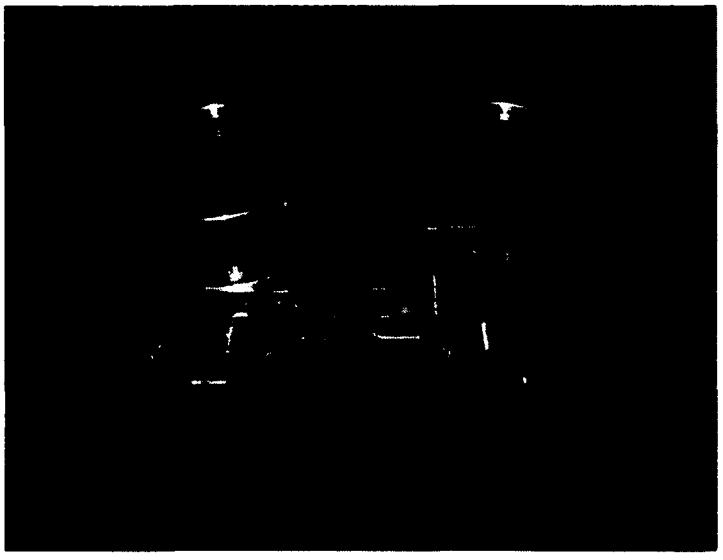


Figure 3.6: Sinclair Interplanetary SS-411

3.1.6 Sun sensor

The sun sensor provides an absolute measurement of heading to *Kapvik*, which is very important when controlling and scanning its environment. The sun sensor included on *Kapvik* is a SS-411 from Sinclair Interplanetary shown in Figure 3.6. The sun sensor provides a sun vector with values along 3 orthogonal axis within 70° of the sensor's +Z axis with an accuracy of ±0.1° [59].

3.2 Rover Kinematic Models

In this thesis two rover forward kinematic models are presented. In SLAM, each particle that is sampled will then be propagated forward each time step according to the kinematic model being used. The reason for using two different models was to allow for testing of the completed algorithms on the test vehicles available to the author. The *Kapvik* rover was still in the stage of being built at the time of writing, and therefore was not available for testing. It was also useful to start with a simple four wheel rover model and design the algorithm, later adapting it to use on a six

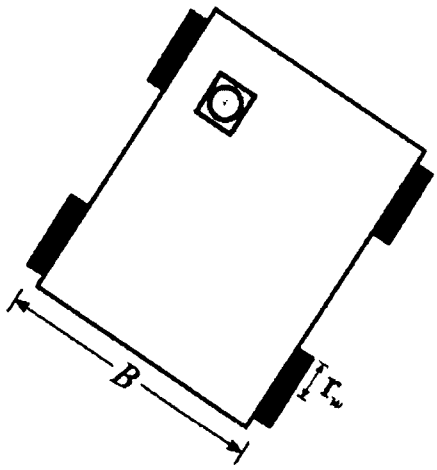


Figure 3.7: Four wheel rover overhead view, the dimensions of the rover affect the calculation of the control inputs from the wheel angular velocities.

wheel rover such as *Kapvik*. The result is an algorithm that can be tested on both rover models in simulation, and on the four wheel variant in real life. When developing the initial four wheel rover model, care was taken to establish a model that was easily adaptable to the six wheel rocker-bogie variant. The following sections detail the development of each rover model and will define the differences between the two vehicles.

3.2.1 Skid-Steer Four Wheel Rover

The skid-steer four wheel rover forward kinematic model was used to represent both the Pioneer and Husky rovers. These testing platforms are discussed in more detail in Section 4.2. A generic four wheel rover will be detailed in this section where dimensions such as the width and length of the rover can be adjusted to fit the specifications of whatever testing platform is being used.

A differential drive rover forward kinematic model can be approximated by a unicycle model if the two wheels on the left side maintain the same angular velocity, $\dot{\theta}_L$, and the two wheels on the right side also have the same angular velocity, $\dot{\theta}_R$. As

this is the common configuration for the test platforms used, this is a valid assumption and is used for this thesis. The model is propagated forward given the current terrain the rover is on, and is therefore a function of the rover's roll, pitch and yaw, denoted Φ_x, Φ_y and Φ_z, following a roll-pitch-yaw convention. The orientation of the rover is a function of the contact angles between each wheel and the ground. This model follows a pseudo-dynamic kinematic form where the model is propagated forward in space and any discrepancies between this model and real life should be contained within the process covariance, $\mathbf{Q}$. This helps simplify the rover model without going into the full dynamic model which is not developed in this thesis. The two unicycle inputs are linear velocity and angular velocity of the rover's geometric center represented in the rover reference frame. The linear velocity, $\boldsymbol{v}(t) \in \mathbb{R}^3$, is completely along the rover's x-axis and the angular velocity, $\boldsymbol{\omega}(t) \in \mathbb{R}^3$ is about the rover's z-axis (ie. $\boldsymbol{v}(t) = [\dot{x}, 0, 0]^T$ and $\boldsymbol{\omega}(t) = [0, 0, \omega_z]^T$) where $\dot{x}$ and ω_z are with respect to the instantaneous rover reference frame. They can be calculated as,

$$\dot{x} = r_w \left(\frac{\dot{\theta}_L(t) + \dot{\theta}_R(t)}{2} \right), \tag{3.1}$$

$$w_z = \frac{r_w}{B} \left(\dot{\theta}_L(t) - \dot{\theta}_R(t) \right), \tag{3.2}$$

where r_w is the radius of each wheel, assumed to be the same for all four wheels, $\dot{\theta}_L$ and $\dot{\theta}_R$ represent the left and right wheel angular velocities respectively, and B is the distance between the left and right wheels as shown in Figure 3.7. It is important to note that the left and right wheel angular velocities are measured from the wheel encoders discussed in Section 3.1.3.

These inputs are specified in the rover reference frame but the rover states are specified in the roll-pitch-yaw convention so the linear and angular velocity input vectors must be rotated into the global reference frame and the rover's roll, pitch and yaw rates. For the complete rover pose, the rotation of the linear and angular velocity

inputs must be done separately. Given the rover's orientation, the rotation from the rover frame to the global frame, $\boldsymbol{C}_R^G$, is calculated as

$$\boldsymbol{C}_R^G = \begin{bmatrix} c_{\Phi_y}c_{\Phi_z} & -c_{\Phi_x}s_{\Phi_z} + s_{\Phi_x}s_{\Phi_y}c_{\Phi_z} & s_{\Phi_x}s_{\Phi_z} + c_{\Phi_x}s_{\Phi_y}c_{\Phi_z} \\ c_{\Phi_y}s_{\Phi_z} & c_{\Phi_x}c_{\Phi_z} + s_{\Phi_x}s_{\Phi_y}s_{\Phi_z} & -s_{\Phi_x}c_{\Phi_z} + c_{\Phi_x}s_{\Phi_y}s_{\Phi_z} \\ -s_{\Phi_y} & s_{\Phi_x}c_{\Phi_y} & c_{\Phi_x}c_{\Phi_y} \end{bmatrix}, \tag{3.3}$$

where Φ_z, Φ_y and Φ_x represent successive rotations in the roll-pitch-yaw convention, and c, s denote cosine and sine respectively.

The transformation of the angular velocity input in the rover reference frame to the Euler angle rates, $\dot{\boldsymbol{\Phi}}$, denoted $\boldsymbol{S}_\omega^G$, is calculated as

$$\boldsymbol{S}_\omega^G = \begin{bmatrix} 1 & s_{\Phi_x}t_{\Phi_y} & c_{\Phi_x}t_{\Phi_y} \\ 0 & c_{\Phi_x} & -s_{\Phi_x} \\ 0 & \frac{s_{\Phi_x}}{c_{\Phi_y}} & \frac{c_{\Phi_x}}{c_{\Phi_y}} \end{bmatrix}, \tag{3.4}$$

where t denotes tan. With these two rotations the rover forward kinematic equation is calculated as

$$\begin{bmatrix} \dot{X}_{Rover} \\ \dot{Y}_{Rover} \\ \dot{Z}_{Rover} \\ \dot{\Phi}_x \\ \dot{\Phi}_y \\ \dot{\Phi}_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_R^G & 0 \\ 0 & \boldsymbol{S}_\omega^G \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}, \tag{3.5}$$

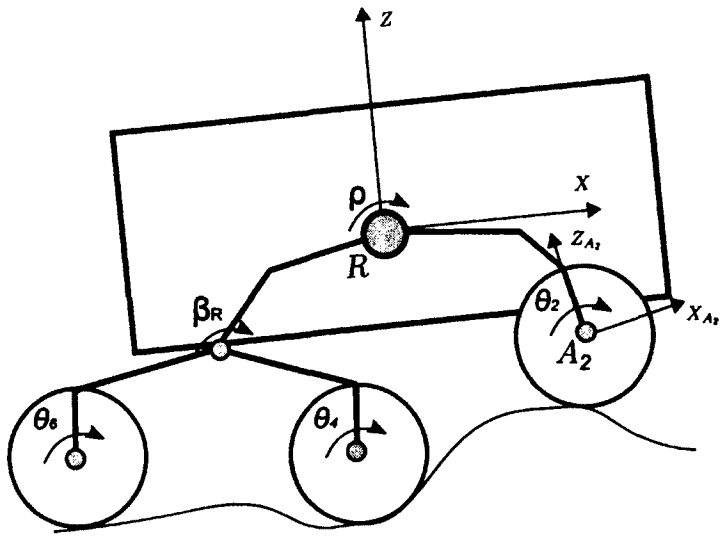


Figure 3.8: Kapvik diagram showing wheel axle coordinate frame, joint angles and wheel rolling angles.

where X_{Rover}, Y_{Rover}, Z_{Rover} are the rover's global position.

3.2.2 *Kapvik* Rover

The *Kapvik* rover has six independently controlled wheels. To determine the complete motion of the rover it would be necessary to include a dynamic model, which is beyond the scope of this thesis. Because the rover is moving at a very slow speed a quasi-dynamic model has been used that makes use of an estimated contact angle rate change, and the measurements from the rocker-bogie potentiometers and the IMU. The kinematic model for an articulated rover used in this thesis is based in part on the model presented in [11]. The rover pose is described by the configuration vector, $\boldsymbol{U} = [X\ Y\ Z\ \Phi_x\ \Phi_y\ \Phi_z]^T \in \mathbb{R}^6$ in the global reference frame, G, whereas the lower case values, (ie. ϕ_x, ϕ_y, ϕ_z) are with respect to the instantaneous rover reference frame, R. Each wheel affects the rover's pose and it is the composite effect of all six rover wheels that will determine the rover's motion. By describing a coordinate frame for each of the wheels, A_i, i = 1...6 the effects of each wheel on the rover

body can be transformed to the rover reference frame by a series of transformations defined by the rover's rocker-bogie linkages and joint angles $\boldsymbol{q} = [\rho, \beta_L, \beta_R]^T \in \mathbb{R}^3$ where ρ is the rocker angle and β_L, β_R are the left and right bogie angles respectively. The transformation from the wheel reference frame to the rover reference frame is denoted $\boldsymbol{T}_{R,Ai}(\boldsymbol{q})$. These last two reference frames and the joint and wheel rolling angles are depicted in Figure 3.8. Each wheel is considered as a rigid disc, with a single contact point. This assumption has been made due to the fact that multiple contact point kinematic analysis is very complex. The wheel is also assumed not to jump off the terrain or penetrate into the terrain. One more coordinate frame, denoted C_i is defined by each wheel's contact points, as shown in Figure 3.9. As can be seen in the figure, the x-axis of this reference frame is aligned with the tangent to the terrain at the contact point and is rotated about the local y-axis by the angle, δ_i, which is the contact angle. A corresponding transformation matrix from the wheel contact frame, C_i, to the wheel frame, A_i, is denoted $\boldsymbol{T}_{Ai,Ci}$ and is given by,

$$\boldsymbol{T}_{Ai,Ci} = \begin{bmatrix} c\delta_i & 0 & s\delta_i & -r_w s\delta_i \\ 0 & 1 & 0 & 0 \\ -s\delta_i & 0 & c\delta_i & -r_w c\delta_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.6}$$

where s and c denote the sine and cosine respectively.

To describe motion, the transformation from one time step, $t-1$, to the next, t of each wheel frame must be calculated. This transformation includes calculations of slip in the wheel reference frame, C_i, visually depicted by Tarokh *et al.* in Figure 3.10. Here η_i describes the side slip of each wheel in the y direction, $r_w\theta_i + \xi_i$ describes the rolling motion of each wheel which includes the desired wheel rate in addition to

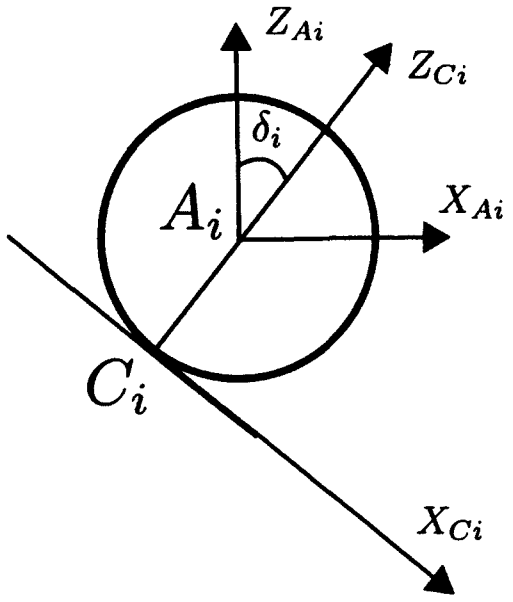


Figure 3.9: Wheel contact coordinate frame

rolling slip in the x direction, and finally ζ_i is the turning slip about the wheel's z axis. It is important to note that even if the environment is assumed to cause no slip, a differential rover such as *Kapvik* when turning will include rotational slip determined from the difference between the left and right wheel rates, $\dot{\theta}_L$ and $\dot{\theta}_R$. Together, these three slip components form the slip vector $\epsilon_i = [\xi_i\ \zeta_i\ \eta_i]^T \in \mathbb{R}^3$. The transformation from one time step to the next due to the wheel rates and slip is denoted $T_{\bar{C}_i,C_i}$ and is described as,

$$\boldsymbol{T}_{\bar{C}_i,C_i} = \begin{bmatrix} c\zeta_i & -s\zeta_i & 0 & r\theta_i + \xi_i \\ s\zeta_i & c\zeta_i & 0 & \eta_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

The transformation from the wheel frame at time step $t-1$ to the rover frame at

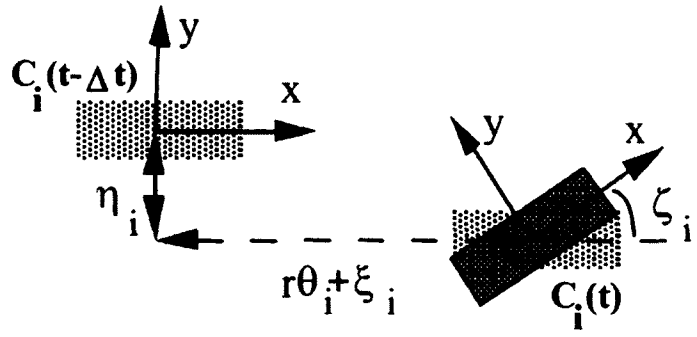


Figure 3.10: Effects of various types of slip shown by Tarokh et al. in [11]

time step t is described as,

$$\boldsymbol{T}_{\bar{C}_i,R} = \boldsymbol{T}_{\bar{C}_i,C_i}(\theta_i, \epsilon_i)\boldsymbol{T}_{Ci,Ai}(\delta_i)\boldsymbol{T}_{Ai,R}(\boldsymbol{q}), \tag{3.8}$$

where $\boldsymbol{T}_{Ci,Ai}$ and $\boldsymbol{T}_{Ai,R}$ are the respective inverses of $\boldsymbol{T}_{Ai,Ci}$ described in (3.6) and $\boldsymbol{T}_{R,Ai}$ which will be described for each wheel in the following section.

To describe the motion, the changes in the configuration vector in the rover reference frame, $\dot{\boldsymbol{u}}_c = [\dot{x}\ \dot{y}\ \dot{z}\ \dot{\phi}_x\ \dot{\phi}_y\ \dot{\phi}_z]^T \in \mathbb{R}^6$, must be determined using the rocker bogie rates, described by $\dot{\boldsymbol{q}}$, the wheel rates, $\dot{\boldsymbol{\theta}} = \left[\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \dot{\theta}_5, \dot{\theta}_6\right]^T \in \mathbb{R}^6$, and the slip rate vector, $\dot{\boldsymbol{\epsilon}} = [\dot{\xi}_1, \dot{\xi}_2, \dot{\xi}_3, \dot{\xi}_4, \dot{\xi}_5, \dot{\xi}_6,\ \dot{\zeta}_1, \dot{\zeta}_2, \dot{\zeta}_3, \dot{\zeta}_4, \dot{\zeta}_5, \dot{\zeta}_6,\ \dot{\eta}_1, \dot{\eta}_2, \dot{\eta}_3, \dot{\eta}_4, \dot{\eta}_5, \dot{\eta}_6]^T \in \mathbb{R}^{18}$. These three sets of quantities make up the input vector for a six wheel articulated rover like *Kapvik*. To do this, as Tarokh shows, an instantaneous coincident transformation matrix is used, $T_{\bar{R},R}$, that describes the rover's transformation from time step $t-1$ to t. This instantaneous coincident matrix can be written as $\boldsymbol{T}_{\bar{R},R} = \boldsymbol{T}_{\bar{R},\bar{C}i}\boldsymbol{T}_{\bar{C}i,R}$. $\boldsymbol{T}_{\bar{R},\bar{C}i}$ is independent of time, so the derivative of $\boldsymbol{T}_{\bar{R},R}$ is

$$\dot{\boldsymbol{T}}_{\bar{R},R} = \boldsymbol{T}_{\bar{R},\bar{C}i}\dot{\boldsymbol{T}}_{\bar{C}i,R}. \tag{3.9}$$

$\dot{\boldsymbol{T}}_{\tilde{C}i,R}$ can be calculated by taking the derivative of (3.8) which is

$$\dot{\boldsymbol{T}}_{\tilde{C}i,R} = \frac{\partial \boldsymbol{T}_{\tilde{C}i,R}}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial \boldsymbol{T}_{\tilde{C}i,R}}{\partial \theta_i}\dot{\theta}_i + \frac{\partial \boldsymbol{T}_{\tilde{C}i,R}}{\partial \epsilon_i}\dot{\epsilon}_i + \frac{\partial \boldsymbol{T}_{\tilde{C}i,R}}{\partial \delta_i}\dot{\delta}_i. \tag{3.10}$$

As Tarokh shows, $\dot{\boldsymbol{T}}_{\tilde{R},R}$ is calculated using the position and orientation rates as

$$\dot{\boldsymbol{T}}_{\tilde{R},R} = \begin{bmatrix} 0 & -\dot{\phi}_z & \dot{\phi}_y & \dot{x} \\ \dot{\phi}_z & 0 & -\dot{\phi}_x & \dot{y} \\ -\dot{\phi}_y & \dot{\phi}_x & 0 & \dot{z} \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.11}$$

substituting (3.10) and (3.11) into (3.9) and equating like elements on both sides the rover configuration rate vector $\dot{\boldsymbol{u}}$ can be found in terms of the input rate vectors: $\dot{\boldsymbol{q}}, \dot{\boldsymbol{\delta}}, \dot{\boldsymbol{\epsilon}}$. The resulting equation is

$$[\dot{x}\ \dot{y}\ \dot{z}\ \dot{\phi}_x\ \dot{\phi}_y\ \dot{\phi}_z]_i^T = \boldsymbol{J}_i[\dot{\boldsymbol{q}}^T\ \dot{\theta}_i\ \dot{\epsilon}_i{}^T\ \dot{\delta}_i]^T, \tag{3.12}$$

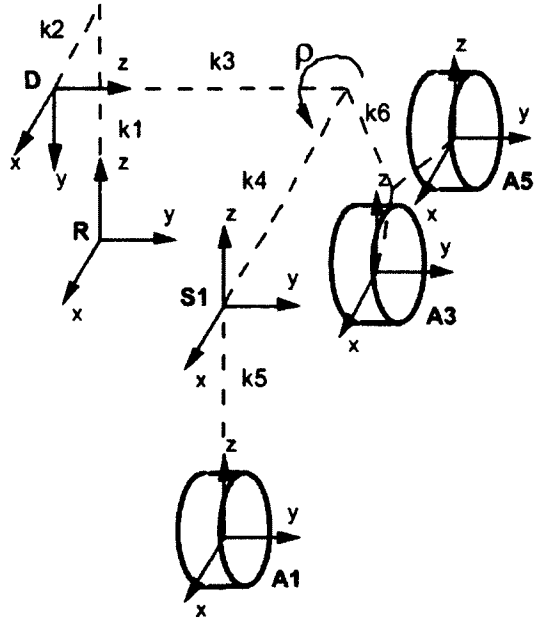
where $\boldsymbol{J}_i$ is a 6 x 8 wheel Jacobian matrix for each wheel and i represents the i^{th} wheel. The equation in (3.12) describes the contribution of each wheel to the overall configuration vector, $\dot{\boldsymbol{u}}_c$. These are averaged together to describe the rover's motion in the rover frame of reference. To use these techniques for *Kapvik*, the $\boldsymbol{q}$ vector that describes the joint angles of the rocker bogie system must still be defined and the rocker-bogie system must be fully described in terms of transformations from the wheel reference frame C_i to the rover reference frame R.

In Section 3.1.1 the *Kapvik* chassis was described, and in Section 3.1.4 the potentiometers and encoders were described, which will be used to form the vectors $\dot{\boldsymbol{q}}$ and $\dot{\boldsymbol{\theta}}$. As was described in those sections, the rocker-bogie system includes a rocker on

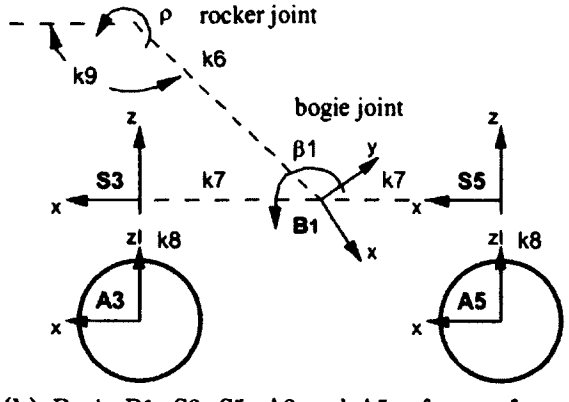
either side that is attached by a differential across the rover body. Because of this, the left rocker's angle, ρ_L, is equal and opposite to the right side rocker angle, ρ_R. Therefore only one side is measured, and is simply defined as ρ. The angle of the rocker as the rover moves over a flat surface is zero. Each of the bogies act independently, and so there are two bogie angles measured and denoted β_L and β_R. When the rover moves over flat ground, $\beta_L = \beta_R = 0$. Together these make up the $\boldsymbol{q}$ vector, $\boldsymbol{q} = [\rho\ \beta_L\ \beta_R]^T$.

While each side's three wheels move at the same rate (ie. $\theta_1 = \theta_3 = \theta_5 = \theta_L$ and $\theta_2 = \theta_4 = \theta_6 = \theta_R$), the kinematic model and actual construction of *Kapvik* provide for the possibility of independently actuating each wheel. This has potential in traction control among other techniques that are beyond the scope of this thesis, however the kinematic model presented in this thesis could be expanded upon to include these. This may require more sensors for the system to be observable such as a velocimeter or load sensors within the Kalman filter.

Just as in [11], a set of reference frames have been defined along the linkages of the rover. Figure 3.11, illustrated by Tarokh et al. in [11], shows the location of the reference frames with respect to one another on the left side of the rover. The first reference frame is simply the rover reference frame, located at the centroid of the rover, R. The differential reference frame is D. B_L and B_R are the left and right bogie reference frames, and S_i, A_i, and C_i are the front axle, wheel, and contact reference frames. To form the transformation matrices from the rover reference frame to the wheel reference frames Denavit-Hartenberg (D-H) parameters, introduced by Denavit and Hartenberg and detailed in [60], are used and defined in Table 3.1 for each reference frame transition as shown in Table 3.1.



(a) Rover R, differential D, S1, A1, A2 and A3 reference frames [11].



(b) Bogie B1, S3, S5, A3 and A5 reference frames [11].

Figure 3.11: Reference frames used on rover's left side.

