Adaptive Rover Navigation in the Presence of Unmodelled Slip

by

Mark A. Swartz

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 2009

Copyright ©

2009 - Mark A. Swartz

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: 978-0-494-52032-1
Our file Notre référence
ISBN: 978-0-494-52032-1

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Autonomous navigation of planetary robotic rovers through sandy, rock-strewn terrain requires that onboard algorithms provide accurate models of the rover kinematics, sensors, and environment in order to localize the robot in its map accurately. Rover slip presents a major challenge in such an environment since formulating analytic slip models a priori using terramechanics can be difficult. This thesis develops a slip-adaptive navigation algorithm that can model slip empirically online as the rover traverses the environment. The slip-adaptive algorithm essentially augments the rover kinematic model with terms that approximate the unmodelled slip. These slip terms are estimated by a multi-layer feedforward neural network (MLFN), which uses an extended Kalman filter (EKF) to estimate the network weights while it also provides the rover state estimates. Slip is learned and thus detected autonomously, eliminating the need for laborious human commands uploaded once per Martian sol. The Mars Exploration Rovers have already demonstrated the benefits of implementing more autonomy into these systems, including shortened traverse times, increased scientific exploration, and prolonged rover survivability.

Acknowledgments

It is with much gratitude that I wish to thank Professor Alex Ellery, Canada Research Chair in Space Robotics and Space Technology, for taking me on as one of his students in the middle of my masters program and providing me with the opportunity to conduct research in the field of space robotics. As my Thesis Supervisor, his repeated expression of confidence in my abilities combined with the the freedom and flexibility he afforded me, allowed me to take the project in a direction that suited my interests and thus helped me make the project my own.

This thesis would not have been possible without the keen engineering insight and guidance of my Co-Supervisor, Professor Joshua Marshall. Over the course of many hours of constructive feedback and problem solving which he tirelessly and generously provided, this thesis evolved into a more theoretically grounded work of research. I am indeed fortunate for his timely arrival at Carleton, coming from MacDonald, Detwiler and Associates Ltd., as I became the beneficiary of the expertise of this highly professional and talented control systems engineer. If there was ever an engineer to emulate in terms of dedication and competence, it would be him.

I would also like to thank Mr. Ali Morbi, a highly talented engineering M.A.Sc. candidate, for the insights he shared over the course of my project. His knowledge into neural network architectures, their benefits and deficiencies, provided me with many additional and unexpected learning opportunities which helped me to advance the goals of this thesis.

A special acknowledgement is due to Professor Anton de Ruiter who kindly pointed me in the right direction twice over. Thanks to his suggestions full observability was realized at last!

Finally, I would like to acknowledge all the graduate student engineers, both M.A.Sc. and Ph.D candidates alike, for their support and friendship. As well, I owe much thanks and gratitude to the staff of the Department of Mechanical and Aerospace Engineering, for the professional and timely assistance they provided over the course of my studies.

Table of Contents

Abstract	iii
Acknowledgments	iv
Table of Contents	vi
List of Tables	x
List of Figures	xi
Nomenclature	xx
1 Introduction	1
1.1 Autonomous Navigation of Mars	3
1.1.1 Slip Estimation using Visual Odometry	5
1.1.2 Future Missions to Mars	7
1.2 Problem Statement	8
1.2.1 Simulation Environment	9
1.2.2 Slip-Adaptive EKF-MLFN Algorithm	10
1.2.3 Research Scope	12
1.3 Thesis Organization	13
2 Rover Navigation	15

2.1	The Extended Kalman Filter	17
2.1.1	EKF Algorithm	19
2.1.2	Observability and Detectability	21
2.2	Rover Localization Algorithm	24
2.2.1	Environment Model	24
2.2.2	Rover Kinematic Model	26
2.2.3	Measurement Model	30
2.2.4	Summary of Model Assumptions	32
2.2.5	EKF Observability for Rover Localization	33
2.3	Rover Slip Model	34
2.4	Rover Localization in the Presence of Unmodelled Slip	37
2.4.1	Example 1: Perfect Rover Model	37
2.4.2	Example 2: Unmodelled Slip-Induced Dynamics	39
2.4.3	Research Justification	39
2.5	Literature Review:	
	Current Slip modelling Research	41
2.5.1	A Priori modelled Slip	41
2.5.2	MER Slip Model Development	42
2.5.3	Remote Slip Estimation	45
3	Neural Network Enabled Adaptation	48
3.1	Multi-Layer Feed-Forward Neural Networks	50
3.1.1	Universal Function Approximators	50
3.1.2	Error Back-Propagation Training	55
3.2	Literature Review:	
	Hybrid EKF-Neural Networks	57
3.2.1	EKF-Based Training	58

3.2.2	Neural Networks as Nonparametric Plant Dynamics	59
3.2.3	Hybrid Navigation Applications	61
3.3	EKF Trained Neural Networks	65
3.3.1	EKF Training Algorithm	66
3.3.2	EKF Training Examples	70
3.3.3	Algorithm Convergence and Approximation Capability	96
3.4	Adaptive EKF Using Neural Networks	97
4	Adaptive Rover Navigation	100
4.1	Slip Model Formulation	100
4.1.1	Environment Model	101
4.1.2	Rover Kinematic Model	103
4.1.3	Slip Kinematic Model	104
4.1.4	Measurement Model	108
4.1.5	Summary of Model Assumptions	109
4.2	Adaptive EKF-NN Algorithm	110
4.2.1	State Space Augmentation	110
4.2.2	Linerization	111
4.3	Slip-Adaptive Navigation Simulations	116
4.3.1	Training the MLFN Slip-Adaptive Model	116
4.3.2	Proof-of-Match	133
4.3.3	Slip Model Validation	138
5	Conclusions	147
5.1	Summary of Contributions	149
5.2	MLFN-Slip Modelling Recommendation	149
5.3	Future Research Areas	151
5.4	Final Remarks	152

List of References	153
Appendix A Slip-Adaptive Navigation Algorithm:	
MATLAB Implementation	160
Appendix B To Observe and Detect: Policing the Kalman Filter	186
B.1 Observability and Detectability	186
B.2 Decomposition of Observable and Unobservable Components	189
Appendix C Weight Convergence: 10 Neuron MLFN	193
Appendix D Slip-Model Validation Results Over the Training Path	204
Appendix E Computing Observability in MATLAB	206

List of Tables

1	Mean Squared Error versus Number of Neurons	124
2	First Simulation	146
3	Second Simulation	146

List of Figures

1	Mars Exploration Rover. Online source: NASA.	2
2	Spirit's view of the Martian terrain. Online source: NASA.	8
3	A GNC system with the navigation component high-lighted. Adapted from [1].	16
4	The rover simulation environment.	25
5	Time history of terrain inputs about the rover pitch and roll axes. . .	26
6	Range and bearing to a feature detected in the simulated 2.25D environment.	31
7	Slip experienced in the rover axes is transformed to the global axes: $\mathbf{S}_g = [S_X \ S_Y \ S_\Psi]$	35
8	Perfect Rover Model: The EKF is able to track the commanded path perfectly resulting in estimate errors that fall within the predicted 3σ error bounds.	38
9	Unmodelled Slip-Induced Dynamics: The EKF is barely able to track the true path resulting in estimate errors that far exceed 3σ error bounds.	40
10	Longitudinal (top) and lateral (bottom) slip versus slope measurements taken from [2].	44
11	A three-layer multi-layer feed-forward neural network.	51
12	A sigmoid activation function is continuous and everywhere differentiable.	53
13	EKF training algorithm for the MLFN.	68

14	Example 1: MLFN approximation of $\sin(45^\circ) = 0.707$. $N = 1, W = 3$, $r_q = 1$. Oscillation is caused by the noise parameters.	72
15	Example 1: Proof-of-Match results for $\sin(\mathbf{x}_k)$, with $\mathbf{x}_k = \{30^\circ, 45^\circ,$ $60^\circ\}$	72
16	Example 1: Weight 1 convergence over 600 seconds.	74
17	Example 1: Weight 2 convergence over 600 seconds.	74
18	Example 1: Weight 3 convergence over 600 seconds.	75
19	Example 1: Observability Grammian for $N = 1$ and $r_q = 1$ indicates the system is observable, $n_q = 3$	75
20	Example 1: Weight 1 convergence over 2000 seconds. $N = 1, W = 3$, $r_q = 2$	76
21	Example 1: Weight 2 convergence over 2000 seconds. $N = 1, W = 3$, $r_q = 2$	76
22	Example 1: Weight 3 convergence over 2000 seconds. $N = 1, W = 3$, $r_q = 2$	77
23	Example 2: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{30^\circ, 45^\circ, 60^\circ\}$, $N = 1, W = 3, r_q = 3$	79
24	Example 2: Weight 1 convergence over 600 seconds.	79
25	Example 2: Weight 2 convergence over 600 seconds.	80
26	Example 2: Weight 3 convergence over 600 seconds.	80
27	Example 2: Observability Grammian for $N = 1$ and $r_q = 3$ indicates the system is observable, $n_q = 3$	81
28	Example 2: Proof-of-Match results for $\sin(\mathbf{x}_k)$, $\mathbf{x}_k = \{30^\circ, 45^\circ, 60^\circ\}$. .	81
29	Example 2: Proof-of-Match results for $\sin(\mathbf{x}_k)$, $\mathbf{x}_k = \{10^\circ, 25^\circ, 50^\circ, 80^\circ\}$.	82
30	Example 2: Proof-of-Match results versus angle input for $\sin(\mathbf{x}_k)$, with $\mathbf{x}_k = \{0^\circ : 90^\circ\}$	82

31	Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{30^\circ, 45^\circ, 60^\circ, 75^\circ\}$, $N = 1$, $W = 3$, $r_q = 4$	84
32	Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{10^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ\}$, $N = 1$, $W = 3$, $r_q = 5$	84
33	Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{0^\circ, 10^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ\}$, $N = 1$, $W = 3$, $r_q = 7$	85
34	Example 4: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = [30^\circ, 45^\circ, 60^\circ]$, $N = 2$, $W = 6$, $r_q = 3$	87
35	Example 4: Proof-of-Match results for $\sin(\mathbf{x}(j)_k)$, $\mathbf{x}(j)_k = \{30^\circ, 45^\circ, 60^\circ\}$	87
36	Example 4: Slow Weight 1 convergence over 2000 seconds.	89
37	Example 4: Slow Weight 2 convergence over 2000 seconds.	89
38	Example 4: Slow Weight 3 convergence over 2000 seconds.	90
39	Example 4: Slow Weight 4 convergence over 2000 seconds.	90
40	Example 4: Slow Weight 5 convergence over 2000 seconds.	91
41	Example 4: Slow Weight 6 convergence over 2000 seconds.	91
42	Example 4: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ, 90^\circ\}$, $N = 2$, $W = 6$, $r_q = 6$	92
43	Example 4: Proof-of-Match results for $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{0^\circ, 10^\circ, 20^\circ, 30^\circ, 40^\circ, 50^\circ, 60^\circ, 70^\circ, 80^\circ, 90^\circ\}$	92
44	Example 4: Fast Weight 1 convergence over 600 seconds.	93
45	Example 4: Fast Weight 2 convergence over 600 seconds.	93
46	Example 4: Fast Weight 3 convergence over 600 seconds.	94
47	Example 4: Fast Weight 4 convergence over 600 seconds.	94
48	Example 4: Fast Weight 5 convergence over 600 seconds.	95
49	Example 4: Fast Weight 6 convergence over 600 seconds.	95
50	The EKF-MLFN slip-adaptive algorithm framework (for simulation).	101

51	Single hidden layer feed-forward neural network for adaptive slip modelling.	105
52	The commanded and true path of the simulated rover during training.	117
53	Time history of training velocity control inputs.	119
54	Time history of exogenous terrain inputs from ground geometry. . . .	119
55	The EKF achieved full observability for $N = 1$	122
56	Poor MLFN-slip approximation results for $N = 1$	122
57	Large localization errors in the rover state estimates for $N = 1$	123
58	The slip-adaptive algorithm performs poorly with $N = 1$	123
59	The EKF achieved full observability for $N = 10$	125
60	Good MLFN-slip approximation results for $N = 10$	125
61	Improved localization errors in the rover state estimates for $N = 10$. .	126
62	The slip-adaptive algorithm performs well with $N = 10$	126
63	The EKF achieved full observability for $N = 20$	127
64	Excellent MLFN-slip approximation results for $N = 20$	127
65	Improved localization errors in the rover state estimates for $N = 20$. .	128
66	The slip-adaptive algorithm performs very well with $N = 20$	128
67	1 Neuron MLFN: Weight convergence for W_1	130
68	1 Neuron MLFN: Weight convergence for W_2	130
69	1 Neuron MLFN: Weight convergence for W_3	131
70	1 Neuron MLFN: Weight convergence for W_4	131
71	1 Neuron MLFN: Weight convergence for W_5	132
72	1 Neuron MLFN: Weight convergence for W_6	132
73	1 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis. . .	134
74	1 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis. . .	134
75	10 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis. .	135
76	10 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis. .	135

77	20 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis. .	136
78	20 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis. .	136
79	20 Neuron MLFN: Five-Run Proof-of-Match for rover slip in the rover x -axis.	137
80	20 Neuron MLFN: Five-Run Proof-of-Match for rover slip in the rover y -axis.	137
81	The commanded and true path of the simulated rover during validation.	139
82	Time history of new path validation velocity control inputs.	140
83	Time history of exogenous terrain inputs over the new path.	140
84	MLFN-slip validation results over the new path before tuning.	141
85	The trained (not tuned) slip-adaptive algorithm provides an excellent slip approximation over the new path.	141
86	Validation rover state errors for the slip-adaptive algorithm using train- ing process noise (not tuned).	143
87	Validation rover state errors for the tuned slip-adaptive algorithm. . .	143
88	Tuned MLFN-slip validation results over the new path.	144
89	The trained (and tuned) slip-adaptive algorithm provides an excellent slip approximation over the new path.	144
90	Weight convergence for W_1	194
91	Weight convergence for W_2	194
92	Weight convergence for W_3	194
93	Weight convergence for W_4	194
94	Weight convergence for W_5	194
95	Weight convergence for W_6	194
96	Weight convergence for W_7	195
97	Weight convergence for W_8	195
98	Weight convergence for W_9	195

99	Weight convergence for W_{10} .	195
100	Weight convergence for W_{11} .	195
101	Weight convergence for W_{12} .	195
102	Weight convergence for W_{13} .	196
103	Weight convergence for W_{14} .	196
104	Weight convergence for W_{15} .	196
105	Weight convergence for W_{16} .	196
106	Weight convergence for W_{17} .	196
107	Weight convergence for W_{18} .	196
108	Weight convergence for W_{19} .	197
109	Weight convergence for W_{20} .	197
110	Weight convergence for W_{21} .	197
111	Weight convergence for W_{22} .	197
112	Weight convergence for W_{23} .	197
113	Weight convergence for W_{24} .	197
114	Weight convergence for W_{25} .	198
115	Weight convergence for W_{26} .	198
116	Weight convergence for W_{27} .	198
117	Weight convergence for W_{28} .	198
118	Weight convergence for W_{29} .	198
119	Weight convergence for W_{30} .	198
120	Weight convergence for W_{31} .	199
121	Weight convergence for W_{32} .	199
122	Weight convergence for W_{33} .	199
123	Weight convergence for W_{34} .	199
124	Weight convergence for W_{35} .	199
125	Weight convergence for W_{36} .	199

126	Weight convergence for W_{37}	200
127	Weight convergence for W_{38}	200
128	Weight convergence for W_{39}	200
129	Weight convergence for W_{40}	200
130	Weight convergence for W_{41}	200
131	Weight convergence for W_{42}	200
132	Weight convergence for W_{43}	201
133	Weight convergence for W_{44}	201
134	Weight convergence for W_{45}	201
135	Weight convergence for W_{46}	201
136	Weight convergence for W_{47}	201
137	Weight convergence for W_{48}	201
138	Weight convergence for W_{49}	202
139	Weight convergence for W_{50}	202
140	Weight convergence for W_{51}	202
141	Weight convergence for W_{52}	202
142	Weight convergence for W_{53}	202
143	Weight convergence for W_{54}	202
144	Weight convergence for W_{55}	203
145	Weight convergence for W_{56}	203
146	Weight convergence for W_{57}	203
147	Weight convergence for W_{58}	203
148	Weight convergence for W_{59}	203
149	Weight convergence for W_{60}	203
150	MLFN-slip validation results over the original training path.	205
151	The trained slip-adaptive algorithm provides an excellent slip approx- imation over the original path.	205

152	Erroneous loss of rank of the observability Grammian for an observable system.	208
153	Expected rank development of the observability Grammian for an observable system.	208

Nomenclature

DOF	Degree of Freedom
EKF	Extended Kalman Filter.
LTI	Linear time-invariant
LTV	Linear time-varying
MLFN	Multi-Layer Feedforward Neural Network.
$f(\cdot)$	Nonlinear rover dynamics model.
$\mathbf{F}_k$	Linearized rover dynamics model at time step k .
g_o	Neural network output from the o^{th} output neuron.
$\mathbf{g}_{O \times 1}$	Neural network output vector.
$h(\cdot)$	Nonlinear sensor measurement model.
$\mathbf{H}_k$	Linearized sensor measurement model at time step k .
$\mathbf{i}$	Neural network inputs.
$\mathbf{I}_W$	Identity matrix of square dimension W .
k	The k^{th} time step.
$\mathbf{K}_k$	Kalman gain at time step k .

m_q	Dimension of the EKF control input vector.
M	Number of neural network inputs.
n_q	Dimension of the EKF state vector.
$n_{(\cdot)}$	Noise on $(\cdot) = \{v, \omega, \phi, \theta, \psi, r_{f,i}, \beta_{f,i}\}$.
N	Number of neural network hidden layer neurons.
$\mathcal{N}(\bar{\mathbf{q}}, \mathbf{P})$	Gaussian distribution with mean, $\bar{\mathbf{q}}$, and covariance $\mathbf{P}$.
O	Number of neural network outputs.
$\mathbf{O}_{obs,k}$	Observability matrix at time step k .
$\mathbf{P}_k$	Covariance matrix at time step k .
$\mathbf{q}_k$	State vector at time step k .
$\mathbf{Q}$	Process noise covariance matrix.
$r_{f,i}$	Range distance to feature i with respect to the rover [m].
r_q	Dimension of the EKF measurement vector.
r_w	Rover wheel radius [m].
R	Linear regression correlation coefficient.
$\mathbf{R}$	Measurement noise covariance matrix.
$\mathbf{R}_{XYZ}^{xyz}$	Rotation matrix from local to global frame.

$\mathbf{R}_X$	Rotation matrix about the global X -axis.
$\mathbf{R}_Y$	Rotation matrix about the global Y -axis.
$\mathbf{R}_Z$	Rotation matrix about the global Z -axis.
$\mathbf{S}_g$	Neural network approximated rover slip vector.
$\mathbf{S}_r$	Rover slip vector.
Sx_r	MLFN-predicted rover slip in the rover x-axis [%].
Sy_r	MLFN-predicted rover slip in the rover y-axis [%].
$S\psi_r$	MLFN-predicted rover slip about the rover z-axis [%].
Sx_t	Environmental truth slip in the rover x-axis [m].
Sy_t	Environmental truth slip in the rover y-axis [m].
$S\psi_t$	Environmental truth slip about the rover z-axis [deg].
T	The sample time or length of each k^{th} time step in seconds [s].
$\mathbf{T}$	Unobservable part separation transformation matrix.
$\mathbf{u}_{c,k} = [v \ \omega]^T$	Control input vector at time step k .
$\mathbf{u}_{e,k} = [\phi \ \theta]^T$	Terrain input vector at time step k .
v	Commanded rover forward velocity, $v = \omega r_w$, [m/s].
v_t	True rover forward velocity [m/s].
v_n^1	n^{th} hidden neuron local field of a neural network.
$\mathbf{v}_{n,k}$	Zero-mean, white, Gaussian measurement noise vector.

$w_{nm,k}^L$	L^{th} layer weight for the m^{th} input and the n^{th} neuron.
W	The total number of neural network weights.
$\mathbf{w}_k$	Weight state vector.
$\mathbf{w}_{n,k}$	Zero-mean, white, Gaussian process noise vector at time step k .
$\mathbf{W}_{N \times M}^1$	Hidden layer weight matrix.
$\mathbf{W}_{O \times N}^1$	Outer layer weight matrix.
(X_f, Y_f)	Global feature coordinates [m].
(X, Y, Z)	Global axes Cartesian coordinates [m].
$\mathbf{z}_k$	Measurement vector at time step k .
$\beta_{f,i}$	Bearing to feature i with respect to the rover [deg].
ω	Rover angular velocity [deg/s].
ϕ	Terrain pitch input angle [deg].
θ	Terrain roll input angle [deg].
$\sigma_{\cdot,i}^2$	Variance of the i^{th} EKF variable ($\cdot$).
Φ_k	State transition matrix at time step k .
Φ^1	Hidden layer sigmoid activation function.
(Θ, Φ, Ψ)	Pitch, roll, yaw about the global axes [deg].

Chapter 1

Introduction

The current state of the art in autonomous planetary mobile robotic systems are the Mars Exploration Rovers (MERs), Spirit and Opportunity, shown in Figure 1. Since their arrival on Mars, the twin rovers have been diligently exploring the Red Planet as surrogate geologists commanded by a team of human scientists on Earth. Originally sent four years ago with a 90-day mission objective and a goal to traverse a minimum of 1 km, both rovers continue to operate today with a combined total distance of 16 km traversed, 35% of which was accomplished autonomously [3]. While there are several factors that have contributed to the MERs exceptional success, the integration of autonomous navigation systems in the rover design have provided a crucial factor to overall rover survivability in the dynamic and harsh extraterrestrial Martian environment.