Table 3.1: *Kapvik* D-H parameters

Frame	d	γ	a	α
D	k_1	0	k_2	-90
B_L	k_3	$k_9 + \rho$	k_6	0
B_R	$-k_3$	$k_9 - \rho$	k_6	0
S_1	ρ	k_3	k_4	90
S_2	$-\rho$	$-k_3$	k_4	90
S_3	$\beta_1 - k_9$	0	k_7	90
S_4	$\beta_2 - k_9$	0	k_7	90
S_5	$\beta_1 - k_9$	0	$-k_7$	90
S_6	$\beta_2 - k_9$	0	$-k_7$	90
A_1	0	$-k_5$	0	0
A_2	0	$-k_5$	0	0
A_3	0	$-k_8$	0	0
A_4	0	$-k_8$	0	0
A_5	0	$-k_8$	0	0
A_6	0	$-k_8$	0	0

In Table 3.1, d is a translation along the initial reference frame's z-axis, γ is a rotation

about the initial frame's z-axis, a is a translation along the x-axis and α is a rotation about the x-axis. The D-H transformation is computed as

$$\boldsymbol{T}_{D-H} = \left[\begin{array}{cccc} cos\gamma & -sin\gamma cos\alpha & sin\gamma sin\alpha & acos\gamma \\ sin\gamma & cos\gamma cos\alpha & -cos\gamma sin\alpha & asin\gamma \\ 0 & sin\alpha & cos\alpha & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \equiv \left[\begin{array}{ccc|c} & Rotation & & Translation \\ \hline 0 & 0 & 0 & 1 \end{array}\right]. \tag{3.13}$$

Each of the k values in Table 3.1 is determined by the dimensions of the rover chassis as detailed in section 3.1.1. These parameters along with the wheel radius are listed in Table 3.2.

Table 3.2: Chassis constants

Constant	Value	Description
k_1	0 m	vertical offset between reference frame R and D
k_2	0 m	forward offset between R and D
k_3	0.356125 m	horizontal distance between D and wheel reference frames
k_4	0.35 m	distance from D to front wheel axis
k_5	0.175 m	height of D from front wheel axis
k_6	0.1904 m	distance along link between rocker joint to bogie joint

Continued on next page

Table 3.2 – continued from previous page

Constant	Value	Description
k_7	0.175 m	length between bogie joint and rear wheel axis
k_8	0.1 m	height of bogie joint from rear wheel axis
k_9	156.8	angle between rocker and bogie joints
r_w	0.075 m	wheel radius

Using the D-H parameters, the transformation from rover frame to contact angle frame for each wheel can be made. The transformation from wheel to rover coordinate frames is formed using (3.13) and successively premultiplying each corresponding row in Table 3.1, written as

$$\begin{aligned} \boldsymbol{T}_{R,Ai} &= (\boldsymbol{T}_{R,D})(\boldsymbol{T}_{D,Si})(\boldsymbol{T}_{Si,Ai}) \qquad i = 1,2 \quad , \\ \boldsymbol{T}_{R,Ai} &= (\boldsymbol{T}_{R,D})(\boldsymbol{T}_{D,B_1})(\boldsymbol{T}_{B_1,Si})(\boldsymbol{T}_{Si,Ai}) \qquad i = 3,5 \quad , \\ \boldsymbol{T}_{R,Ai} &= (\boldsymbol{T}_{R,D})(\boldsymbol{T}_{D,B_2})(\boldsymbol{T}_{B_2,Si})(\boldsymbol{T}_{Si,Ai}) \qquad i = 4,6 \quad . \end{aligned} \tag{3.14}$$

The final transformation from the wheel coordinate frame to the contact reference frame is made by substituting (3.14) into (3.8). Using (3.10) and (3.11), the expression in (3.12) may now be found for each wheel as follows. For wheels 1 and 2, (3.12)

becomes

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{bmatrix}_i = \begin{bmatrix} 0 & J_{x,\theta} & J_{x,\xi} & J_{x,\zeta} & J_{x,\eta} & J_{x,\delta} \\ 0 & J_{y,\theta} & J_{y,\xi} & J_{y,\zeta} & J_{y,\eta} & J_{y,\delta} \\ 0 & J_{z,\theta} & J_{z,\xi} & J_{z,\zeta} & J_{z,\eta} & J_{z,\delta} \\ 0 & 0 & 0 & J_{\phi_x,\zeta} & 0 & J_{\phi_x,\delta} \\ b_i & 0 & 0 & J_{\phi_y,\zeta} & 0 & J_{\phi_y,\delta} \\ 0 & 0 & 0 & J_{\phi_z,\zeta} & 0 & J_{\phi_z,\delta} \end{bmatrix} \begin{bmatrix} \dot{\rho} \\ \dot{\theta}_i \\ \dot{\xi}_i \\ \dot{\zeta}_i \\ \dot{\eta}_i \\ \dot{\delta}_i \end{bmatrix}, \qquad i = 1,2 \qquad (3.15)$$

where $b_i = (-1)^i$. For wheels 3 through 6, (3.12) becomes

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi}_x \\ \dot{\phi}_y \\ \dot{\phi}_z \end{bmatrix}_i = \begin{bmatrix} 0 & J_{x,\beta_L} & J_{x,\beta_R} & r_w c\sigma_i & c\sigma_i & J_{x,\zeta} & 0 & J_{x,\delta} \\ 0 & 0 & 0 & 0 & 0 & J_{y,\zeta} & 1 & 0 \\ 0 & J_{z,\beta_L} & J_{z,\beta_R} & -r_w s\sigma_i & -s\sigma_i & J_{z,\zeta} & 0 & J_{z,\delta} \\ 0 & 0 & 0 & 0 & 0 & s\sigma_i & 0 & 0 \\ b_i & J_{\phi_y,\beta_L} & J_{\phi_y,\beta_R} & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & c\sigma_i & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\rho} \\ \dot{\beta}_L \\ \dot{\beta}_R \\ \dot{\theta}_i \\ \dot{\xi}_i \\ \dot{\zeta}_i \\ \dot{\eta}_i \\ \dot{\delta}_i \end{bmatrix}, \qquad i = 3,4,5,6 \qquad (3.16)$$

where

$$\sigma_i = \begin{cases} \rho + \beta_L + \delta_i & i = 3,5 \\ -\rho + \beta_R + \delta_i & i = 4,6 \end{cases}, \tag{3.17}$$

and all J_i elements are given in appendix A.

By forming the equation in this manner it is easy to see which inputs affect which states. For instance it is apparent taking a glance at (3.15) and (3.16) that the rocker only affects the pitch of the rover, and that the wheel rates (θ_i), rolling slip (ξ_i) and side slip (η_i) have no affect on the orientation. These simple observations give some insight into the motion of the rover.

Taking each of the equations in (3.15) and (3.16) and averaging the resulting $\dot{\boldsymbol{u}}$ value gives the overall configuration rate vector in the instantaneous rover reference frame. The orientation components of the $\dot{\boldsymbol{u}}$ vector, $\dot{\boldsymbol{\phi}}$, are equal to $\boldsymbol{\omega}$, the angular velocity in the rover reference frame, if $\boldsymbol{\phi}$ represents the rotation over an infinitesimally small time interval, which is the case. To transform this into $\dot{\boldsymbol{U}}$, the global configuration rate vector is calculated using the same method used on a 4 wheel differential rover as in (3.5) which transforms the position and orientation into the global frame, written as

$$\begin{bmatrix} \dot{X}_{Rover} \\ \dot{Y}_{Rover} \\ \dot{Z}_{Rover} \\ \dot{\Phi}_x \\ \dot{\Phi}_y \\ \dot{\Phi}_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_R^G & 0 \\ 0 & \boldsymbol{S}_\omega^G \end{bmatrix} \frac{1}{6} \sum_{i=1}^{6} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_i . \tag{3.18}$$

3.3 Hybrid FastSLAM and Kalman Filter Estimation Scheme

The basic FastSLAM 2.0 framework discussed in Section 2.2 assumes a measurement will be taken at each time step, t. While this would be beneficial for localisation, there are a few reasons that this is not feasible within the *Kapvik* framework. First, the LIDAR and tilt unit as designed take too long to do a full 3D scan. Even with only ten 2D scans, the time to complete a scan is on the order of several seconds, at least an order of magnitude larger than the length of each time step. The rover must also be stationary during a scan, unless further data processing is done to account for any movement, which introduces yet more error into the measurements. Second, the processing time for a point cloud of data from a LIDAR scan is fairly substantial, especially given the limited processing power of *Kapvik*. While in this thesis, a specific processing efficiency constraint is not set, the overall speed of the algorithm is kept as low as possible. One solution to these two problems is to limit the number of scans *Kapvik* takes. If *Kapvik* takes less scans, less time will be needed for scanning and processing LIDAR scans.

To solve these problems a hybrid estimation scheme has been developed that makes use of both the FastSLAM algorithm as well as a Kalman filter between scans. The basic algorithm is divided into two parts: the estimation that takes place in between scans, deemed the pre-scan estimation component which is explained in Section 3.3.1 and the post-scan correction that makes use of the FastSLAM algorithm to update both the pose of the rover and the map explained in Section 3.3.2. In Section 3.3.3 the determination of when to take a scan and do the post-scan step is discussed.

3.3.1 Pre-Scan Estimation

In between each scan the rover will propagate N particles forward. Each of these particles represents a hypothesis of the path the rover has taken, as discussed in Section 2.2.2. Each particle is propagated forward each time step, t, by the inputs to the rover forward kinematic model. This step serves as the *a priori* step in a cubature Kalman filter for each of the particles. The Kalman measurement update step is then applied to each particle where each particle state is updated by the measurements from the IMU and sun sensor. This gives a very accurate estimate of the rover's orientation which is very important when performing data association in SLAM. However the estimation of the rover's position is based on dead reckoning methods, and therefore SLAM is used to update the rover position at various intervals. This continues each time step until a scan is taken. What follows is a description of the process and measurement models used in the cubature Kalman filter described in Section 2.1.2.

States and Process Model

The process model follows from the equations in (3.5) and (3.18) depending on the rover being used. In both cases the configuration state's being estimated are:

$$\boldsymbol{s}_t = [X, Y, Z, \Phi_x, \Phi_y, \Phi_z]^T \in \mathbb{R}^6. \tag{3.19}$$

In the case of the *Kapvik* kinematic model, slip rates and contact angle rates are also estimated and used as inputs. This results in an augmented state vector:

$$\boldsymbol{s}_t^a = \left[X, Y, Z, \Phi_x, \Phi_y, \Phi_z, \boldsymbol{\xi}^T, \dot{\boldsymbol{\xi}}^T, \boldsymbol{\zeta}^T, \dot{\boldsymbol{\zeta}}^T, \boldsymbol{\eta}^T, \dot{\boldsymbol{\eta}}^T, \boldsymbol{\delta}^T, \dot{\boldsymbol{\delta}}^T\right]^T \in \mathbb{R}^{54}, \tag{3.20}$$

where

$$\begin{aligned}
\boldsymbol{\xi} &= [\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6]^T \in \mathbb{R}^6, \\
\dot{\boldsymbol{\xi}} &= \left[\dot{\xi}_1, \dot{\xi}_2, \dot{\xi}_3, \dot{\xi}_4, \dot{\xi}_5, \dot{\xi}_6\right]^T \in \mathbb{R}^6, \\
\boldsymbol{\zeta} &= [\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5, \zeta_6]^T \in \mathbb{R}^6, \\
\dot{\boldsymbol{\zeta}} &= \left[\dot{\zeta}_1, \dot{\zeta}_2, \dot{\zeta}_3, \dot{\zeta}_4, \dot{\zeta}_5, \dot{\zeta}_6\right]^T \in \mathbb{R}^6, \\
\boldsymbol{\eta} &= [\eta_1, \eta_2, \eta_3, \eta_4, \eta_5, \eta_6]^T \in \mathbb{R}^6, \\
\dot{\boldsymbol{\eta}} &= [\dot{\eta}_1, \dot{\eta}_2, \dot{\eta}_3, \dot{\eta}_4, \dot{\eta}_5, \dot{\eta}_6]^T \in \mathbb{R}^6, \\
\boldsymbol{\delta} &= [\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6]^T \in \mathbb{R}^6, \\
\dot{\boldsymbol{\delta}} &= \left[\dot{\delta}_1, \dot{\delta}_2, \dot{\delta}_3, \dot{\delta}_4, \dot{\delta}_5, \dot{\delta}_6\right]^T \in \mathbb{R}^6.
\end{aligned}$$

The process model is a non-linear discrete-time model and is propagated using

$$\boldsymbol{s}_t^a = f\left(\boldsymbol{s}_{t-1}^a, \boldsymbol{u}_t\right), \tag{3.21}$$

where f will be defined in (3.28), and $\boldsymbol{u}_t$ is the input vector defined by the inputs of the wheel encoders and potentiometers:

$$\boldsymbol{u}_t = \left[\dot{\rho}, \dot{\beta}_L, \dot{\beta}_R, \dot{\boldsymbol{\theta}}^T\right]^T \in \mathbb{R}^9. \tag{3.22}$$

It can be seen here that $\boldsymbol{u}_t$ requires the rates of values such as the rocker angle and wheel encoders yet the measurements are not rate values. To obtain the rate value the higher order finite difference method described in [61] is applied to the input measurements.

$$\dot{\boldsymbol{z}}_t = \frac{\boldsymbol{z}_t - 4\boldsymbol{z}_{t-1} + 3\boldsymbol{z}_{t-2}}{2T_s}, \tag{3.23}$$

where $\dot{\boldsymbol{z}}_t$ and $\boldsymbol{z}_t$ could represent any of the two types of inputs mentioned above and T_s is the time between time steps.

To calculate (3.21) a numerical integration of the kinematic model in (3.18) must be performed, which is denoted $\dot{\boldsymbol{U}}_t\left(\boldsymbol{s}^a_{t-1}, \boldsymbol{u}_t\right)$, as well as the estimated slip and contact angles, along with their associated rates that augment the state vector. Because no *a priori* information is known about the slip rates and contact angle rates (due to unknown terrain characteristics) they follow a random walk model and the process model for the contact angle rates is simply:

$$\dot{\boldsymbol{\xi}}_{i,t} = \dot{\boldsymbol{\xi}}_{i,t-1}, \tag{3.24}$$

$$\dot{\boldsymbol{\zeta}}_{i,t} = \dot{\boldsymbol{\zeta}}_{i,t-1}, \tag{3.25}$$

$$\dot{\boldsymbol{\eta}}_{i,t} = \dot{\boldsymbol{\eta}}_{i,t-1}, \tag{3.26}$$

$$\dot{\boldsymbol{\delta}}_{i,t} = \dot{\boldsymbol{\delta}}_{i,t-1}. \tag{3.27}$$

These quantities are numerically integrated by the timestep, T_s, to arrive at the slip and contact angle quantities, for instance: $\boldsymbol{\delta}_{i,t} = \boldsymbol{\delta}_{i,t-1} + \dot{\boldsymbol{\delta}}_{i,t-1}T_s$. While there are issues with the observability of slip and contact angles [13] for articulated rovers, this process model will suffice due to the use of SLAM. After each SLAM update the augmented state estimates are reset to zero and the process model continues onwards. This is valid because the slip rates and contact angle rates will be zero at the end of each SLAM update, due to the rover not moving. Any accrued error in the estimation of these values will be reduced by the SLAM update when the particle that most closely fits the rover motion is weighted highly. Using this information, f can be constructed as:

$$f = s^a_{t-1} + \begin{bmatrix} \dot{U}(s^a_{t-1}, u_t) \\ \dot{\xi}_{i,t-1} \\ 0_{6\times1} \\ \dot{\zeta}_{i,t-1} \\ 0_{6\times1} \\ \dot{\eta}_{i,t-1} \\ 0_{6\times1} \\ \dot{\delta}_{i,t-1} \\ 0_{6\times1} \end{bmatrix} T_s. \tag{3.28}$$

For most terrain, typical rover motions and an adequate number of particles in SLAM, this should be fine. However for more extreme terrain the number of particles that would be needed to cover all possibilities may grow too large. In this case additional sensors may be needed to handle these observability issues such as load sensors that allow the contact angle to be directly measured within a fully dynamic model.

Measurements

The external measurements during pre-scan estimation available to *Kapvik* come from the IMU and sun sensor. The IMU provides a raw measurement of $z_{IMU} = [a_x, a_y, a_z, \omega_x, \omega_y, \omega_z]^T \in \mathbb{R}^6$ which represent measurements of acceleration and angular velocity along the three orthogonal axis of the rover reference frame. If for any reason the encoders and potentiometers are not available on *Kapvik* or are providing noisy results (such as moving over very bumpy terrain), it may be beneficial

to numerically integrate the accelerometer and gyro measurements for an estimate of position and orientation. The switching back and forth between the two forms of measurements is a technique called gyrodometry [62]. However, this hasn't been implemented in this thesis and therefore the gyro measurements are not used. The accelerometer measurements allow for a measurement of the rover's pitch and roll to be constructed using the direction of the gravity vector sensed by the IMU. It is important to note that this method will work under slow acceleration and steady-state conditions. Because *Kapvik* travels very slowly (approximately 2 cm/s) it does not experience large accelerations in comparison to the affects of gravity. The roll and pitch measurements are calculated according to:

$$z_{\Phi_x} = \tan^{-1}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right), \tag{3.29}$$

$$z_{\Phi_y} = \tan^{-1}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right), \tag{3.30}$$

it is important to note that these measurements depend on the rover maintaining a roll and pitch between $-\pi/2$ and $\pi/2$ which is reasonable given the environment the rover operates in. These measurements are direct measurements of Φ_x and Φ_y.

The sun sensor provides a raw measurement of $\boldsymbol{z}_{SS} = [ss_x, ss_y, ss_z]^T \in \mathbb{R}^3$ that corresponds to the unit vector pointed toward the sun from the sun sensor along the three orthogonal axis of the rover reference frame as shown in Figure 3.12. It is shown by Furgale *et al.* in [63] that through a series of transformations a corresponding heading $\boldsymbol{z}_{\Phi_z}$ can be provided, also a direct measurement of Φ_z. For this reason the simulation does not simulate the sun sensor, but simply provides a direct measurement of Φ_z with Gaussian, zero mean, white noise. The sun sensor has been tested and verified to work within a Kalman Filter on Kapvik using an implementation of [63] but the measurement model was not developed by the author and so was not used

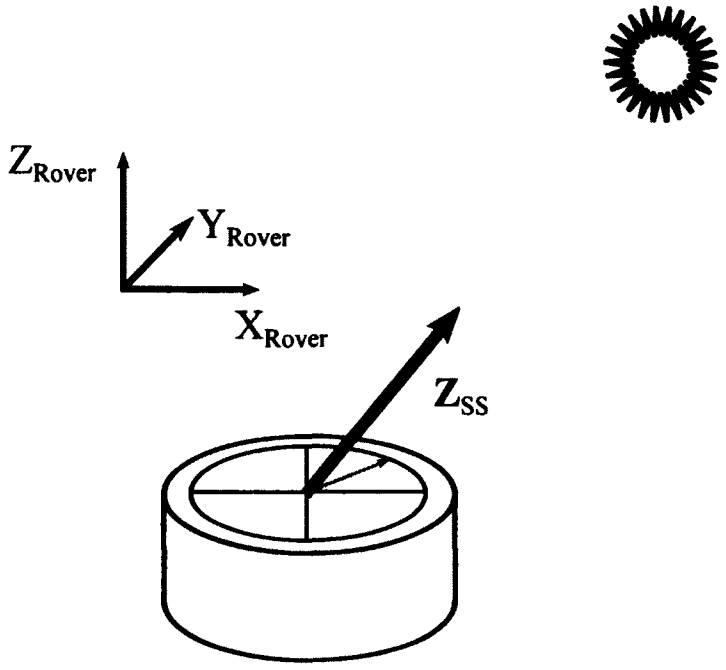


Figure 3.12: Sun sensor measurement with respect to the rover reference frame

for the simulation in this thesis.

Measurement Model

The measurement model h, shown in (3.32), predicts the measurements of each of the absolute sensors in Section 3.1 that make up the measurement vector, $\boldsymbol{z}$, using the current augmented state estimate, $\hat{\boldsymbol{s}}_t^a$. The predicted measurement vector is denoted $\hat{\boldsymbol{y}}_t$,

$$\hat{\boldsymbol{y}}_t = h\left(\hat{\boldsymbol{s}}_t^a\right) \in \mathbb{R}^3. \tag{3.31}$$

The measurement vector includes measurements of the rover's orientation (Φ_x, Φ_y, Φ_z). To form this measurement vector, the IMU and sun sensor measurements are combined as discussed in Section 3.1. Because Φ_x, Φ_y, Φ_z make up parts of

the state vector, $\boldsymbol{s}_t^a$, the predicted measurement vector is straight forward:

$$\hat{\boldsymbol{y}}_t = \left[\hat{\Phi}_x, \hat{\Phi}_y, \hat{\Phi}_z\right]^T. \tag{3.32}$$

3.3.2 Post-Scan Localization and Mapping

In Section 2.2.2 FastSLAM was discussed and the two versions of the basic algorithm were compared. In most cases, FastSLAM 2.0 is superior to FastSLAM 1.0 due to the incorporation of new measurements into the proposal distribution instead of completely relying on the motion model of the rover. However, FastSLAM 2.0 also uses inaccurate linear approximations and Jacobian calculations. Calculating the Jacobian matrices is difficult and the inaccurate approximation of the posterior covariance will affect the filter accuracy and consistency.

For these reasons, a nonlinear Kalman filter was desired for use on *Kapvik*. In [33], it was shown that by using an unscented Kalman filter (UKF), FastSLAM 2.0 performed substantially better. The UKF was used to compute the proposal distribution with measurement updates as well as updating the feature estimates.

However, in practise the author has noticed that the covariance matrix will often become negative. This can happen due to the matrix square root step in the UKF, matrix inversion, amplified roundoff errors and subtraction of two positive definite matrices [16]. To alleviate this the square root Cubature Kalman filter (SCKF) discussed in Section 2.1.2 is used, which avoids computing a matrix inversion explicitly. There are also potential computation savings in the way that the matrices are computed, however this is not implemented for this thesis. It has also been shown that the SCKF is more numerically stable than a comparable square root UKF [16].

One of the contributions of this thesis is the implementation of the SCKF in the SLAM framework. In addition, much of the literature bypasses the data association problem which significantly affects the way the proposal distribution is calculated in

the first part of the SLAM algorithm. For SLAM to run in a completely unknown 3D environment like *Kapvik* requires, it is obvious that the data association problem must be handled. The Hungarian algorithm described in Section 2.4 was used for this purpose. To implement data association in FastSLAM 2.0 adjustments to the FastSLAM algorithm were made which will be discussed.

The LIDAR measurement model will be explained in the following section as it is used to update the proposal distribution. Next, using the measurement model and the CKF modifications to the FastSLAM 2.0 algorithm the proposal distribution is updated and a pose estimate is sampled for each particle. To do this, data association has been implemented that runs in tandem with the proposal distribution update. Afterwards, feature estimates in the map are updated, new features are assigned and the importance weight for each particle is calculated. Finally the particles are resampled and the algorithm then moves back to the Kalman filter estimation between scans discussed in Section 3.3.1.

LIDAR Inverse Measurement Model

A typical LIDAR will return a range and bearing which can be converted to a Cartesian position with knowledge of the pose of the LIDAR device. To avoid the expense of a 3D LIDAR scanner, 2D LIDAR scanners can be made to tilt about an axis using a tilt device, and at set intervals take 2D scans as shown in Figure 3.13. The combination of 2D scans and tilt intervals can be used to create a 3D scan that approximates scans that a 3D scanner creates. For the purposes of this thesis, the scans are taken with the rover in a stationary pose, devoid of any moving objects such as people or vehicles. The same concepts can easily be extended to full 3D LIDAR scanners.

For this thesis, a Directed Perception PTU D46 is used as the tilt unit, and a Hukuyo URG-04LX range finder as well as a SICK LMS 111 LIDAR, shown in Figure 3.3, are used for the LIDAR scanners. The LIDAR and tilt unit are mounted

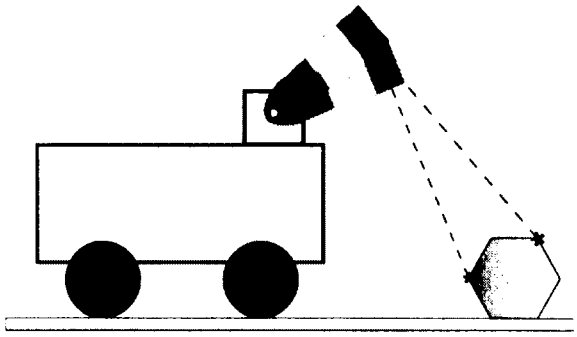


Figure 3.13: 2D Scanner used for 3D scans

together on a Pioneer and Husky rover respectively, centred at the front of the rover as seen in Figure 3.14.