Over the last four years, the rovers have traversed across sandy and hard-packed rocky landscapes littered with rock debris and sand dunes, and sometimes on slopes with inclinations as high as 31° [4]. These types of terrain make rover navigation over Martian soil no trivial task and pose a constant risk to the rover should it become stuck, bogged down, or tip over. In general, a navigation algorithm provides the position and orientation estimates (robot pose) that allow the rover to localize itself within a map of the environment it has a priori (robot localization) or that it is

Figure 1: Mars Exploration Rover. Online source: NASA.

building concurrently with the pose estimates (simultaneous localization and mapping - SLAM). To perform the pose estimate, the algorithm typically relies on an idealized mathematical model of the mobile robot that leaves certain dynamics unmodelled. These unmodelled dynamics may arise from assumptions made about the system behaviour, linearization, vehicle wear and damage, or environmental disturbances such as wheel slip. The resulting navigation algorithm may be able to cope in this scenario if it is robust, but more likely divergence of the estimate will result, especially in situations of wheel slip where the disturbance experienced is large or occurs over an extended period of time [5].

The research presented in this thesis explores the development of a rover navigational technique capable of autonomously adapting the rover dynamics to compensate for unmodelled nonlinear slip at the wheel-terrain interface. Autonomy is achieved in the sense that the rover can correct its position estimates independently from human operators on Earth saving valuable time that can be used conducting scientific exploration. In the following sections a closer examination of the problem of planetary

rover navigation in the presence of slip is presented. Specifically, the experience of the MERs is used as a primary example that illustrates the perils faced in navigating Martian terrain and the solutions that were implemented using on-board autonomous slip estimation.

1.1 Autonomous Navigation of Mars

In January 2004, NASA's Mars Exploration Rovers (MERs) began their primary 90-day mission to search for evidence of water on Mars by means of geological exploration of two distinct sites. Spirit began its journey at Gusev Crater and Opportunity at Meridiani Planum located on the opposite side of the planet. Now, almost five years after their original expiration date, and with over 16 km of traverse completed, the success of the MERs can be attributed to their ability to traverse Martian terrain in the presence of high levels of slip. The details of the MERs navigation system discussed below are presented in detail in [6, 7].

On average, commands from human operators on Earth are uploaded to the MERs once per Martian day (called a sol). These instructions can include navigational commands such as directions to the next area of interest. The rovers have three primary navigation modes: low-level commands specifying specific actuator positions (wheel turn, steering angle), directed driving primitives (for example, turn in place) for circular arc trajectories, and autonomous driving where the rover can determine the optimal driving primitives to execute in order to avoid obstacles and steer towards its goal location.

The autonomous driving mode was developed in order to satisfy the requirement that the MERs be capable to traversing 100 m per sol of flat terrain, sparsely distributed with rocky obstacles, with no human assistance once the drive commenced. Also, because the maximum distance traversed each sol would be limited by several

factors including position error estimation due to wheel slip, obstacle-occlusion of the full range of vision sensors, and limited communication opportunities with Earth, the decision to integrate autonomous driving capability was motivated.

Using stereovision sensors (Spirit: 125° Field of View (FOV) front and rear HAZ-CAMs, Opportunity: 45° FOV NAVCAMs), the MER navigation system is comprised of several autonomous driving modes that can be used separately or in combination. These modes are enumerated below.

1. *Terrain Assessment* using Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) autonomously detects geometric hazards such as rocks, cliffs, or steep slopes. A traversability map is generated and candidate paths projected and weighted according to traverse difficulty. No estimate of slip is performed in this mode.
2. *Local Path Selection* in which autonomous control is given to the rover to choose the drive direction based upon multiple path directions to a goal point. In the simplest case, this mode relies on the position estimate from wheel encoders and inertial measurement unit (IMU) (i.e. no vision processing). This mode cannot measure longitudinal slip but can compensate for yaw errors resulting from lateral slip down sandy slopes. This mode is often combined with *Terrain Assessment* to provide hazard detection and enables autonomous steering around obstacles.
3. *Visual Odometry* is the only means by which the MERs can detect longitudinal slip. This mode uses a stereo image pair taken over a small motion step, to update the rover position estimate by comparing feature locations in the image sequence. Traversing steep slopes, mixed sand and rocky terrain, and sandy dunes on flat plains often results in high slip that, without the ability to measure it, would severely compromise the ability of the MERs to navigate as the

position estimates, based solely on odometry and IMU sensors, would drift away from the true values. Combining this mode with *Local Path Selection* mode allowed the rovers to navigate the sandy slopes of Mars and provided operators on Earth a strategy for determining “keep-out zones” of high slip.

The integration of these autonomous drive modes on the MERs have cut the time required to reach areas of scientific interest. Consequently, more time becomes available for actual scientific exploration. Spirit’s trek to the Columbia Hills, for instance, was cut in half (6 months to cover 2 km) by using *Terrain Assessment* over one third of the distance (the other two thirds being human commanded *Directed Drives*). On sol 125, Spirit made it’s longest single-day traverse of 124 m of which 62 m was autonomously driven using *Terrain Assessment* and *Local Path Selection*. Opportunity drove 141 meters on sol 82 [8]. On sols 383 to 385, Opportunity drove an initial 106 m using *Local Path Selection*, then another 284 m using *Local Path Selection* and *Terrain Assessment*.

Autonomous navigation on the MERs has enabled them to average approximately a 1000 m every four months. As of September 2006, Spirit (sol 960) had traversed a total of 6876 m, of which 3126 m was performed autonomously, and Opportunity (sol 940) had traversed 9130 m of which 2465 m was performed autonomously [3]. Compare these traverse distances to the cumulative distance achieved by the 1997 Mars Pathfinder rover, Sojourner, over its entire 90-day mission: just over 100 m [8].

1.1.1 Slip Estimation using Visual Odometry

Concerning the accuracy of the position estimate of the MERs, the design requirement called for no more than a 10% drift over 100 m of drive. On flat, benign terrain, the position estimates, based on a combination of wheel encoder (wheel rotations)

and IMU measurements (rover attitude - pitch, roll, yaw), easily satisfied that requirement. However, on steep slopes and sandy terrain Visual Odometry emerged as a vital tool to measure the extreme slip that was encountered when wheels lost traction [4].

Visual Odometry provides a 6-DOF (degree-of-freedom) rover pose position update in the (X, Y, Z) Cartesian global coordinates and about these axes, (Θ, Φ, Ψ) , or roll, pitch, and yaw respectively. The updates are determined by tracking the motion of autonomously selected terrain features called *Harris corners*¹ between two stereo image pairs collected over a short motion traverse (0.5 m). This autonomous drive mode evolved into a critical safety check used to terminate a planned drive early if insufficient progress toward a goal was made after 5 to 10 m of traverse [7]. Several examples of where Visual Odometry essentially rescued Spirit and Opportunity are detailed in [4]. Two of these scenarios are described next.

Opportunity and the Purgatory Ripple

On sol 446, after a blind drive of 50 m over an pile of sand, it was found that not only had Opportunity failed to climb over the pile, it had become bogged down. A strategy was devised to free the rover and prevent a similar situation from occurring again. The solution required Opportunity to execute two meters of driving with the wheels steered to perform 23° of heading change. After this, Visual Odometry was used to estimate the actual motion of the rover. This process was repeated from sol 463 to sol 483 over which progress of 1 mm of actual motion was achieved for every 2 m of commanded driving (measured slip was approximately 99%). Finally, after 39 days (sol 484), Opportunity managed to free itself from what came to be known as the *Purgatory Ripple*.

¹A Harris corner is a point where local image intensity changes in two independent directions indicating the presence of two intersecting edges [9].

Subsequently, rover driving over sandy terrain was limited to 5 m after which Visual Odometry would be used to perform a “Slip Check” to assess progress. If progress was negligible, the MER could be easily backed out since the wheels would not be buried and an alternate route could be determined. Slip Check prevented Opportunity from digging in on sols 501 and 603, allowing exploration to continue after only a single day of delay [4].

Opportunity has required the use of Visual Odometry more often than Spirit due to the nature of the slippery terrain it has crossed. On sol 555, Visual Odometry was used on 682 m of the 5947 m traversed by Opportunity [6].

Spirit and the Wheel Malfunction

On sol 125, Spirit’s front right wheel was found to be drawing more current than the others and was subsequently deactivated to conserve power. In this driving mode, the wheel was dragged along as the others were driven which resulted in unbounded navigation position estimate errors even on flat terrain. Visual Odometry was used to correct for the position error due to the inoperative wheel allowing Spirit’s mission to continue on Mars [4].

1.1.2 Future Missions to Mars

Next generation planetary rovers sent to Mars, such as NASA’s Mars Science Lab (2009) or ESA’s ExoMars (2013), must be able to autonomously navigate slippery Martian terrain, such as that shown in Figure 2. As evidenced by the MERs, autonomy can increase the amount of scientific exploration and improve chances of overall mission success and rover survivability. However, whereas the MERs average a distance of 1 km every four months [3], the mission requirements for the ExoMars rover, for instance, require the navigation system to autonomously traverse 1 km every 10

Figure 2: Spirit's view of the Martian terrain. Online source: NASA.

sols [10]. In order to meet these demanding design requirements, research into methods of increasing rover navigational autonomy and the ability to deal with high levels of slip is already underway [11–15].

1.2 Problem Statement

From the preceding discussion it is evident that a rover placed on Mars to conduct exploration missions must include some level of autonomy in terms of navigation due to the limited communication opportunities with Earth that prevent human controlled driving from achieving fast real-time responses to new navigational issues as they arise. Thus, an autonomous navigation system must be able to cope with

- Geometric obstacles including rocks, cliffs, and hills;
- Non-geometric obstacles such as slip-inducing sandy slopes and dunes.

While this thesis does not address the first item listed, the slip-detecting navigation algorithm developed herein will attempt to directly tackle the last point.

A rover placed on Mars and required to travel farther distances autonomously must be able to automatically detect slip so that the correct position updates can be determined for accurate navigation. As well, in terms of rover survivability, real-time slip estimation must be available so that corrective manoeuvres can be executed in

cases where the level of slip might cause the vehicle to dig itself into the terrain. Real-time slip estimation would act as a *Slip Check* technique similar to the policy adopted by the MER operators, except that this algorithm would always be active and completely automatic.

Provided below is an overview of the rover, terrain, and sensor models that will be used in this thesis. As well, the underlying algorithm used to construct the slip-adaptive navigation algorithm is introduced. Finally, the research scope of this thesis is outlined.

1.2.1 Simulation Environment

A simulated rover, sensor, and terrain model are overviewed here and developed in detail in Chapter 2. All simulations are performed in MATLAB. The kinematics of the rover are simplified to a unicycle representation, where the rover states required to navigate are the X and Y coordinate positions and orientation Ψ of the rover in the environment. The corresponding state space is $\mathbf{q} = [X, Y, \Psi]^T$. Since rovers in general have the ability to stop and turn in place, this simplification is not far from actual rover kinematic behaviour.

Terrain slip is modelled by creating regions within the environment associated with a particular nonlinear slip model. As discussed in Chapter 2, rover slip is a function of various parameters including the terrain type (loose rock, hard-pack rock, sand, and combinations of the same), slope of the terrain in pitch and roll directions, and also the vehicle mass, velocity, and wheel design. After several simplifying assumptions are made (see Chapter 2), slip remains a function of only the slope of the terrain being traversed.

Definition of 2.25D

The environment model is a 2D plane randomly populated with landmarks or more generally, “features”. These are represented as small points fixed to certain locations in the map and have their own corresponding Cartesian coordinates, (X_f, Y_f) . A slight modification of the well known 2.5D surface, a simplified 3D surface where each point in the $\{X, Y\}$ plane has a single Z -elevation value, is adopted in this thesis [16]. While elevation is not represented in order to simplify the terrain model, each map point will be associated with a specific amount of slope in the roll and pitch directions resulting in a what is here called a 2.25D representation of the environment. Since elevation is not modelled, 2.5D does not apply, but since the rover can pitch and roll in the 2D plane, a 2.25D moniker seems appropriate. Thus, the .25 does not represent a numerical quantity, only the fact that the rover is allowed to rotate about the axes in the 2D plane.

In order to navigate, and ultimately detect slip, the rover must have some means by which to sense the features present in the environment model. This work does not adopt any one sensor configuration specifically, but assumes that the rover is equipped with a sensor capable of detecting the range and bearing of these features relative to itself.

1.2.2 Slip-Adaptive EKF-MLFN Algorithm

The well-known extended Kalman filter (EKF), widely used in navigation, guidance, and control applications, will be adopted for rover navigation in this thesis. As will be discussed in more detail in Chapter 2, the EKF is a recursive state estimation algorithm that can be used to construct the rover position and orientation (called rover localization) in a map of the environment. The EKF requires an exact model of the rover kinematics (or dynamics) that map control inputs into rover motions across

the terrain. However, if certain dynamics are hard to model or remain unmodelled altogether, the EKF estimates will not be optimal at best and diverge completely at worst. One such hard-to-model quantity is the slip that can occur at the wheel-terrain interface.

In order to provide autonomous slip-adaptive capability to the EKF navigation algorithm, this thesis will focus on fusing together the EKF with the “learning” capability of a multi-layer feedforward neural network (MLFN).

The EKF provides a well-understood technique for state estimation in a probabilistic sense. Also a variety of mathematical tools exist for the filters analysis such as assessing system performance and accuracy, as well as determining state observability or detectability in the case the system is unobservable.

While several techniques could be used to allow for real-time modelling of terrain slip, the approach investigated here is focused on the learning capabilities of neural networks. A MLFN is relatively simple to implement and possesses nonlinear function modelling capability. As traditional learning algorithms, such as error back-propagation, are known for their slow convergence characteristics, this thesis instead uses an EKF to provide neural network training.

Since an EKF is used to navigate the rover and train the MLFN, this thesis uses a hybrid algorithm developed first by Lobbia, Stubberud, and Owen in [17] in which the same EKF is used to estimate the plant states and the so-called neural “weights” that enable the neural network’s function approximation capability. As the MLFN augments the plant model by capturing unmodelled or missing dynamics (see Chapter 3), the EKF plant state and neural network weight estimates are improved resulting in a synergistic effect. Thus, while this general hybrid approach is not introduced here, its specific application to the problem of slip-adaptive navigation is the primary contribution of this thesis.

1.2.3 Research Scope

The development of a slip-adaptive rover navigation algorithm requires the employment of techniques from several areas including kinematic modelling, Kalman filtering, neural network implementation, system identification, and terrain slip mechanics and modelling. As such, given the problem formulation above, the scope of this dissertation encompasses the following research objectives:

1. A literature review covering planetary rover navigation, slip modelling and estimation, EKF training of neural networks, and applications of hybrid EKF-neural network techniques;
2. A review of necessary background theory pertaining to navigation using the EKF and multi-layer feedforward neural network design and implementation;
3. Investigation of the performance of the EKF training algorithm for MLFNs through numerical analysis of system observability, state convergence and function approximation capability;
4. Development of the simulation environment in MATLAB with models representing the rover, sensors, the terrain environment with its corresponding landmark features, and a nonlinear slip model based on actual experimental data;
5. Formulation of the slip-adaptive navigation algorithm using a hybrid EKF-MLFN approach;
6. Numerical study of the slip-adaptive algorithm's observability, state convergence, and MLFN slip-approximation capability;
7. Validation via simulations of the algorithm in the synthetic environment.

The systems investigated in this thesis are linearized and time-varying. Unlike observable linear time-invariant systems where control over the placement of the system

eigenvalues guarantees stability, even observable time-varying systems may not exhibit stable behaviour. As such, a detailed analysis of the stability of the algorithms developed herein is beyond the scope of this thesis. Instead we focus on proving these algorithms through a numerical study of their stability by means of computer simulation.

1.3 Thesis Organization

This chapter introduced the motivation behind the research presented in this thesis. The problem of navigating rovers on other worlds in regions of high slip presents an important area of research if overall rover mission length and survivability are to be improved and enhanced for future missions requiring increased onboard autonomy. As such, this thesis presents the development of a slip-adaptive navigation technique to autonomously model the slip environment in real time as the rover navigates. To this end, the following chapters cover each of the corresponding elements that are ultimately used to construct the slip-adaptive navigation algorithm.

In Chapter 2, rover navigation in the presence of unmodelled slip is investigated. The chapter begins by presenting the general EKF algorithm. Then the kinematic rover, sensor and environmental models are derived and a navigation algorithm is formulated where slip is not accounted for in the model. The performance of the algorithm is investigated when used to navigate a rover in the presence of slip. This section concludes by reviewing the current literature on slip estimation and presents the environmental slip model that will be used in this thesis.

Chapter 3 introduces the multi-layer feedforward neural network (hereafter exclusively referred to as MLFN). Its attributes as a universal function approximator are discussed and an overview of the error back-propagation training algorithm is provided. A literature review highlighting current areas of research in combining EKFs

and neural networks is presented and includes papers on EKF-based MLFN training, using neural networks as plant models, and solving navigation problems by applying a hybrid approach. The EKF-based neural network training algorithm is developed and investigated in detail with examples. Finally, the hybrid EKF-MLFN approach that is used to develop the slip-adaptive navigation algorithm is presented.

The main contribution of this thesis is presented in Chapter 4 where the slip-adaptive algorithm for rover navigation is developed using the hybrid EKF-MLFN. The models developed in Chapter 2 are reformulated to include the MLFN-based slip model and are then incorporated into the hybrid EKF-MLFN algorithm presented in Chapter 3. Using the simulated MATLAB environment, the slip-adaptive algorithm is validated by demonstrating how the MLFN captures the environmental nonlinear slip-model online as the rover navigates various terrain profiles and slip conditions. System observability is determined at each step while path dependence is observed through numerical investigation.

Finally, conclusions and areas for continued research are discussed in Chapter 5.

Chapter 2

Rover Navigation

Rover *navigation* as defined in this thesis is the problem of determining the position and orientation of the vehicle in its operational environment. A general diagram of a guidance, navigation, and control (GNC) system is shown in Figure 3. The problems of rover trajectory *planning*, *guidance* and *control*, which involves planning the optimal trajectory or path to take and the corresponding control actions, are beyond the scope of this thesis. However, accurate navigation provides the current state of the rover and thus is a necessary capability in order to implement high-level planning tasks, guidance algorithms, and control strategies successfully [1].

Navigation can be divided into two subproblems:

- Rover localization, which determines vehicle position and orientation given an environment map;
- Mapping, which is the process of constructing an environment map given a rover location.

A rover placed into an environment without knowledge of its current location or a map will have to solve both localization and mapping problems concurrently in what is known as Simultaneous Localization and Mapping (SLAM). While formulating a SLAM algorithm could provide a challenging scenario in which to validate a

Figure 3: A GNC system with the navigation component high-lighted. Adapted from [1].

slip-adaptive navigation technique, the additional requirement of the rover to map its environment as well as localize offers no real benefit to the research focus here. Of course, a more accurate position and orientation estimate will also impact the accuracy of SLAM-constructed maps, however, this research will focus on the primary problem of autonomous slip estimation and therefore only the localization component of navigation, using an a priori known map, need be analyzed. A detailed discussion on solving the SLAM problem can be found in [18] and [19].

In this thesis, rover localization will be determined using the well-known extended Kalman filter (EKF) which has been used extensively for nonlinear navigation applications such as the Global Positioning System (GPS) and for onboard aircraft/spacecraft Inertial Navigation Systems (INS). Developed from the original linear Kalman filter formulation in 1961 for the navigation of NASA’s Apollo spacecraft, and later in 1966 for the navigation system onboard the Lockheed C-5A aircraft, the EKF has continued to find extensive application to navigational problems in the aerospace industry [20].

2.1 The Extended Kalman Filter

The EKF is a recursive probabilistic state estimator for nonlinear dynamical systems. The algorithm, common in the literature [1, 18, 19, 21, 22], is an attractive one for navigation applications as it provides the ability for the estimation of unmeasurable states, such as vehicle position, from a dynamic system model and sensor measurements all in the presence of noise. The EKF algorithm assumes the noise is white, zero mean, and Gaussian; an assumption which is used to model the noise in the simulations performed in this thesis but which is never satisfied in actual implementations. In these cases, the non-Gaussian noise probability density function can be approximated arbitrarily closely by the weighted sum of several Gaussian probability density functions [23, 24]. Thus, the EKF acts as an estimator for the system states and as a filter on the noisy measurements. In applying the EKF to a system, the state variables are also assumed to have Gaussian distributions, denoted $\mathcal{N}(\bar{\mathbf{q}}, \mathbf{P})$, with multivariate mean $\bar{\mathbf{q}}$ and covariance matrix $\mathbf{P}$.

Consider a discrete-time nonlinear system

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}, \mathbf{u}_{c,k}) + \mathbf{w}_{n,k}, \quad (1)$$

$$\mathbf{z}_k = h(\mathbf{q}_k) + \mathbf{v}_{n,k}, \quad (2)$$

where

$\mathbf{q}_k$ = the state vector at time step k of dimension n_q ;

$\mathbf{u}_{c,k}$ = control input vector at time step k of dimension m_q ;

$f(\cdot)$ = nonlinear rover kinematic model;

$\mathbf{w}_{n,k}$ = zero-mean, white, and Gaussian process noise vector of dimension n_q ;

$\mathbf{z}_k$ = sensor measurement vector of dimension r_q ;

$\mathbf{v}_{n,k}$ = zero-mean, white, and Gaussian sensor noise vector of dimension r_q .