By estimating the pose of the rover over its traverse, a transformation matrix can be used to convert the range, bearing, and tilt values returned by the combination of a LIDAR and tilt unit to data in global Cartesian components. The range and bearing values are first converted to Cartesian points in the LIDAR frame of reference (represented by L in the following equations). The LIDAR frame of reference is aligned such that the X' axis is located along the zero bearing as shown in Figure 3.15 and the Y' axis is aligned with the tilt unit's axis of rotation. Therefore the Z' coordinate of each scanned point will remain the same in the LIDAR reference frame, being equal to the distance between the LIDAR and the tilt axis of rotation, L_a. The position of each scanned point in the LIDAR frame is defined as

$$\begin{aligned} X^L_{meas,i,j} &= r_i \cos\psi_i, \\ Y^L_{meas,i,j} &= r_i \sin\psi_i \qquad i = 1, ...N \quad , \\ Z^L_{meas,i,j} &= L_a, \end{aligned} \tag{3.33}$$

where i is the measurement index for one 2D scan, N is the number of measurements in one 2D scan, r_i is the range measurement, ψ_i is the bearing measurement and L_a is the distance between the axis of rotation and the LIDAR head, aligned with the

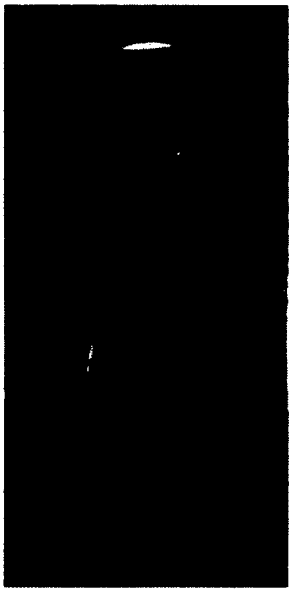


Figure 3.14: Pioneer rover with tilt unit and LIDAR attached

Z' axis of the LIDAR frame.

To obtain the global position the rotation between the LIDAR frame to the rover frame, $C^R_{L,j}$, is calculated as

$$C^R_{L,j} = \begin{bmatrix} \cos\theta_j & 0 & \sin\theta_j \\ 0 & 1 & 0 \\ -\sin\theta_j & 0 & \cos\theta_j \end{bmatrix} \qquad j = 1, ...M \quad , \tag{3.34}$$

where j is the tilt angle index for one 2D scan, M is the total number of tilt angles, θ is the angle about the tilt axis the LIDAR is oriented, with respect to the rover's X axis as shown in Figure 3.15. The translation will be given by the position of the LIDAR in the rover reference system as shown in Figure 3.16. Together the transformation from the LIDAR reference frame to the rover reference frame is formed as

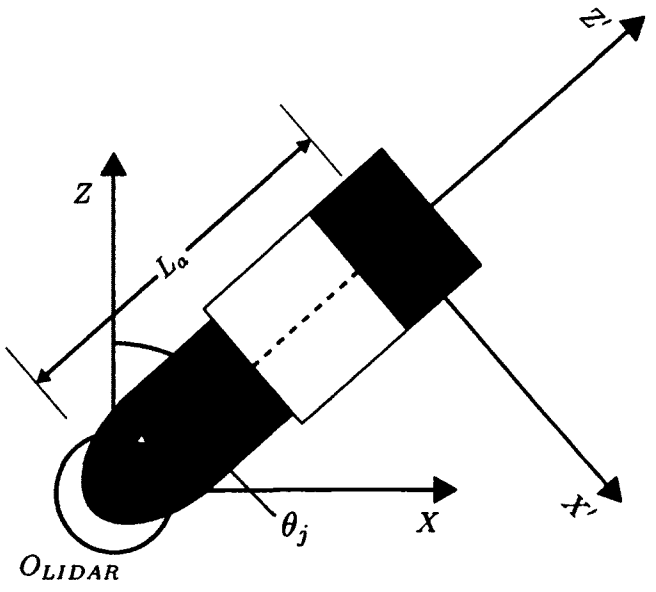


Figure 3.15: 2D Scan in LIDAR reference frame

$$\boldsymbol{T}_{L,j}^{R} = \left[\begin{array}{ccc|c} & \boldsymbol{C}_{L,j}^{R} & & \begin{array}{c} -X_{LIDAR} \\ -Y_{LIDAR} \\ -Z_{LIDAR} \end{array} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]. \tag{3.35}$$

Now, given the rover's pose estimate rotation between the rover frame and the global frame, $\boldsymbol{C}_{R}^{G}$ from (3.3) is used to rotate from the rover frame to the global frame

$$\boldsymbol{T}_{R}^{G} = \left[\begin{array}{ccc|c} & \boldsymbol{C}_{R}^{G} & & \begin{array}{c} -X_{Rover} \\ -Y_{Rover} \\ -Z_{Rover} \end{array} \\ \hline 0 & 0 & 0 & 1 \end{array}\right], \tag{3.36}$$

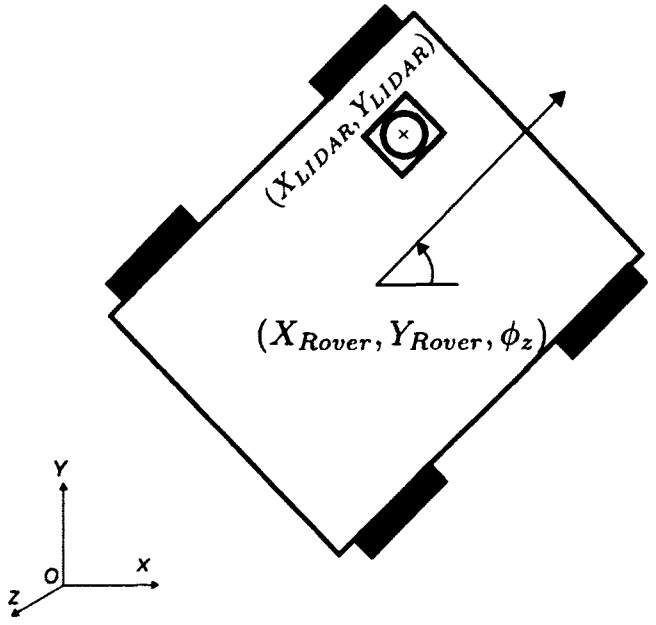


Figure 3.16: LIDAR position in rover reference frame

where X_{Rover}, Y_{Rover}, Z_{Rover} represent the position of the rover in the global reference frame. Finally, the position of each measurement in the global reference frame is calculated as

$$\begin{bmatrix} X^G_{meas,i,j} \\ Y^G_{meas,i,j} \\ Z^G_{meas,i,j} \\ 1 \end{bmatrix} = \boldsymbol{T}^G_R \boldsymbol{T}^R_{L,j} \begin{bmatrix} X^L_{meas,i,j} \\ Y^L_{meas,i,j} \\ Z^L_{meas,i,j} \\ 1 \end{bmatrix}, \tag{3.37}$$

which results in $N \times M$ measurements that make up a single 3D point cloud. These measurements that have been transformed into the global reference frame and are now ready to be added to the map in the SLAM algorithm. Once again, referring to Algorithm 2.2, the features are added along with an associated covariance, $\boldsymbol{P}^{[m]}_{\hat{n}_t,t}$.

LIDAR Measurement Model

A measurement model must be developed so that a prediction, $\hat{\boldsymbol{z}}$, can be made for each feature within the rover's perceptual range. Using the map, N_{t-1}, the rover's pose, $\boldsymbol{s}_t = [X_{Rover}, Y_{Rover}, Z_{Rover}, \Phi_x, \Phi_y, \Phi_z]^T$, and the constraints defined by the tilt unit and LIDAR, $\hat{\boldsymbol{z}}$ can be computed.

The first step to forming a measurement model is to define the reference systems for the LIDAR with respect to the rover. Two different lasers were used in testing so the model was kept fairly general allowing parameters to be changed with different mounting locations for the LIDAR on a variety of rovers.

Starting from the global reference frame the difference in the feature's estimated position, $\hat{\boldsymbol{\lambda}}_{t-1} \in \mathbb{R}^3$, and the rover's estimated position is computed as:

$$\boldsymbol{d}_G = \hat{\boldsymbol{\lambda}}_{t-1} - \begin{bmatrix} X_{Rover} \\ Y_{Rover} \\ Z_{Rover} \end{bmatrix}, \tag{3.38}$$

$\boldsymbol{d}_G$ is then rotated into the rover's reference frame

$$\boldsymbol{d}_R = \boldsymbol{C}_G^R \boldsymbol{d}_G, \tag{3.39}$$

where $\boldsymbol{C}_G^R$ is computed as:

$$\boldsymbol{C}_G^R = \begin{bmatrix} c_{\Phi_y} c_{\Phi_z} & c_{\Phi_y} s_{\Phi_z} & -s_{\Phi_y} \\ -c_{\Phi_x} s_{\Phi_z} + s_{\Phi_x} s_{\Phi_y} c_{\Phi_z} & c_{\Phi_x} c_{\Phi_z} + s_{\Phi_x} s_{\Phi_y} s_{\Phi_z} & s_{\Phi_x} c_{\Phi_y} \\ s_{\Phi_x} s_{\Phi_z} + c_{\Phi_x} s_{\Phi_y} c_{\Phi_z} & -s_{\Phi_x} c_{\Phi_z} + c_{\Phi_x} s_{\Phi_y} s\Phi_z & c_{\Phi_x} c_{\Phi_y} \end{bmatrix}. \tag{3.40}$$

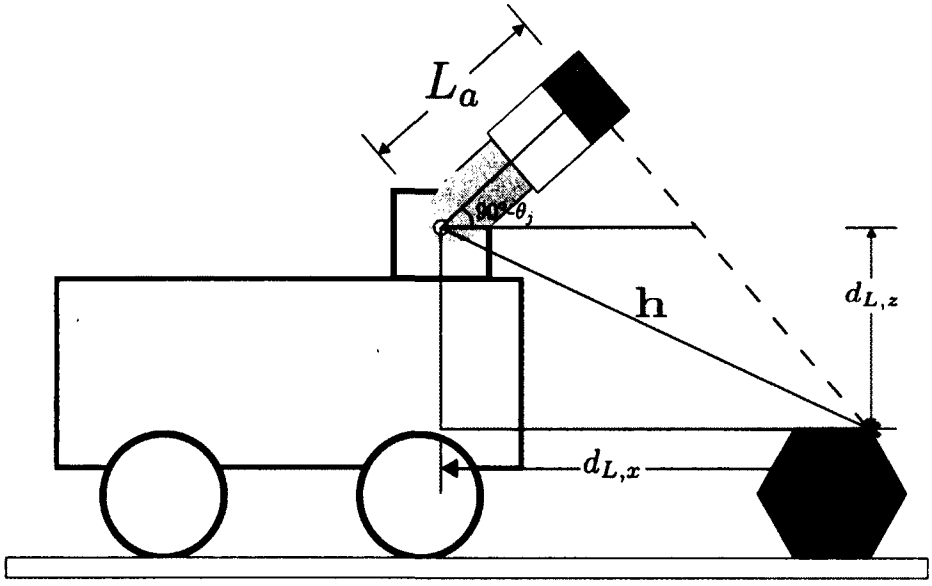


Figure 3.17: Visualization of the trigonometry used for the LIDAR model calculations

The difference between $\boldsymbol{d}_R$ and the LIDAR position in the rover reference frame, denoted $\boldsymbol{d}_L$ is then computed as:

$$\boldsymbol{d}_L = \boldsymbol{d}_R - \begin{bmatrix} X_{LIDAR} \\ Y_{LIDAR} \\ Z_{LIDAR} \end{bmatrix}, \tag{3.41}$$

$\boldsymbol{d}_L$ is the distance between the base of the LIDAR and the position of a feature in the map in the rover reference frame. If the LIDAR head is aligned along the z axis of the tilt unit, as shown in Figure 3.17, then through some trigonometric calculations the associated measurements of range, $\hat{r}$, bearing, $\hat{\psi}$ and tilt, $\hat{\theta}$ can be calculated. Using this diagram it is straightforward to show that

$$h = \sqrt{d_{L,x}^2 + d_{L,z}^2}, \tag{3.42}$$

and the tilt is calculated as

$$\hat{\theta} = \frac{\pi}{2} - \cos^{-1}\left(\frac{L_a}{h}\right) + \tan^{-1}\left(\frac{d_{L,z}}{d_{L,x}}\right). \tag{3.43}$$

Using the predicted tilt, $\hat{\theta}$, the predicted range, $\hat{r}$, and bearing, $\hat{\psi}$, can be found by first finding the difference between the LIDAR head position and the feature position in the rover's reference frame, denoted $\boldsymbol{d}_H^R$

$$\boldsymbol{d}_H^R = \begin{bmatrix} d_{L,x} - L_a \cos\left(\hat{\theta} + \frac{\pi}{2}\right) \\ d_{L,y} \\ d_{L,z} - L_a \sin\left(\hat{\theta} + \frac{\pi}{2}\right) \end{bmatrix}. \tag{3.44}$$

Rotating this difference into the tilt unit's frame of reference will allow the computation of the predicted range and bearing:

$$\boldsymbol{d}_H^L = \boldsymbol{C}_R^L \boldsymbol{d}_H^R, \tag{3.45}$$

where $\boldsymbol{C}_R^L$ is computed as

$$\boldsymbol{C}_R^L = \begin{bmatrix} \cos\hat{\theta} & 0 & -\sin\hat{\theta} \\ 0 & 1 & 0 \\ \sin\hat{\theta} & 0 & \cos\hat{\theta} \end{bmatrix}. \tag{3.46}$$

Using the range, $\hat{r}$, is computed as the norm of $\boldsymbol{d}_H^R$

$$\hat{r} = \| \boldsymbol{d}_H^R \|. \tag{3.47}$$

The bearing, $\hat{\psi}$ can be calculated as

$$\hat{\psi} = \tan^{-1}\left(\frac{(d_H^R)_y}{(d_H^R)_x}\right). \tag{3.48}$$

For the purposes of the rest of this section, the preceding calculations leading to $\hat{\boldsymbol{z}} = [\hat{r}, \hat{\psi}, \hat{\theta}]^T$ are represented by the function h such that:

$$h(\boldsymbol{s}_t, \hat{\boldsymbol{\lambda}}_{t-1}) = [\hat{r}, \hat{\psi}, \hat{\theta}]^T \in \mathbb{R}^3. \tag{3.49}$$

Pose Update

To update the pose, FastSLAM 2.0 updates the proposal distribution with the latest LIDAR measurements. If there are no features currently in the map, then the following steps and the feature update step are skipped and the measurements are added to the map as they are. However, if there are features in the map, the pose will be updated. Unlike FastSLAM 2.0, this thesis incorporates a square root cubature Kalman filter approach that does not linearize the motion or measurement models. A description of the Cubature Kalman Filter is contained in Section 2.1.2.

For each $\hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]}$ feature (out of a total $N_{\lambda,t-1}^{[m]}$ features), a set of $2L$ measurement prediction sigma points are generated where L is the size of the state vector using the latest set of state sigma points, $\boldsymbol{\chi}_t^{[i][m]}$, estimated from the pre-scan localization. These two sets of sigma points are used in (3.53)-(3.58) to calculate the innovation square root covariance, $\boldsymbol{S}_{zz,t}^{[m]}$, and the cross-covariance, $\boldsymbol{P}_{xz,k,t}^{[m]}$, which are then used to compute the appropriate Kalman gain, $\boldsymbol{K}_t^{[m]}$, and vehicle square root covariance,

$\boldsymbol{S}_{P,k,t}$:

$$\boldsymbol{\xi}_i = \sqrt{L}\,(\boldsymbol{I}_{L\times L})_i \qquad i = 1,...,L \quad , \tag{3.50}$$

$$\boldsymbol{\xi}_{i+L} = -\sqrt{L}\,(\boldsymbol{I}_{L\times L})_i \qquad i = 1,...,L \quad , \tag{3.51}$$

$$\boldsymbol{\chi}_t^{[i][m]} = \boldsymbol{S}_{P,t-1}\boldsymbol{\xi}_i + \boldsymbol{s}_t^a \qquad i = 1,...,2L \quad , \tag{3.52}$$

where the i on the right hand side of (3.50) and (3.51) represents the i^{th} column of $\boldsymbol{I}_{L\times L}$. Using the sigma points, the rest of the values are calculated:

$$\hat{\boldsymbol{Z}}_t^{[i][m]} = h\left(\boldsymbol{\chi}_t^{[i][m]}, \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]}\right), \tag{3.53}$$

$$\hat{\boldsymbol{z}}_t^{[m]} = \sum_{i=0}^{2L} w_c^{[i]}\hat{\boldsymbol{Z}}_t^{[i][m]}, \tag{3.54}$$

$$\boldsymbol{S}_{zz,t}^{[m]} = Tria\left(\left[\frac{1}{\sqrt{2L}}\left[\hat{\boldsymbol{Z}}_t^{[1][m]} - \hat{\boldsymbol{z}}_t^{[m]} \quad \hat{\boldsymbol{Z}}_t^{[2][m]} - \hat{\boldsymbol{z}}_t^{[m]} ... \hat{\boldsymbol{Z}}_t^{[2L][m]} - \hat{\boldsymbol{z}}_t^{[m]}\right] \qquad \boldsymbol{S}_R\right]\right), \tag{3.55}$$

$$\boldsymbol{P}_{xz,k,t}^{[m]} = \frac{1}{2L}\sum_{i=1}^{2L}\left(\boldsymbol{\chi}_t^{[i][m]} - \boldsymbol{s}_{t|t-1}^{[m]}\right)\left(\hat{\boldsymbol{Z}}_t^{[i][m]} - \hat{\boldsymbol{z}}_t^{[m]}\right)^T, \tag{3.56}$$

$$\boldsymbol{K}_t^{[m]} = \left(\boldsymbol{P}_{xz,k,t}/\left(\boldsymbol{S}_{zz,t}^{[m]}\right)^T\right)/\boldsymbol{S}_{zz,t}^{[m]}, \tag{3.57}$$

$$\begin{aligned}\boldsymbol{S}_{P,k,t}^{[m]} = Tria\Bigg(\Bigg[&\frac{1}{\sqrt{2L}}\left[\boldsymbol{\chi}_t^{[1][m]} - \boldsymbol{s}_{t|t-1}^{[m]} \quad \boldsymbol{\chi}_t^{[2][m]} - \boldsymbol{s}_{t|t-1}^{[m]} \cdots \boldsymbol{\chi}_t^{[2L][m]} - \boldsymbol{s}_{t|t-1}^{[m]}\right] \\ &- \boldsymbol{K}_t^{[m]}\frac{1}{\sqrt{2L}}\left[\hat{\boldsymbol{Z}}_t^{[1][m]} - \hat{\boldsymbol{z}}_t^{[m]} \quad \hat{\boldsymbol{Z}}_t^{[2][m]} - \hat{\boldsymbol{z}}_t^{[m]} ... \hat{\boldsymbol{Z}}_t^{[2L][m]} - \hat{\boldsymbol{z}}_t^{[m]}\right] \qquad \boldsymbol{K}_t^{[m]}\boldsymbol{S}_R\Bigg]\Bigg).\end{aligned} \tag{3.58}$$

Using $\boldsymbol{K}_t^{[m]}$ and $\boldsymbol{S}_{P,k,t}^{[m]}$, each measurement, denoted $\boldsymbol{z}_{j,t}$ $(j = 1...N_{z,t})$, is cycled through to calculate an updated state, $\boldsymbol{s}_{j,k,t}^{[m]}$, as well as a likelihood, $p_{j,k}$, that the

measurement $z_{j,t}$ came from feature $\hat{\lambda}_{k,t-1}^{[m]}$:

$$\mu_{s_t,j,k} = s_{t|t-1}^{[m]} + K_t^{[m]} \left(z_{j,t} - \hat{z}_t^{[m]} \right). \tag{3.59}$$

$$s_{j,k,t}^{[m]} \sim \mathcal{N} \left(\mu_{s_t,j,k}, S_{P,t}^{[m]} \right), \tag{3.60}$$

$$\hat{z}_{j,k} = h \left(s_{j,k,t}^{[m]}, \hat{\lambda}_{k,t-1}^{[m]} \right), \tag{3.61}$$

$$p_{j,k}^{[m]} = \left| 2\pi (S_{zz,t}^{[m]})^T S_{zz,t}^{[m]} \right|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} \left(\left((z_{j,t} - \hat{z}_{j,k})^T / \left(S_{zz,t}^{[m]} \right)^T \right) / S_{zz,t}^{[m]} \right) (z_{j,t} - \hat{z}_{j,k}) \right\}. \tag{3.62}$$

The remaining likelihoods that make up $p^{[m]}$ represent the likelihood of a new measurement. These elements are set to a tunable parameter, p_0:

$$p_{j,e+N_{\lambda,t-1}^{[m]}}^{[m]} = p_0 \qquad e = 1 \ldots N_{z,t}. \tag{3.63}$$

With the likelihood matrix $p^{[m]}$ computed, data association can be done using the Hungarian algorithm described in Section 2.4. The set of data associations is represented by:

$$\hat{c} = HungarianAlgorithm(p^{[m]}), \tag{3.64}$$

where $\hat{c}$ is a matrix equal in size and dimension to $p^{[m]}$. Elements are either 1 or 0, where 1 represents features each measurement has been assigned to.

If no measurements are assigned to a feature, then the pose is not updated and the algorithm continues on to add the features to the map. However, if any measurements have been assigned to features, the pose and vehicle covariance are updated to the weighted average of $s_{j,k,t}^{[m]}$ and $S_{P,k,t}^{[m]}$ that correspond to the associated measurement

and feature pairs (i.e. $\hat{c}(k) > 0$):

$$\boldsymbol{s}_t^{[m]} = \frac{\sum_{\substack{k=1 \\ \hat{c}(k)>0}}^{N_{\lambda,t-1}^{[m]}} p_{\hat{c}(k),k}^{[m]} \cdot \boldsymbol{s}_{\hat{c}(k),k,t}^{[m]}}{\sum_{\substack{k=1 \\ \hat{c}(k)>0}}^{N_{\lambda,t-1}^{[m]}} p_{\hat{c}(k),k}^{[m]}}, \tag{3.65}$$

$$\boldsymbol{S}_{P,t}^{[m]} = \frac{\sum_{\substack{k=1 \\ \hat{c}(k)>0}}^{N_{\lambda,t-1}^{[m]}} p_{\hat{c}(k),k}^{[m]} \cdot \boldsymbol{S}_{P,k,t}^{[m]}}{\sum_{\substack{k=1 \\ \hat{c}(k)>0}}^{N_{\lambda,t-1}^{[m]}} p_{\hat{c}(k),k}^{[m]}}. \tag{3.66}$$

Feature Update

The feature update will adjust the position estimate of each of the associated features from $\hat{\boldsymbol{c}}$ using the measurement update step of the CKF. With each update, the contribution of the feature ($\hat{w}$) to the importance weight of each particle ($w^{[m]}$) is calculated to take into account the probability of the assignment in $\hat{\boldsymbol{c}}$.