As time, t , evolves from time step $k - 1$ to k , where $t = kT$ (T is the length of each time step in seconds) and thus

$$\mathbf{q}_k \equiv \mathbf{q}(kT), \quad (3)$$

the EKF propagates $\mathcal{N}(\bar{\mathbf{q}}, \mathbf{P})$ via the nonlinear dynamic model, $f(\mathbf{q}_{k-1}, \mathbf{u}_{c,k})$, with mean $\mathbf{q}_k$ and variance of the i^{th} state $\sigma_{q,i}^2$. The initial state estimate error covariance matrix is then constructed as follows,

$$\mathbf{P}_0 = \begin{bmatrix} \sigma_{q,1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{q,2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \sigma_{q,n_q}^2 \end{bmatrix}, \quad (4)$$

where the initial variance of each state (determined from statistical knowledge of the initial states) is initialized at $\sigma_{q,i}^2 = \sigma_{0,i}^2$. Even if the initial state variance is unknown, a large initial guess (indicating a high level of uncertainty) can suffice in most cases [22]. Squaring these terms and multiplying by three represent the 3σ error bounds on each state estimate.

Control inputs are $\mathbf{u}_{c,k}$, and output is $\mathbf{z}_k$ with nonlinear measurement model $h(\mathbf{q}_k)$. Process and measurement noise, $\mathbf{w}_{n,k}$ and $\mathbf{v}_{n,k}$, are white and distributed according to $\mathcal{N}(0, \mathbf{Q})$ and $\mathcal{N}(0, \mathbf{R})$, where $\mathbf{Q}$ and $\mathbf{R}$ represent the time-invariant and independent process and measurement noise covariance matrices, defined by:

$$\mathbf{Q} = \begin{bmatrix} \sigma_{w_n,1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{w_n,2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_{w_n,n_q}^2 \end{bmatrix}, \quad (5)$$

$$\mathbf{R} = \begin{bmatrix} \sigma_{v_n,1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{v_n,2}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_{v_n,r_q}^2 \end{bmatrix}, \quad (6)$$

where $\sigma_{w_n,i}^2$ is the process noise variance on the i^{th} state¹ while $\sigma_{v_n,i}^2$ is the sensor noise variance on the i^{th} measurement. The values for these statistics are a function of the associated sensor (laser range finder, IMU, encoders, etc.) which have known statistics determined by the manufacturer. Alternatively, the statistics can be derived and modelled as a Gaussian distribution with standard deviation σ if the sensor error is known. For example, if the error on a laser range finder is $+/-0.5m$, then assuming a standard deviation of $3\sigma = 0.5$, the initial variance would be $\sigma^2 = (0.5/3)^2$.

2.1.1 EKF Algorithm

The EKF algorithm consists of two steps, shown below, where the “-” superscript indicates pre-update value and the “+” indicates post-update value:

¹Later in this thesis the process noise is placed directly onto the control inputs.

Prediction Step

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial q} \right|_{q=\hat{q}_{k-1}^+}, \quad (7)$$

$$\hat{\mathbf{q}}_k^- = f(\hat{\mathbf{q}}_{k-1}^+, \mathbf{u}_{c,k}), \quad (8)$$

$$\mathbf{P}_k^- = \mathbf{F}_k \mathbf{P}_{k-1}^+ \mathbf{F}_k^T + \mathbf{Q}. \quad (9)$$

Measurement Update Step

$$\mathbf{H}_k = \left. \frac{\partial h}{\partial q} \right|_{q=\hat{q}_k^-}, \quad (10)$$

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T [\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}]^{-1}. \quad (11)$$

$$\hat{\mathbf{q}}_k^+ = \hat{\mathbf{q}}_k^- + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\mathbf{q}}_k^-)), \quad (12)$$

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K} \mathbf{H}_k \mathbf{P}_k^-. \quad (13)$$

In the prediction and update steps above, equations (7) and (10) represent the linearization of the nonlinear system - equations (1) and (2) respectively - about the current state estimate. The linearization is necessary since the EKF uses the linear Kalman filter equations for the propagation of the error covariance determined from equations (9) and (13), and the calculation of the Kalman gain matrix in equation (11). It is these linearization steps that put the *extended* in the name “extended Kalman filter” thus enabling the application of the linear Kalman filter technique to nonlinear systems.

In the prediction step, the nonlinear system model is used to predict the state $\hat{\mathbf{q}}_k^-$ at the next time step k based on the previous state estimate at time step $k - 1$ and the current control input $\mathbf{u}_{c,k}$; see equation (8). This step is sometimes called dead-reckoning and is susceptible to error accumulation in the state estimate due to noise in the control input sensors as well as other sources. To prevent the state

estimate errors from growing without bound, a periodic correction is provided in the measurement step of the algorithm (hence the use of the superscript “-” in the first step and “+” in the second step). Using feedback from the environment obtained from sensor measurements, an update is applied to the original state estimate. A so-called *innovation* error is computed from the difference between the sensor measurements, $\mathbf{z}_k$, and the measurement model, $h(\mathbf{q}_k)$. The measurement model is a function of the state prediction from the first step, and thus the measurements are correlated with the states of the system. The Kalman gain matrix then maps the innovation errors into a corrective state update as shown in equation (12).

2.1.2 Observability and Detectability

Since, in general, the states cannot be directly measured by the sensors (e.g. the rover cannot measure its position (X, Y) directly in the environment map using its own sensors), the EKF must construct the states from the sensor measurements. To do this, these constructed states must be observable. If the initial state, $\mathbf{q}_0$, can be reconstructed from knowledge of the output, $\mathbf{y}_k$ from time step zero to k , then the system is observable [21, 25]. Assessing system observability allows the analyst to determine if the states can in fact be constructed from the measurements. For linear time-invariant (LTI) systems, a system is fully observable when the observability matrix, $\mathbf{O}_{\text{obs}}$ has rank equal to n_q (i.e. full rank) [21]. The time-invariant observability matrix is defined by

$$\mathbf{O}_{\text{obs}} = \begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix}. \quad (14)$$

Similarly, for linear time-varying (LTV) systems, the observability Grammian matrix must be full rank at each time step [21]:

$$\mathbf{O}_{\text{obs},k} = \sum_{j=0}^k \Phi_j^T \mathbf{H}_j^T \mathbf{H}_j \Phi_j, \quad (15)$$

where $\Phi_j = \Pi_{i=0}^j \mathbf{F}_i$.

Since this thesis considers the EKF for solving the navigation problem of rover localization where the rover system is nonlinear, the observability of the EKF can be determined by using equations (14) or (15) developed for linear systems (since we linearize the system). A note of caution is provided here when determining the rank of these matrices using algorithms that perform singular value decomposition (SVD). While this algorithm is the most reliable, care must be taken that it does not discard non-zero singular values resulting in a loss of rank of the observability Grammian. Appendix E presents a computational example of how to successfully use MATLAB's rank function.

If the system is LTI but not completely observable, then it can be separated into the observable and unobservable parts (see the Appendix B for an example). In this scenario, only the observable part can be estimated leaving the unobservable part at the mercy of system "detectability". Detectability is the assessment of the stability of the unobservable part of the system, which, for discrete time systems, depends on the location of the eigenvalues in relation to the unit circle. The stable eigenvalues lie in

the unit circle (state estimate convergence), marginally stable eigenvalues are on the unit circle (state estimate oscillatory motion), and unstable eigenvalues lie outside the unit circle (state estimate divergence). Therefore, the system is detectable if the eigenvalues of the unobservable part lie inside the unit circle.

The consequence of a system with unobservable parts is that, while the EKF Kalman gains can drive the convergence of the observable part, the states associated with the unobservable part will converge or not depending on the stability of the unobservable part of the system. It is important to note that each state may contain both observable and unobservable components.

Appendix B contains a complete discussion and derivation of the LTI observability matrix and an example that illustrates the process of separating a system into its observable and unobservable parts.

However, for LTV systems, even if the system is observable, state estimate convergence is not guaranteed as the placement of the EKF eigenvalues no longer corresponds to the stability of the solution [26]. Thus, in this thesis, since the proof of stability is difficult to show analytically, a numerical study is presented through the presentation of the simulation results generated over the course of this research.

It is worthwhile to note the distinction between the concept of model identifiability and system observability. As already defined, observability is the problem of reconstructing the states of the system from measurements. Model identifiability is the problem of uniquely estimating the parameters of a model given a set of measurements. In this thesis, no distinction between a state and a parameter is made, as states which are modelled as constants can in effect be called the “parameters” of the model. In this context, observability of the EKF is a necessary, but not sufficient condition (in the LTV case) to establish model identifiability when we use the EKF later in this thesis to estimate the constant weights of a neural network which provides an adaptive model of rover slip. State observability and parameter identifiability is

treated by Rusnak et al. in [27].

2.2 Rover Localization Algorithm

The EKF algorithm can be applied to rover navigation by treating the localization problem as one of estimation. The states to be estimated are the position and orientation of the rover in the environment map. A simple rover kinematic model serves as the EKF dynamic model in the predictive step by providing the initial estimate of the rover states. In the update step, sensor measurements are used to detect map features (landmarks) in the environment with respect to the rover. These help localize the rover since the position of the features with respect to the rover are related to the absolute position of the rover in the environment.

In the following sections, models for the terrain environment, rover kinematics, and rover sensors are developed. Subsequently, two simulations are presented to demonstrate the cases where (1) the EKF's component models represent the the environment and rover accurately, and (2) where these models do not accurately capture the true system dynamics. Of course, in this latter scenario, slip will be introduced at the wheel-terrain interface, which has not been modelled in the EKF.

2.2.1 Environment Model

This thesis uses a simple 2.25D (see Section 1.2.1) environment randomly populated with “features” in all simulations. On Mars, these features might correspond to random rock debris distributed throughout the environment and detected by external sensors such as cameras or laser range finders. Feature detection may be provided by algorithms which can identify Harris corners [9] or scale-invariant features via Scale Invariant Feature Transforms (SIFT features) [28]. In the simulated environment, detected features are represented as small fixed points in the 2.25D map and have

Figure 4: The rover simulation environment.

their own corresponding Cartesian coordinates, (X_f, Y_f) . In this thesis, the map is supplied to the rover a priori and thus only the localization problem is solved and not the SLAM problem. Also it is assumed that each feature is uniquely identifiable and thus the so-called correspondence problem — that of determining if a detected feature is new or has been seen before — is solved a priori as well. This is a significant assumption since it assumes that each new measurement is associated with the correct known set of mapped features, where in practice, the correspondence between these subsequent measurements and known features is not always apparent. While certain non-robust methods have been developed in order to overcome this problem, such as determination of the Mahalanobis Distance [18, 29], the development of a robust method is beyond the scope of this thesis.

The simulation environment is shown in Figure 4. While this environment is

Figure 5: Time history of terrain inputs about the rover pitch and roll axes.

essentially a fictitious 2D landscape, the rover is permitted to pitch by ϕ degrees and roll by θ degrees about its center of gravity (CG). Thus the terrain provides these exogenous inputs, $\mathbf{u}_{e,k} = [\phi \ \theta]$, to the rover kinematics (as opposed to the control inputs $\mathbf{u}_{c,k} = [v \ \omega]^T$). As discussed in Chapter 1, we refer to this environment as *2.25D* since the rover can pitch and roll in the 2D plane. An example of the type of terrain inputs the rover would experience while traversing a series of rolling hills is depicted in Figure 5.

2.2.2 Rover Kinematic Model

The rover kinematics are modelled on a unicycle, where the rover states required to navigate the 2D environment are $\mathbf{q}_k = [X_k, Y_k, \Psi_k]^T$ in the global frame. Since rovers in general have the ability to stop and turn in place, this simplification is not far from

actual rover kinematic behaviour. The first two states are the Cartesian positions with respect to some global coordinate system with the origin possibly located at the landing site. The last state represents the rover yaw angle measured about the z -axis.

Since the environment models the effects of roll and pitch on the rover, this complicates the derivation of the rover kinematic model. Control inputs provided in the local rover local frame must be transformed to motion in the global frame. A rotation matrix is thus defined that describes the orientation of the rover-fixed reference frame, $\{x, y, z\}$, with respect to a global coordinate frame, $\{X, Y, Z\}$. Since the terrain model described in section 2.2.1 permits the rover to roll (θ), pitch (ϕ), and yaw (Ψ) in the 2.25D plane, then the rotation matrix can be defined as (setting $\cos(\cdot) = c(\cdot)$ and $\sin(\cdot) = s(\cdot)$):

$$\mathbf{R}_{XYZ}^{xyz} = \mathbf{R}_Z \mathbf{R}_Y \mathbf{R}_X \quad (16)$$

$$= \begin{bmatrix} c(\Psi) & -s(\Psi) & 0 \\ s(\Psi) & c(\Psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c(\phi) & 0 & s(\phi) \\ 0 & 1 & 0 \\ -s(\phi) & 0 & c(\phi) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\theta) & -s(\theta) \\ 0 & s(\theta) & c(\theta) \end{bmatrix}, \quad (17)$$

which yields

$$\mathbf{R}_{XYZ}^{xyz} = \begin{bmatrix} c(\Psi)c(\phi) & c(\Psi)s(\phi)s(\theta) - s(\Psi)c(\theta) & c(\Psi)s(\phi)c(\theta) + s(\Psi)s(\theta) \\ s(\Psi)c(\phi) & s(\Psi)s(\phi)s(\theta) + c(\Psi)c(\theta) & s(\Psi)s(\phi)c(\theta) - c(\Psi)s(\theta) \\ -s(\phi) & c(\phi)s(\theta) & c(\phi)c(\theta) \end{bmatrix}. \quad (18)$$

The linear control input provided to the rover in the *rover reference frame* is the

forward velocity which acts only in the rover x -axis, $v = \dot{x}_r$. Thus, $\mathbf{v}_{xyz} = [v \ 0 \ 0]^T$ and the control input in the *global frame* is

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} v \cos(\Psi) \cos(\phi) \\ v \sin(\Psi) \cos(\phi) \\ -v \sin(\phi) \end{bmatrix}. \quad (19)$$

Since the rover manoeuvres in a 2.25D plane, the last element in the rover velocity vector is constrained by the terrain, i.e. $\dot{Z} = 0$.

For simplicity, the transformation of rover angular manoeuvres from the local rover frame to the global frame is approximated by applying the same rotation matrix to the rover angular control input, $\dot{\psi} = \omega$ where $\omega_{xyz} = [0 \ 0 \ \omega]^T$, despite the fact that is only valid for small rotational manoeuvres. The approximate rotation of the rover about the global Z -axis is given by $\dot{\Psi} = \omega c(\phi) c(\theta)$, where ω is the angular velocity control input in the rover frame. Rotation about the global X and Y axes is neglected.

The rover continuous time model can be written as

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} v \cos(\Psi) \cos(\phi) \\ v \sin(\Psi) \cos(\phi) \\ \omega c(\phi) c(\theta) \end{bmatrix}. \quad (20)$$

Although the rover moves in continuous time, the nonlinear rover kinematics must

eventually be implemented on computer hardware and thus approximated as a discrete time model. In discretized form, the nonlinear rover kinematics represented in equation (20) can be expressed as follows:

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}, \mathbf{u}_{c,k}) + \mathbf{q}_{k-1}, \quad (21)$$

where

$\mathbf{q}_k$ = the rover states at time step k ;

$\mathbf{u}_{c,k}$ = control inputs, $[v \ \omega]^T$, forward and angular velocity, respectively;

$f(\cdot)$ = nonlinear rover kinematic model.

In expanded form

$$\begin{bmatrix} X_k \\ Y_k \\ \Psi_k \end{bmatrix} = \begin{bmatrix} T \cos(\psi_{k-1}) \cos(\phi_k) & 0 \\ T \sin(\psi_{k-1}) \cos(\phi_k) & 0 \\ 0 & T \cos(\theta_k) \cos(\phi_k) \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \Psi_{k-1} \end{bmatrix}, \quad (22)$$

where

T = the sample time or length of the time step, k , in seconds.

Equation (22) provides the rover states at each time step given a set of control inputs. From the authors experience in data acquisition and simulation, T can range from as high as 0.01 s/sample (100 Hz) down to 0.1 s/sample (10 Hz). In the simulations performed in this thesis, the latter was selected in order to shorten the length of each simulation.

2.2.3 Measurement Model

In order to navigate, the rover must have some means by which to sense any unique features that may be present in the environment. As already explained, the simulated environment is populated with point features that are uniquely identified by their location in a known map. Thus the mapping and correspondence problem are solved in advance since this thesis is concerned only with the ability of the rover to localize in the map using the neural network to estimate the slip being experienced.

This thesis assumes the rover is equipped with a sensor that can measure the range and bearing to all point features within 150 meters of the rover and within a 180 degree field of view. Some examples of sensors which could provide the range and bearing measurements required include stereo vision cameras [30], the sensor of choice on all future rover missions to Mars, and laser range finders.

Stereo cameras (passive vision systems) can easily detect thousands of 3D data points for navigational processing but are subject to solar lighting conditions [6]. They are also slow due to the time it takes to process each image. Visual Odometry, for example, can take up to three minutes to process a single time step and also requires that the difference between two images be small in order for the algorithm to be able to track the features it has detected [4]. This restriction limits the maximum velocity of the MERs to 10 m/h while Visual Odometry is online.

Alternatively, laser range finders (active vision systems) were used on Sojourner, although with limited success [6]. In this scenario, a mathematical model of the expected locations of environment features would be compared to the laser-sensed scan of the environment revealing the actual feature locations. For this scenario to work, the features would have to be uniquely identifiable which presents the challenge to this kind of motion estimation technique. However, while this method has not been perfected, research into its application to terrain modelling is underway at the

Figure 6: Range and bearing to a feature detected in the simulated 2.25D environment.

Canadian Space Agency [31]. Also, Neptec, a NASA-contracted company specializing in 3D laser imaging, has made recent advances in using active vision to model three-dimensional objects. Neptec's Laser Camera System (LCS) is currently used to detect damage on the Space Shuttle while docked with the International Space Station [32–34]. Active vision also renders feature detection independent of solar illumination.

Whether active vision or passive vision systems are used for ranging purposes does not matter, so long as enough information is provided so that the rover can localize itself within the environment.

By using Figure 6, a model of the generic rover measurement sensor which provides the range $(r_{f,i})$ and bearing $(\beta_{f,i})$ to each i^{th} detected feature is presented below in equation (23).

$$h_{i,k} = \begin{bmatrix} r_{f,i} \\ \beta_{f,i} \end{bmatrix} = \begin{bmatrix} \sqrt{(X_{f,i} - X_k)^2 + (Y_{f,i} - Y_k)^2} \\ \arctan2\left(\frac{Y_{f,i} - Y_k}{X_{f,i} - X_k}\right) - \Psi_k \end{bmatrix} \quad (23)$$

where

- $h_{i,k}$ = is the i^{th} feature detected at time step k ;
- $(X_{f,i}, Y_{f,i})$ = are the global coordinates of the i^{th} feature;
- (X_k, Y_k, Ψ_k) = are the rover states at time step k .

In addition to the measurement sensor, it is assumed that a sensor is equipped that can provide direct measurements of the rovers orientation (one of the EKF states), Ψ_k . The measurement model provides the expected values for rover orientation and the range and bearing of each detected feature. By subtraction from the actual measurements, the innovation error is produced. For m features at time step k , the expected values are combined into a single vector in order to determine the innovation error.

$$h_k = [h_{1,k} \quad \dots \quad h_{i,k} \quad \dots \quad h_{m,k} \quad \psi_k]^T. \quad (24)$$

Since this generic sensor model is assumed perfect, this thesis neglects the estimation of sensor biases and offsets that form a typical part of the state space in the EKF algorithm in practical applications.

2.2.4 Summary of Model Assumptions

The models discussed above made several assumptions and simplifications in order to reduce the difficulty of implementing the simulation performed in this thesis. These included:

- Rover state and noise probability density functions are Gaussian. Noise is also zero mean and white.
- A map is provided to the rover a priori. In the real case of exploration to other planets this would not be the case, but this assumption allows the scope of the problem to be reduced to one of pure localization instead of treating the entire SLAM problem.
- The correspondence problem is solved a priori and is assumed to be perfect. Again, this would not be the case in the real world application. The problem of identifying unique features is the subject of ongoing research primarily using stereoscopic sensors.
- The rover behaves like a unicycle. The MERs have six wheels which allow it to move forward, backward, and turn-in-place. Therefore, this approximation is valid.
- The terrain environment is a 2.25D plane (see Section 1.2.1). Mars contains a variety of landscape profiles from flat and barren, to mountainous and rock-strewn each with various distributions of rock and sand. The inclusion of terrain geometry in the environment model, i.e. inclinations in the pitch and roll directions, increase the fidelity of the terrain model (putting the .25 in the 2.25D).

2.2.5 EKF Observability for Rover Localization

As the rover kinematic and measurement models are both nonlinear they must be linearized so that implementation into the EKF algorithm outlined in section 2.1.1 can be accomplished. Once linearized, this system constitutes a linear time-varying system. Using the observability Grammian of equation (15), the observability of this system can be determined at each time step. It is noted here that given the above

modelled system, the EKF is completely observable at every time step, $\text{rank}[\mathbf{O}_{obs,k}] = n_q = 3$, and thus the rover states of position and orientation can be constructed from the measurements.