To begin, a set of $2n$ cubature points are generated, where n is the dimension of the feature state, using the previous estimate of the associated feature:

$$\boldsymbol{\xi}_i = \sqrt{n}\,(\boldsymbol{I}_{n\times n})_i \qquad i = 1, ..., n \quad , \tag{3.67}$$

$$\boldsymbol{\xi}_{i+n} = -\sqrt{n}\,(\boldsymbol{I}_{n\times n})_i \qquad i = 1, ..., n \quad , \tag{3.68}$$

$$\boldsymbol{\chi}_{\lambda_k}^{[i][m]} = \boldsymbol{S}_{\lambda_k,t-1}^{[m]}\boldsymbol{\xi}_i + \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]} \qquad i = 1, ..., 2n \quad , \tag{3.69}$$

These are then propagated through the measurement model:

$$\bar{\boldsymbol{Z}}_t^{[i][m]} = h(\boldsymbol{s}_t^{[m]}, \boldsymbol{\chi}_{\lambda_k}^{[i][m]}) \quad i = 0, ..., 2n \quad , \tag{3.70}$$

$$\bar{\boldsymbol{z}}_t^{[m]} = \frac{1}{\sqrt{2n}} \sum_{i=1}^{2n} \bar{\boldsymbol{Z}}_t^{[i][m]}, \tag{3.71}$$

The innovation square root covariance, $\bar{\boldsymbol{S}}_{zz,t}^{[m]}$, cross covariance, $\bar{\boldsymbol{P}}_{xz,t}^{[m]}$, and Kalman

gain, $\bar{\boldsymbol{K}}_t^{[m]}$ are then computed as:

$$\bar{\boldsymbol{S}}_{zz,t}^{[m]} = Tria\left(\left[\frac{1}{\sqrt{2n}}\left[\bar{\boldsymbol{Z}}_t^{[1][m]} - \bar{\boldsymbol{z}}_t^{[m]} \quad \bar{\boldsymbol{Z}}_t^{[2][m]} - \bar{\boldsymbol{z}}_t^{[m]} \ldots \bar{\boldsymbol{Z}}_t^{[2n][m]} - \bar{\boldsymbol{z}}_t^{[m]}\right] \qquad \boldsymbol{S}_R\right]\right), \tag{3.72}$$

$$\bar{\boldsymbol{P}}_{xz,t}^{[m]} = \frac{1}{2n}\sum_{i=1}^{2n}\left(\boldsymbol{\chi}_{\lambda_k}^{[i][m]} - \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]}\right)\left(\bar{\boldsymbol{Z}}_t^{[i][m]} - \bar{\boldsymbol{z}}_t^{[m]}\right)^T, \tag{3.73}$$

$$\bar{\boldsymbol{K}}_t^{[m]} = \left(\bar{\boldsymbol{P}}_{xz,t} / \left(\bar{\boldsymbol{S}}_{zz,t}^{[m]}\right)^T\right) / \bar{\boldsymbol{S}}_{zz,t}^{[m]}. \tag{3.74}$$

The estimated mean and covariance of the k^{th} feature are then updated by:

$$\hat{\boldsymbol{\lambda}}_{k,t}^{[m]} = \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]} + \bar{\boldsymbol{K}}_t^{[m]}\left(\boldsymbol{z}_{\hat{c}(k),t} - \bar{\boldsymbol{z}}_t^{[m]}\right), \tag{3.75}$$

$$\begin{aligned}\boldsymbol{S}_{\lambda_k,t}^{[m]} = Tria\Bigg(\Bigg[&\frac{1}{\sqrt{2n}}\left[\boldsymbol{\chi}_{\lambda_k}^{[1][m]} - \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]} \quad \boldsymbol{\chi}_{\lambda_k}^{[2][m]} - \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]} \cdots \boldsymbol{\chi}_{\lambda_k}^{[2n][m]} - \hat{\boldsymbol{\lambda}}_{k,t-1}^{[m]}\right] \\ &- \bar{\boldsymbol{K}}_t^{[m]}\frac{1}{\sqrt{2n}}\left[\bar{\boldsymbol{Z}}_t^{[1][m]} - \bar{\boldsymbol{z}}_t^{[m]} \quad \bar{\boldsymbol{Z}}_t^{[2][m]} - \bar{\boldsymbol{z}}_t^{[m]} \ldots \bar{\boldsymbol{Z}}_t^{[2n][m]} - \bar{\boldsymbol{z}}_t^{[m]}\right] \qquad \bar{\boldsymbol{K}}_t^{[m]}\boldsymbol{S}_R\Bigg]\Bigg).\end{aligned} \tag{3.76}$$

Finally, the importance weight contribution is calculated:

$$\hat{w}^{[m]} = \left|2\pi \boldsymbol{L}_t^{[m]}\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\boldsymbol{z}_{\hat{c}(k),t} - \bar{\boldsymbol{z}}_t^{[m]}\right)^T \left(\boldsymbol{L}_t^{[m]}\right)^{-1}\left(\boldsymbol{z}_{\hat{c}(k),t} - \bar{\boldsymbol{z}}_t^{[m]}\right)\right\}, \tag{3.77}$$

where

$$\boldsymbol{L}_t^{[m]} = \left[\left(\left(\boldsymbol{P}_{xz,k,t}^{[m]}\right)^T / \left(\boldsymbol{S}_{P,t}^{[m]}\right)^T\right) / \left(\boldsymbol{S}_{P,t}^{[m]}\right)\right] \boldsymbol{P}_{xz,k,t}^{[m]}. \tag{3.78}$$

Feature Addition

Before resampling the particle filter, the measurements from $\boldsymbol{z}_t$ that were not associated with features must be added to the map. Because in FastSLAM the feature state estimator is separated from the vehicle state estimator, the initial feature covariance is simply the measurement covariance [33]. In the case of the square root Cubature

FastSLAM used in this thesis, this corresponds to S_{R_L} which is a Cholesky factorization of the covariance, R_L in Section 3.1.2. This covariance is used to calculate the measurement sigma points that are then passed through the inverse measurement model, h^{-1}. The following calculations lead to an initial feature mean, $\hat{\boldsymbol{\lambda}}_{k,t}^{[m]}$ and square root covariance, $\boldsymbol{S}_{\lambda_k,t}^{[m]}$ that are added to the particle's map:

$$\boldsymbol{\chi}_z^{[i][m]} = \boldsymbol{S}_{R_L}\boldsymbol{\xi}_i + \boldsymbol{z}_t, \tag{3.79}$$

$$\boldsymbol{\Lambda}_t^{[i][m]} = h^{-1}(\boldsymbol{s}_t^{[m]}, \boldsymbol{\chi}_z^{[i][m]}) \quad i = 0, ..., 2n \quad , \tag{3.80}$$

$$\hat{\boldsymbol{\lambda}}_{k,t}^{[m]} = \frac{1}{\sqrt{2n}} \sum_{i=1}^{2n} \boldsymbol{\Lambda}_t^{[i][m]}, \tag{3.81}$$

$$\boldsymbol{S}_{\lambda_k,t}^{[m]} = \textbf{Tria}\left(\left[\frac{1}{\sqrt{2n}}\left[\boldsymbol{\Lambda}_t^{[1][m]} - \hat{\boldsymbol{\lambda}}_{k,t}^{[m]} \quad \boldsymbol{\Lambda}_t^{[2][m]} - \hat{\boldsymbol{\lambda}}_{k,t}^{[m]} ... \boldsymbol{\Lambda}_t^{[2n][m]} - \hat{\boldsymbol{\lambda}}_{k,t}^{[m]}\right] \quad \boldsymbol{S}_{R_L}\right]\right), \tag{3.82}$$

where $k = (N_{\lambda,t-1}^{[m]} + 1 ... N_{\lambda,t}^{[m]})$.

3.3.3 Scanning Decision

Given the premise that scans will not be taken at each time step, another problem emerges: how does *Kapvik* determine when a scan should be taken? A simple choice would be to select a scan after a certain distance has been travelled or time has passed, however this may not be the most efficient method. If the sensors being used change or the rover is driving straight more than it is turning, the uncertainty in its pose will be related to the path it took to reach that pose, rather than simply the distance it has driven. Because the rover has an estimate of not only its pose but also the uncertainty in these estimates, it is a good indicator for when a scan should be taken as these are the state estimates that the scan is trying to improve. The uncertainty threshold that is set for taking scans will also be consistent across different sensors and different manoeuvres, as it is a direct estimate of the uncertainty in each of the

rover's states. In addition to this threshold, a threshold of distance travelled can be used as well, so that the rover does not drive outside of the map it has already built. This ensures that even if the rover's pose uncertainty is small, it is still mapping its environment sufficiently.

3.4 Obstacle Identification for Path Planning and SLAM

When performing autonomous navigation, it is desirable to classify features from a LIDAR generated map as obstacles, untraversable terrain or simply traversable terrain. It is important to classify the LIDAR data for two reasons. First, basic obstacle detection is necessary for path planning. A LIDAR scan will typically contain at least tens of thousands of data points [10, 64]. In the case of *Kapvik's* LIDAR each data point contains a measurement of range, bearing and tilt. There must be a way for the autonomous system to differentiate which part of the scan constitutes an obstacle, and which part can be safely navigated over. The second reason relates to accurate SLAM. Data association is the ability of the autonomous system to correctly identify measurements it receives as belonging or not belonging to its current map and is of critical importance in solving any SLAM problem [30]. The stability of SLAM depends on the accuracy of data association. Data association will commonly attribute measurements of uniform surfaces, such as ground or walls, to nearby previous measurements. These data association errors can quickly lead to divergence in SLAM algorithms. By classifying the scan, these types of terrain can be removed from the data association problem. Removing large portions of the scan that are not useful also allow for higher resolution scans to be associated when computational power is limited. To do this, the scan is classified and only data that has been classified as an

obstacle is used by SLAM.

Neural networks are often used to solve classification problems [20, 53, 65]. They have been applied in speech recognition, character recognition, and signal processing problems. It has been shown that Neural networks can be viewed as universal approximators, and it has been demonstrated that a single hidden layer neural network can approximate any continuous function with support in a unit hypercube [66]. Neural networks would seem worthy of investigation for the 3D LIDAR classification problem.

Part of the contribution of this thesis is an easily adaptable training system that can be applied to a wide range of terrain types and avoids time consuming hand-labeled target scans to be used in neural network training. It makes use of more information from LIDAR scans than has been used in previous examples and includes information such as the rover's estimated pose. An effort has been made to keep the overall neural network structure simple, allowing easy adjustment to the inputs and outputs.

In Section 3.4.1 the Neural Net structure is explained along with the methodology for training. The simulation used to train the neural network is described in Section 4.1. Test results in a real world environment are presented in Section 5.2.

3.4.1 Neural Network Development

The neural network structure used in this thesis is a multi-layer perceptron neural network (MLPN). An MLPN has an input layer, output layer, and in between one or more so called hidden layers as shown in Figure 3.18. In neural network diagrams it is common to depict neurons as circles, connected in layers by weighting factors that are represented by lines. The input layer neuron values are multiplied by the weighting factors, and the result is summed with all other weighted input neuron values as the input to an activation function in the next layer neurons. The activation function

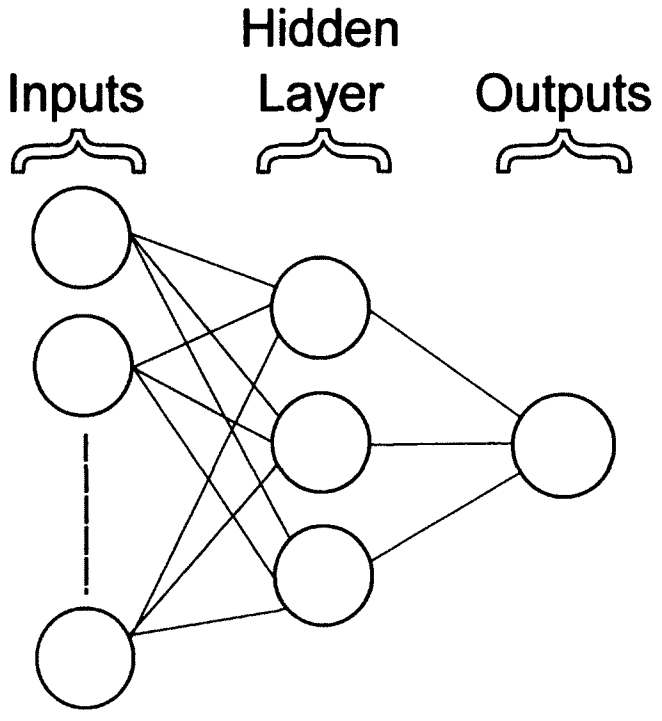


Figure 3.18: Multilayer perceptron neural network

used in this thesis is the sigmoid function,

$$f(x) = \frac{1}{1+e^{-x}}, \tag{3.83}$$

where x is the summation of all weighted neuron outputs from the previous layer written as:

$$x = \sum_{i=1}^{N_n} \mathbf{w}_i N_{i,ab}, \tag{3.84}$$

where $\mathbf{w}_i$ represents the weight of the connection between each previous layer neuron with a neuron in the next layer. $N_{i,ab}$ represents the inputs from the previous layer and N_n represents the number of neurons in the previous layer.

The sigmoid function was used so that the output of the neural network conforms to values between 0 and 1. This is done for each neuron in the layer and repeats until

the output layer is reached, the values of which represent the neural network output.

Neural networks must be trained so that the weighting factors represented by lines in neural network diagrams produce an output that is expected based on the initial input to the network. The network in this thesis undergoes training on a data set representing LIDAR scans of various terrains, and is shown to generalize well enough that when presented with new measurements from actual outdoor environments it will correctly classify the terrain. What follows is a detailed description of the neural network input, output, and training formulation.

Neural Network Cell Grid Input

As demonstrated in [55], the points in a new set of measurements can be organized in a cell grid overlaying the map.

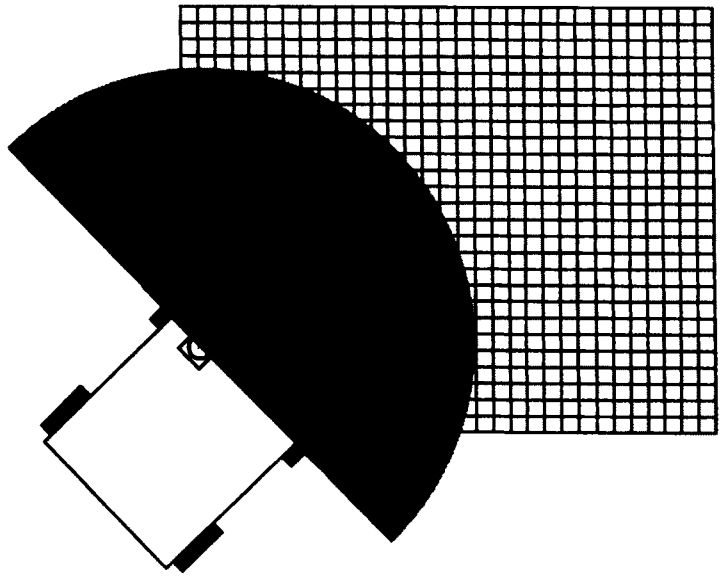


Figure 3.19: Cell grid to partition LIDAR scans

Each cell, C_{ab}, represents a certain area of the map with a and b denoting the row and column index respectively, starting from bottom left as seen in Figure 3.19. Some associated values are assigned to each cell based on the LIDAR measurements

contained within it. The dimensions of the cell were chosen to be approximately the diameter of a rover wheel, as this is the typical definition of an untraversable obstacle for a rocker bogie type rover.

For instance, if a cell that covers the range of points within one wheel diameter in the x and y directions is selected somewhere on the map, all points that fall within this range will be considered. The values the cell takes on will include things such as the mean height in comparison to the rover's estimated height, the variance of the features in the Z direction, or the difference between the maximum and minimum positions of features in the Z direction.

The neural net will classify each cell in the grid based on these inputs. To train the neural net, a set of measurements that have been classified previously will be presented to the neural network and based on the difference between its output and the known classification, the weights will be adjusted as discussed later in this section.

The number of neurons chosen for the hidden layers was chosen empirically by testing the accuracy of the neural network at various numbers of neurons, with the goal of being as accurate as possible using as few as possible neurons in each hidden layer. The comparison of these is shown in Section 5.1.2.

To begin testing the neural network, the output was limited to a single neuron, with a value between 0 and 1. Different regions between 0 and 1 represent different classifications. For instance, in a binary classification 0 represents traversable terrain and 1 represents untraversable terrain. The output is then assigned a classification based on the following rule:

$$O_{ab}(N_{o,ab}) = \begin{cases} \text{Untraversable} & \text{if } N_{o,ab} \geq 0.5 \\ \text{Traversable} & \text{if } N_{o,ab} < 0.5, \end{cases} \tag{3.85}$$

Where O_{ab} is the classification for C_{ab}, and $N_{o,ab}$ is the neural network output for C_{ab}.

The activation function used for this neural network is the sigmoid activation function, which has limits of 0 and 1 as x goes to $\pm\infty$. If the network is trained for these outputs, the weights will be driven to large values that are unstable [65]. Therefore the network should be trained on less extreme values. In the case of this neural network, the outputs range is between 0.1 and 0.9, with traversable terrain being labelled 0.1 and untraversable terrain being labelled 0.9. It is important to note that these are the values the network is trained on, but the output of the network still follows the rule in (3.85).

With added classification types such as partially traversable, the range between 0.1 and 0.9 is divided further. When working with multiple classification types another potential neural network structure has several outputs neurons, with the output consisting of one neuron exclusively set to one, the rest being set to 0. The output that is equal to one corresponds to the classification that has been made. In the case of the first example, with only two terrain types, the network would contain two output neurons.

The neural net requires a relatively large and varied data set for training. This helps the neural net generalize which is important when it is presented later with unfamiliar terrain measurements. To offset the large amount of time and effort that would be required to classify many LIDAR scans manually, a simulation may be used to generate a set of simulated LIDAR scans belonging to randomly generated terrain. The terrain is then automatically classified as it is generated. For instance when terrain is generated, the ground is labelled as traversable terrain whereas shapes representing rocks are labelled as obstacles. When the simulated LIDAR scans these different areas, the training set contains the correct labels for each scanned point to be used for supervised training. The simulation developed for this thesis is discussed in Section 4.1.

The input layer of the neural network contains the information that has been

received after separating each measurement into its respective grid cell. Information about each grid cell is then used as an input to the neural network. Inputs might be the mean height of the cell with reference to the rover position or the difference in height between the lowest and highest feature. The information that is given to the neural network should have some relation to whether the cell is potentially an obstacle or not.

Neural Network Input Types

After dividing up the LIDAR range data into a cell format, various inputs can be derived for each cell that will help the neural network determine what type of terrain that cell represents. While some cells will be empty, and thus will not be classified, the ones that are not empty will contain one or more LIDAR data points. As an input, specific LIDAR data points can be considered, or characteristics of the group as a whole. Information about the rover's pose can also be used to influence how the neural network perceives a cell. Anything that might influence the way the LIDAR scanner scans the environment as well as may describe differences between types of terrain could be a potential input to the neural network. What follows is a brief description of each potential input to the neural network.

- Mean height: The first input to the neural network will be the cell's mean height. The calculation for the mean height is taken as the position of the rover in the z direction, Z_{Rover}, subtracted from the mean position in the z direction of all LIDAR data points, $Z_{Mean_{a,b}}$:

$$Z_{H,ab} = Z_{Rover} - Z_{Mean,ab}, \tag{3.86}$$

  If the rover was operating on a 2D ground plane with obstacles scattered

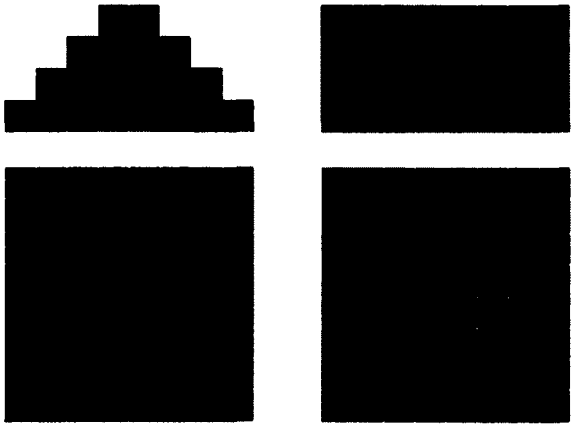


Figure 3.20: Comparison of adjacent cell values for slopes (left) and obstacles (right)

about, this input alone could be used to determine whether a cell represented an obstacle or not. However, in 3D terrain there is the potential for hills and valleys. To the rover these might look like obstacles, but in fact they could be smooth slopes that the rover can easily traverse. To be able to differentiate between the two is of obvious importance, so other inputs from each cell must be considered to differentiate the two.

- Variance: The variance of the height of LIDAR data points in a given cell can be an indication of what type of terrain it represents. Solid obstacles and terrain will have a small variance, whereas terrain that includes grass may have a larger variance in height. The variance in Z for each cell is calculated as,

$$\sigma^2_{Z_{a,b}} = \frac{1}{N^m_{a,b}} \sum_{i=1}^{N^m_{a,b}} (Z^i_{a,b} - \mu_{Z_{a,b}})^2, \tag{3.87}$$

where

$$\mu_{Z_{a,b}} = \frac{1}{N^m_{a,b}} \sum_{i=1}^{N^m_{a,b}} Z^i_{a,b}, \tag{3.88}$$

where $N^m_{a,b}$ is the number of measurements in each cell, and $Z^i_{a,b}$ is the Z position of each measurement.

- Absolute height difference: The difference between the maximum and minimum height in a given cell. Data points belonging to an obstacle will have larger differences in height over the span of a single cell than traversable terrain.

- Number of LIDAR data points: The number of points in a cell will in general be higher for obstacles.

- Rover orientation: The rover orientation in the pitch and roll directions will have an effect on what is considered an obstacle. For instance if the rover is pitched forward, looking down a hill, an obstacle on the hill will have a larger height difference than if the rover had seen a similar obstacle on flat terrain.

- Mean height of adjacent cells: Slopes are more likely to have adjacent cells that change gradually in height than cells that include obstacles where the edges will typically have large drop offs in height. This is illustrated in Figure 3.20

Methodology for Training

The methods used to train a neural network are extremely important. Improper training can lead to neural networks that are not well generalized. They may perform well on a training set, but because the training set doesn't fully represent the range of data encountered in real measurements it may fail to generalize well enough to work in a real scenario. While part of the reason a simulation was used to train the neural

network was to reduce the time required to label the training set, it was also used to increase the range of terrain encountered by the rover. For instance, if the data was obtained from a few locations in and around a university campus with the actual LIDAR, and then hand labelled, the dataset might not resemble the terrain located in a park or another common outdoor environment.

There is also question of too much training data, where the neural net is not able to encode the information from all possibilities adequately. By using a simulation, the type and volume of terrain the neural net is trained on is easily controlled. However, simulations present new problems. The terrain generated is a smooth surface described by a polynomial, and the obstacles are polyhedron shapes. The question is whether the neural network will be able to generalize these more basic geometries from the training set to real data it receives while driving outdoors. In this thesis the training and validation steps are done through simulation, and the testing set is from real world scans. The goal will be for the neural network trained on the simulated data to correctly classify scans taken in the real world.

If the neural network fails on real data, the question becomes whether some fundamental information is not being given to the neural network causing it to fail or if the training set itself needs to be changed, to more accurately represent data the rover will encounter in the real world.

The neural network presented in this thesis is trained using a Kalman filter. The neural network weights are treated as estimated states, $\hat{\mathbf{w}}_n$, with the output of the neural network being treated as a measurement, $\mathbf{y}_n$. The training set consists of a number of sample cells with inputs based on the properties discussed in Section 3.4.1 denoted $\mathbf{u}_n$ and outputs based on the origin of LIDAR measurements in each cell, denoted $\mathbf{d}_n$. For instance if any of the data points in a cell belong to an obstacle, the cell is labelled an obstacle and $\mathbf{d}_n$ will be equal to 0.9. If the output is divided further into different types beyond traversable and obstacle terrain, a hierarchy is used to

determine how to classify cells with more than one type of data.

To properly estimate each of the weight states, the training set should be at least as large as the number of weights. For the initial training, a new surface was generated for each training set input/output pair and only one measurement was taken. This ensured that the terrain was different in each scan, and to account for the fact that at present a path planning or control methodology hasn't been applied to the simulated rover, it simply orients itself with respect to the terrain at a random location in the map. After one scan, each of the inputs are calculated and formed into the neural network input vector $\mathbf{u}_n$. A cubature Kalman filter is used to train the weights of the neural network as seen in Figure 3.21. This avoids the complicated Jacobians that would need to be computed with an extended Kalman filter. The form of the Kalman filter training algorithm was developed in whole as presented by Haykin in [20]. The system state model is described by a random walk equation

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \boldsymbol{\omega}_n, \tag{3.89}$$

where $\boldsymbol{\omega}_n$ is a white Gaussian noise of zero mean and covariance matrix $\mathbf{Q}_\omega$. The measurement model is described by

$$\mathbf{d}_n = \mathbf{b}(\mathbf{w}_n, \mathbf{u}_n) + \boldsymbol{\nu}_n, \tag{3.90}$$

where $\mathbf{b}_n$ represents the neural network, and $\boldsymbol{\nu}_n$ represents the measurement noise and is a white Gaussian noise of zero mean and covariance matrix $\mathbf{R}_n$.

When the training for that particular scan has been accomplished, the weights are saved and used to start training with a newly generated training set input/output pair and presented to the neural network in the same fashion until the training set has been exhausted.