2.3 Rover Slip Model

Wheel slip is defined as the difference between the commanded wheel velocity, $v = \omega r_w$, where r_w is the wheel radius, and the actual velocity of the wheel center, v_t :

$$S_{wheel} = \omega r_w - v_t, \quad (25)$$

which may be normalized by the commanded velocity to form the dimensionless quantity, $S = \frac{S_{wheel}}{\omega r_w}$, with values in the range $-1 \leq S_{wheel} \leq 1$ [35].

Slip has also been defined in terms of the entire vehicle as the difference between the velocity estimate of each degree of freedom (DOF) and the actual velocity at the vehicles center of mass [14]. In this work, the latter definition is adopted, except instead of dealing with slip *velocity* this thesis works with slip *distance*.

We define here longitudinal, lateral, and yaw slip components in the rover frame: $\mathbf{S}_r = [Sx_r \ Sy_r \ S\psi_r]$ shown in Figure 7. These slip components represent the values that must be estimated by the slip-adaptive model developed in this thesis. These terms must be transformed to the global frame via the rotation matrix developed previously in equation 18 that describes the orientation of the rover-fixed reference frame, $\{x, y, z\}$, with respect to a global coordinate frame, $\{X, Y, Z\}$.

The slip experienced by the rover in the *rover reference frame* in the longitudinal and lateral directions is $\mathbf{S}_{xyz} = [Sx_r \ Sy_r \ 0]^T$ and thus the rover slip in the *global frame* is

Figure 7: Slip experienced in the rover axes is transformed to the global axes: $\mathbf{S}_g = [S_X \ S_Y \ S_\Psi]$

$$\begin{aligned}
 \mathbf{S}_{XYZ} &= \mathbf{R}_{XYZ}^{xyz} \mathbf{S}_{xyz} \\
 \begin{bmatrix} S_X \\ S_Y \\ S_Z \end{bmatrix} &= \begin{bmatrix} Sx_r(c(\Psi)c(\phi)) + Sy_r(c(\Psi)s(\phi)s(\theta) - s(\Psi)c(\theta)) \\ Sx_r(s(\Psi)c(\phi)) + Sy_r(s(\Psi)s(\phi)s(\theta) + c(\Psi)c(\theta)) \\ Sx_r(-s(\phi)) + Sy_r(c(\phi)s(\theta)) \end{bmatrix}. \quad (26)
 \end{aligned}$$

Since the rover manoeuvres in a 2.25D plane, the last element in the vehicle slip vector, S_Z , is neglected. Performing a similar analysis for the slip about the rotation directions, i.e. $\mathbf{S}_{Rot} = \mathbf{R}_{XYZ}^{xyz} \mathbf{S}_{\theta\phi\Psi}$, where $\mathbf{S}_{\theta\phi\Psi} = [0 \ 0 \ S\psi_r]^T$, yields

$$\mathbf{S}_{Rot} = \begin{bmatrix} S_{\Theta} \\ S_{\Phi} \\ S_{\Psi} \end{bmatrix} = \begin{bmatrix} S\psi_r(c(\Psi)s(\phi)c(\theta) + s(\Psi)s(\theta)) \\ S\psi_r(s(\Psi)s(\phi)c(\theta) - c(\Psi)s(\theta)) \\ S\psi_rc(\phi)c(\theta) \end{bmatrix}. \quad (27)$$

As with the rover kinematic model formulated previously, the above represents an approximation since in general, large rotations cannot be treated in this way. The first two elements indicate slip about the global X and Y axes. But since the rover is constrained by the terrain, these two terms are set to zero leaving only the last element of equation (27), the slip about the yaw axis. Thus the final slip vector is formed by concatenating the global longitudinal and lateral slip distances from equation (26) with the global slip yaw angle from equation (27):

$$\mathbf{S}_g = \begin{bmatrix} S_X \\ S_Y \\ S_{\Psi} \end{bmatrix} = \begin{bmatrix} Sx_r(c(\Psi)c(\phi)) + Sy_r(c(\Psi)s(\phi)s(\theta) - s(\Psi)c(\theta)) \\ Sx_r(s(\Psi)c(\phi)) + Sy_r(s(\Psi)s(\phi)s(\theta) + c(\Psi)c(\theta)) \\ S\psi_rc(\phi)c(\theta) \end{bmatrix}. \quad (28)$$

Of course there is a lot more to modelling slip than merely transforming it from one frame to another. As will be discussed later in this chapter, slip is a nonlinear function of soil mechanics, terrain geometry, and wheel design. The truth model of the rover slip terms, $\mathbf{S}_r = [Sx_r \ Sy_r \ S\psi_r]$, that will be used in the terrain model for simulation is presented in the literature review of the final section of this chapter, along with a description of the slip-modelling research conducted at NASA's Jet Propulsion Lab and the California Institute of Technology. But first we present two examples that demonstrate the importance of slip modelling in the EKF rover localization algorithm.

2.4 Rover Localization in the Presence of Unmodelled Slip

In the following two examples, the capability of the rover localization algorithm to navigate a rover is investigated. The first simulation assumes that the rover, sensors, and environment have been modelled perfectly. The second simulation introduces slip into the terrain model which remains unmodelled in the localization algorithm.

2.4.1 Example 1: Perfect Rover Model

In Figure 8 a map depicting the commanded, true, and estimated paths of the rover through an environment populated with randomly generated features is presented. The commanded path is the desired path for the rover to traverse and represents the one set by human controllers in the absence of all knowledge of ground slope and slip that may affect the actual performance of the rover to navigate. The true path takes into account the actual environment and terrain characteristics such as slope and slip. The estimated path plots the EKF estimated rover states in the map using the kinematics and sensor models developed in subsections 2.2.2 and 2.2.3.

In this first simulation it is assumed that the rover kinematics and sensor models provide an accurate representation of the actual physical systems and that no environmental disturbances or external inputs are present. Thus, in this scenario, the rover is assumed to have perfect traction with the terrain and therefore the commanded and true paths are identical.

Figure 8 presents the path results and EKF estimate errors with 3σ error bounds. The EKF rover localization algorithm is shown to produce consistent estimates since 99% of the EKF estimate errors fall within the 3σ bounds.

Figure 8: Perfect Rover Model: The EKF is able to track the commanded path perfectly resulting in estimate errors that fall within the predicted 3σ error bounds.

2.4.2 Example 2: Unmodelled Slip-Induced Dynamics

This example demonstrates the performance of the EKF navigation algorithm when slip is present in the environment. In this scenario, the commanded and true paths will be different since the commanded path is a blind set of directions that do not take into account the terrain characteristics such as soil mechanics, slip, or slope. These parameters will have an effect on the actual position and orientation of the rover as it traverses the environment. But since these effects are not modelled, it is expected that the EKF-estimated path and the true path will diverge by some degree, let alone the divergence from the commanded path which, as discussed, constitutes a blind set of path instructions to the rover.

Figure 9 presents the results of the simulation in which slip was continuously acting on the rover. Clearly the EKF cannot follow the commanded path as unmodelled slip causes the rover to diverge. Since the EKF incorporates sensor measurements, the estimate is able, to some degree, to follow the true path. In fact, if the slip had only been periodic (and not continuous) then the EKF may have had time, during the brief periods of full traction where the kinematic rover model represented the actual system, to recover from the state estimate divergence. However, in the extreme case presented here, slip was imposed continuously, a realistic scenario on the Martian surface, resulting in large divergence errors.

2.4.3 Research Justification

These examples demonstrate the need to model the dynamic system more accurately. Even with the incorporation of sensor measurements, if a component of the system dynamics is missing, the EKF will not provide accurate estimates of the rover location in the environment. In this example, the missing component was a model for the slip present in the environment. As discussed in Chapter 1, slip poses a major problem

Figure 9: Unmodelled Slip-Induced Dynamics: The EKF is barely able to track the true path resulting in estimate errors that far exceed 3σ error bounds.

for rovers on Mars and, coupled with the fact that slip is a complex, hard-to-model system, represents an important area of research for those interested in the advancing the effectiveness of robotic planetary exploration.

2.5 Literature Review:

Current Slip modelling Research

The experience of the Mars Exploration Rovers, as discussed in Chapter 1, has demonstrated how important slip estimation can be for accurately navigating the slopes and sandy dunes that populate the Martian terrain. Originally only an extra credit capability, slip-detecting Visual Odometry was not originally planned for regular mission operation, but as of sol 555, 682 m of 5947 m was traversed by Opportunity using Visual Odometry [6]. Numerous other MER examples have shown how slip estimation is a vital component to mission success and rover survivability [4]. As such, research into slip detection using stereoscopic techniques has been conducted at NASA's Jet Propulsion Lab and Caltech.

2.5.1 A Priori modelled Slip

In this thesis we propose an algorithm that “learns” online the slip the rover is experiencing as a function of several measurable variables, which could include rover speed, terrain slope, and other terrain properties. This constitutes a regression of variables to find a linear or nonlinear relationship for slip estimation. We refer to this as our empirical slip model. Here we distinguish this learned or empirical slip model from efforts to construct slip models based on the physical properties of the terrain. Such slip models can be derived a priori and thus learning is not required.

Efforts have been made to model soil behaviour for rovers conducting planetary

exploration missions. These includes the work of Andrade to model the rover-soil interaction of the Marsokhod robot [36], and the characterization of Martian surface properties in-situ by both Sojourner in 1997 [37] and Opportunity in 2004 [38]. The mechanics of vehicle-terrian interaction, or terramechanics, is investigated in-depth by Wong including terrain stress distribution, plastic equilibrium theory, characterization of pressure-sinkage, repetitive load response, and shear stress-shear displacement relationships [35]. Additional coverage into soil mechanics is provided by Bekker [39].

Analytic slip models for robotic exploration rovers have also been formulated by Ward and Iagnemma in [40, 41] by using an EKF to estimate the true longitudinal rover velocity with modelled vehicle dynamic forces including braking, traction, and wheel rolling resistance along with wheel encoder and inertial measurement unit sensors. In their proposed algorithm, the tire traction/braking model and sensor measurements are used to estimate the external resistive forces acting to “immobilize” the robot. These estimates are then used to calculate the rover’s acceleration and velocity. In another work, wheel sinkage is assessed by Brooks, Iagnemma, and Dubowsky in which they formulate a visual method for rover mobility characterization by assessing rover wheel sinkage in deformable terrain [42].

In one sense these models are superior to learned models since they provide a complete definition of slip behaviour. However, the point of the work presented in this thesis is that slip is complex and thus inevitably such models are in fact incomplete leaving some aspects of the system behaviour unmodelled. Which brings us back to the justification for the inclusion of a slip-adaptive technique for slip modelling that uses online learning to capture the unmodelled dynamics.

2.5.2 MER Slip Model Development

Spirit and Opportunity were designed with a set of mobility requirements that included the ability to climb over obstacles of 25 cm in height, traverse slopes with

inclinations of at most 45 degrees without tipping over, and cover distances of over 1 km in terrain consisting of both high-traction hard-packed rocky soil and soft sandy deformable soil. These requirements were put to the test at JPL/Caltech by Lindemann and Voorhees using the MER-identical Dynamic Test Model (DTM) rover and a 16 ft square Variable Terrain Tilt Platform (VTTP) [2].

Over 200 tests were conducted where the DTM was commanded to traverse VTTP with a constant slope ranging from 0 to 20 degrees. The tests were also divided into two representative terrain surfaces including a hard high-traction surface and 6 in deep soft deformable sand. The orientation of the DTM with respect to the VTTP during the traverse was also varied and included drives up slope, down slope, across slope, and turn-in-place. Additionally, on both types of terrain, obstacles up to 25 cm in height were placed in the path of the DTM. Considering the slip of the rover CG and not at each individual wheel, the slip was measured as a percentage of total drive as follows:

$$S = \frac{v_c - v_t}{v_c} \times 100,$$

where v_t is the true traverse velocity, and v_c is the commanded traverse velocity in the local rover x -axis. Similarly, slip can be calculated in the rover y -axis, and angularly about the z -axis.

While excellent traction was experienced on the hard high-traction terrain, the slip measurements for the sand-covered VTTP over the range of table slopes indicated a direct nonlinear correlation between the table inclination and the resulting slip experienced by the DTM rover. Also, a different nonlinear correlation was observed depending on the orientation of the rover traverse (up slope, down slope, cross slope, turn-in-place). The slip data set, digitally extracted from [2], is presented below in Figure 10.

Figure 10: Longitudinal (top) and lateral (bottom) slip versus slope measurements taken from [2].

The graphs presented in Figure 10 combine the up and down slope curves into a continuous data set for slip in the longitudinal rover x -axis as shown in the top graph of Figure 10. The combined data set provides for positive and negative pitch (ϕ) inclinations. Likewise, in the rover lateral y -axis, positive and negative roll (θ) inclinations are provided by mirroring the cross slope slip measurements over the graphs slip-axis. The lateral slip is presented in the bottom graph of the figure. A linear regression was performed to produce the polynomial curve fit shown on each graph along with the correlation coefficient R .

This experimentally determined slip model will provide the “truth” slip experienced by the rover in the simulations developed later in this thesis. The simulations will be used to demonstrate the ability of the slip-adaptive rover navigation algorithm to model the slip online using a neural network as the rover traverses the simulated environment (see Chapter 4).

2.5.3 Remote Slip Estimation

Vision sensors can be used to extract texture and geometric information from the region over which a planned traverse path overlays, providing an means to predict slip remotely before it is actually experienced. Angelova et al. have developed just such a technique using stereo imagery information that yields the terrain type (soil, gravel, sand, asphalt, wood chips) and terrain geometry (roll, pitch) that corresponds to a particular learned slip model [13–15]. The image information I , consists of the terrain texture and colour based on appearance characteristics, A , while the terrain geometry, denoted G , are estimated from the visual information and an inertial measurement unit (IMU). Thus, $I = (A, G)$. The approach partitions the input space into subregions of terrain type, T , with the probability of the terrain type dependent on the terrain appearance A , or $P(T|A)$. Letting $f(S|I)$ be a regression function of slip given the visual information, then

$$f(S|I) = \sum P(T|A)f(S|T, G).$$

The above partitions the learning problem into one of terrain recognition, $P(T|A)$, and slip prediction, $f(S|T, G)$. Thus once the terrain has been classified, then the slip learning algorithm can concentrate on the slip behaviour for a single terrain type.

The learning algorithm is based on *receptive field regression* which learns a locally linear regression model (the receptive field) in a small space of the actual nonlinear function [43]. Each receptive field is learned independently of the others and the set of fields form a so-called *mixture-of-experts*. This approach allows for local variations in the in nonlinear slip model to be learned as new information is encountered.

In the neural network approach used in this thesis, new information is generally applied globally to the entire network. In Chapter 3 it will be shown how over time, the neural network learns by averaging the weights to the combination that provides the “best fit”, in a least squared sense (if error back-propagation is used for training) or a minimum variance sense (if the EKF is used for training), for the nonlinear function output being approximated. This means that if there are discontinuities in the function, the neural network may not provide a good estimation of the output, whereas the receptive field approach would be able to capture the discontinuity.

In their experimental configuration, slip is measured directly using visual odometry, which is the current method used by the MERs. Two images are taken in sequence from which the movement of visual features such as Harris corners are tracked to determine the actual distance traversed. Comparing this value to the commanded traverse distance results in the slip of the rover. A complete navigation system overview, including remote slip prediction, is provided by Helmick et al. in [12].

Unfortunately, in order to learn the slip function that corresponds to a particular terrain type so that it can be predicted remotely, the rover must first traverse several

paths locally. Also, some problems were experienced in their terrain classification algorithm to determine the correct terrain type. Ultimately, the learning algorithm was conducted off-line and slip estimation occurred only in the x-axis direction [15].

Chapter 3

Neural Network Enabled Adaptation

By using an EKF-based navigation technique for rover localization, this thesis has illustrated how wheel slip can cause serious deviations in the rover position estimate even when sensor information is supplied to correct the dynamic model-based estimate. This is because a significant component of the overall system dynamic model that would incorporate the effect of terrain inputs was missing. The missing component is the *slip model* which contributes as much to the rover position as the *rover kinematic* model developed in the previous chapter. To provide the slip model, one approach would be to derive an analytic model based on the physics of the system. Such is the approach used with the rover kinematic model where control inputs to the wheel are mapped to position outputs via simple geometry.

In this thesis, we instead focus on the development of a *slip-adaptive* approach capable of modelling slip empirically as the rover “experiences” it. Justification for this might rest on the fact that if slip-induced errors can be modelled adaptively in this way, perhaps other errors from unknown internal and external factors could be captured the same way and provide mission-saving advantages over systems without such an adaptive technique. Unfortunately, blind formulation of dynamic system models is simply not practical since it is difficult to model a system empirically without knowing in advance what input and output parameters define the system.

These parameters must also be correlated in order for an empirical model to be constructed. Thus, whether an analytic or empirical model is implemented, some “engineering thinking” must be carried out in advance in order to fully understand the nature of the modelled system. This will be demonstrated with the slip-adaptive technique developed in this work.

Still, there are advantages to adaptively and empirically modelling slip. These include the ability of the adaptive model to autonomously adjust its slip model as environmental parameters change which can provide for safer and more accurate Martian traverses of extremely slippery terrain. Linearization of highly nonlinear systems, common in many analytic model formulations, can also be avoided in this approach. Also, with more onboard autonomy, longer traverses would become possible since less human-specified navigation commands would be required and thus less overhead costs in time are spent solving navigation problems and more time is spent achieving scientific and exploration goals. Finally, this approach would provide a way to capture the full complexity of the slip automatically without the need to invest the resources that would be required to develop the analytic equivalent model. As discussed previously, slip is a complex function of soil mechanics, terrain geometry, and wheel design.

The approach proposed in this thesis uses a multi-layer feed-forward neural network (MLFN) to provide the slip-adaptive model. Thus, the overall system dynamic model would consist of the sum of the analytic rover kinematic model that maps control inputs into rover position state estimates, and the slip-adaptive neural network empirical model that maps terrain inputs into rover slip estimates. The next section provides an overview of MLFNs.

3.1 Multi-Layer Feed-Forward Neural Networks

There are several different types of neural network architectures, however, this thesis investigates the suitability of the most widely used form called the multi-layer feed-forward neural network (MLFN). The MLFN can easily be incorporated into the EKF algorithm for training and slip approximation functions, as will be investigated later in this thesis. An introduction to the other types of neural networks can be found in [44] and [45]. This section will first introduce the specific terminology used when describing the components of a neural network.

An overview of MLFNs as given by Haykin [44] is described here. Shown in Figure 11 is a typical MLFN architecture and is, as the name suggests, composed of multiple layers. In the figure, there are three layers illustrated and are respectively called the input, hidden, and output layers. If there were more than three layers, then the additional layers would also be called hidden layers. As will be discussed in the following section, at least one hidden layer is required to provide function approximation capability to the neural network. Each layer is composed of a specific number of so-called “neurons” that, with the exception of the input-layer which simply propagates the input signal unchanged, perform a weighted sum computation. Subsequently a so-called activation function may be applied to the resulting sum that determines whether the neuron has received sufficient input to be “on” or “off”. This result is then passed forward to the neurons in the next layer. In the next section this qualitative description of the MLFN architecture is placed into a mathematical framework and more of the details shown in Figure 11 will be explained.

3.1.1 Universal Function Approximators

Neural networks with as few as one hidden layer are universal function approximators. It has been rigorously proven by Hornik et al [46] that a three-layer MLFN, such as

Figure 11: A three-layer multi-layer feed-forward neural network.

that shown in Figure 11, with an arbitrary sigmoid activation function in the hidden layer neurons, can approximate any arbitrary mapping, from one dimensional space to another, to any desired degree of accuracy. This approximation assumes that there are no constraints on the number of neurons (i.e. there are *sufficiently* many) in the hidden layer. It has also been shown by Gallant and White and Hecht-Nelson that a MLFN can be used to generate the Fourier series approximation using one hidden layer containing cosine activation functions [47, 48]. It is well known that the Fourier series can be used to approximate any function to any degree of accuracy [45], and so the MLFN can be viewed in a similar sense.

A MLFN is presented in Figure 11. The network consists of a total of three layers with inputs, $\mathbf{i}$, entering the nodes of the first layer (*input layer*), the signal propagates forward through the second layer (*hidden layer*), and finally the network outputs, $\mathbf{g}$, exit the network from the third layer (*output layer*). The number of M inputs and O outputs depends upon the nature of the function being approximated. For example, if the neural network was trained to approximate a sine function, one angle input

(and typically a bias with fixed input value of 1) would be required, thus $M = 2$, and one output argument would result, $O = 1$. In the general case, the MLFN can be trained to map any set of input arguments from $\mathbb{R}^M$ to any set of output arguments in $\mathbb{R}^O$, $M \neq O$.