The next step is to validate the neural network using another set of scans that

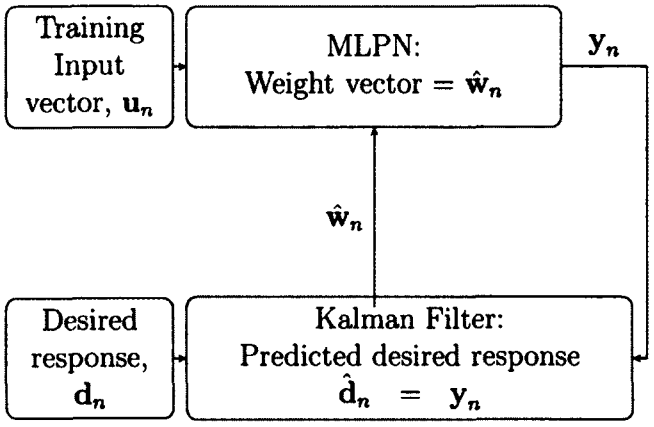


Figure 3.21: Neural network trained by Cubature Kalman Filter

were not included in the training set. This is called the validation set. The neural network is run on these unique scans and the output is compared to the known values for each scan. To confirm the neural network is working correctly, these validation scans should be classified correctly by the trained neural network. If the validation set has a mean square error above some set threshold the neural net continues with another iteration of training using both sets until the validation set is classified below the mean square error threshold.

3.5 Complete Algorithm

Throughout this chapter each section of the proposed algorithm has been outlined in detail. The motion and measurement model for *Kapvik* are introduced and from there the estimation scheme between scans is developed using the CKF presented in Section 2.1.2. When the uncertainty in the rover pose reaches a certain threshold, or when the rover drives a threshold distance, a scan is taken. When a scan is taken, SLAM takes over from the basic Kalman filter estimation to update both the rover pose as well as the map. The novel square root cubature FastSLAM algorithm presented in this thesis is developed for this purpose. Finally a neural network is designed that can

parse LIDAR data to differentiate between traversable and non-traversable objects, reducing the number of extraneous points in a LIDAR point cloud and also providing valuable information to path planning algorithms. All of these algorithms make up a complete navigation algorithm, which is outlined as follows:

Algorithm 3.1 *Kapvik* Navigation Algorithm

1: **while** Distance or uncertainty threshold has not been met **do**
2: Predict mean and covariance of the vehicle at time step t using motion model presented in Section 3.3.1.
3: Update mean and covariance with observations from IMU and sun sensor at time step t, also presented in Section 3.3.1 using CKF.
4: **end while**
5: **for** $m = 1 \rightarrow M$ **do** {Loop over all particles}
6: Retrieve $\boldsymbol{s}_t^{[m]}$ estimated from last CKF update.
7: Scan environment and classify scan data using neural network described in Section 3.4
8: **for** each feature **do**
9: **for** each measurement **do**
10: Calculate an updated mean and covariance of vehicle as in FastSLAM 2.0 for each classified measurement and map feature pair using (3.50)-(3.62). Calculate likelihood for each of these updates.
11: **end for**
12: **end for**
13: Calculate data associations (3.64) based on the likelihood of each assignment.
14: Calculate a weighted average of the pose and covariance given the data association made for each measurement and the likelihood calculated as in (3.65)-(3.66).
15: **for** each associated feature **do**
16: Update mean and covariance of feature using (3.67)-(3.77).
17: **end for**
18: **for** each new feature **do**
19: Calculate new feature mean and covariance using (3.79)-(3.82).
20: **end for**
21: Resample particles with probability $\propto \hat{w}^{[m]}$.
22: **end for**
23: Return to Line 1 with updated particle set.

Chapter 4

Testing

4.1 Simulation

A simulation was designed in Matlab for this thesis to test the various algorithms presented in Chapter 3. The simulation generates a "ground truth" for the rover as it moves over terrain that is meant to resemble Mars. It also simulates the various sensors located on the rover including the IMU, sun sensor and LIDAR. This simulation allows all aspects of this thesis to be tested, from the Rover kinematic model, to the SLAM algorithm as well as the neural net classification of LIDAR data. It also provides a useful tool to automate the neural network training process, and one of the contributions of this thesis is demonstrating the ability of the neural net to generalize the data it is trained on in simulation well enough that it can be used in a wide variety of environments and reliably classify terrain from real scans. For this to work, the simulation must produce LIDAR scans that are realistic.

For this thesis, a simulation is developed that models 3D terrain such as sloping hills and rocks, a rover traversing over this terrain taking measurements with simulated sensors, and a LIDAR measuring the terrain from the rover's location.

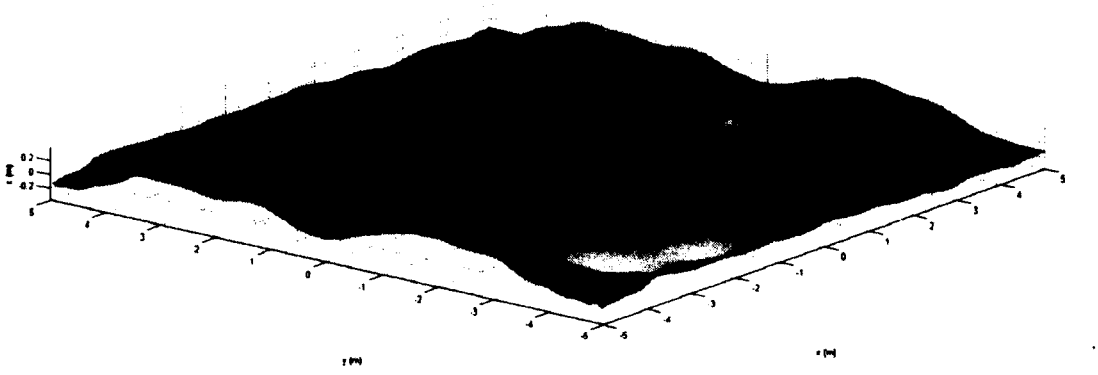


Figure 4.1: Randomly generated ground terrain

4.1.1 3D Terrain Generation

For the purposes of this simulation, there are two basic types of terrain. The first is ground terrain, which represents sloping hills and valleys as well as flat surfaces. In the real world tests done for this thesis this includes surfaces consisting of cement/soil as well as grass. The second type of terrain are obstacles. These include objects such as rocks, trees, bushes and other objects that are laying on the ground terrain.

The ground terrain is constructed by first randomly generating a function for elevation with the x and y position as an input. A polynomial for sloping terrain is used,

$$Z_{Terrain} = f(X_{Terrain}, Y_{Terrain}), \tag{4.1}$$

where the polynomial coefficients are randomly adjusted with each training set. By randomly adjusting these coefficients a different ground shape is generated each time a training set is created, as shown in Figure 4.1.

To generate obstacles, a convex polyhedron model was used for the purpose of speeding up LIDAR simulation. Unlike an open ended surface, closed surfaces such as rocks can be represented by a polyhedron. Efficient ray tracing operations for polyhedron are well established in computer graphics literature and the method used

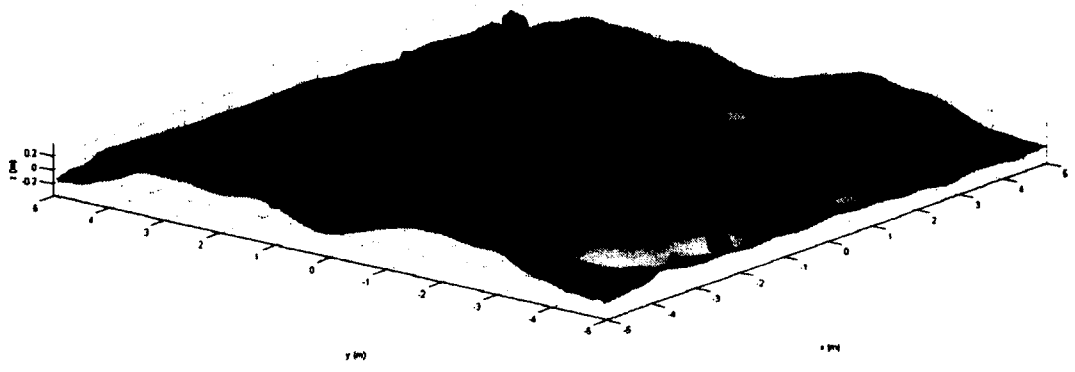


Figure 4.2: Randomly generated terrain with obstacles

by Haines in [67] is used for this thesis. This allows the creation of many such objects in a given map. These objects are varied in size and location randomly, with a mean Z position of zero in relation to the ground it resides on. Much like rocks in nature, some of the polyhedron resides below the surface it is laying on.

The end result of these two processes is a terrain with varying degrees of slope, and a random distribution of rocks at different heights above the ground terrain as shown in Figure 4.2. From the viewpoint of a LIDAR, this type of terrain closely matches the type of terrain encountered in an outdoor environment. In addition to these two main types of terrain, one can adjust the simulation to account for trees by using cylindrical shapes instead of polyhedron shapes. To simulate grass or bushes, adding variance to the measurements generated from designated areas of the ground or obstacle terrains will differentiate these terrain from the "regular" terrain in a realistic way.

4.1.2 3D Motion Model

The rover used in this simulation was modelled after both a differential drive rover such as the Pioneer rover that has been used for the outdoor tests as well as an articulated rover like the *Kapvik* micro-rover. A differential drive rover forward kinematic

model was simplified to a unicycle model, with inputs of $\boldsymbol{v}(t)$ and $\boldsymbol{\omega}(t)$ for linear and angular velocity respectively at time t. Differential drive rovers are generally capable of turning in place so this simplification is close to actual kinematic behaviour. These inputs are specified in the rover reference frame so using (3.3) the input control vector for linear velocity in the global reference frame and hence the new rover position can be calculated. For the complete rover pose the transformation of angular velocity is calculated using $\boldsymbol{S}_{\omega}^{G}$ from (3.4) and together, the kinematic equations are calculated as in (3.5).

After the rover moves through one time step, the pose is adjusted to conform to the terrain beneath it. To simplify the simulation, it has been assumed that the surface is smooth, and that the wheels are always in contact with the surface at a single point of contact. In [11], Tarokh et al. discuss a kinematic model to determine the rover wheel configuration given the constraints of the rover's X, Y position and its heading, Φ_z. This model is adjusted for use on both differential drive rovers as well as rocker bogie type vehicles for this thesis.

The wheel contact angles, δ_i, $i = 1, ..., N_w$ are calculated as,

$$\delta_i = \tan^{-1} \gamma_i, \tag{4.2}$$

where γ_i is the gradient of the terrain in the Z direction at the wheel position (X_i, Y_i) along the wheel heading $\Phi_{zc,i}$, shown in Figure 3.9.

Through a series of transformations from the wheel contact frame to the rover frame, the pose of the wheels are determined in the rover reference frame. Considering the rover pose in the global reference frame as $\boldsymbol{U} = [X_{Rover}, Y_{Rover}, Z_{Rover}, \Phi_x, \Phi_y, \Phi_z]^T$ the transformation matrix from wheel contact frames C_i, $i = 1, ..N_w$ to the global frame for the rover is

$$\boldsymbol{T}_{Ci}^{G}(\boldsymbol{U},\delta_i) = \boldsymbol{T}_{R}^{G}(\boldsymbol{U})\boldsymbol{T}_{Ai}^{R}\boldsymbol{T}_{Ci}^{Ai}(\delta_i). \tag{4.3}$$

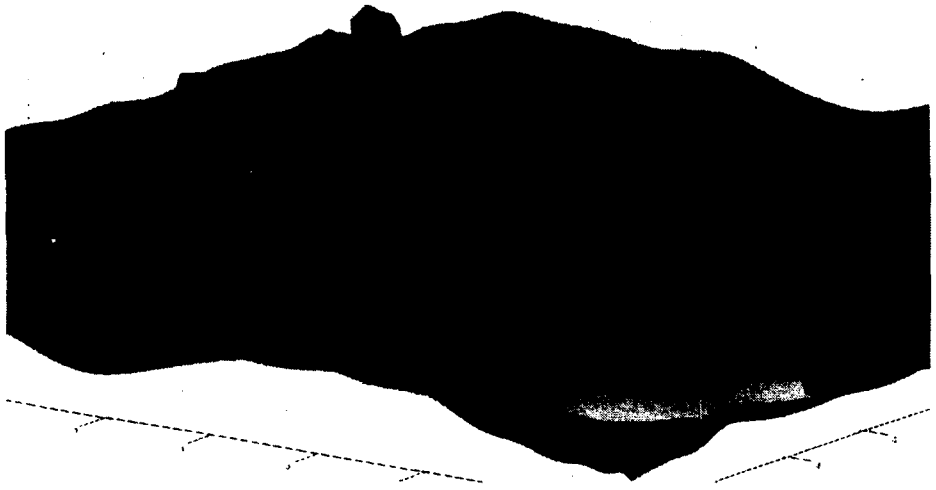
$\boldsymbol{T}_{R}^{G}(U)$ is obtained using (3.36) and $\boldsymbol{T}_{Ai}^{R}$ can be found for each of the N_w wheels by performing a series of Denavit-Hartenberg (DH) transformations that represent a kinematic chain from each wheel's axle to the rover reference frame as shown in Figure 3.8 for a differential drive rover and computed in equation (3.14) for the *Kapvik* micro-rover. The transformation from the contact frame to the axle frame, $\boldsymbol{T}_{Ci}^{Ai}$ is calculated using (3.6).

This rotation about the contact angle is visualized in Figure 3.9. In addition, at each time step, the simulation determines slope and height of the surface at the X and Y position of each of the rover's wheels.

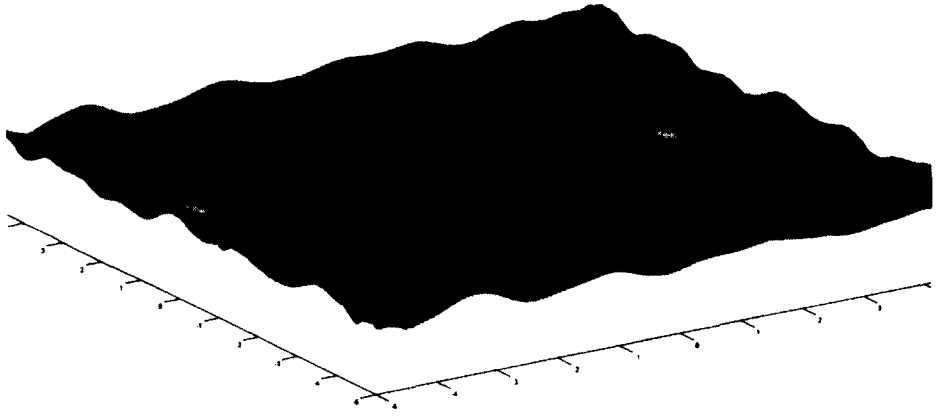
Based on the slope and height of the surface mesh under each wheel the pose of the rover is optimized under the constraints of its position in X and Y and its current direction using the fminsearch function in Matlab. Tarokh et al. in [11] show that (4.3) is a homogeneous transformation matrix and can be used to obtain the pose of each of the wheel contact points calculated as

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ \Phi_{xc} \\ \Phi_{yc} \\ \Phi_{zc} \end{bmatrix}_i = \begin{bmatrix} \boldsymbol{T}_{Ci}^{G}[1,4] \\ \boldsymbol{T}_{Ci}^{G}[2,4] \\ \boldsymbol{T}_{Ci}^{G}[3,4] \\ \tan^{-1}(\boldsymbol{T}_{Ci}^{G}[3,2]/\boldsymbol{T}_{Ci}^{G}[3,3]) \\ \sin^{-1}(-\boldsymbol{T}_{Ci}^{G}[3,1]) \\ \tan^{-1}(\boldsymbol{T}_{Ci}^{G}[2,1]/\boldsymbol{T}_{Ci}^{G}[1,1]) \end{bmatrix}, \tag{4.4}$$

where i denotes the wheel and $\boldsymbol{T}_{Ci}^{G}[j,k]$ denotes the jk^{th} element in the transformation



(a) Simulated differential drive rover



(b) Simulated *Kapvik* micro-rover

Figure 4.3: Simulated rovers driving over sloping terrain

matrix. The left hand side of (4.4) is the pose vector for the i^{th} wheel contact point. There is an assumption that the pitch of each wheel contact is limited to $-\pi/2 < \Phi_{yc} < \pi/2$ which is valid for this simulation and most terrain the rover is expected to travel over. The $\tan^{-1}$ function is calculated in matlab using the atan2 function in Matlab to resolve the yaw quadrant.

Each of the wheels will be constrained to be in contact with the terrain, thus $Z_{c,i}$ should be equal to the surface mesh height below it. An error vector is set up that

contains the difference in height of each of the wheels and the surface mesh below

$$E_i(\boldsymbol{U}, \delta_i) = Z_i(\boldsymbol{U}, \delta_i) - Z_{surface}(X_{c,i}, Y_{c,i}). \tag{4.5}$$

The objective function that is minimized in Matlab is represented by the norm of the $N_w \times 1$ error vector (4.5) as shown in [11]. The end result can be seen in Figure 4.3 which shows both types of rovers navigating over bumpy terrain. To simulate bumpy and uneven terrain, some noise is added to the input vector in (3.5).

For the purposes of the simulation, the rover is started at a random position above the randomly generated terrain. It is in general placed in a location somewhat close to an obstacle so that at least one valid scan containing both ground and obstacle terrain is taken, but this can be adjusted. The trajectory the rover takes can be changed to avoid obstacles using a path planning algorithm, but for the initial tests the rover drove a limited distance so this was unnecessary.

This simulation provides the "true" pose of the rover and is used in both neural network training as well as testing the proposed SLAM algorithm. In the case of the SLAM algorithm, measurements are taken from this ground truth and summed with noise generated from a white, zero mean, gaussian noise as described by the appropriate measurement model for each sensor.

4.1.3 LIDAR Simulation

To simulate LIDAR scans, a set of lines is generated, each representing a single LIDAR beam. The LIDAR consists of a set number of scans at each tilt interval. Each line is calculated in its parameterized form based on the position of two points. These positions are calculated using the transformation matrices calculated in (3.35) and (3.36). The first point is the position of the LIDAR head, where the beam originates from. The second point is defined by the tilt and bearing angles as well as the

maximum range of the LIDAR, r_{max}. The first position for each scan is calculated as

$$\boldsymbol{P}_1^{ij} = \boldsymbol{T}_R^G \boldsymbol{T}_{L,j}^R \begin{bmatrix} 0 \\ 0 \\ L_a \end{bmatrix}, \tag{4.6}$$

where i represents the bearing measurement index for the j^{th} tilt measurement. The second position is calculated as

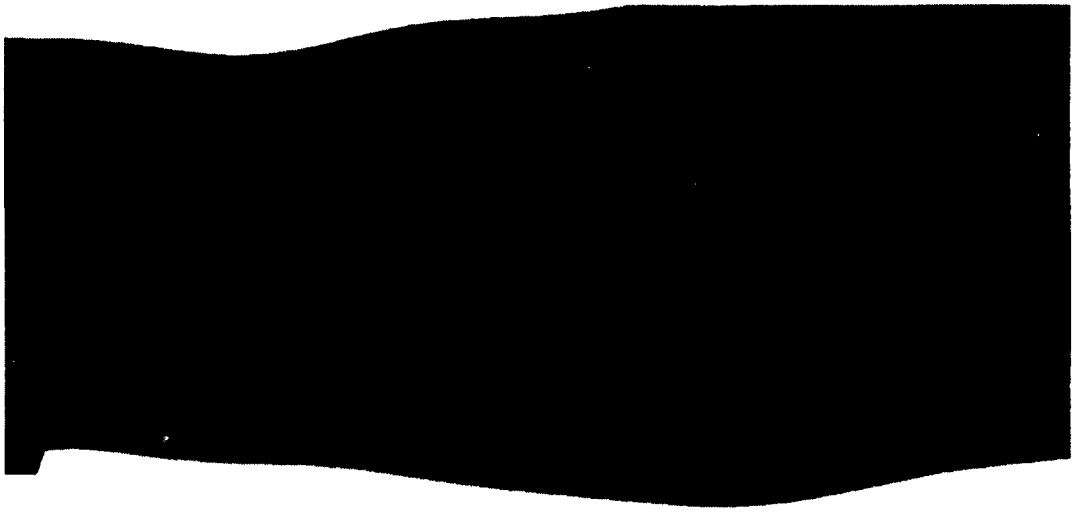
$$\boldsymbol{P}_2^{ij} = \boldsymbol{T}_R^G \boldsymbol{T}_{L,j}^R \begin{bmatrix} r_{max} \cos\psi_i \\ r_{max} \cos\psi_i \\ L_a \end{bmatrix}. \tag{4.7}$$

The parameterised equation of the line that connects the two points is then calculated as

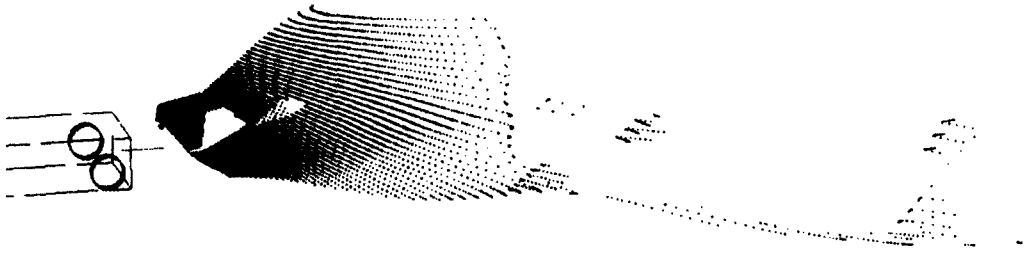
$$\boldsymbol{P}^{ij} = \boldsymbol{P}_1^{ij} + s(\boldsymbol{P}_2^{ij} - \boldsymbol{P}_1^{ij}), \tag{4.8}$$

where s is a value between 0 and 1.

The next step is to use the $M \times N$ lines that have been generated from (4.8) and determine the intersection points, if any, between these lines and the surface mesh that represents the ground as well as the polyhedrons representing objects. The ultimate goal is to determine, for each line, the intersection point that is the closest distance to the LIDAR. To do this, the intersection points are calculated for the ground first. A system of nonlinear equations is set up using (4.1) and (4.8) and using Matlab's fmincon function a solution for the intersection point is found for each line within the boundaries of the LIDAR field of view. These intersection points are then compared, line by line, to any intersection points with obstacles.



(a) Simulated terrain



(b) Simulated LIDAR scan

Figure 4.4: Comparison of actual simulated terrain and simulated LIDAR scan

The distance of each closest intersection point, $\boldsymbol{P}_2^{ij}$, from the LIDAR head position, $\boldsymbol{P}_1^{ij}$, corresponds to each range measurement. After this selection, Gaussian noise is added to the range measurement using the specified standard deviation for the Hokuyo or SICK range finder. The bearing and tilt measurements are predefined

by the resolution of the LIDAR bearing and the tilt unit. Together, these three measurements make up the simulated LIDAR beam, and $M \times N$ measurements are calculated. A side by side comparison of the actual simulated environment and a simulated LIDAR scan of that environment are shown in Figure 4.4.

4.2 Pioneer and Husky Sensor Platforms

4.2.1 Rovers

The algorithms presented in this thesis are tested when possible on two rover platforms, the Mobile Robotics Pioneer 3-AT and the Clearpath Husky rovers. Each rover is a differential drive rover with 4 wheels. The left and right side wheels are independently driven as explained in Section 3.2 and the rovers are driven using skid steering. While the Husky is larger and generally more robust in outdoor environments, both rovers were operated in the same way, at speeds similar to the Kapvik rover with a maximum speed of $0.02m/s$. A detailed discussion of the differences between these two rovers is not needed, as the differences did not affect the way the algorithms were run beyond the difference in sizes which were presented as general quantities in Section 3.2. The two rovers are pictured in Figure 1.2 and Figure 1.3.