As the signal propagates through the network the neurons act on the inputs to transform them into the desired outputs. For example, consider the first neuron. In a fully connected network, all inputs will have a path to the first neuron with an associated weight¹. As the inputs flow to neuron one, a linear combination of the weights and inputs is performed and a bias term is applied². Thus, for the n^{th} hidden layer neuron, $n \in \{1, 2, \dots, N\}$, the m^{th} input, i_m where $m \in \{1, 2, \dots, M\}$, and where the weight connecting the m^{th} input to the n^{th} neuron is w_{nm} , the first step in the neuron calculation produces the *induced local field* [44], v_n^1 :

$$v_n^1 = \sum_{m=1}^M w_{nm} i_m. \quad (29)$$

The “1” superscript refers to the first set of neurons in the hidden layer. The second step applies a nonlinear activation function which, in the context of simulating biological neural behaviour, determines if the artificial neuron is “activated” by the weighted input. A popular choice of activation function is the sigmoid function, shown in Figure 12, since they are continuous, differentiable everywhere, and asymptotically approach their saturation values (0 and 1 as indicated in the figure). The first two properties are attractive for neural network implementation since training algorithms such as Error Back-Propagation involve taking the derivative of the neural network. The asymptotical saturation limit, on the other hand, provides a mechanism for

¹Throughout this thesis, references to “weights” are the neural network variety.

²The bias has the effect of shifting the output of the linear combination of weights and inputs in the resulting output space much the same way as “ b ” shifts the curve in the line equation $y = mx + b$.

Figure 12: A sigmoid activation function is continuous and everywhere differentiable.

controlling neuron “firing” or rather, determining neural “on-off” status. The non-linearity of the activation function also enables the neural network to approximate nonlinear functions.

Applying the sigmoid function to the weighted input determined previously is shown below.

$$g'_n = \Phi^1(v_n^1) = \frac{1}{1 + \exp(-v_n^1)}. \quad (30)$$

This process is repeated for each neuron in the hidden layer. A similar process is carried out as the signal propagates from the hidden layer to the output layer. However, for the purpose of this thesis, no activation function will be applied to the final set of network outputs since these values will ultimately represent the wheel-slip approximation, and thus do not “saturate” to values between 0 and 1. Thus, in

Figure 11, the output layer neuron calculations are defined by

$$v_o^2 = \sum_{n=1}^N w_{on} g'_n, \quad (31)$$

$$g_o = \Phi^2(v_o^2) = v_o^2. \quad (32)$$

This last step is repeated for each o^{th} neuron in the output layer, $o \in \{1, 2, \dots, O\}$. Knowing the number of inputs, outputs, and hidden-layer neurons provides enough information to determine the total number of weights of a MLFN as defined in this thesis. The total number of weights, W , can be determined from

$$W = N(M + O). \quad (33)$$

The output of the entire MLFN given a vector set of inputs at time step k can be represented as a series of nested matrix operations as follows (subscript represents the matrix dimension):

$$\mathbf{g}_{O \times 1} = \Phi^2(\mathbf{v}_{O \times 1}^2) = \Phi^2(\mathbf{W}_{O \times N}^2 \Phi^1(\mathbf{v}_{N \times 1}^1)) = \Phi^2(\mathbf{W}_{O \times N}^2 \Phi^1(\mathbf{W}_{N \times M}^1 \mathbf{i}_{M \times 1})), \quad (34)$$

where we define the input vector, $\mathbf{i}$ (note the bias input of 1), and weight matrices for the hidden and output layers, $\mathbf{W}^1$ and $\mathbf{W}^2$ respectively, to be

$$\mathbf{i} = [i_1 \ i_2 \ \dots \ i_M \ 1]^T, \quad (35)$$

$$\mathbf{W}_{N \times M}^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & \dots & w_{1M}^1 \\ w_{21}^1 & w_{22}^1 & \dots & w_{2M}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^1 & w_{N2}^1 & \dots & w_{NM}^1 \end{bmatrix}, \quad (36)$$

$$\mathbf{W}_{O \times N}^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & \dots & w_{1N}^2 \\ w_{21}^2 & w_{22}^2 & \dots & w_{2N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_{O1}^2 & w_{O2}^2 & \dots & w_{ON}^2 \end{bmatrix}, \quad (37)$$

and also the intermediate vectors of weighted sums in the hidden and output layers, respectively are

$$\mathbf{v}_{N \times 1}^1 = [v_1^1 \ \dots \ v_n^1 \ \dots \ v_N^1]^T, \quad (38)$$

$$\mathbf{v}_{O \times 1}^2 = [v_1^2 \ \dots \ v_o^2 \ \dots \ v_O^2]^T. \quad (39)$$

$$(40)$$

3.1.2 Error Back-Propagation Training

In a fully connected MLFN, each input is connected to every neuron in the hidden layer, which in turn is connected to every output neuron in the output layer. Each pathway connecting the inputs to the hidden neurons, and the hidden neurons to

the output neurons, has a specific weight value which is assigned via the training algorithm. The most well known training algorithm for neural networks is *Error Back-Propagation*, which assigns to each weight a correcting value based upon the minimization of the error between the true output and the neural network-predicted output. For example, at a particular time step k , if the known output value of the neural network at the o^{th} output neuron is d_o , then the error is

$$e_o = d_o - g_o. \quad (41)$$

The sum of squared errors over all neurons in the output layer at this instant is called the *instantaneous error energy* [44] and is defined by

$$E_k = \frac{1}{2} \sum_{o=1}^O e_o^2. \quad (42)$$

Since the instantaneous error energy is a function of the neural network weights (i.e. refer to equation (41) and note that the neural network output is a function of the weights, $g_o(\mathbf{i}, \mathbf{W}^1, \mathbf{W}^2)$), E_k can be minimized by adjusting the neural network weights. The Error Back-Propagation algorithm uses the following approximation, given in [44], to generate at each time step the weight correction:

$$\Delta w_{nm} = -\eta \frac{\partial E_k}{\partial w_{nm}}. \quad (43)$$

Equation (43) represents the gradient decent (negative partial derivative) in weight space that forces the weights to approach the values that minimize the error. The term, η , is the learning rate parameter, usually set to $0 < \eta \leq 1$. Further steps are required in order to “back-propagate” the error correction throughout the weights in the hidden layer of the MLFN. A complete derivation of the algorithm is provided by Haykin [44]. While the original Error Back-Propagation algorithm was proposed

in the 1960's by Rosenblatt [49], successful application did not occur until the 1980's based on the method proposed by Werbos in 1974 [50], and was later popularized in the work by Rumelhart et al in 1986 [51].

3.2 Literature Review:

Hybrid EKF-Neural Networks

A unifying perspective on the relationship between the Kalman filter and the neural network is obtained when they are both viewed as data fusion methods. Data fusion is the process of combining the information from several different sources and integrating them into a single process such that the performance of the process is improved over using the individual information components. Deng and Hwang classify data fusion into two types [52]. The first is parametric for which the Kalman filter is an example. The second is called information theoretic exemplified by the neural network. Sasiadek goes further, classifying three types of data fusion algorithms: probabilistic model-based fusion such as Bayesian reasoning, least-squares fusion such as the Kalman filter, and so-called intelligent fusion such as the neural network [53].

However one chooses to classify these data fusion methods, it seems natural to attempt to combine them in order to achieve a hybrid approach that combines their respective advantages. The EKF provides a probabilistic framework for the estimation of unknown parameters in the presence of noise and the construction of parameters that cannot be measured directly (assuming the system is observable). The disadvantage is that the dynamic process must be modelled a priori and perfectly as was shown in Chapter 2. Also, the noise must be modelled as zero-mean, white, and Gaussian. The MLFN can provide online learning capability, thus doing away with the need for a prior modelled or perfectly modelled systems since unknown or complex

model dynamics can potentially be captured online using a MLFN. However, neural networks are only as good as the information used to train them.

Several examples from literature that illustrate a variety of techniques for combining the Kalman filter and neural network into hybrid algorithms are presented in this section.

3.2.1 EKF-Based Training

The popular error back-propagation learning algorithm, discussed in section 3.1.2 of this thesis, has been widely used since the mid-1980's but due to the characteristics of the error surface over which the algorithm searches in its gradient descent, convergence can be slow, many training epochs³ can be required, solutions can get stuck in local minima, and the solution can be sensitive to weight initialization [54]. Due to these inherent problems in the error back-propagation algorithm various training alternatives have been formulated in order to speed up convergence and reduce training epochs.

A promising alternative was formulated by Singhal and Wu by using an EKF for training a MLFN [55]. Later, Iiguni and Sakai proposed essentially the same technique and giving a brief acknowledgement to Singhal and Wu [56]. In this training algorithm, the problem is treated as a nonlinear system identification problem. Singhal and Wu attribute the slow convergence of error back-propagation to the fact that the algorithm does not incorporate the knowledge gained from previous training data and thus requires possibly thousands of iterations for convergence of the weights to be achieved. The Kalman filter, alternatively, incorporates the knowledge gained from all previous information and computes the minimum variance estimate causing faster convergence and requiring fewer iterations in training. In fact, it was shown by Ruck et al. [57] that under certain assumptions, the EKF will reduce to

³Neural network techno-speak for "iteration".

the back-propagation algorithm. As this EKF-based training method is adopted for later use in this thesis, a detailed discussion of the method along with several training examples can be found in section 3.3.

In application, Singhal and Wu’s technique has been used by Mosavi, Mirzaeepour, and Nabavi in order to train a MLFN to provide the DGPS⁴ error correction to a vehicle navigation system during temporary signal outages [54].

Other research has provided further insight into the performance optimization of the EKF-training algorithm and its application to other types of neural networks besides the simple MLFN. Heimes presents various implementations of EKF training for recurrent and static neural networks using so-called “decoupling” and “multi-streaming” methods [58]. The training of Recurrent Neural Networks is investigated in the paper by Puskorius and Feldkamp using a Decoupled EKF training algorithm [59]. In another paper by Zhang, Wu, Zhang, and Zheng the Unscented Kalman filter is proposed as a training algorithm for Minimal Resource Allocation Neural Networks and training performance is compared with the EKF-training algorithm [60].

3.2.2 Neural Networks as Nonparametric Plant Dynamics

In the last section, the EKF was used to train the neural network to provide some input-output mapping. In this sense the states were the weights of the network and the EKF produced the optimal weight estimate in the minimum variance sense. Alternatively, the neural network can also augment the parameter-based analytically modelled dynamic system in the predictive step of the EKF, providing a technique for modelling unknown system nonlinearities and error sources in a nonparametric⁵ way. The following literature review investigates examples of this type of hybrid approach to combining EKFs with neural networks.

⁴Differential GPS

⁵Nonparametric models are based upon empirical data as opposed to analytic models which use physical parameters, such as mass, to define the system characteristics.

The algorithm developed by Lobbia, Stubberud, and Owen in [17] uses an EKF to estimate the states of an analytically modelled cantilever beam dynamics system while a MLFN is used to provide corrections to the state estimates. The MLFN thus acts as an empirical model that captures the unknown highly nonlinear part of the dynamics. Additionally, along with providing the state estimates for the plant dynamics, the EKF simultaneously estimates the weights of the MLFN. Thus, this approach takes advantage of the fact that the EKF output, based on sensor measurements and sensor model, will be able to construct both the weights of the network and the states of the plant dynamics providing that the system is observable. Further, this system is formulated such that the MLFN is trained online.

This powerful and innovative technique was adopted as the primary algorithm upon which the slip adaptive navigation technique, developed as the main contribution of this thesis, is based. Thus, a detailed development of this algorithm postponed until section 3.4 of this thesis.

An alternative approach to the above was provided by Zhan and Wan by using an unscented Kalman filter (UKF) algorithm thus avoiding the need to linearize the plant dynamics in order to propagate the error covariance matrices and compute the Kalman gain as is the case with the EKF approach [61]. However, application of the UKF was not considered in this thesis but is proposed as an ideal area for future research.

In the next section, several research papers are overviewed that demonstrate how a hybrid approach to estimation by combining an EKF and neural network can be accomplished. While the algorithm developed by Lobbia, Stubberud, and Owen will be adopted in this thesis, many other formulations exist and are presented below. For the purposes of this thesis the discussion is limited to navigation applications.

3.2.3 Hybrid Navigation Applications

An autonomous and adaptive system for interplanetary satellite navigation that uses EKF, single-layer neural networks, and genetic algorithms is presented by Crain, Bishop, and Ely in [62]. The algorithm they develop identifies and corrects anomalous tracking behaviour caused by changes in the system environment causing deviations between the true satellite behaviour and the modelled plant dynamics and sensors. The space environment offers many challenges to providing accurate interplanetary navigation, such as accurately modelling the solar radiation pressure surface on Mars Pathfinder or the thrust performance of Mars Climate Orbiter and Mars Polar Lander.

The architecture of this navigation algorithm consists of two components: a near real-time autonomous monitoring component and an off-line adaptive modelling component. Operating in parallel to the real-time navigation algorithm, the autonomous monitor consists of a bank of EKFs contained in a so-called hierarchical mixture-of-experts (HME). Each EKF bank is designed to provide “expertise” in one of the following three unmodelled environmental modes: (1) impulsive velocity changes (e.g. thruster misfire), (2) sun radiation pressure mismodelling (e.g. dynamic environmental change), (3) unmodelled impulses and measurement noise changes. Each individual filter provides a specific estimation expertise over a region of the navigation parameter space. The EKF banks are regulated by single-layer neural networks called gating networks that effectively determine which unmodelled mode is presently occurring. This is accomplished by using the gating networks to rank each EKF bank with a set of weights based on the collective residual errors in the estimates produced by the EKFs within a particular bank. Once the unmodelled mode has been identified, the off-line adaptive modelling component, consisting of a genetic algorithm, is engaged to correct the set of navigation parameters and provide these updates to the real-time navigation algorithm.

In a more direct application, Jow, Chang, and Lin present neural network-aided Kalman filter for GPS navigation [63]. In this approach, a multi-layer neural network is trained using error back-propagation to provide a correction to the system dynamics model which estimates the current position and velocity of a vehicle equipped with a GPS receiver. In this sense, the neural network becomes a nonlinear dynamic error model. In its implementation, an off-line training phase is commenced in which a vehicle is driven at 100 m/s over a known trajectory to gather the training data. At each time step, the network is provided with a vector consisting of 18 EKF innovation errors and the two previously corrected estimates (20 values in total), and maps these using a hidden layer of 16 neurons and output layer of two neurons into the state estimate corrections on vehicle position and velocity. The trained network was then verified online in a vehicle traveling at 70 m/s over a different trajectory.

Another GPS application presented by Wang, Wang, Sinclair, and Watts uses a neural network to learn the relationship between the change in vehicle velocity and orientation to the corrected vehicle dynamic states issued by an EKF while GPS measurements are available [64]. Then, in the situation where a short-term GPS outage occurs, the neural network is used to continue to provide accurate position and velocity estimates until GPS reception is reestablished.

Neural networks can also be used in conjunction with Kalman filters in order to estimate the process and measurement noise covariance matrices thereby helping to improve the performance of the filter in navigation applications. In [65], Cao, Fang, and Sheng investigate this approach using a UAV⁶ equipped with a micro inertial measurement unit (MIMU), GPS receiver, and a micro magnetic compass (MMC). Each sensor has its own associated bias term and noise statistics which may not necessarily be zero-mean, white, or Gaussian. While their work is not clear on

⁶Unmanned Aerial Vehicle

how exactly the neural network is trained, or whether training occurs on- or off-line, a so-called neural network adaptive controller (NNAC) scheme is set up such that the EKF innovation errors, estimated states, linearized dynamics matrix, and the difference between the previous corrected noise covariance and current predicted covariance matrices, $\mathbf{Q}$ and $\mathbf{R}$, are provided to the NNAC. The output is then the true noise covariance matrices associated with the EKF process and sensor models.

Adaptive target tracking provides another problem to which a combination of Kalman filtering and neural networks can provide potential benefits. Typical target tracking techniques, such as the interacting multiple model (IMM), employ several motion models to capture, a priori, the manoeuvring capabilities of potential targets. Based on the work of Lobbia, Stubberud, and Owen [17], Kramer and Stubberud develop an adaptive algorithm to track multiple targets without prior knowledge of their manoeuvring capabilities and using only a single system model [66]. The target kinematics are described using a straight line motion model defined by

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}),$$

$$\begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \\ y_{k-1} \\ \dot{y}_{k-1} \end{bmatrix},$$

where $\mathbf{q}_k = [x_k \ \dot{x}_k \ y_k \ \dot{y}_k]^T$ are the position and velocity states in an $\{X, Y\}$ plane, T is the sample time, and k is the k^{th} time step. In order to adapt this simple model to a variety of targets, a neural network is added to the kinematics model to provide the unmodelled (in a parametric sense) components of the target. Thus,

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}) + g(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}),$$

where $g(\cdot)$ represents the neural network output (corrections to the four states), and $\mathbf{w}_{k-1}$ are the vector of weights. Kramer's approach also takes the interesting step of also using the EKF to estimate the weights of the neural network along with the states of the target. Thus, the neural network weights also become the states of the system. Therefore, the final state space system must be augmented with the weights of the neural network as follows:

$$\begin{bmatrix} x_k \\ \dot{x}_k \\ y_k \\ \dot{y}_k \\ \mathbf{w}_k \end{bmatrix} = \begin{bmatrix} 1 & T & 0 & 0 & \mathbf{0}_{1 \times W} \\ 0 & 1 & 0 & 0 & \mathbf{0}_{1 \times W} \\ 0 & 0 & 1 & T & \mathbf{0}_{1 \times W} \\ 0 & 0 & 0 & 1 & \mathbf{0}_{1 \times W} \\ \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{I}_W \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \\ y_{k-1} \\ \dot{y}_{k-1} \\ \mathbf{w}_{k-1} \end{bmatrix} + \begin{bmatrix} g_1(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}) \\ g_2(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}) \\ g_3(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}) \\ g_4(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}) \\ \mathbf{0}_{W \times 1} \end{bmatrix},$$

where g_1 , g_2 , g_3 , and g_4 represent the state corrections from the output nodes of a single neural neural network. The subscript W is the number of weights and is used above to indicate the dimension of the state space. As indicated in the above equation, the dynamics of the weights are assumed to be constants in the prediction step of the EKF. The weights will therefore be updated in the measurement step, assuming system observability, causing the initial values to approach their correct constant values.

In their validation tests, the so-called neural EKF (NEKF) is trained online in

three tracking experiments, one for a simulated ground vehicle, a second for a simulated watercraft, and a third for an actual F-4 aircraft flying a series of racetrack manoeuvres. Comparing their results to that of a regular EKF, the tracking performance of the NEKF provided a general improvement.

The guidance system of an interceptor missile using a Kalman filter augmented with a so-called General Regression Neural Network (GRNN) was presented by Fisher and Rauch in [67]. The authors pose that one approach to parameter estimation for systems containing large nonlinearities and unknown initial conditions is to implement multiple EKF's to model the system: the one with the minimum residual becomes the winner. As an alternative, Fisher and Rauch develop a GRNN, training off-line, to provide corrections to the EKF initial conditions and system parameters when its estimates produce large errors from the expected values in areas of high nonlinearity. The GRNN is described as a “supervisor” that allows the EKF-linearized model to track nonlinear regions of the parameter space.

To demonstrate this method, they use the guidance control law of an interceptor missile over the terminal flight (last ten seconds). During hand-off from a tracking radar, the target may be manoeuvring and the guidance system must detect these manoeuvres in order to update the control law. Eleven state variables are estimated including the target positions (3), velocity (3), acceleration (1), attitude (3), and angular rate (1).

3.3 EKF Trained Neural Networks

While error back-propagation is well-known, it comes at a cost of slow convergence which has resulted in the development of other training algorithms. Among these was the development of the EKF training technique. While the EKF may result in a faster convergence, according to Ruck et al. [57], back-propagation is a degenerate

form of the EKF training algorithm, which in turn is shown to require three orders of magnitude more floating-point operations than back-propagation. Conversely, the computational complexity arises from information the EKF retains that aids in improving convergence performance but which is lost when using back-propagation.

The EKF training algorithm for neural networks was proposed by Singhal and Wu [55] in 1989 and Iiguni and Sakai [56] in 1992. In these expositions, however, observability is merely assumed. As a contribution of this thesis, this section, after first providing an overview of the EKF training technique for MLFNs, several examples are presented that investigate the EKF observability, weight convergence characteristics, and MLFN function approximation capability.

This EKF training technique holds significant relevance to the slip-adaptive navigation technique developed in this thesis since the EKF will also be used to provide the localization estimate of the rover. Later on it will be proposed that the EKF can perform both rover localization and MLFN training simultaneously thus providing a potentially robust and elegant solution to online slip-adaptive rover navigation.

3.3.1 EKF Training Algorithm

Training using an EKF can be viewed as a nonlinear dynamical system identification problem where the weights of the MLFN become the states of the system. In this approach, it is assumed that the outputs of the nonlinear system, $\mathbf{y}_k = f(\mathbf{x}_k)$, can be observed directly and this provides the basis for which this method can be used for neural network training. By providing the same known system input, $\mathbf{i}_k = [\mathbf{x}_k \ 1]^T$, to the MLFN, the network output, $\hat{\mathbf{y}}_k = g(\mathbf{i}_k, \mathbf{w}_k)$, can be used to approximate the true system output, $\mathbf{y}_k = f(\mathbf{x}_k)$, and the innovation error can be determined from

$$\begin{aligned}
\mathbf{e}_{inov} &= \mathbf{y}_k - \hat{\mathbf{y}}_k, \\
&= f(\mathbf{x}_k) - g(\mathbf{i}_k, \mathbf{w}_k).
\end{aligned} \tag{44}$$

In the above expression, $f(\mathbf{x}_k)$ represents the sensor measurement of the nonlinear system output and $g(\mathbf{i}_k, \mathbf{w}_k)$ is the nonlinear MLFN-measurement model. In order for the MLFN to approximate the system outputs, however, the weights of the network must be trained so that the innovation error of equation (44) converges to zero. To accomplish this, the system dynamics model must be derived which provides the predicted MLFN weight values. In this scenario, the true MLFN weight values are assumed to be constants that the EKF is trying to estimate. Thus, in the prediction step of the EKF algorithm, the dynamic system is represented by

$$\mathbf{w}_k = \mathbf{F}\mathbf{w}_{k-1}, \tag{45}$$

where the time-invariant dynamics matrix is $\mathbf{F} = \mathbf{I}_W$ and $\mathbf{I}_W$ is the identity matrix of dimension $W \times W$ where W is the total number of weights. Therefore, in the prediction step, the system is linear time-invariant (LTI). The states are represented by $\mathbf{w}_k$ which are the vector of weights of the MLFN⁷. Note that even though the weights are modelled as constants, the estimated value of the weights will change over time until convergence occurs. Thus, the subscript k is used to denote the specific value of the weights at time step k during estimation.

Taking the linear system dynamic model and nonlinear system measurement model developed above, the EKF algorithm developed in Chapter 2 can now be applied for MLFN training. A block diagram depicting the training algorithm is

⁷For the purpose of the EKF, the weights are ordered into a vector format when used as states, and in matrix format when used in the neural network as shown in the previous section.

Figure 13: EKF training algorithm for the MLFN.

shown in Figure 13. For the linearization steps of the EKF algorithm, the linearized dynamics matrix is trivial since the dynamics are already linear. The measurement model, however, which is constructed using the MLFN, is nonlinear and must be linearized at each time step in the algorithm. Thus the overall system represents a linear time-varying (LTV) system. Denoting the linearized measurement model at time step k using the usual variable $\mathbf{H}_k$, and the o^{th} neural network output as $g_{o,k}$, where the states are the weight estimates of the L^{th} layer are defined by $w_{nm,k}^L$, then the linearized measurement model is determined from

$$\mathbf{H}_k = \frac{\partial g_{o,k}}{\partial w_{nm,k}^L}, \quad (46)$$

where the derivatives of the MLFN (dropping the k from the notation) with respect to the the weights in the hidden and output layers are, respectively

$$\frac{\partial g_o}{\partial w_{nm}^1} = w_{on}^2 \frac{\partial \Phi_n}{\partial v_n^1} i_m, \quad (47)$$

$$\frac{\partial g_o}{\partial w_{on}^2} = \Phi_n, \quad \frac{\partial g_o}{\partial w_{c \neq on}^2} = 0. \quad (48)$$

To clarify equation (48), the derivative of the MLFN with respect to the output weights depends on whether the weight is directly linked to the particular output neuron, $o \in \{1, 2, \dots, O\}$. If it is linked then the equation on the left is used, if it is not linked as shown in the subscripts $c \neq o$, then the equation on the right is used.

Assembling the linear MLFN weight dynamic model and nonlinear function MLFN measurement model into the EKF algorithm produces a technique for training the MLFN to approximate any nonlinear (or linear) function. The dynamics model is based on the assumption that there exists a set of weights with constant values that can approximate the nonlinear function in a MLFN framework. The nonlinear measurement model is the MLFN that is required to model the function output. The difference between the neural network output and the actual or true function output provides the innovation error. Then using the Kalman gains produced by the EKF algorithm, the correction required to update the original estimate of the network weights is determined. In algorithmic form, this description can be expressed as

Prediction Step: Linear Dynamics Model Estimation

$$\hat{\mathbf{w}}_k^- = \mathbf{F} \hat{\mathbf{w}}_{k-1}^+, \quad (49)$$

$$\mathbf{P}_k^- = \mathbf{F} \mathbf{P}_{k-1}^+ \mathbf{F}^T + \mathbf{Q}. \quad (50)$$

Update Step: Nonlinear Measurement Model Correction

$$\mathbf{H}_k = \frac{\partial g_{o,k}}{\partial w_{nm,k}^L}, \quad (51)$$

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R})^{-1}, \quad (52)$$

$$\hat{\mathbf{w}}_k^+ = \hat{\mathbf{w}}_k^- + \mathbf{K}_k (f(\mathbf{x}_k) - g(\mathbf{i}_k, \mathbf{w}_k)), \quad (53)$$

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K}_k \mathbf{H}_k \mathbf{P}_k^-. \quad (54)$$

To complete the training algorithm, the initial states, $\hat{\mathbf{w}}_{k-1}^+ = \hat{\mathbf{w}}_0$, and initial estimate error covariance, $\mathbf{P}_{k-1}^+ = \mathbf{P}_0$, must be chosen and the covariance matrices for process and measurement noise, $\mathbf{Q}$ and $\mathbf{R}$, defined before training can begin.

3.3.2 EKF Training Examples

To demonstrate the EKF MLFN training algorithm developed in the previous section it is employed here to approximate a sine function. Thus it is desired to model with a MLFN the function $\mathbf{y}_k = f(\mathbf{x}_k) = \sin(\mathbf{x}_k)$ where $\mathbf{x}_k$ represents a vector of r_q angles. In the following examples, the system observability is assessed and it is shown that despite observability, convergence of the states is not guaranteed. Moreover, it is demonstrated how providing a additional measurements can improve state convergence but conversely have adverse effects on the ability of the MLFN to adequately approximate the desired function. These results are demonstrated through a series of examples in which the effects of first increasing the number of training inputs to the MLFN and then increasing the number of hidden layer neurons are investigated.

For the following examples the initial error covariance is set to $\mathbf{P}_{k-1}^+ = 10 \times \mathbf{I}_W$, which represents the level of uncertainty in the initial estimate of the weights. The process noise covariance is assumed to be constant and is defined as $\mathbf{Q} = 0.01 \times \mathbf{I}_W$. The measurement noise covariance is set to $\mathbf{R} = \sigma\pi/180 \times \mathbf{I}_r$, where $\sigma = 0.1$ degrees

and $r = r_q$ is the number of measurements at the k^{th} time step. Finally, the time step is set to $T = 0.25$ seconds.

Example 1: Single Neuron MLFN - One Constant Input Angle

Beginning with a simple example, we set the number of neurons to $N = 1$ and the number of neural network inputs to $M = 2$ (the angle input and bias); thus $\mathbf{i}_k = [\mathbf{x}_k \ 1]^T$ where $\mathbf{x}_k = 45^\circ$ is the input angle to the MLFN. The MLFN that is used to approximate the sine function is thus

$$\hat{\mathbf{y}}_k = g(\mathbf{i}_k, \mathbf{w}_k), \quad (55)$$

where initial set of weights, $\hat{\mathbf{w}}_{k-1}^+ = \hat{\mathbf{w}}_0$, are determined using a random number generator that generates numbers with zero mean, standard deviation of one, and Gaussian distribution. The total number of weights is determined from equation (33), which in this case with $N = 1$, $M = 2$, and number of outputs $O = 1$ produces $W = 3$. This means that the rank of the observability Grammian for the EKF must be $n_{obs} = 3$.

With the above parameters defined, the EKF training algorithm was used to train the MLFN to approximate a sine function. The function approximation is shown in Figure 14.

From the figure, it appears that the EKF training technique can approximate the sinusoidal function for the given constant input of 45° . Note that the noise on the approximated value is due to the process and measurement noise statistics imposed during training and defined prior to this example. If the training was successful, the final set of weights should be able to approximate the same function when supplied with the same input. Another important result to investigate is how well the MLFN

Figure 14: Example 1: MLFN approximation of $\sin(45^\circ) = 0.707$. $N = 1$, $W = 3$, $r_q = 1$. Oscillation is caused by the noise parameters.

Figure 15: Example 1: Proof-of-Match results for $\sin(\mathbf{x}_k)$, with $\mathbf{x}_k = \{30^\circ, 45^\circ, 60^\circ\}$.

approximation generalizes to other inputs. To test how well the trained MLFN approximates sine, a “Proof-of-Match” (POM) program is constructed which simply re-runs the original input through the trained MLFN to see if the EKF training did indeed succeed. As well, additional inputs are provided to test the generalizability of the trained network. In turn, POM is run for $\mathbf{x}(j)_k = \{30^\circ, 45^\circ, 60^\circ\}$, with $\mathbf{i} = [\mathbf{x}(j)_k \ 1]^T$ for each j^{th} angle input, $j \in \{1, 2, 3\}$. The results are plotted in the Figure 15.

From the figure it can be seen that while the MLFN was able to approximate sine for the 45° input used during training, it provided a poor approximation for the other two angles that were never used in training. Thus, the weights do not generalize well to other inputs.

It is important to note that the failure of the network to generalize in this case is not due to a failure of the EKF to properly train the weights of the MLFN. It is well known that two key aspects are important to achieve proper neural network training: (1) a sufficient number of neurons must be present in the network to provide adequate “neural capacity”, and (2) sufficient “richness” of input training patterns (angles in this case) must be supplied in order for the neural network to generalize well to values in between and on the boundaries of those used for training.

Another side of the EKF training algorithm must not be overlooked which pertains to the proper application of the EKF itself. In order for the EKF to estimate each weight and thus train the network the system must be completely observable. As discussed in Chapter 2, the system observability is assessed from the rank of the observability Grammian.

Due to the fact that the linearized measurement model varies in time, this system is LTV, and thus even if the system is observable, there is no guarantee the states will converge. Figures 16 - 18 display the weight estimate time history for the three weights trained in this example. Clearly, over the 600 second time interval none of

Figure 16: Example 1: Weight 1 convergence over 600 seconds.

Figure 17: Example 1: Weight 2 convergence over 600 seconds.

Figure 18: Example 1: Weight 3 convergence over 600 seconds.

Figure 19: Example 1: Observability Grammian for $N = 1$ and $r_q = 1$ indicates the system is observable, $n_q = 3$.

Figure 20: Example 1: Weight 1 convergence over 2000 seconds. $N = 1$, $W = 3$, $r_q = 2$.

Figure 21: Example 1: Weight 2 convergence over 2000 seconds. $N = 1$, $W = 3$, $r_q = 2$.

Figure 22: Example 1: Weight 3 convergence over 2000 seconds. $N = 1$, $W = 3$, $r_q = 2$.

the weights converge. An analysis of the system observability, shown in Figure 19, computes the observability Grammian at each time step and reveals that the rank of the observability Grammian is 3 (although with some noticeable drops in rank), and thus the system is observable.

Since the weight estimates in Figures 16 - 18 appear to keep diverging, re-running the simulation for a longer period may allow the EKF more time to drive the weight estimates closer to their correct values. The weight estimates for a training period of 2000 seconds are shown in Figures 20 - 22. Unfortunately, the estimate time histories do not provide a clear indication of convergence. In the next example, we investigate how additional measurements can help improve the convergence characteristics of the EKF-MLFN trainer.

Example 2: Single Neuron MLFN - Three Constant Input Angles

We now investigate the effect of increasing the number of measurements provided to the EKF at each time step. Since the MLFN is configured to take only two inputs at a time (an angle and bias), multiple measurements must be concatenated into a single vector (stacked) so that during time step k , $\mathbf{H}_k$ has rank equal to the number of states being estimated, $n_q = W$. To achieve this, the number of measurements at the k^{th} step must equal the number of states to be estimated. Therefore, $r_q = W$ separate inputs denoted as $\mathbf{x}(j)_k$ where $j \in \{1, 2, \dots, r_q\}$, must be provided to the MLFN to calculate the corresponding measurement output. Because the MLFN takes in only $M = 2$ inputs at each time step, each j^{th} input is supplied in turn to the MLFN such that the neural network input is $\mathbf{i}_k = [\mathbf{x}(j)_k \ 1]^T$ (i.e. $M = 2$) at each time step. Also, since this is occurring over a single time step, the MLFN uses the current set of estimated weights to transform the supplied inputs into the corresponding modelled measurement output. The resulting set of MLFN-generated measurement/function outputs is used to calculate the innovation error by taking the difference between these values and the corresponding values produced by the true measured observation/function output.

Additionally, the MLFN must be linearized about the current set of weights to determine the linearized measurement model matrix

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial g_{1,k}}{\partial w_{nm,k}^L} \\ \frac{\partial g_{2,k}}{\partial w_{nm,k}^L} \\ \vdots \\ \frac{\partial g_{r=W,k}}{\partial w_{nm,k}^L} \end{bmatrix}. \quad (56)$$

Returning now to the single neuron case, but now supplying three constant angle

Figure 23: Example 2: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{30^\circ, 45^\circ, 60^\circ\}$, $N = 1$, $W = 3$, $r_q = 3$.

Figure 24: Example 2: Weight 1 convergence over 600 seconds.

Figure 25: Example 2: Weight 2 convergence over 600 seconds.

Figure 26: Example 2: Weight 3 convergence over 600 seconds.

Figure 27: Example 2: Observability Grammian for $N = 1$ and $r_q = 3$ indicates the system is observable, $n_q = 3$.

Figure 28: Example 2: Proof-of-Match results for $\sin(\mathbf{x}_k)$, $\mathbf{x}_k = \{30^\circ, 45^\circ, 60^\circ\}$.

Figure 29: Example 2: Proof-of-Match results for $\sin(\mathbf{x}_k)$, $\mathbf{x}_k = \{10^\circ, 25^\circ, 50^\circ, 80^\circ\}$.

Figure 30: Example 2: Proof-of-Match results versus angle input for $\sin(\mathbf{x}_k)$, with $\mathbf{x}_k = \{0^\circ : 90^\circ\}$.

inputs to the neural network at each time step, $\mathbf{i}_k = [\mathbf{x}(j)_k \ 1]^T$ where $\mathbf{x}(j)_k = \{30^\circ, 45^\circ, 60^\circ\}$, it is now possible to train the MLFN to approximate three input angles instead of only single angle. Figures 23 - 26 display the MLFN approximations and weight convergence time histories.

Full observability is again obtained, as shown in Figure 27. The inclusion of the additional measurements has provided for a “stronger” observable system as the Grammian no longer drops rank and the weights indeed converge “decisively” and quickly. As a side benefit, the trained MLFN may also be able to provide a better approximation of the sine function since it was trained with two more inputs than before; that is, it may generalize better to other input values.

As with the previous example, it is important to ascertain if the final set of weights are indeed trained by performing POM on the original input and new input to check how well the trained network generalizes to new information. Figures 28 and 29 show the result of running POM with the original training inputs and a new set, $\mathbf{x}(j)_k = \{10^\circ, 25^\circ, 50^\circ, 80^\circ\}$. Clearly, the MLFN approximates the training input quite well but notable errors occur with the new input angles, particularly with $\mathbf{x}(j)_k = 10^\circ$. If we use the MLFN to plot a curve over a range of angles from $\mathbf{x}(j)_k = \{0^\circ : 90^\circ\}$ instead of time as shown in Figure 30, it becomes more clear where the current approximation fails to capture the true sine curve. And indeed, the actual value of $\sin(10^\circ)$ lies in the region where the MLFN curve diverges from the true one. To correct for these regions of divergence, more training examples over these areas would be required. This solution is investigated in the next example.

Example 3: Single Neuron MLFN - Multiple Constant Input Angles

In the last example it was shown how adding more measurements to the EKF can help improve the state convergence characteristics. This allows the EKF to train the weights such that they converge to steady-state values. By implication, adding

Figure 31: Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{30^\circ, 45^\circ, 60^\circ, 75^\circ\}$, $N = 1$, $W = 3$, $r_q = 4$.

Figure 32: Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{10^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ\}$, $N = 1$, $W = 3$, $r_q = 5$.

Figure 33: Example 3: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = \{0^\circ, 10^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ\}$, $N = 1$, $W = 3$, $r_q = 7$.

more measurements requires additional inputs to be provided to the MLFN at each time step which results in a better approximation of the desired function, in this case a sine curve. It was observed that in regions of input where the MLFN was not trained the network approximation could diverge from the true values. This normally occurs at the boundaries of the input space and thus it is necessary to provide a set of training inputs that span the entire input space that the system is expected to encounter during operation. It might be assumed that one can simply continue to stack measurements as was done in the previous example until the desired input space is adequately covered. Unfortunately this is not the case.

As it turns out, a single-neuron MLFN is limited to the number of inputs it can map to corresponding outputs. Thus, in providing the measurement model, the MLFN is limited by its “neural capacity” or the number of neurons in the hidden and output layers that provide the computational power of the network. Therefore, once

some particular number of inputs is exceeded, the MLFN's performance as a function approximator will be compromised.

Consider the neural network from the last example. Sine wave approximations are shown in Figures 31 - 33 for various numbers of training data. In Figure 31 four angle inputs are input to the MLFN sequentially at each time step and the neural network approximation continues to produce excellent results. However, when five inputs are input to the MLFN, the approximation begins to degrade as shown in Figure 32. Figure 33 shows further degradation with seven angle inputs.

These results demonstrate the need for sufficiently many neurons in the hidden layer in order for the MLFN to become a *universal function approximator* [46]. This result may seem counter-intuitive since EKF observability is satisfied. However, despite the fact that the EKF can train the $W = 3$ weights for this single neuron network, there simply are not enough neurons available to map the inputs into the desired outputs. As a result, the function approximation capability of the network begins to suffer as the number of inputs increases above the number of states being estimated. In the next example it will be demonstrated that by simply increasing the number of neurons the function approximation capability of the MLFN can be improved.

Example 4: Two Neuron MLFN - Multiple Constant Input Angles

Consider now a MLFN with two neurons, two inputs (the angle and the bias), and one output. By equation (33) the number of weights that we need to estimate is $W = 6$. This means up to six angle inputs could be cycled through the MLFN to produce a sine function estimate. First assume that we only have three inputs: $\mathbf{x}(\mathbf{j})_k = \{30^\circ, 45^\circ, 60^\circ\}$ such that the input to the MLFN at time step k is cycled through each j^{th} input so that $\mathbf{i}_k = [\mathbf{x}(j)_k \ 1]^T$. This is the same input configuration as Example 2 but now there are three extra weights due to the additional neuron. This is also similar

Figure 34: Example 4: MLFN approximation of $\sin(\mathbf{x}(j))$, $\mathbf{x}(j) = [30^\circ, 45^\circ, 60^\circ]$, $N = 2$, $W = 6$, $r_q = 3$.

Figure 35: Example 4: Proof-of-Match results for $\sin(\mathbf{x}(j)_k)$, $\mathbf{x}(j)_k = \{30^\circ, 45^\circ, 60^\circ\}$.

to Example 1 where there were more weights than angle inputs.

In the figures below are the results of training a MLFN with two neurons. As with Example 2, the MLFN provides an excellent approximation of the sine function for the three training inputs provided as shown in Figure 34. Performing POM, Figure 35, with the final set of weights confirms this conclusion.

Referring to Figures 36 - 41, some weights are obviously converging, while the rest seem to wander aimlessly. This behaviour can be attributed to the fact that for LTV systems, system observability does not guarantee convergence of the states.

Taking this example further, in order to improve the convergence time of the states, it is a simple matter of simply increasing the number of angles passed as input to the MLFN at each time step. Since there are six weights that need to be estimated, then six angles are required. Setting these angles to $\mathbf{x}(\mathbf{j})_k = \{0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ, 90^\circ\}$ so that a fairly equal span of the input space is covered, all weights should converge to values that model the sine function. Figure 42 shows the results of the function approximation during training and Figure 43 presents the POM results for the inputs $\mathbf{x}(\mathbf{j})_k = \{0^\circ, 10^\circ, 20^\circ, 30^\circ, 40^\circ, 50^\circ, 60^\circ, 70^\circ, 80^\circ, 90^\circ\}$. The six weights clearly converge as shown in Figures 44 - 49.

It was shown in the previous example how increasing the number of measurements above the number of states (weights) can have a negative effect on the ability the MLFN to approximate the desired function. This was due to the limited ability of a single neuron to map the large number of inputs into the desired output. In this example, the number of neurons was increased to two, allowing the MLFN to cope with an increased number of inputs. The end result is an observable training algorithm that, despite not being able to guarantee convergence due to the time-varying nature of the system, does indeed provide strong weight convergence characteristics as shown numerically, and also produces a better trained MLFN due to increased neural capacity that can handle a larger array of inputs.

Figure 36: Example 4: Slow Weight 1 convergence over 2000 seconds.

Figure 37: Example 4: Slow Weight 2 convergence over 2000 seconds.

Figure 38: Example 4: Slow Weight 3 convergence over 2000 seconds.

Figure 39: Example 4: Slow Weight 4 convergence over 2000 seconds.

Figure 40: Example 4: Slow Weight 5 convergence over 2000 seconds.

Figure 41: Example 4: Slow Weight 6 convergence over 2000 seconds.

Figure 42: Example 4: MLFN approximation of $\sin(x(j))$, $x(j) = \{0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ, 90^\circ\}$, $N = 2$, $W = 6$, $r_q = 6$.

Figure 43: Example 4: Proof-of-Match results for $\sin(x(j))$, $x(j) = \{0^\circ, 10^\circ, 20^\circ, 30^\circ, 40^\circ, 50^\circ, 60^\circ, 70^\circ, 80^\circ, 90^\circ\}$.

Figure 44: Example 4: Fast Weight 1 convergence over 600 seconds.

Figure 45: Example 4: Fast Weight 2 convergence over 600 seconds.

Figure 46: Example 4: Fast Weight 3 convergence over 600 seconds.

Figure 47: Example 4: Fast Weight 4 convergence over 600 seconds.

Figure 48: Example 4: Fast Weight 5 convergence over 600 seconds.

Figure 49: Example 4: Fast Weight 6 convergence over 600 seconds.