4.2.2 Data Recording

To record data a Lenovo laptop was used running the Ubuntu Linux operating system. The rovers and all sensors were connected to the laptop through USB ports and the player/stage robotic operating system was used to control each of the sensors as well as record the data they output. In some cases player/stage drivers did not exist for a sensor (such as the SICK LMS 111 LIDAR and the Memsense H3-IMU), so drivers were written in C++ for use in the player/stage framework. To run the algorithms

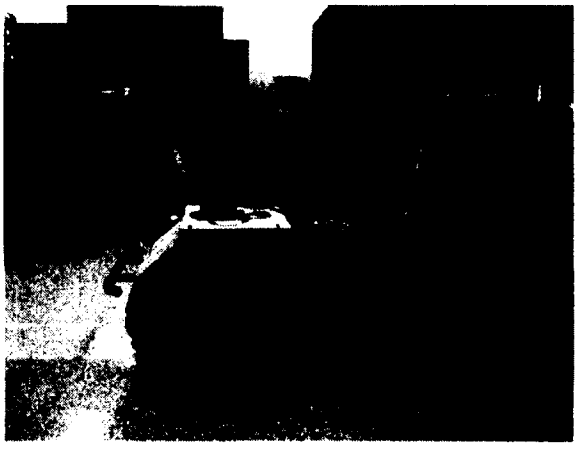


Figure 4.5: Typical indoor testing environment

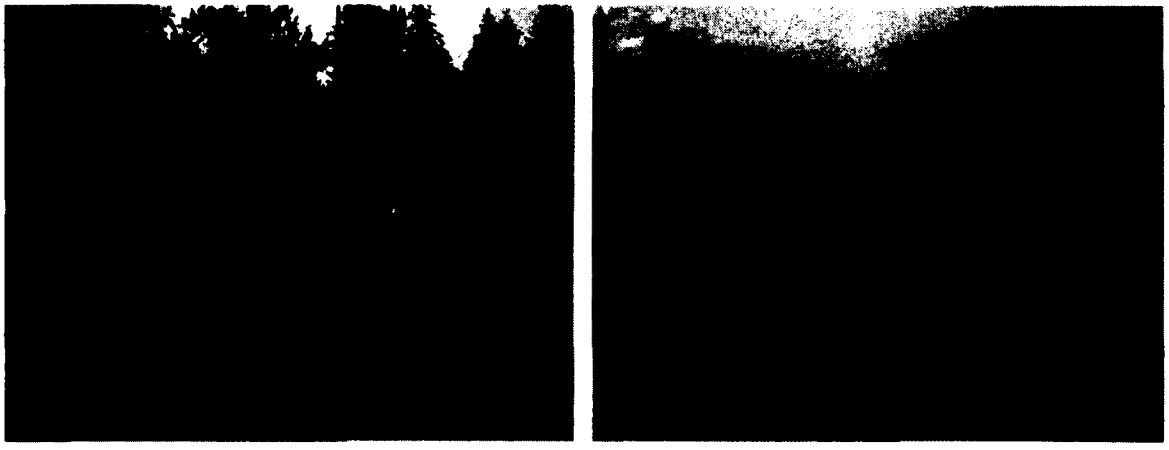
Matlab was used along with a player/stage toolbox written for Matlab.

4.3 Test Environments

An attempt was made while developing the algorithms presented in Chapter 3 to test them in a real world environment. The neural network was extensively tested in indoor and outdoor environments and showed promising results which will be discussed in Section 5.2. This is of crucial importance as one of the contributions of this thesis is demonstrating that a neural network trained on simulated data can operate in a real world environment. This section outlines some of the environments that were used for testing.

4.3.1 Indoors

Indoor environments are not the focus of this thesis as *Kapvik* will operate in an outdoor environment. However initial testing was carried out in a lab setting. A lab setting provides more obstacles for the neural network to detect usually, but it is also a less challenging environment to test the navigation techniques in, due to a flat,



(a) Carleton University campus. (b) Petrie Island.

Figure 4.6: Typical outdoor environments.

uniform surface of travel. Figure 4.5 shows a typical test environment.

4.3.2 Outdoors

Because *Kapvik* is meant to operate in an outdoor environment it was important to test in outdoor environments. Most outdoor testing was done around the Carleton University campus as shown in Figure 4.6a. Some tests were also done on Petrie Island, Ottawa, Canada on sandy terrain with various sized rocks and slopes included. However, during the Petrie Island tests only the neural network testing was completed. Figure 4.6b displays the environment the neural network was tested on.

Chapter 5

Results

5.1 Simulation

Before testing any of the algorithms presented in this thesis in the real world, simulations were developed to test and tune the algorithms. In the case of the SLAM algorithm that includes an articulated rover kinematic model, the algorithm was only able to be tested in simulation. The LIDAR classification algorithm was also tested in simulation before it was used in the real world.

5.1.1 *Kapvik* Kinematics

To test the validity of the kinematic simulation that is developed in Chapter 3 and Chapter 4 the simulation was run over a 180 second time period. No scans were taken during this time and only the "ground truth" model was run, therefore no localization or mapping occurs. The results of this test will determine whether the simulation accurately reflects the kinematic motion of *Kapvik* over rolling terrain. The analysis was carried out as was done in [11]. In [11], the terrain was kept flat for one side of the rover, and bumpy for the other side, however the model built for this thesis involves more complex terrain so this is used for testing. The result of these tests are compared and contrasted to the results in [11].

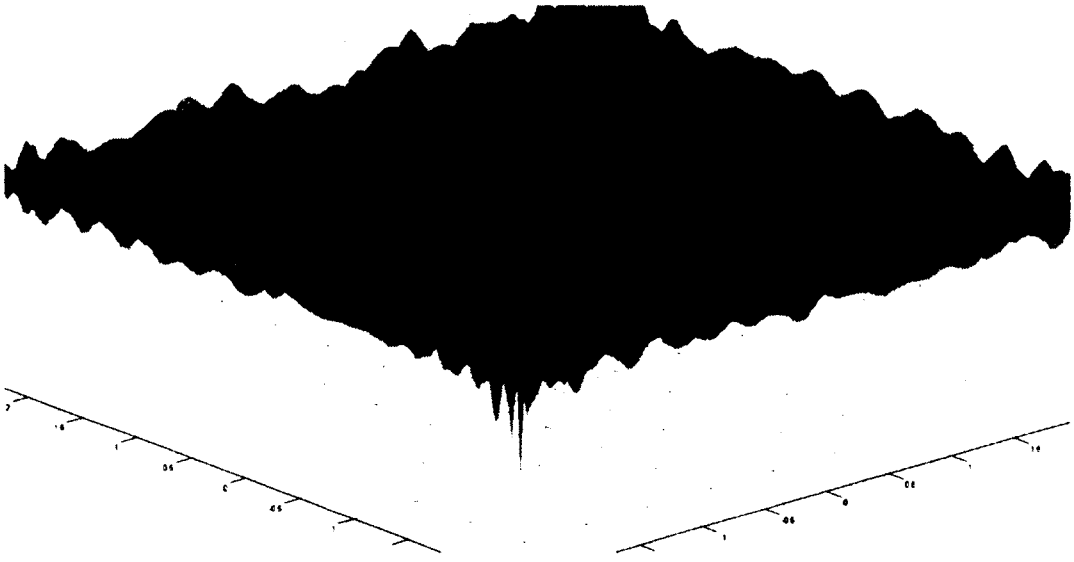


Figure 5.1: Traces of front wheels over bumpy terrain

To begin testing, bumpy terrain was generated. As in [11], the rover was commanded to do both a straight path and a "serpentine" path. The straight path results will be presented here and the serpentine results are listed in Appendix B.

The left and right wheels were set to an angular speed of $\dot{\theta}_L = \dot{\theta}_R = \frac{0.02m/s}{r_w}$. The front two wheel positions were traced along the path of the rover and are displayed in Figure 5.1. On visual inspection of the wheel traces it is apparent that both sets of wheels traverse different terrain profiles that will result in a different bogie characteristics. What the results of this test should show are an overall roll and pitch that remain small compared to the contact angles of the wheels. The rocker should also exhibit less motion than the two bogies. Finally the contact angles should be independent from one another as each wheel is traversing a different part of the terrain.

Figure 5.2 shows the rover body roll Φ_x and pitch Φ_y as the result of the combined motion of all six wheels. It can be seen that the roll and pitch are kept relatively small, and seem to oscillate about zero degrees as it moves over the bumpy terrain. It is also seen that the rover roll is trending downwards, which follows the overall slope

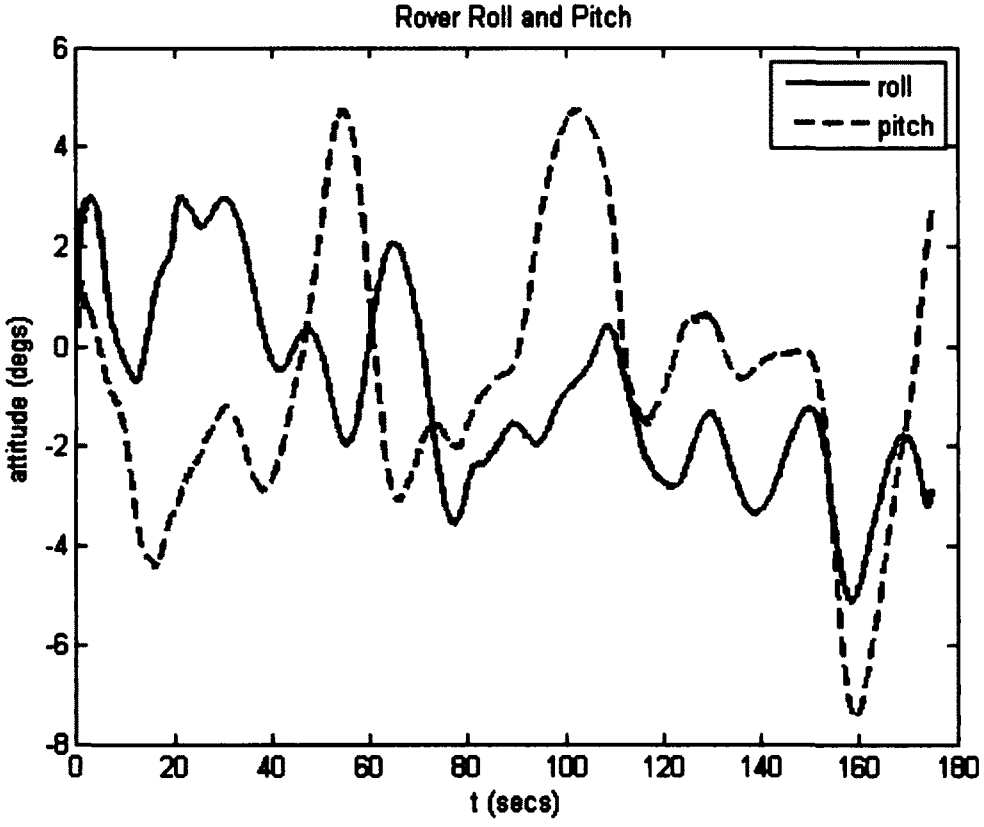


Figure 5.2: Rover roll and pitch for straight path

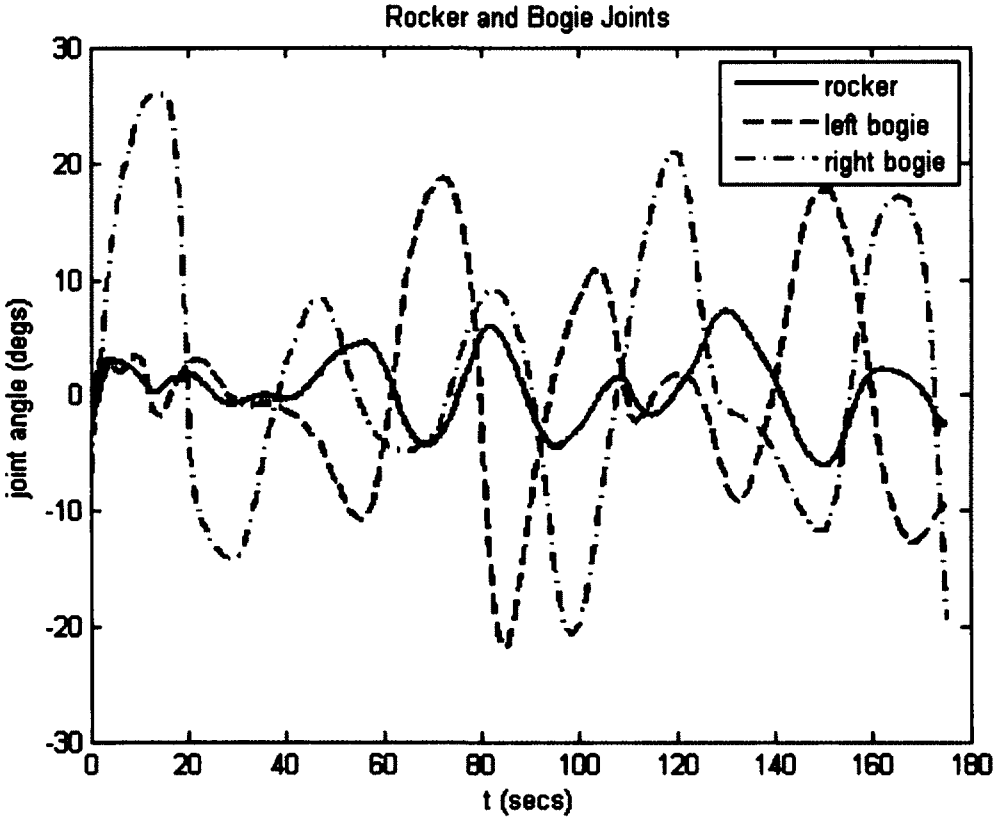


Figure 5.3: Rocker and bogie joint angles for straight path

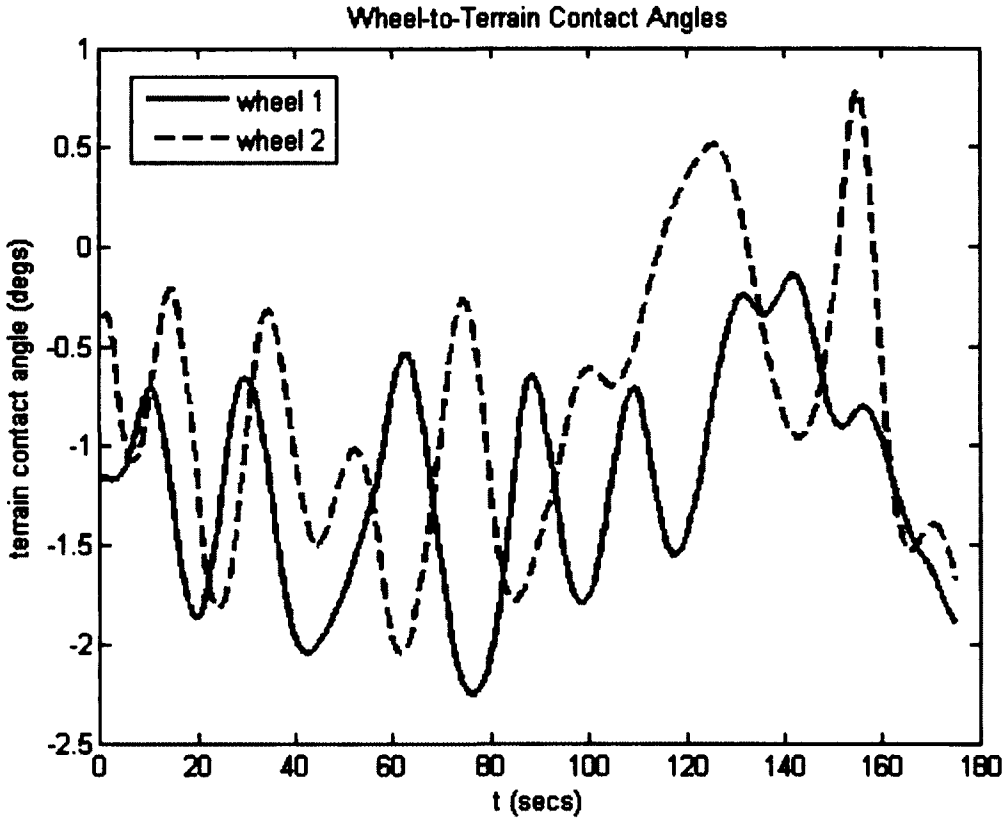


Figure 5.4: Contact angle for the front wheels on straight path

of the terrain, independent of the small bumps the rover traverses. This shows the behaviour that is desired from a rocker-bogie system, where the rover body roll and pitch are kept at a minimum as it moves over nonuniform terrain.

Figure 5.3 shows the rocker ρ and two bogie angles β_1 and β_2. The rocker angle is shown as expected to be much smaller in magnitude than the two bogie angles. The two bogie angles also exhibit oscillating, independent behaviour from one another. In Figure 5.4 the contact angles are shown for the two front wheels. The contact angles follow an oscillating path as the rover traverses the bumpy terrain, with each wheel having an independent contact angle based on the terrain it is traversing.

5.1.2 LIDAR Classification

With the neural network developed in Section 3.4.1 and the simulation in Section 4.1 training of a neural network can be accomplished.

The algorithm was run on a test set generated by the same simulation as the training and validation sets. The neural networks were trained with all inputs and the binary outputs discussed in Section 3.4.1. The grid cell size was set to $r_w \times r_w$ and a varying number of hidden layers and their corresponding number of neurons were tested on the same data set.

The maximum total number of neurons was limited arbitrarily to 32 and the number of hidden layers and neurons was varied as follows: The first test included just one hidden layer and one neuron, which was incremented up to a total of one hidden layer and 32 neurons. After this was completed, 2 hidden layers were used, with the first test including 2 layers, with 1 neuron each. The second layer had its number of neurons increased until it reached 16, and then was reset to 1 with the number of neurons in the first layer increasing by 1 until both layers had 16 neurons each. This same procedure was repeated for 3 layers, with a final design of 11 neurons in the first and second layer, and 10 neurons in the third layer.

The mean squared error (mse) is defined as the average of the squared error in the output of the neural network and the known classification for all grid cells in the test set. The mse associated with each type of test was plotted to see if any trends were present in changing the size and complexity of the neural network, and also to see what the average root mse was across all types of networks. It is apparent by visual inspection of the plot that the error decreases as the number of neurons and hidden layers increases until the 3rd layer is added, where a noticeable increase in error occurs. For these reasons a network of 2 layers with 16 neurons in each layer was used for testing purposes in a real world environment. The overall average root mean squared error was approximately 0.1, which in the binary case is accurate enough for correct classification according to the rule presented in 3.85.

5.1.3 SLAM

The proposed method of SLAM presented in Algorithm 3.1 was tested in simulation using the rover kinematic model and the measurement model in Section 3.3.1. LIDAR was simulated using the method in Section 4.1.3. The rover was driven in a circle over sloping terrain with rocks randomly distributed. The rover's wheel rates on the left were set to a constant $\dot{\theta}_L = (0.01m/s)/r_w$ and $\dot{\theta}_R = (0.02m/s)/r_w$. No path planning algorithm was used for this thesis, so if rocks are in the way of the rover, the rover will drive through them despite having labelled them as obstacles. This will not affect the results of the SLAM algorithm.

The hybrid SLAM algorithm was run with a time step of $T = 0.25s$ and scans were taken every 5 seconds. The particle filter included 50 particles. A rolling slip was introduced of 20% of the velocity of each wheel to the ground truth model. To test the effectiveness of the algorithm at localizing the rover the simulation was run with localization being provided by only the square root CKF, using the IMU and sun sensor measurements. This meant that the orientation was observable, but the

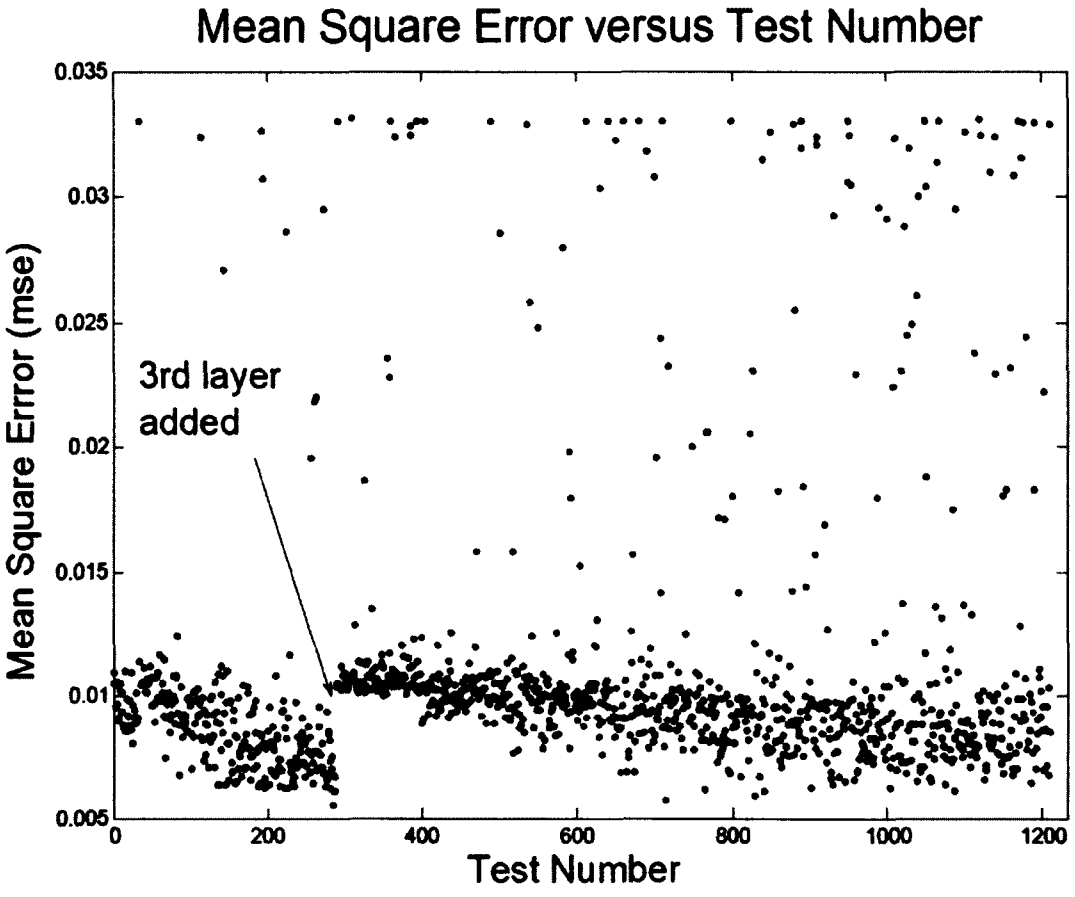


Figure 5.5: Neural Network Performance Tests

position was based purely on dead reckoning using the motion model. This was compared to running the same simulation with the hybrid algorithm presented in this thesis that incorporates the scans taken at intervals. In Figure 5.6 the rover's maximum weighted particle position error in both cases is plotted against time step. It is clear comparing these two sets of plots that the SLAM algorithm provides improved estimates to the position.

Each of the three position errors have also been plotted along with their associated 3σ bounds error derived from the positional uncertainty, $\boldsymbol{P}_t$, in Figure 5.7-Figure 5.9. It is interesting to note that most of the error introduced into the state estimate is in the Z direction. This may be because there is no direct measurement of the contact

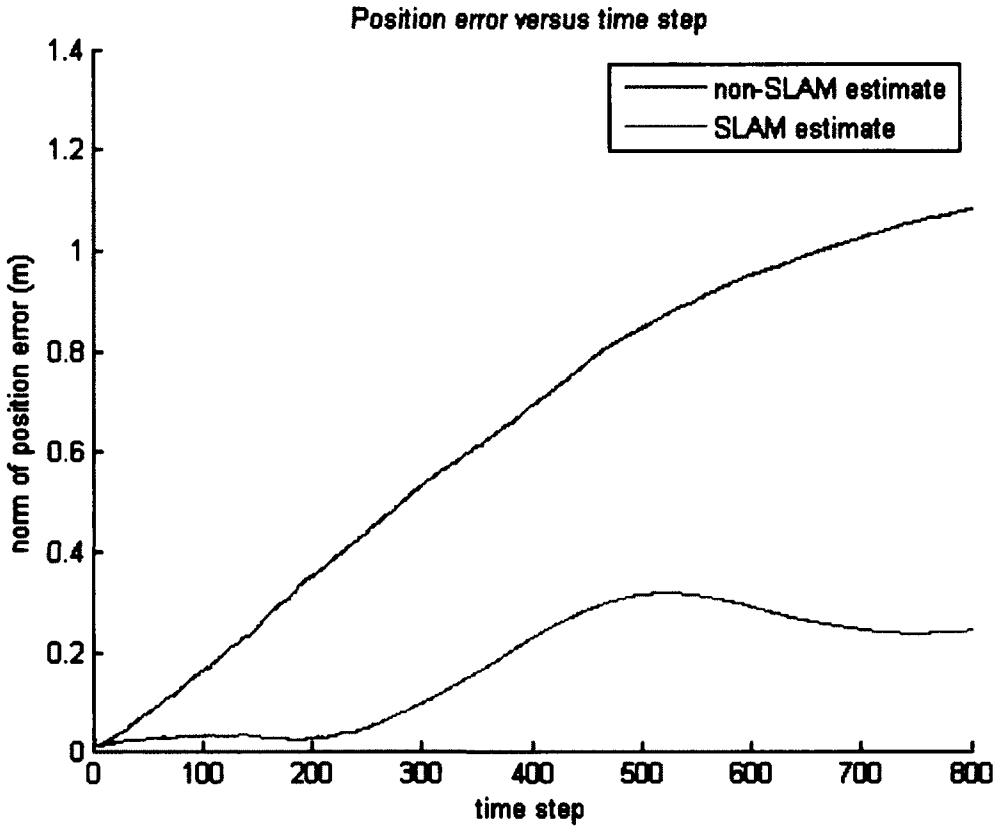


Figure 5.6: Position estimate error comparison.

angle, δ_i, of each wheel. The contact angle error will most affect the Z position of scan points and in turn will have a larger affect on the Z position estimate of the rover. The large jump in error seen in Figure 5.9 can be explained by the fact that before this, the contact angle was relatively accurately estimated. Before the large jump, a scan was taken with an inaccurate contact angle estimate. Because all points in the scan were effected by this wrong contact angle estimate, data association was able to match the points to the previous map, however the change in the Z position of the features is erroneous and has led to a sudden, large error introduced into the Z position of the rover by SLAM.