3.3.3 Algorithm Convergence and Approximation Capability

In Example 2, we saw how increasing the number of measurements at every time step k , so that $\text{rank}(\mathbf{H}_k) = n_q = W$, provides faster weight convergence. This indicates that the EKF training technique may be able to drive all weight estimates to converged values. This was achieved by stacking measurements such that $r_q = W$ separate MLFN inputs (in this case the inputs to the MLFN were the measurements of the EKF) were provided at each time step. Formally, in order to provide improved convergence characteristics to the EKF, set the minimum number of measurements equal to the number of states/neural network weights:

$$n_q = W = r_q. \quad (57)$$

As was demonstrated in Example 3, however, if too many measurements are taken at the k^{th} time step such that there are insufficient neurons to map the inputs to the desired outputs, then the MLFN will not behave as a universal function approximator [46]. The numerical investigation revealed that by matching the number of measurements to the number of weights in the state space, the MLFN universal function approximation capability was improved.

Thus, given a system with M inputs (j^{th} angle input and bias) and O outputs, the number of weights and corresponding identical number angle inputs can be determined from equation (33) for any selected number of MLFN neurons, N . For example, with $M = 2$, $O = 1$, and $N = 1$, the number of weights and measurements are

$$W = r_q = 1(2 + 1) = 3.$$

3.4 Adaptive EKF Using Neural Networks

In the work first published by Lobbia et al in [17] and later by Zhan and Wan in [61] a hybrid estimation algorithm is derived that provides for both analytically modelled system dynamics and adaptive nonparametric system dynamics to be integrated into a single online algorithm. By using an EKF to estimate the states of a typical analytically modelled nonlinear dynamic system, their contribution augments that model with a neural network to provide online learning of unmodelled and highly nonlinear dynamics. Thus, the neural network (a MLFN) allows the EKF dynamic model to adapt autonomously to changes in the actual plant or capture unmodelled plant components.

The real elegance of this technique comes from the fact that the EKF also provides the learning algorithm for the weights of the MLFN. Therefore, the weights of the network augment the state space of the system dynamics and the EKF estimates the weights for the MLFN while also estimating states of the plant. Of course, the usefulness of this approach depends acutely on the ability of the EKF to construct the plant states and MLFN weights from system output measurements; that is, the observability of the system.

The formulation of this algorithm is straightforward. We begin by simply adding to a known, analytic nonlinear plant model $f(\mathbf{q}_{k-1})$, the adaptive nonlinear MLFN output $g(\mathbf{q}_{k-1}, \mathbf{w}_{k-1})$, with plant states $\mathbf{q}_{k-1}$ and network weights $\mathbf{w}_{k-1}$.

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}) + g(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}). \quad (58)$$

Since the EKF will estimate the weights as well as the plant states, the state vector is augmented with a vector of neural network weights, and the dynamic model chosen for the weights is identity: that is, we assume that there exist a set of constant weights that can provide the function approximation capability to the MLFN, $g(\cdot)$,

that will result in the minimization of the EKF innovation error. The augmented system is thus

$$\bar{\mathbf{q}}_k = \begin{bmatrix} \mathbf{q}_k \\ \mathbf{w}_k \end{bmatrix} = \begin{bmatrix} f(\mathbf{q}_{k-1}) \\ \mathbf{0}_{W \times 1} \end{bmatrix} + \begin{bmatrix} g(\mathbf{q}_{k-1}, \mathbf{w}_{k-1}) \\ \mathbf{w}_{k-1} \end{bmatrix}, \quad (59)$$

where W is the number of weights and is used above to indicate the dimension of the state space.

Equation (59) requires that the general EKF algorithm also be augmented in order to accommodate the inclusion of the weights into the state space. The adjusted EKF algorithm becomes:

Prediction Step

$$\bar{\mathbf{F}}_k = \frac{\partial f}{\partial \bar{\mathbf{q}}} \big|_{q=\bar{\mathbf{q}}_{k-1}^+} + \frac{\partial g}{\partial \bar{\mathbf{q}}} \big|_{q=\bar{\mathbf{q}}_{k-1}^+}, \quad (60)$$

$$\bar{\mathbf{q}}_k^- = f(\bar{\mathbf{q}}_{k-1}^+, \mathbf{u}_{c,k}) + g(\bar{\mathbf{q}}_{k-1}^+, \mathbf{u}_{c,k}, \mathbf{w}_{k-1}^+), \quad (61)$$

$$\mathbf{P}_k^- = \bar{\mathbf{F}}_k \mathbf{P}_{k-1}^+ \bar{\mathbf{F}}_k^T + \mathbf{Q}. \quad (62)$$

Measurement Update Step

$$\bar{\mathbf{H}}_k = \frac{\partial h}{\partial \bar{\mathbf{q}}} \big|_{q=\bar{\mathbf{q}}_k^-}, \quad (63)$$

$$\mathbf{K}_k = \mathbf{P}_k^- \bar{\mathbf{H}}_k^T [\bar{\mathbf{H}}_k \mathbf{P}_k^- \bar{\mathbf{H}}_k^T + \mathbf{R}]^{-1}. \quad (64)$$

$$\bar{\mathbf{q}}_k^+ = \bar{\mathbf{q}}_k^- + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\bar{\mathbf{q}}}_k^-)), \quad (65)$$

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K}_k \bar{\mathbf{H}}_k \mathbf{P}_k^-. \quad (66)$$

As a consequence of the linearized measurement model, $\mathbf{H}_k$, not being a function of the MLFN weights, the derivative of the measurement model with respect to augmented states $\bar{\mathbf{q}}_k$ results in

$$\bar{\mathbf{H}}_k = [\mathbf{H}_k \quad \mathbf{0}_{r \times W}], \quad (67)$$

where $\mathbf{H}_k$ is the linearization of the measurement model with respect to the plant states only, and the derivatives of the same model with respect to the network weights results in the matrix of zeros with r_q rows and W columns. It should be noted that the dimensions of the estimate error covariance $\mathbf{P}_k$ and the process noise covariance $\mathbf{Q}$ will have square dimensions augmented to $n_q + W$ where there are n_q plant states and W MLFN weights. The dimension of the measurement covariance $\mathbf{R}$ depends only on the number of measurements and thus remains square at dimension r_q .

The above algorithm will be used as the foundation on which the slip-adaptive navigation technique is developed in the next chapter. In Chapter 2, the EKF was used to navigate by providing the localization estimate of the rover. In this chapter, the EKF was used to train the MLFN by providing the weight estimates. In Chapter 4, the two EKFs are combined via the above algorithm to perform both rover localization and MLFN training simultaneously.

Chapter 4

Adaptive Rover Navigation

In this chapter the main contribution of this thesis is presented. By using the JPL-Caltech slip model as slip truth, the slip-adaptive navigation algorithm is formulated by combining the EKF rover navigation algorithm developed in Chapter 2 with the hybrid EKF-MLFN filter presented at the end of Chapter 3. First, each of the component models including the simulated environment, rover, sensors, and slip are revisited to define additional elements. Then the models are assembled into the overall slip-adaptive framework for rover navigation, which is shown in Figure 50. Next, the system observability is determined and several simulations are presented that demonstrate the strengths and weaknesses of the slip-adaptive navigation technique. Finally, EKF-MLFN technique is validated against slip truth via MATLAB simulation and the issue of path independence is discussed.

4.1 Slip Model Formulation

In Chapter 2 the models for the simulated terrain environment, rover kinematic, and sensor models were developed for the case of rover navigation without accounting for the effects of slip that present significant terrain inputs (exogenous inputs) to the rover state estimate. In the sections that follow, these models are revisited in order

Figure 50: The EKF-MLFN slip-adaptive algorithm framework (for simulation).

to compensate for the effects of rover slip.

4.1.1 Environment Model

The simulated environment will continue to be a 2D plane randomly populated with features. Terrain slip will be modelled by creating regions within the environment associated with the longitudinal and lateral slip models developed by JPL and presented in Chapter 2. In reality, slip is a complex function of soil mechanics (loose rock, hard-pack rock, sand, and combinations) [39], terrain slope in the pitch and roll directions [2], and also the vehicle mass, velocity, and wheel design [35]. Since the rover mass and wheel design are fixed and its wheel input velocity can be controlled to some nominal constant value, slip will become independent of these parameters. To simplify further, it is assumed that the rover traverses only a single type of terrain and thus the slip experienced by the rover will only be a function of the terrain slope in the roll and pitch directions.

Although this thesis has discussed how slip occurs at the wheel-terrain interface, the simulated environmental slip model developed in Chapter 2 will be used to determine the slip experienced by the rover body (with respect to some central reference

point) as opposed to each individual wheel. This model is reproduced below in equation (68) which provides the slip values experienced in the rover-fixed local axis. The longitudinal slip (rover x -axis) is a function of the rover pitch inclination (ϕ) on the terrain. The lateral slip (rover y -axis) is a function of the rover roll angle (θ) on the terrain. While terrain features such as rocks and other obstacles are modelled as point entities in the environment for localization purposes, they are not modelled as obstacles over which the simulated rover must climb over or even avoid. This simplification avoids the need to account for the additional slip caused by such obstacles and also avoids pitch and roll measurement errors that occur when real rovers climb over such obstacles. It is also assumed that zero slip is experienced in the rover yaw direction, which in general is not the case.

$$\begin{bmatrix} Sx_t \\ Sy_t \\ S\psi_t \end{bmatrix} = \begin{bmatrix} -1.696E^{-06}\phi^6 - 2.402E^{-05}\phi^5 + 7.985E^{-04}\phi^4 + 1.805E^{-02}\phi^3 \dots \\ \dots - 1.471E^{-02}\phi^2 + 2.319E^{-01}\phi + 1.753 \\ 2.361E^{-04}\theta^4 + 2.238E^{-15}\theta^3 + 5.128E^{-02}\theta^2 - 4.873E^{-12}\theta + 1.471 \\ 0 \end{bmatrix}, \quad (68)$$

where

ϕ, θ = represent terrain inputs of pitch and roll, respectively;

Sx_t = longitudinal body CG slip truth along x -axis of the rover frame;

Sy_t = lateral body CG slip truth along y -axis of the rover frame;

$S\psi_t$ = angular body slip truth about z -axis of the rover frame.

The rover-frame slip truth components are then transformed to the global axis by means of the following transformation derived in Chapter 2:

$$\mathbf{S}_g = \begin{bmatrix} S_X \\ S_Y \\ S_\Psi \end{bmatrix} = \begin{bmatrix} Sx_t(c(\Psi)c(\phi)) + Sy_t(c(\Psi)s(\phi)s(\theta) - s(\Psi)c(\theta)) \\ Sx_t(s(\Psi)c(\phi)) + Sy_t(s(\Psi)s(\phi)s(\theta) + c(\Psi)c(\theta)) \\ S\psi_t c(\phi)c(\theta) \end{bmatrix}. \quad (69)$$

The simulation environment that was described in Chapter 2, is shown in Figure 4. While this environment is essentially a fictitious 2D landscape, the rover is permitted to pitch by ϕ degrees and roll by θ degrees about its CG. Thus the terrain provides these exogenous inputs, $\mathbf{u}_{e,k} = [\phi \ \theta]$, to the rover kinematics. Of course, in a real environment such terrain inputs would result in a change of altitude along the z -axis of the rover with respect to some specified datum plane. In this context, the environment model represented here could be referred to as *2.25D* since elevation is neglected but the rover is allowed to orient itself about the roll, pitch, and yaw axes. An example of the type of terrain inputs the rover would experience while traversing a series of rolling hills is depicted in Figure 5.

4.1.2 Rover Kinematic Model

The rover kinematics will continue to be modelled on a unicycle, where the rover states required to navigate the 2D environment are $\mathbf{q}_k = [X, Y, \Psi]^T$. These states provide the simulated rover the ability to stop and turn in place not unlike actual rovers such as Spirit and Opportunity. The first two states are the Cartesian positions with respect to some global coordinate system with the origin possibly located at the landing site. The last state represents the rover yaw angle measured about the z -axis. The nonlinear rover kinematics, repeated here for convenience, can be represented in discretized form as follows

$$\begin{bmatrix} X_k \\ Y_k \\ \Psi_k \end{bmatrix} = \begin{bmatrix} T \cos(\Psi_{k-1}) \cos(\phi_k) & 0 \\ T \sin(\Psi_{k-1}) \cos(\phi_k) & 0 \\ 0 & T \cos(\theta_k) \cos(\phi_k) \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \Psi_{k-1} \end{bmatrix}, \quad (70)$$

where

T = the time step in seconds.

Equation (70) provides the rover states at each time step. However, if there is slip at the wheel-terrain interface, then the rover kinematics model described above does not provide the entire picture with regard to the current rover states. An additional model is required in order to capture the slip component of the kinematics.

4.1.3 Slip Kinematic Model

The rover kinematic model presented above describes how the control inputs, the forward velocity and angular velocity of the rover, determine the resulting rover state. However, if the rover is operating in the presence of unmodelled slip, then these state estimates will diverge from the true values. The slip model developed by JPL, presented in Chapter 2, and included as part of the environmental model presented in section 4.1.1 is the *unknown* truth slip that remains unmodelled in the rover kinematic formulation derived in section 4.1.2. It is this unknown slip that we wish to learn online as the rover navigates the environment. Thus we require an *adaptive slip model* to empirically model online the true, unknown slip experienced by the rover in the simulated environment.

To complete the rover kinematic model shown in equation (70), an adaptive slip

model for the purpose of modelling the unknown slip and thus correcting the state estimates and allowing the rover to accurately localize itself and navigate the environment is required. For this purpose, a MLFN is used to capture the unmodelled slip in the rover body and apply a correction to the state estimates supplied by rover kinematic model of equation (70).

Figure 51: Single hidden layer feed-forward neural network for adaptive slip modelling.

The MLFN is shown in Figure 51. Based upon the same design discussed in Chapter 3, the neural network contains only a single hidden layer of N neurons. Thus the number of neurons in this layer can be adjusted until the desired performance parameters are met (i.e. maximum slip estimate quality versus minimum computational time, etc.).

As was shown in Chapter 2, slip is a function of the terrain geometry; specifically, the amount of slope being traversed in the pitch and roll directions. Therefore, these parameters are provided as inputs to the neural network assuming the availability of onboard sensors to provide rover pitch and roll angle measurements. This is similar to measuring the control inputs of the rover required by the rover kinematic model.

In that case, the control inputs are internal to the system. In the case of the terrain inputs, roll and pitch, they are exogenous or external inputs to the system but nevertheless also impact the resulting states of the rover. Since these inputs are measured, the signals will contain noise with corresponding noise statistics that must be accounted for when implementing the EKF.

The neural network output is the rover body slip percentage (the units of the slip truth model) at time step k in the rover x , y , and ψ directions:

$$\mathbf{S}_r(\phi_{k-1}, \theta_{k-1}, 1, \mathbf{w}_{k-1}) = [Sx_r \ Sy_r \ S\psi_r]^T. \quad (71)$$

Since the EKF rover states are in units of distance, a fractional term, $\frac{v_c T}{100}$, is necessary in order to convert the MLFN slip estimates in percent to those in units of distance (meters in this thesis). As an example, consider the slip in a single axis: since the definition of slip is $S = \frac{v_c - v_t}{v_c} \times 100$ for a known commanded velocity v_c and an unknown true velocity v_t , and given an estimate of S in percent, then the slip distance is

$$S_{dist} = \frac{v_c T}{100} S \quad (72)$$

$$= \frac{v_c T}{100} \left(\frac{v_c - v_t}{v_c} \times 100 \right), \quad (73)$$

$$= T(v_c - v_t). \quad (74)$$

The estimated slip terms must also be transformed into the global frame from the rover frame as shown in Chapter 2 and above in equation (69) for slip truth: $\mathbf{S}_g = \mathbf{R}_{XYZ}^{xyz} \mathbf{S}_r$. Thus, combining the rover kinematics of equation (70) with the neural network slip model, the complete kinematic model is

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}, \mathbf{u}_{c,k}) + \mathbf{q}_{k-1} - \mathbf{S}_g(\mathbf{u}_{e,k}), \quad (75)$$

where

$\mathbf{q}_k$ = the rover state at time k ;

$\mathbf{u}_{c,k}$ = control inputs, $[v \ \omega]^T$, forward and angular velocity, respectively;

$\mathbf{u}_{e,k}$ = terrain inputs, $[\phi \ \theta]^T$, pitch and roll angles, respectively;

$f(\cdot)$ = nonlinear rover kinematic model;

$\mathbf{S}_g$ = rover slip model as observed in the global frame.

In expanded form

$$\begin{bmatrix} X_k \\ Y_k \\ \Psi_k \end{bmatrix} = \begin{bmatrix} Tc(\Psi_{k-1})c(\phi_k) & 0 \\ Ts(\Psi_{k-1})c(\phi_k) & 0 \\ 0 & Tc(\theta_k)c(\phi_k) \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \Psi_{k-1} \end{bmatrix} - \quad (76)$$

$$\begin{bmatrix} \frac{v_k T}{100} \{ Sx_r c(\Psi_{k-1})c(\phi_k) + Sy_r (s(\theta_k)s(\phi_k)c(\Psi_{k-1}) - c(\theta_k)s(\Psi_{k-1})) \} \\ \frac{v_k T}{100} \{ Sx_r s(\Psi_{k-1})c(\phi_k) + Sy_r (s(\theta_k)s(\phi_k)s(\Psi_{k-1}) + c(\theta_k)c(\Psi_{k-1})) \} \\ S\psi_r Tc(\theta_k)c(\phi_k) \end{bmatrix}, \quad (77)$$

where

$s(\text{angle}), c(\text{angle})$ = represent sine and cosine, respectively;

T = the time step in seconds;

Sx_r = longitudinal body CG slip percent along x -axis of the rover frame;

Sy_r = lateral body CG slip percent along y -axis of the rover frame;

$S\psi_r$ = angular body slip percent about z -axis of the rover frame.

4.1.4 Measurement Model

In order to navigate and ultimately detect slip, the rover must have some means by which to sense any unique features that may be present in the environment. As already explained, the simulated environment is populated with point features that are uniquely identified by their location in a known map. Thus the mapping and correspondence problem are solved in advance since this thesis is concerned only with the ability of the rover to localize in the map using the neural network to estimate the slip being experienced.

This work assumes that the rover is equipped with a sensor that can measure range and bearing to each detected feature. By using Figure 6, a model for the range ($r_{f,i}$) and bearing ($\beta_{f,i}$) to each i^{th} detected feature is presented below in equation (78).

$$\mathbf{h}_{i,k} = \begin{bmatrix} r_{f,i} \\ \beta_{f,i} \end{bmatrix} = \begin{bmatrix} \sqrt{(X_{f,j} - X_k)^2 + (Y_{f,j} - Y_k)^2} \\ \arctan2\left(\frac{Y_{f,j} - Y_k}{X_{f,j} - X_k}\right) - \Psi_k \end{bmatrix} \quad (78)$$

where

$\mathbf{h}_{i,k}$ = is the i^{th} feature detected at time k ;

$(X_{f,i}, Y_{f,i})$ = are the global coordinates of the i^{th} feature;

(X_k, Y_k, Ψ_k) = are the rover states in the global frame at time k .

In addition to the laser range finder, an electronic compass is provided to determine the orientation, Ψ_k , of the rover directly. The measurement model provides

the expected values for rover orientation and the range and bearing of each detected feature. By subtraction from the actual measurements, the innovation error that is subsequently transformed by the Kalman gain into state estimate updates at each time step is produced. For m features at time k , the expected values are combined into a single vector in order to determine the innovation error.

$$\mathbf{h}_k = [\mathbf{h}_{1,k} \quad \dots \quad \mathbf{h}_{j,k} \quad \dots \quad \mathbf{h}_{m,k} \quad \dots \quad \Psi_k]^T. \quad (79)$$

4.1.5 Summary of Model Assumptions

In addition to the assumptions made in Chapter 2, this additional condition is made regarding the application of slip in the environmental simulation.

- Assuming vehicle mass, velocity, wheel design, and terrain soil mechanics are constant, slip is only a function of terrain geometry, pitch and roll, ϕ and θ , respectively. While this assumption is perfectly valid for the first three items in the list, the assumption regarding constant soil mechanics serves only to simplify the simulation of slip in this thesis. Unfortunately, as was discussed in the first two chapters, a rover will encounter a variety of rocky and sandy terrain profiles over the course of its lifetime. However, it is reasonable, in the candidate's view, that demonstration of the EKF-MLFN technique to successfully estimate nonlinear slip over single kind of terrain would necessarily indicate a high potential to do the same for other terrain types.

4.2 Adaptive EKF-NN Algorithm

The formulation of the slip adaptive navigation algorithm is developed next using the hybrid EKF-MLFN filtering framework presented in the last section of Chapter 3 and the models developed in the preceding sections of this chapter. This involves embedding the MLFN into the EKF algorithm and appending the neural network weights to the vector of system states requiring the EKF algorithm to estimate both the three rover states, $[X_k \ Y_k \ \Psi_k]^T$, and the vector of W neural network weights, $\mathbf{w}_k$, simultaneously.