To solve this issue, a measurement from a sensor such as a load sensor could be introduced to improve the contact angle estimate. It is also important to note that

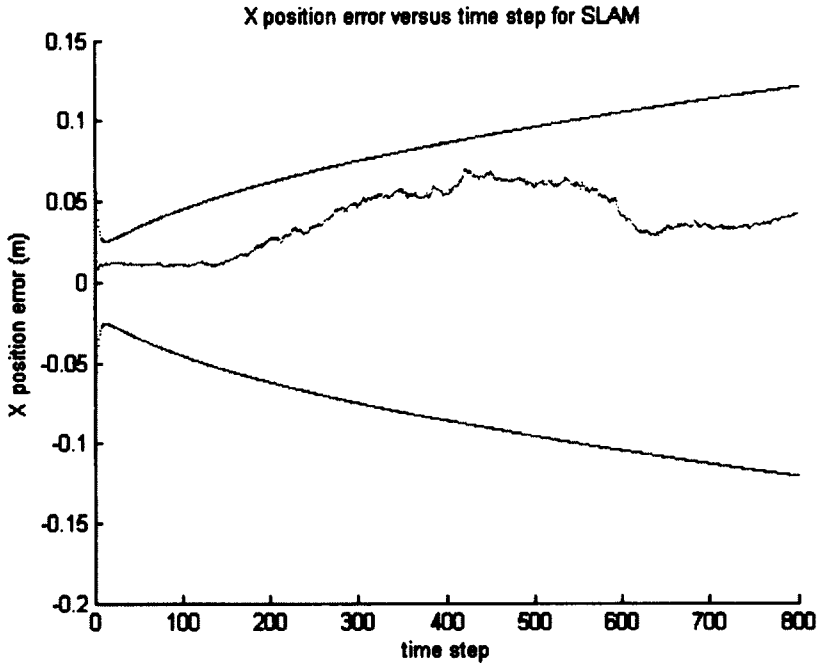


Figure 5.7: X position error.

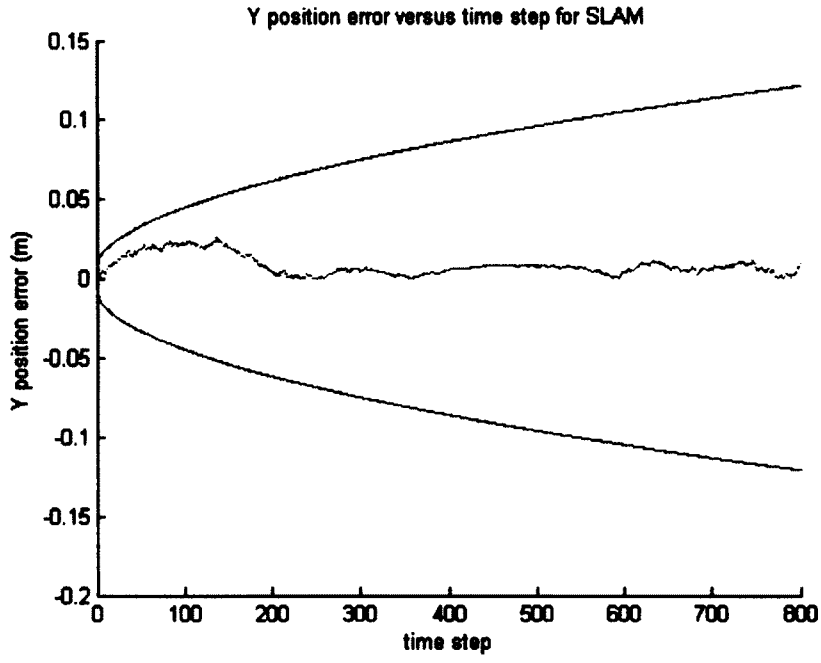


Figure 5.8: Y position error.

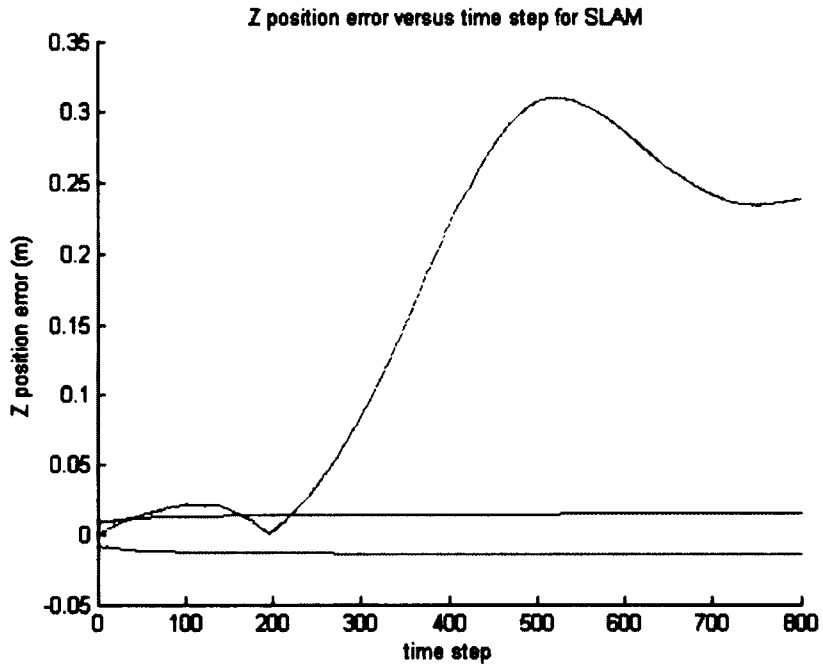


Figure 5.9: Z position error.

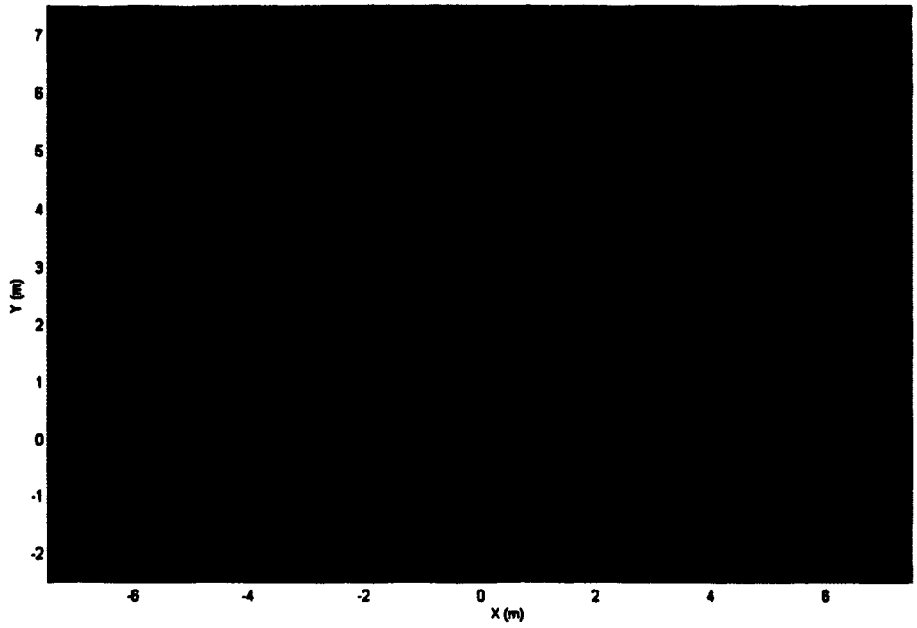


Figure 5.10: Matlab SLAM simulation.

the uncertainty will still rise with each time step unless the "loop" is closed as shown in Figure 2.3. This proves difficult when the distance travelled before closing the loop becomes larger, and often a measure of performance for SLAM is the distance the rover can travel and still successfully close the loop. With the error in Z, this distance is not large. Although SLAM will improve the estimate of the rover, it is not an absolute measurement of position before closing the loop and will introduce some error with each new measurement. Increasing the frequency of scans, increasing the number of particles and introducing measurements of the contact angle would improve the estimate, but that must be balanced against the increased computation necessary to implement these changes. In Figure 5.10 a look at the map and rover estimate in the Matlab simulation is shown.

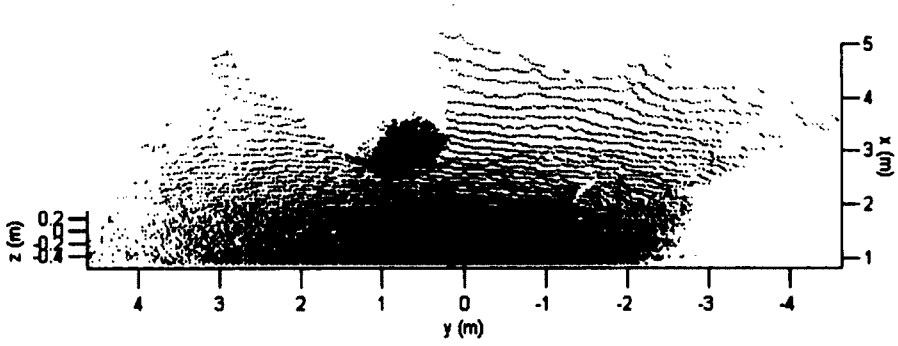
5.2 Real World LIDAR Classification Results

The results in Figure 5.5 show that the neural network was successfully trained to differentiate terrain such as obstacles or non-obstacles in a simulated environment with an average root mean squared error of 0.1 for the classification. The next step is to test the trained neural networks on actual outdoor data taken with the Hokuyo LIDAR scanner. The goal is to observe scans that separate navigable terrain from obstacles by visually inspecting each classification and comparing to pictures taken at the same time to check for accuracy. The obstacles that have been classified can then be used in path planning or data association problems.

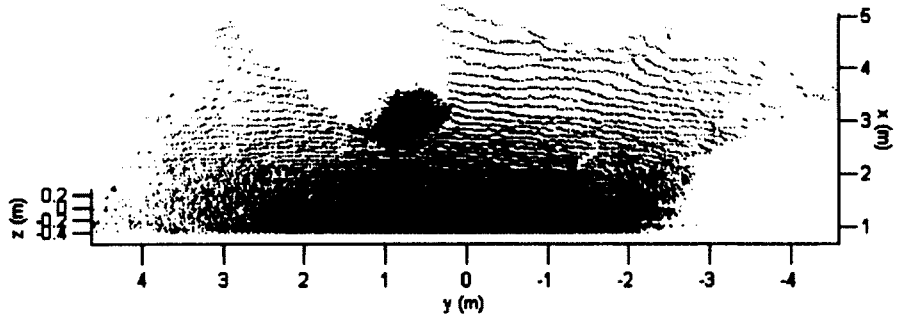
Scans were taken on the Carleton University campus. The environments chosen included large boulders, sidewalks, grass, trees, shrubs and brick walls. This presented a number of varied terrain types for the neural network to classify. An example of a classified scan is shown in Figure 5.12 with an original scan, classified scan, and picture taken from slightly above the LIDAR's point of view. Lightly coloured points

represent traversable terrain whereas non-traversable terrain is darkly coloured. In Figure 5.11b the large boulder to the left seen in Figure 5.11c can be seen to be classified correctly as an obstacle, along with the small rock shown to the extreme right of the image. The boulder that appears in the middle-right of the image does not appear in the LIDAR scan due to its distance from the rover and the limited range of the Hokuyo range finder.

Petrie Island beach was used as a testing ground for *Kapvik's* chassis so the LIDAR classification was tested out on this terrain as well. This terrain is more similar to a Martian environment than the other outdoor measurements taken at Carleton's campus. The SICK LMS LIDAR and Husky rover were used instead of the Hokuyo range finder and Pioneer rover so the range was much larger. To test a variety of features in the laser scans, sand was shovelled and shaped to form slopes. In addition rocks and a bucket were placed in front of the rover to act as obstacles. The sloping terrain should not be identified by the rover as an obstacle in most cases, whereas the rocks and bucket should. A picture of the scan area is shown in Figure 5.12a with the slopes and obstacles labelled. A scan was taken and classified over this area. As in the previous scans, the lightly coloured points represent traversable terrain and darkly coloured points represent obstacles. The corresponding shapes in the classified scan are labelled in Figure 5.12b. The plot shows that all features were correctly identified while leaving the sloping terrain classified as traversable. Due to the success of the outdoor trials the neural network can be implemented within the proposed SLAM algorithm when tested on real world terrain.



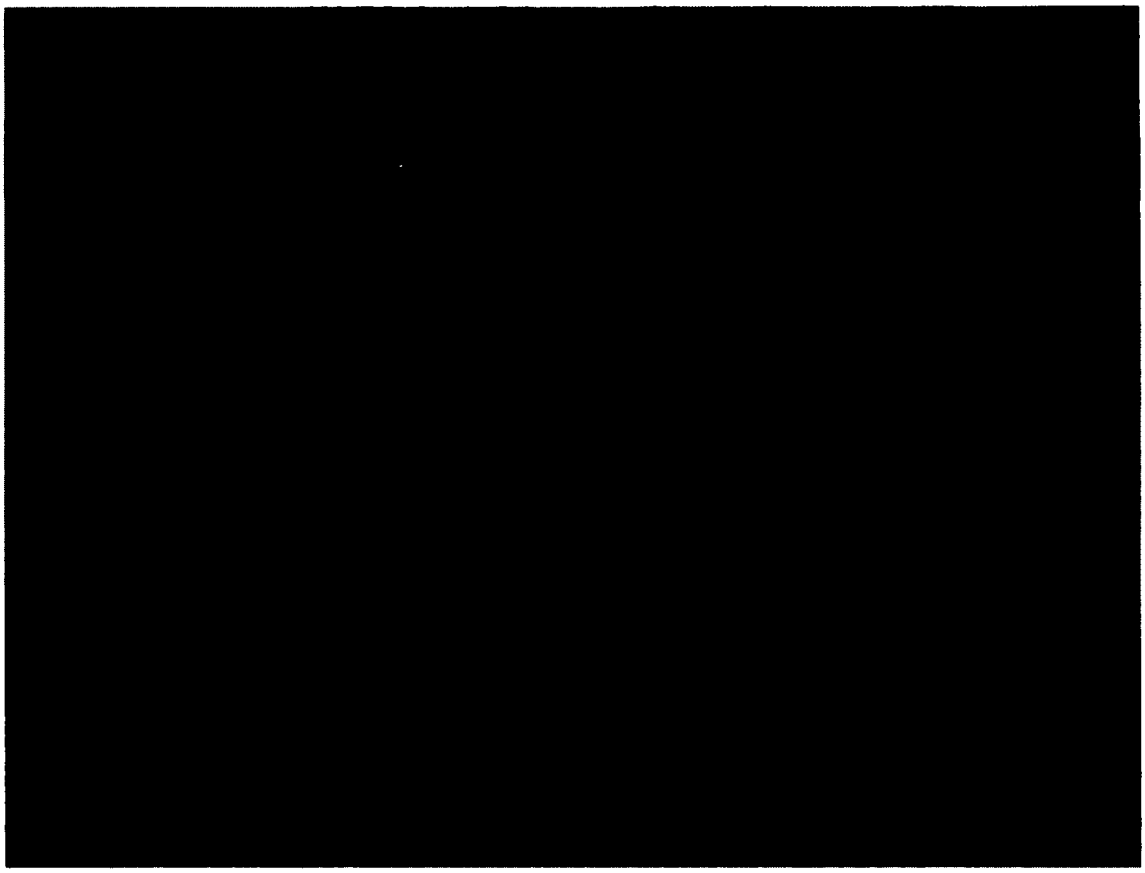
(a) Original scan



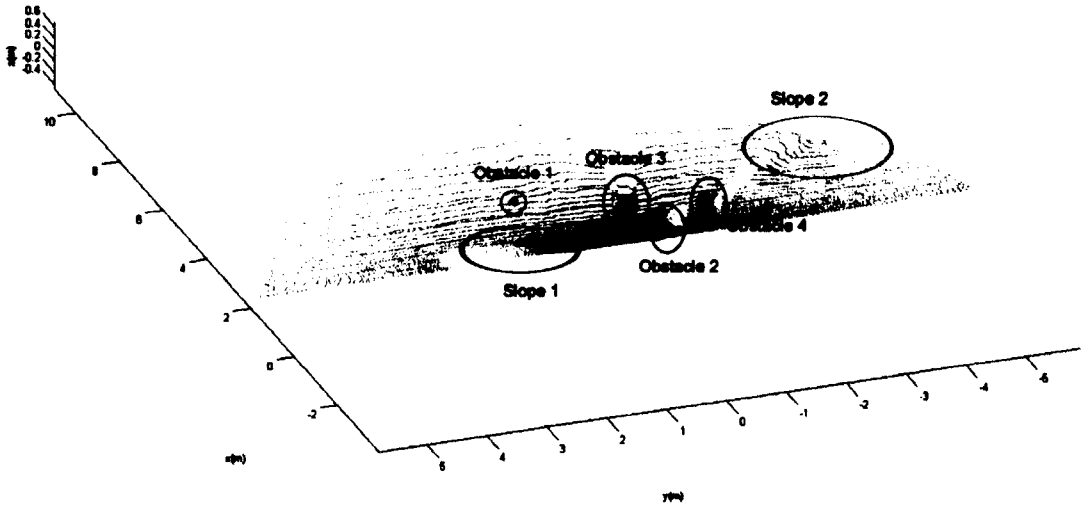
(b) Classified scan with obstacle cells coloured blue

(c) Photograph of scene

Figure 5.11: Comparison of original and classified LIDAR scans on Carleton University campus



(a) Photograph of scene



(b) Classified scan with obstacle cells coloured blue

Figure 5.12: Classified Scan at Petrie Island, Ottawa, Canada

Chapter 6

Conclusions and Recommendations

The goal of thesis is to develop a form of SLAM for an articulated rover that fits the specifications of the *Kapvik* rover. The algorithm estimates the 6D pose of the rover as well as a 3D map of its environment. This problem was approached through efficient LIDAR classification and 6D SLAM that included use of the sensors available to it on *Kapvik*. To validate the algorithms and help train the neural network used for LIDAR classification, a simulation was designed. The performance of the SLAM and neural network LIDAR classification algorithms were validated in simulation using realistic noise properties. Real world tests show that the neural network accurately classifies LIDAR data after being trained through simulation. Thus the over-arching goals of this thesis have been satisfied, however there remain several areas of future work that could improve on the work presented.

6.1 Summary of Contributions

1. The creation of an efficient LIDAR classification technique using neural networks. This method can process large amounts of data in a short amount of time and can also automatically train itself through use of a simulation. By training the neural network on different classes of terrain in the simulation, it

can be easily adapted to multiple types of terrain.

2. The design of a new SLAM algorithm that makes use of an articulated rover motion model and a large suite of sensors including LIDAR. Data association was implemented in a more realistic way that takes into account the whole set of scan data through the use of the Hungarian algorithm. The SLAM algorithm works within the paradigm of a Mars rover mission with limited computation time, doing the majority of computation at "sub-goals" where the rover stops and scans its environment. To improve accuracy and stability a square root Cubature Kalman filter implementation of FastSLAM was developed.

3. The design of a simulation environment that includes a motion model for an articulated rover driving over hilly terrain and a LIDAR sensor model that produces scans of the terrain as well as rocks that are randomly distributed over the surface. This simulation was used for training and validation of the LIDAR classification neural network as well as validation of the SLAM algorithm.

6.2 Recommendations for Future Work

This thesis involves several different areas of research, including neural networks, Baysian Filters, SLAM and kinematics. As a result there are several areas that could be expanded upon and improved from the initial results presented in this thesis. This section presents some of the areas that could be expanded upon, focusing on the areas of LIDAR classification, SLAM and the kinematic model:

(i) The LIDAR classification in this thesis presents a multi-layer feed forward framework that could be extended to many different types of terrain. However, in this thesis only traversable and non-traversable terrain are identified. By simulating other types of terrain within the simulation it would be possible to identify

objects such as walls, trees and other types of objects that classifying beyond a binary relationship may improve the use of LIDAR mapping in path planning and science. It would also be worth investigating more complex forms of neural networks beyond the multi-layer feed forward network such as the recurrent multilayer neural networks or SOM (Self Organizing Maps) that may improve performance or provide alternative ways to approach this problem from a neural network perspective.

(ii) The SLAM algorithm developed in this thesis is a modified version of the Fast-SLAM 2.0 algorithm that makes use of the nonlinear square root Cubature Kalman Filter. The algorithm also incorporates measurements into a cost matrix and applies the Hungarian algorithm to do data association. The main bottleneck in the performance of this algorithm comes in the data association step, where a likelihood must be calculated for all features in the map that are within the perceptual range of the map for each measurement. The form of data association used in this thesis is a consequence of the map representation through point clouds of the LIDAR data. The first area of improvement in the SLAM algorithm would be determining a more appropriate form of map representation such as fitting polygonal shapes to point cloud data, or simply geometric relationships between points in each scan. This could limit the number of "features" in the map, as well as open up more options for data association that are more efficient such as Joint Compatability Branch and Bound (JCBB).

(iii) The kinematic model used for this thesis does not make use of the force sensors located on *Kapvik*. While the model is sound, within the Kalman filter framework the contact angles and their rates for each wheel are following a random

walk model without any direct measurements. For this thesis these had a nominal value of zero so that the overall SLAM algorithm could be tested, but it would be advantageous to develop a model that sensed contact angles through the force sensors located on *Kapvik*. This may require a more developed motion model that makes use of dynamics. It may also be possible to estimate the slip parameters of *Kapvik* by using the change in position estimation before and after the SLAM update. The kinematic model in this thesis is a good basis to work on, incorporating all of these quantities into the motion model.

List of References

[1] J. Fentanes, E. Zalama, and J. Gomez-Garcia-Bermejo, "Algorithm for efficient 3D reconstruction of outdoor environments using mobile robots," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3275–3280, IEEE, 2011.

[2] NASA, "Now a Stationary Research Platform, NASA's Mars Rover Spirit Starts a New Chapter in Red Planet Scientific Studies," 2010.

[3] R. E. Arvidson and L. May, "Mission Concept Study Planetary Science Decadal Survey," Tech. Rep. March, Jet Propulsion Laboratory, Pasadena, California, 2010.

[4] D. Cole and P. Newman, "Using laser range data for 3D SLAM in outdoor environments," *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1556–1563, 2006.

[5] M. Montemerlo and S. Thrun, *FastSLAM - A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics.* New York: Springer, 1 ed., 2007.

[6] Y. Li and E. B. Olson, "A General Purpose Feature Extractor for Light Detection and Ranging Data," *Sensors*, vol. 10, no. 11, pp. 10356–10375, 2010.

[7] J. Yi, J. Zhang, D. Song, and S. Jayasuriya, "IMU-based localization and slip estimation for skid-steered mobile robots," *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2845–2850, 2007.

[8] D. Marzorati, M. Matteucci, and D. Migliore, "Integration of 3d Lines and Points in 6dof Visual Slam by Uncertain Projective Geometry," in *Proceedings of ECMR*, pp. 1–6, 2007.

[9] D. Cole, A. Harrison, and P. Newman, "Using naturally salient regions for SLAM with 3D laser data," in *International Conference on Robotics and Automation, SLAM Workshop*, Citeseer, 2005.

[10] K. Granström and T. Schön, "Learning to close the loop from 3D point clouds," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2089–2095, IEEE, 2010.

[11] M. Tarokh and G. McDermott, "Kinematics modeling and analyses of articulated rovers," *Robotics, IEEE Transactions on*, vol. 21, no. 4, pp. 539–553, 2005.

[12] E. T. Baumgartner, "Rover localization results for the FIDO rover," *Proceedings of SPIE*, vol. 4571, pp. 34–44, 2001.

[13] J. B. Balaram, "Kinematic state estimation for a Mars rover," *Robotica*, vol. 18, no. 3, pp. 251–262, 2000.

[14] D. Simon, *Optimal State Estimation.* Hoboken, New Jersey: John Wiley & Sons, Inc., 2006.

[15] R. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[16] I. Arasaratnam and S. Haykin, "Cubature Kalman Filters," *IEEE Transactions on Automatic Control*, vol. 54, no. 6, pp. 1254–1269, 2009.

[17] J. Bellantoni, "A square root formulation of the Kalman-Schmidt filter.," *AIAA journal*, vol. 5, no. 7, pp. 1309–1314, 1967.

[18] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, vol. 3, p. 26, Spie Bellingham, WA, 1997.

[19] F. Daum, "Nonlinear filters: beyond the Kalman filter," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 20, no. 8, pp. 57–69, 2005.

[20] S. Haykin, *Neural networks and Learning Machines.* Upper Saddle River, New Jersey: Pearson Education, Inc., 3rd ed., 2009.

[21] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes: The Art of Scientific Computing.* New York: Cambridge University Press, 3 ed., 2007.

[22] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the National conference on Artificial Intelligence*, pp. 593–598, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[23] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics*, vol. 5, no. 4, pp. 56–68, 1986.

[24] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proc. 3DIM*, pp. 145–152, Citeseer, 2001.

[25] G. Conte, D. Scaradozzi, S. Zanoli, L. Gambella, and G. Marani, "Underwater SLAM with ICP Localization and Neural Network Objects Classification," in *ISOPE-2008: Eighteenth(2008) International Offshore and Offshore and Polar Engineering Conference Proceedings*, pp. 1–7, International Society of Offshore and Polar Engineers, P. O. Box 189, Cupertino, CA, 95015-0189, USA,, 2008.

[26] A. Nüchter, K. Lingemann, and J. Hertzberg, "6d slam with cached kd-tree search," in *Robot Navigation* (S. Fekete, R. Fleischer, R. Klein, and A. Lopez-Ortiz, eds.), no. 06421 in Dagstuhl Seminar Proceedings, (Dagstuhl, Germany), Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[27] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM with approximate data association," *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pp. 242–249.

[28] P. Newman, J. Leonard, J. Tardos, and J. Neira, "Explore and return: Experimental validation of real-time concurrent mapping and localization," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1802–1809, IEEE, 2002.

[29] P. Corke and J. Trevelyan, *Experimental robotics VI (Lecture Notes in Control and Information Sciences)*, vol. 6. Springer Verlag, 2000.

[30] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *International Joint Conference on Artificial Intelligence*, vol. 18, pp. 1151–1156, Citeseer, 2003.

[31] A. Doucet, N. De Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice.* Springer Verlag, 2001.

[32] S. Thrun, D. Fox, W. Burgard, and Others, "Monte carlo localization with mixture proposal distribution," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 859–865, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.

[33] C. Kim, R. Sakthivel, and W. Chung, "Unscented FastSLAM: A robust and efficient solution to the SLAM problem," *IEEE Transactions on robotics*, vol. 24, no. 4, pp. 808–820, 2008.

[34] T. Bailey, "Slam simulations." `http://www-personal.acfr.usyd.edu.au/tbailey/software/index.html`, January 2012.

[35] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, pp. 661–692, Sept. 2006.

[36] A. Broggi, C. Caraffi, P. P. Porta, P. Zani, and V.-d. Ingegneria, "The Single Frame Stereo Vision System for Reliable Obstacle Detection used during the 2005 DARPA Grand Challenge on TerraMax," in *IEEE Intelligent Transportation Systems Conference*, (Toronto, Canada), pp. 745–752, 2006.

[37] T. Barfoot, "Online visual motion estimation using fastslam with sift features," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Edmonton, Canada), pp. 579–585, IEEE, 2005.

[38] C. Tong, T. Barfoot, and E. Dupuis, "3D SLAM for planetary worksite mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (San Francisco, USA), pp. 631–638, IEEE, Sept. 2011.

[39] C. McManus, P. Furgale, and T. Barfoot, "Towards appearance-based methods for lidar sensors," in *IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 1930–1935, 2011.

[40] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM algorithm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, no. 1, (Beijing, China), pp. 3562–3568, IEEE, 2006.

[41] T. Bailey, J. Nieto, and E. Nebot, "Consistency of the FastSLAM algorithm," in *Proceedings of IEEE International Conference on Robotics and Automation*, no. 1, (Orlando, Florida), pp. 424–429, IEEE, 2006.

[42] M. Cugliari and F. Martinelli, "A FastSLAM algorithm based on the Unscented Filtering with adaptive selective resampling," *Field and Service Robotics*, 2008.

[43] I. Kim, N. Kwak, H. Lee, and B. Lee, "Improved particle fusing geometric relation between particles in FastSLAM," *Robotica*, vol. 27, no. 06, pp. 853–859, 2009.

[44] S. Lacroix, a. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila, "Autonomous Rover Navigation on Unknown Terrains: Functions and Integration," *The International Journal of Robotics Research*, vol. 21, pp. 917–942, Oct. 2002.

[45] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Θ^*: Any-Angle Path Planning on Grids," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, p. 1177, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[46] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *International Joint Conference on Artificial Intelligence*, vol. 18, pp. 1135–1142, LAWRENCE ERLBAUM ASSOCIATES LTD, 2003.

[47] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings*, vol. 1, pp. 206–211, Ieee, 2003.

[48] T. Bailey, E. Nebot, and J. Rosenblatt, "Data association for mobile robot navigation: A graph theoretic approach," *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 3, pp. 2512–2517, 2000.

[49] H. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 52, no. 1, pp. 7–21, 1955.

[50] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied*, vol. 5, no. 1, pp. 32–38, 1957.

[51] J. Bernardo and A. Smith, *Bayesian Theory*. New York: Wiley, 1993.

[52] C. Gamallo, M. Mucientes, and C. Regueiro, "Visual FastSLAM through Omnivision," in *Towards Autonomous Robotic Systems*, (Londonderry, United Kingdom), pp. 128–135, 2009.

[53] L. Fausett, *Fundamentals of neural networks.* New York: Prentice Hall, 1 ed., 1994.

[54] Y. Le Cun, B. Boser, J. Denker, and D. Henderson, "Handwritten Digit Recognition with a Backpropagation Network," in *Advances in Neural Information Processing Systemss*, (San Mateo, CA), pp. 396–404, Morgan Kaufmann, 1990.

[55] A. Hata, D. Wolf, and G. Pessin, "Terrain mapping and classification using neural networks," in *Proceedings of the 2009 International Conference on Hybrid Information Technology*, 2009.

[56] R. Lindemann and C. Voorhees, "Mars Exploration Rover mobility assembly design, test and performance," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 1, pp. 450–455, IEEE, 2005.

[57] I. M. Technology, "Series GL Hollow Shaft Sensors," 2010.

[58] Memsense, "H3-IMU High Performance Inertial Measurement Unit," 2011.

[59] S. Interplanetary, "SS-411 Data Sheet," 2011.

[60] R. Hartenberg and J. Denavit, *Kinematic synthesis of linkages.* New York: McGraw-Hill, 1964.

[61] M. Fogiel, *The Numerical Analysis Problem Solver.* Piscataway, New Jersey: Research & Education Association, 1983.

[62] J. Borenstein and L. Feng, "Gyrodometry: a new method for combining data from gyros and odometry in mobile robots," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 423–428, 1996.

[63] P. Furgale and J. Enright, "Sun sensor navigation for planetary rovers: theory and field testing," *Aerospace and Electronic*, vol. 47, no. 3, p. 1631, 2011.

[64] A. Nüchter, H. Surmann, and J. Hertzberg, "Automatic classification of objects in 3D laser range scans," in *Proc. 8th Conf. Intelligent Autonomous Systems*, pp. 963–970, Citeseer, 2004.

[65] T. Masters, *Practical neural network recipes in C++.* San Diego, CA: Academic Press, Inc., 1993.

[66] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[67] E. Haines, "Fast Ray-Convex Polyhedron Intersection," in *Graphics Gems II* (J. Arvo, ed.), pp. 247–250, San Diego: Academic Press, 1991.

Appendix A

Kinematic Jacobian Elements

Elements of the Jacobian matrices in (3.15) and (3.16) are shown here and were presented by Tarokh et al. in [11]. For (3.15), corresponding to the front two wheels, the Jacobian elements are:

$$
\begin{aligned}
J_{x,\theta} &= k_{10}(b_i s\rho s\delta_i + c\rho c\delta_i), \\
J_{x,\xi} &= b_i s\rho s\delta_i + c\rho c\delta_i, \\
J_{x,\zeta} &= -k_3 s\rho s\delta_i - b_i k_3 c\rho c\delta_i, \\
J_{x,\eta} &= 0, \\
J_{x,\delta} &= -k_5 c\rho + b_i k_4 s\rho, \\
J_{y,\theta} &= 0, \\
J_{y,\xi} &= 0, \\
J_{y,\zeta} &= -k_5 s\delta_i - k_4 c\delta_i, \\
J_{y,\eta} &= 1, \\
J_{y,\delta} &= 0, \\
J_{z,\theta} &= k_{10}(b_i s\rho c\delta_i - c\rho s\delta_i),
\end{aligned}
$$

$$
\begin{aligned}
J_{z,\xi} &= b_i s\rho c\delta_i - c\rho s\delta_i, \\
J_{z,\zeta} &= b_i k_3 c\rho s\delta_i - k_3 s\rho c\delta_i, \\
J_{z,\eta} &= 0, \\
J_{z,\delta} &= -b_i k_5 s\rho - k_4 c\rho, \\
J_{\phi_x,\zeta} &= c\rho s\delta_i - b_i s\rho c\delta_i, \\
J_{\phi_x,\delta} &= 0, \\
J_{\phi_y,\zeta} &= 0, \\
J_{\phi_y,\delta} &= -1, \\
J_{\phi_z,\zeta} &= c\rho c\delta_i + b_i s\rho s\delta_i, \\
J_{\phi_z,\delta} &= 0,
\end{aligned}
$$

where c and s represent the cosine and sine respectively. For (3.16), corresponding to the middle and back wheels, the Jacobian elements are:

$$
\begin{aligned}
J_{x,\beta_L} &= \frac{1}{2}\left(b_i - 1\right) k_6 s\left(k_9 + \rho\right), \\
J_{x,\beta_R} &= -\frac{1}{2}\left(b_i + 1\right) k_6 s\left(k_9 - \rho\right), \\
J_{x,\zeta} &= -b_i k_3 c\sigma_i, \\
J_{x,\delta} &= -k_6 s\left(k_9 - b_i\rho\right) - a_{si} s\left(\sigma_i - \delta_i\right) - k_8 c\left(\sigma_i - \delta_i\right), \\
J_{y,\zeta} &= -k_6 c\left(k_9 - b_i\rho - \sigma_i\right) - a_{si} c\delta_i - k_8 s\delta_i, \\
J_{z,\beta_L} &= \frac{1}{2}\left(b_i - 1\right) k_6 c\left(k_9 + \rho\right), \\
J_{z,\beta_R} &= -\frac{1}{2}\left(b_i + 1\right) k_6 c\left(k_9 - \rho\right), \\
J_{z,\zeta} &= b_i k_3 s\delta_i, \\
J_{z,\delta} &= -k_6 c\left(k_9 - b_i\rho\right) - a_{si} c\left(\sigma_i - \delta_i\right) + k_8 s\left(\sigma_i - \delta_i\right),
\end{aligned}
$$

$$J_{\phi_y,\beta_L} = \frac{1}{2}(b_i - 1),$$
$$J_{\phi_y,\beta_R} = -\frac{1}{2}(b_i + 1),$$

where σ_i is defined in (3.17) and a_{si} is defined as

$$a_{si} = \begin{cases} k_7 & i = 3,3 \\ -k_7 & i = 5,6 \end{cases} . \tag{A.1}$$

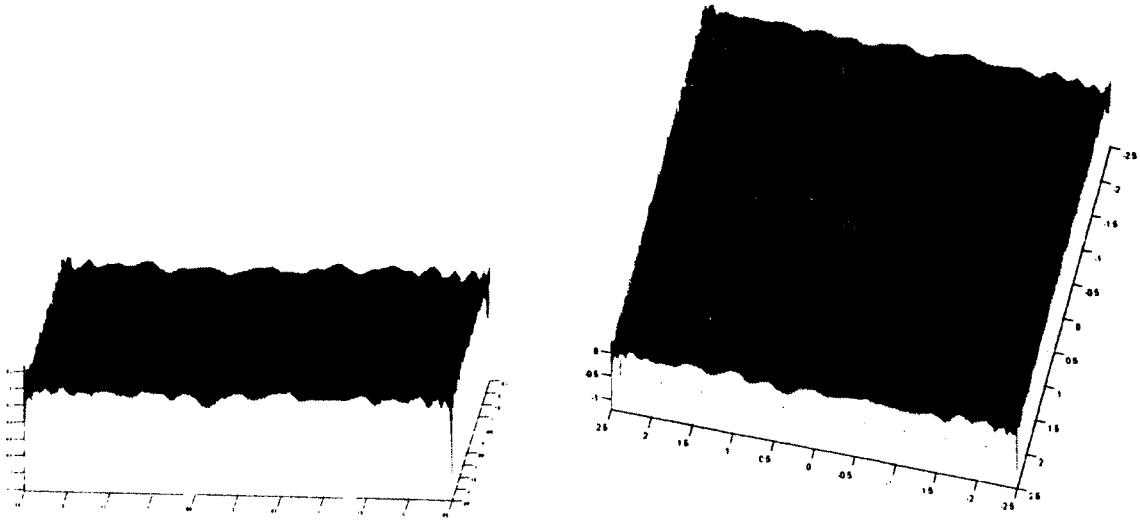
Appendix B

"Serpentine" Path Kinematic Parameters

Two path types were tested with the *Kapvik* kinematic model. The first was a straight path over bumpy terrain and the second was a "serpentine" path over bumpy terrain. This appendix shows the results of the "serpentine" path. The serpentine path was created by setting the wheel angular speeds appropriately. The left wheels were set to an angular speed of $\dot{\theta}_L = \frac{0.02m/s|\sin(\pi t/100)|}{r_w}$ and the right wheels were set to an angular speed of $\dot{\theta}_R = \frac{0.02m/s|\cos(\pi t/100)|}{r_w}$.

The front two wheel positions were traced along the path of the rover and are displayed in Figure B.1. This test should show that the appropriate path is carried out given the rover's input commands for wheel angular velocity. To see this, the rover's front wheel positions are traced out as it drives over the surface. Just as in the straight path tests, the results should show are an overall roll and pitch that remain small compared to the contact angles of the wheels. The rocker should also exhibit less motion than the two bogies. Finally the contact angles should be independent from one another as each wheel is traversing a different part of the terrain.

Figure B.2 shows the rover body roll ϕ_x and pitch ϕ_y as the result of the combined motion of all six wheels. It can be seen that the roll and pitch are kept relatively small, and seem to oscillate about zero degrees as it moves over the bumpy terrain. This shows the behaviour that is desired from a rocker-bogie system, where the rover



(a) Perspective view (b) Overhead view

Figure B.1: Traces of front wheels over bumpy terrain

body roll and pitch are kept at a minimum as it moves over nonuniform terrain. Figure B.3 shows the rocker ρ and two bogie angles β_1 and β_2. The rocker angle is shown as expected to be much smaller in magnitude than the two bogie angles. The two bogie angles also exhibit oscillating, independent behaviour from one another. In Figure B.4 the contact angles are shown for the two front wheels. The contact angles follow an oscillating path as the rover traverses the bumpy terrain, with each wheel having an independent contact angle based on the terrain it is traversing.

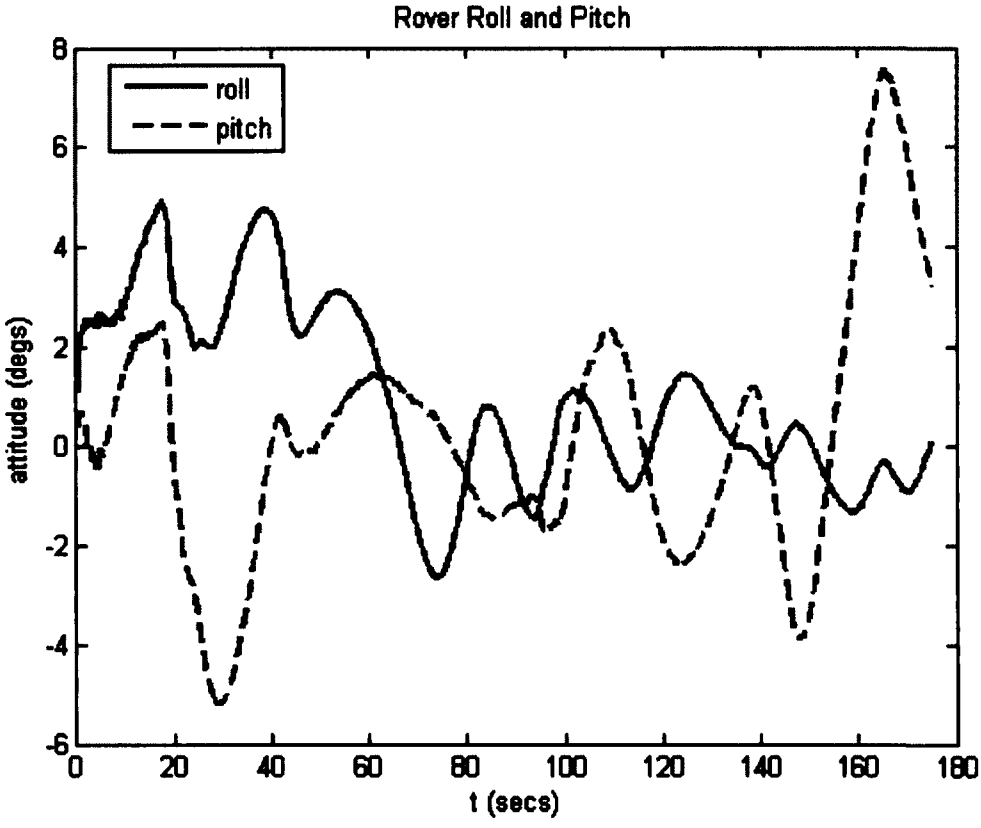


Figure B.2: Rover roll and pitch for "serpentine" path

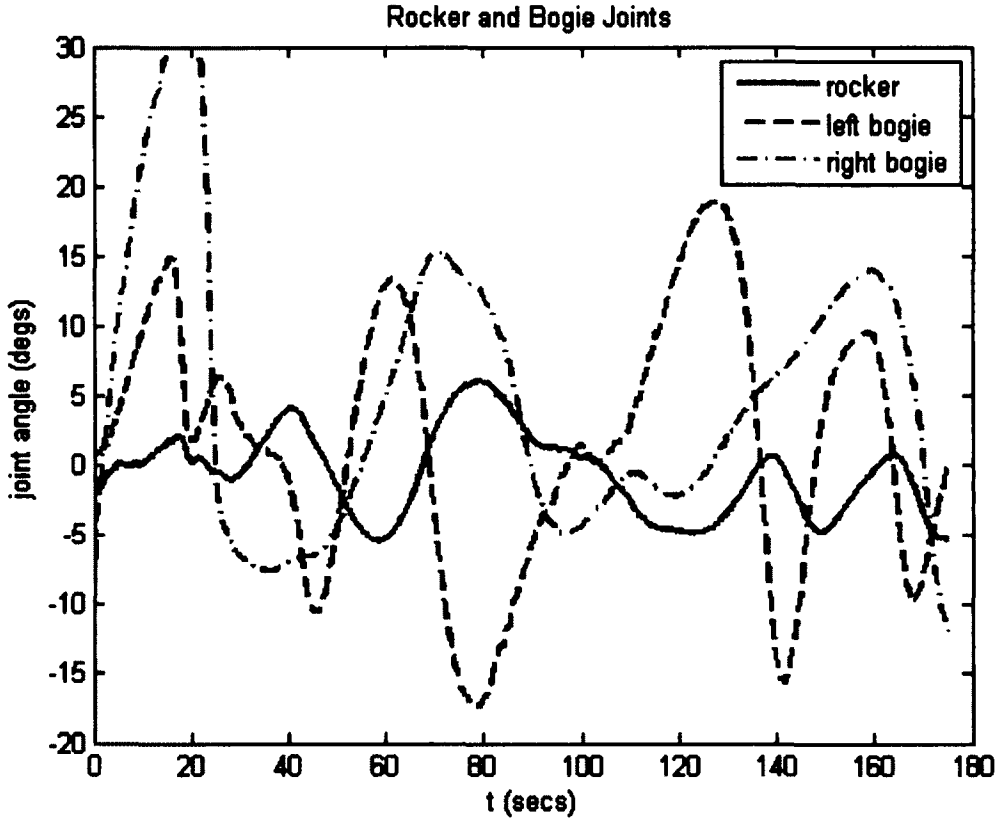


Figure B.3: Rocker and bogie joint angles for "serpentine" path

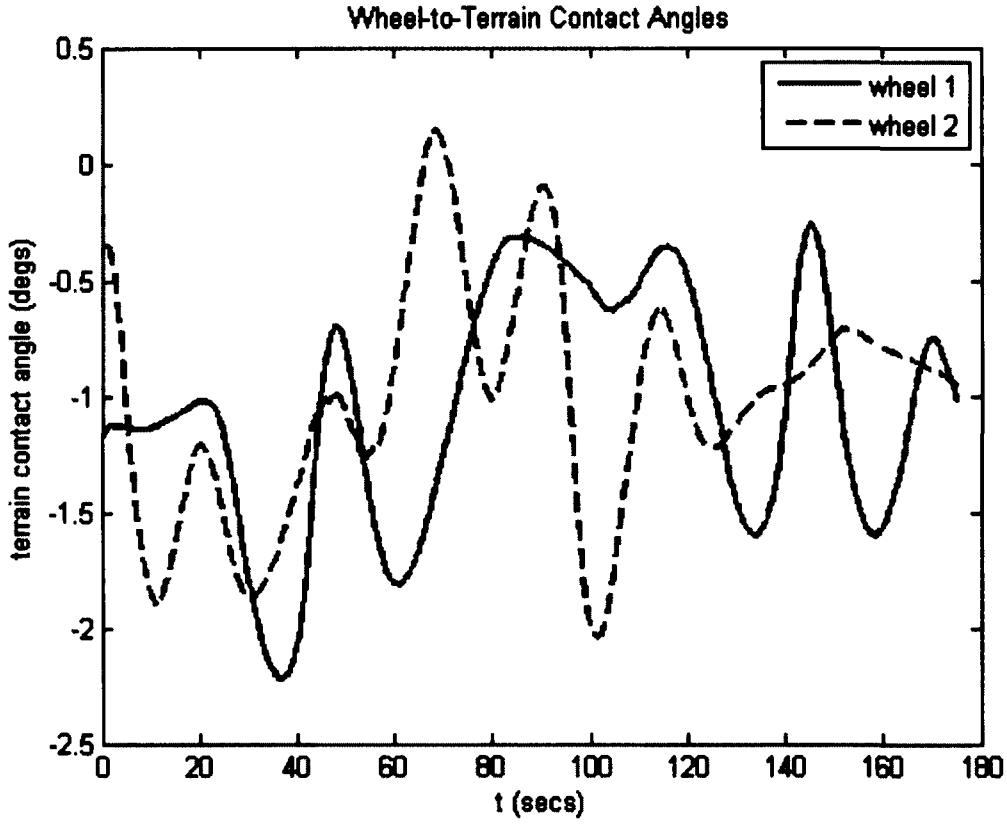


Figure B.4: Contact angle for the front wheels on "serpentine" path