4.2.1 State Space Augmentation

Augmenting the rover states with a vector of neural network weights results in $\mathbf{q}_k = [X_k \ Y_k \ \Psi_k \ \mathbf{w}_k^T]^T$. The system kinematics including slip is written as

$$\mathbf{q}_k = \begin{bmatrix} f(\mathbf{q}_{k-1}, \mathbf{u}_{c,k}) \\ \mathbf{0} \end{bmatrix} + \mathbf{q}_{k-1} - \begin{bmatrix} \mathbf{S}_g(\mathbf{u}_{e,k}, \mathbf{w}_{k-1}) \\ \mathbf{0} \end{bmatrix}. \quad (80)$$

In expanded form

$$\begin{bmatrix} X_k \\ Y_k \\ \Psi_k \\ \mathbf{w}_k \end{bmatrix} = \begin{bmatrix} Tc(\psi_{k-1})c(\phi_k) & 0 \\ Ts(\psi_{k-1})c(\phi_k) & 0 \\ 0 & Tc(\theta_k)c(\phi_k) \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \Psi_{k-1} \\ \mathbf{w}_{k-1} \end{bmatrix} - \quad (81)$$

$$\begin{bmatrix} \frac{v_k T}{100} \{ Sx_r c(\Psi_{k-1}) c(\phi_k) + Sy_r (s(\theta_k) s(\phi_k) c(\Psi_{k-1}) - c(\theta_k) s(\Psi_{k-1})) \} \\ \frac{v_k T}{100} \{ Sx_r s(\Psi_{k-1}) c(\phi_k) + Sy_r (s(\theta_k) s(\phi_k) s(\Psi_{k-1}) + c(\theta_k) c(\Psi_{k-1})) \} \\ S\psi_r T c(\theta_k) c(\phi_k) \\ \mathbf{0} \end{bmatrix}.$$

This is the same as equation (77) but with the state space augmented to include the dimensionality increase due to the inclusion of the neural network weights. Since the slip terms are estimated via the neural network, it is important to note that the slip terms are also functions of the neural network inputs. The MLFN can be represented by equation (71) and shown below for convenience:

$$\mathbf{S}_r(\phi_{k-1}, \theta_{k-1}, 1, \mathbf{w}_{k-1}) = [Sx_r \ Sy_r \ S\psi_r]^T. \quad (82)$$

4.2.2 Linerization

Since both the rover kinematics and the neural network represent nonlinear systems, they must be linearized with respect to the states in order to use the EKF algorithm. The linearized system can be represented as the sum of the linearized rover kinematics (without slip) and the linearized slip kinematics (the neural network slip model) as follows:

$$\mathbf{F}_k = \frac{\partial \mathbf{f}}{\partial \mathbf{q}_k} + \frac{\partial \mathbf{S}_r}{\partial \mathbf{q}_k}. \quad (83)$$

Expanding the first partial derivative of the rover kinematics results in

$$\frac{\partial \mathbf{f}}{\partial \mathbf{q}_k} = \begin{bmatrix} 1 & 0 & \frac{\partial f_1}{\partial \Psi_k} & \mathbf{0}_{1 \times W} \\ 0 & 1 & \frac{\partial f_2}{\partial \Psi_k} & \mathbf{0}_{1 \times W} \\ 0 & 0 & 1 & \mathbf{0}_{1 \times W} \\ \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times W} \end{bmatrix}, \quad (84)$$

where

$$\begin{aligned} \frac{\partial f_1}{\partial \Psi_k} &= -v_k T s(\Psi_k) c(\phi_k) + \frac{v_k T}{100} \left(S x_r s(\Psi_k) c(\phi_k) + S y_r (s(\Psi_k) s(\phi_k) s(\theta_k) + c(\Psi_k) c(\theta_k)) \right), \\ \frac{\partial f_2}{\partial \Psi_k} &= v_k T c(\Psi_k) c(\phi_k) - \frac{v_k T}{100} \left(S x_r c(\Psi_k) c(\phi_k) + S y_r (c(\Psi_k) s(\phi_k) s(\theta_k) - s(\Psi_k) c(\theta_k)) \right). \end{aligned}$$

Now expanding the second partial derivative of the neural network slip model,

$$\frac{\partial \mathbf{S}_r}{\partial \mathbf{q}_k} = \begin{bmatrix} \frac{\partial S x_r}{\partial x_k} & \frac{\partial S x_r}{\partial y_k} & \frac{\partial S x_r}{\partial \Psi_k} & \frac{\partial S x_r}{\partial \mathbf{w}_k} \\ \frac{\partial S y_r}{\partial x_k} & \frac{\partial S y_r}{\partial y_k} & \frac{\partial S y_r}{\partial \Psi_k} & \frac{\partial S y_r}{\partial \mathbf{w}_k} \\ \frac{\partial S \Psi_r}{\partial x_k} & \frac{\partial S \Psi_r}{\partial y_k} & \frac{\partial S \Psi_r}{\partial \Psi_k} & \frac{\partial S \Psi_r}{\partial \mathbf{w}_k} \\ \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{0}_{W \times 1} & \mathbf{I}_{W \times W} \end{bmatrix}, \quad (85)$$

where the partial derivatives of the neural network with respect to the rover states (the first three columns), $[\frac{\partial S_r}{\partial x_k}, \frac{\partial S_r}{\partial y_k}, \frac{\partial S_r}{\partial \Psi_k}]$, are zero since the network is not a function of these parameters. However, the neural network is a function of the weight states and therefore the partial derivatives (last column), $[\frac{\partial S x_r}{\partial \mathbf{w}_k}, \frac{\partial S y_r}{\partial \mathbf{w}_k}, \frac{\partial S \Psi_r}{\partial \mathbf{w}_k}]^T$ are determined from

Hidden Layer:

$$\begin{aligned}
 \frac{\partial Sx_r}{\partial \mathbf{w}_{nm\ k}^1} &= w_{X,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} i_m \\
 \frac{\partial Sy_r}{\partial \mathbf{w}_{nm\ k}^1} &= w_{Y,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} i_m \\
 \frac{\partial S\psi_r}{\partial \mathbf{w}_{nm\ k}^1} &= w_{\Psi,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} i_m
 \end{aligned} \tag{86}$$

Outer Layer:

$$\left[\begin{array}{ccc}
 \frac{\partial Sx_r}{\partial \mathbf{w}_{X,n\ k}^2} = \Phi_n^1 & \frac{\partial Sx_r}{\partial \mathbf{w}_{Y,n\ k}^2} = 0 & \frac{\partial Sx_r}{\partial \mathbf{w}_{\Psi,n\ k}^2} = 0 \\
 \frac{\partial Sy_r}{\partial \mathbf{w}_{X,n\ k}^2} = 0 & \frac{\partial Sy_r}{\partial \mathbf{w}_{Y,n\ k}^2} = \Phi_n^1 & \frac{\partial Sy_r}{\partial \mathbf{w}_{\Psi,n\ k}^2} = 0 \\
 \frac{\partial S\psi_r}{\partial \mathbf{w}_{X,n\ k}^2} = 0 & \frac{\partial S\psi_r}{\partial \mathbf{w}_{Y,n\ k}^2} = 0 & \frac{\partial S\psi_r}{\partial \mathbf{w}_{\Psi,n\ k}^2} = \Phi_n^1
 \end{array} \right] \tag{87}$$

where i_m is the m^{th} input, $\mathbf{i} = [\phi, \theta, 1]$ and $m \in \{1, 2, 3\}$.

As the derivative of the each weight with respect to themselves is one and with respect to any other weight is zero, the identity matrix, $\mathbf{I}_{W \times W}$, is formed as shown in equation (85).

The inputs to the system are comprised of the control inputs to the rover kinematics, $u_c = [v_k \ \omega_k]^T$, and the terrain geometry inputs to the neural network slip model, $u_e = [\phi_k \ \theta_k]^T$. Since these inputs are measured with external sensors, such as wheel encoders (control inputs) and inclinometers (terrain inputs), noise will be present on the signal and will be denoted by $[n_v, n_\omega, n_\phi, n_\theta]$. Assuming this noise is zero-mean Gaussian and white (essential since the EKF is formulated on this premise) with the variance of each signal denoted $[\sigma_v^2, \sigma_\omega^2, \sigma_\phi^2, \sigma_\theta^2]$, then the process noise matrix, $\mathbf{Q}$, is

$$\mathbf{Q} = \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_\omega^2 & 0 & 0 \\ 0 & 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & 0 & \sigma_\theta^2 \end{bmatrix}. \quad (88)$$

The transformation matrix, $\mathbf{L}_k$, is required to map the process noise matrix in input space to the process noise in state space. That is

$$\mathbf{Q}_s = \mathbf{L}_k \mathbf{Q} \mathbf{L}_k^T, \quad (89)$$

where $\mathbf{L}_k$ is defined as

$$\mathbf{L}_k = \begin{bmatrix} \frac{\partial f_1}{\partial n_v} & 0 & \frac{\partial f_1}{\partial n_\phi} & \frac{\partial f_1}{\partial n_\theta} \\ \frac{\partial f_2}{\partial n_v} & 0 & \frac{\partial f_2}{\partial n_\phi} & \frac{\partial f_2}{\partial n_\theta} \\ 0 & \frac{\partial f_3}{\partial n_\omega} & \frac{\partial f_3}{\partial n_\phi} & \frac{\partial f_3}{\partial n_\theta} \end{bmatrix}, \quad (90)$$

and where,

$$\begin{aligned} \frac{\partial f_1}{\partial n_v} &= Tc(\Psi_k)c(\phi_k) - \frac{T}{100} \left(Sx_r c(\Psi_k)c(\phi_k) + Sy_r (c(\Psi_k)s(\phi_k)s(\theta_k) - s(\Psi_k)c(\theta_k)) \right), \\ \frac{\partial f_2}{\partial n_v} &= Ts(\Psi_k)c(\phi_k) - \frac{T}{100} \left(Sx_r s(\Psi_k)c(\phi_k) + Sy_r (s(\Psi_k)s(\phi_k)s(\theta_k) + c(\Psi_k)c(\theta_k)) \right), \\ \frac{\partial f_3}{\partial n_\omega} &= Tc(\phi_k)c(\theta_k), \\ \frac{\partial f_1}{\partial n_\phi} &= -v_k Tc(\Psi_k)s(\phi_k) - \frac{vT}{100} \left(\frac{\partial Sx_r}{\partial n_\phi} c(\Psi_k)c(\phi_k) - Sx_r c(\Psi_k)s(\phi_k) - \frac{\partial Sy_r}{\partial n_\phi} s(\Psi_k)c(\theta_k) \dots \right. \\ &\quad \left. \dots + \frac{\partial Sy_r}{\partial n_\phi} c(\Psi_k)s(\phi_k)s(\theta_k) + Sy_r s(\Psi_k)c(\phi_k)s(\theta_k) \right), \\ \frac{\partial f_2}{\partial n_\phi} &= -v_k Ts(\Psi_k)s(\phi_k) - \frac{vT}{100} \left(\frac{\partial Sx_r}{\partial n_\phi} s(\Psi_k)c(\phi_k) - Sx_r s(\Psi_k)s(\phi_k) + \frac{\partial Sy_r}{\partial n_\phi} c(\Psi_k)c(\theta_k) + \right. \\ &\quad \left. \frac{\partial Sy_r}{\partial n_\phi} s(\Psi_k)s(\phi_k)s(\theta_k) + Sy_r s(\Psi_k)c(\phi_k)s(\theta_k) \right), \\ \frac{\partial f_3}{\partial n_\phi} &= -\omega_k Tc(\theta_k)s(\phi_k) - T \frac{\partial S\psi_r}{\partial n_\phi} c(\theta_k)c(\phi_k) + TS\psi_r c(\theta_k)s(\phi_k), \end{aligned}$$

$$\begin{aligned}
\frac{\partial f_1}{\partial n_\theta} &= -\frac{v_k T}{100} \left(\frac{\partial S x_r}{\partial n_\theta} c(\Psi_k) c(\phi_k) - \frac{\partial S y_r}{\partial n_\theta} s(\Psi_k) c(\theta_k) + S y_r s(\Psi_k) s(\theta_k) + \frac{\partial S y_r}{\partial n_\theta} c(\Psi_k) s(\phi_k) s(\theta_k) + \right. \\
&\quad \left. S y_r c(\Psi_k) s(\phi_k) c(\theta_k) \right), \\
\frac{\partial f_2}{\partial n_\theta} &= -\frac{v_k T}{100} \left(\frac{\partial S x_r}{\partial n_\theta} s(\Psi_k) c(\phi_k) + \frac{\partial S y_r}{\partial n_\theta} c(\Psi_k) c(\theta_k) - S y_r c(\Psi_k) s(\theta_k) + \frac{\partial S y_r}{\partial n_\theta} s(\Psi_k) s(\phi_k) s(\theta_k) + \right. \\
&\quad \left. S y_r s(\Psi_k) s(\phi_k) c(\theta_k) \right), \\
\frac{\partial f_3}{\partial n_\theta} &= -\omega_k T c(\phi_k) s(\theta_k) - T \frac{\partial S \psi_r}{\partial n_\theta} c(\phi_k) c(\theta_k) + T S \psi_r c(\phi) s(\theta_k).
\end{aligned}$$

The derivatives of the neural network with respect to the noise on the terrain inputs are determined from

$$\frac{\partial S x_r}{\partial n_\phi} = \sum_{n=1}^N w_{x,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\phi}^1, \quad (91)$$

$$\frac{\partial S y_r}{\partial n_\phi} = \sum_{n=1}^N w_{y,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\phi}^1, \quad (92)$$

$$\frac{\partial S \psi_r}{\partial n_\phi} = \sum_{n=1}^N w_{\psi,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\phi}^1, \quad (93)$$

$$\frac{\partial S x_r}{\partial n_\theta} = \sum_{n=1}^N w_{x,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\theta}^1, \quad (94)$$

$$\frac{\partial S y_r}{\partial n_\theta} = \sum_{n=1}^N w_{y,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\theta}^1, \quad (95)$$

$$\frac{\partial S \psi_r}{\partial n_\theta} = \sum_{n=1}^N w_{\psi,n}^2 \frac{\partial \Phi_n^1}{\partial v_n^1} w_{n,\theta}^1, \quad (96)$$

where the partial derivative of the neural network transfer function $\frac{\partial \Phi_n^1}{\partial v_n^1}$ in the hidden layer was defined in Chapter 3.

To demonstrate the slip-adaptive navigation algorithm constructed above several simulations will be presented in the next section. Example code written in the MATLAB programming environment is listed in Appendix A.

4.3 Slip-Adaptive Navigation Simulations

In the next sections, several simulations are performed that demonstrate the ability of the derived algorithm to localize the rover in the presence of unmodelled slip. The MLFN is shown to be able to capture the unmodelled slip components in the rover axes thus preventing the divergence of the rover state estimates of position and orientation in the 2.25D environment. An example of such divergence was presented in Example 2 of Chapter 2 where there was no slip-adaptive navigation algorithm implemented. In the next three sections, the tasks of MLFN training, proof-of-match (POM), and model validation are performed.

Training involves the process through which the MLFN “learns” the slip model online while experiencing a set of sensor inputs representative of the typical slip conditions present in the environment. POM is a quick check that compares the trained MLFN’s slip model to the actual slip (the JPL-produced MER slip model used here as slip truth). In actual practice, the true slip model is unknown and thus the POM step used here simply provides additional means to verify that the MLFN has been trained to approximate the rover slip.

The final step, model validation, is a practical way to validate the trained model. Validation is performed by presenting the rover with a new path over the terrain and requires the trained MLFN to predict the slip the rover experiences without any more additional training. The learned slip model is deemed validated if the rover can accurately navigate the environment with minimal localization errors.

4.3.1 Training the MLFN Slip-Adaptive Model

Within the context of MLFN training using an EKF to estimate the neural network weights, the issue of EKF observability will be investigated. It is also desirable to confirm that the weights of the network are indeed converging. In the following

Figure 52: The commanded and true path of the simulated rover during training.

simulations, the unicycle-based rover will be required to traverse terrain with the commanded trajectory shown in Figure 52.

The commanded path is the path the rover would travel in an environment with perfect traction. However, since there is slip present in the environment, the path actually traversed is different from the commanded one. This is called the true path. In a practical implementation, the rover navigation algorithm would form part of the control loop, providing the position and velocity reference signals to the controller in order to drive the rover towards the commanded path. Thus, in the control loop, the true path would approach and become identical to the commanded path. However, in this thesis, we are interested in investigating the localization capabilities of the slip-adaptive EKF-MLFN navigation algorithm. Obviously, if the algorithm can localize the rover in the presence of unmodelled slip, then it could also be used to provide

accurate reference signals to the rover navigational controller. Therefore, the slip-adaptive navigation algorithm will provide estimates of the true path and not the commanded one.

The commanded trajectory is defined by a time history of command inputs, forward and angular velocity defined in the control vector $\mathbf{u}_{c,k} = [v_k \ \omega_k]^T$, that are provided to the rover as shown in Figure 53. Meanwhile, the terrain is influencing the rover position via the exogenous inputs of pitch and roll, $\mathbf{u}_{e,k} = [\phi_k \ \theta_k]^T$, of which it has been shown that rover slip is a function. During training, the terrain input time history experienced by the rover is shown in Figure 54.

For the following simulations, the initial error covariance for the rover states are set to $\sigma_x^2 = \sigma_y^2 = 2^2 \text{ m}^2$, and $\sigma_\psi^2 = (10\frac{\pi}{180})^2 \text{ rad}^2$, and for MLFN weights, $\sigma_W^2 = 500$. The initial error covariance matrix is then

$$\mathbf{P}_0 = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 & \dots & 0 \\ 0 & \sigma_y^2 & 0 & 0 & \dots & 0 \\ 0 & 0 & \sigma_\psi^2 & 0 & \dots & 0 \\ 0 & 0 & 0 & \sigma_{W_1^1}^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \dots & \sigma_{W_{n_q}^2}^2 \end{bmatrix}, \quad (97)$$

which represents the level of uncertainty in the initial estimate of the rover states and weights. The process noise statistics used for the control and terrain input measurements are $\sigma_v = 0.04 \text{ m/s}$, $\sigma_\omega = 0.01 \text{ rad/s}$, $\sigma_\phi = 0.001 \text{ rad}$, $\sigma_\theta = 0.001 \text{ rad}$, and thus the process noise covariance matrix is defined as

Figure 53: Time history of training velocity control inputs.

Figure 54: Time history of exogenous terrain inputs from ground geometry.

$$\mathbf{Q} = \begin{bmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_\omega^2 & 0 & 0 \\ 0 & 0 & \sigma_\phi^2 & 0 \\ 0 & 0 & 0 & \sigma_\theta^2 \end{bmatrix}. \quad (98)$$

Finally, the measurement noise statics are selected to be $\sigma_r = 0.5$ m on range, $\sigma_\beta = 0.009$ rad on bearing, and $\sigma_\psi = 0.001$ rad on rover orientation. The measurement noise covariance for r_m detected features is

$$\mathbf{R}_k = \begin{bmatrix} \sigma_{r,1} & 0 & \dots & 0 & 0 & 0 \\ 0 & \sigma_{\beta,1} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{r,r_m} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\beta,r_m} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_\psi \end{bmatrix}. \quad (99)$$

Observability versus Approximation Capability

The linearized state space dynamics matrix, $\mathbf{F}_k$, formulated for slip-adaptive navigation in equation (83) is time-varying since the matrix is a function of the orientation state of the rover, Ψ_k , and also the weight states of the MLFN. Therefore, in order to determine if all of the rover states and neural network weights can be constructed from measurements, the observability Grammian presented in Chapter 2 must be full rank at each time step as shown below:

$$\mathbf{O}_{obs,k} = \sum_{j=0}^k \Phi_j^T \mathbf{H}_j^T \mathbf{H}_j \Phi_j, \quad (100)$$

$$\text{rank}(\mathbf{O}_{obs,k}) = n_{q,k}. \quad (101)$$

Beginning with a simple one neuron MLFN, with the measured exogenous terrain inputs of pitch and roll supplied to the neural network along with the bias input of one, $\mathbf{i}_k = [\phi_k \ \theta_k \ 1]^T$, and expecting three rover slip approximations from the MLFN, $[Sx_r \ Sy_r \ S\psi_r]^T$, we have $N = 1$, $M = 3$, and $O = 3$, and thus weights of number, as shown in Chapter 3,

$$W = N(M + O), \quad (102)$$

where here $W = 6$. These, along with the rover states, $\mathbf{q}_k = [X \ Y \ \Psi]^T$, form the state vector of dimension $n_q = 9$. The observability Grammian must therefore be of rank $n_{q,k} = 9$ for the EKF to be able to estimate the states. The simulation results are shown in Figures 55 - 58.

While full observability was achieved in this case, see Figure 55, since the MLFN has only one neuron to work with in order to approximate the nonlinear slip, poor results were obtained as shown in Figure 56. If the MLFN cannot model the slip, this translates into poor localization capability overall as the large localization errors demonstrate in Figure 57. The estimated path versus the true and commanded paths is shown in Figure 58.

This example demonstrates the effect of insufficient hidden layer neurons that was investigated in Chapter 3. In order to achieve a good function approximation capability of the MLFN, *enough* neurons must be available to perform computations. If the number of neurons were increased to $N = 10$, then it can be expected that the

Figure 55: The EKF achieved full observability for $N = 1$.

Figure 56: Poor MLFN-slip approximation results for $N = 1$.

Figure 57: Large localization errors in the rover state estimates for $N = 1$.

Figure 58: The slip-adaptive algorithm performs poorly with $N = 1$.

Table 1: Mean Squared Error versus Number of Neurons

Neurons	MSE X	MSE Y	MSE Ψ
1	289.6990	330.5289	0.000248
5	1.031618	1.076890	0.000000
10	0.307306	0.223598	0.000000
20	0.041706	0.030846	0.000000

MLFN-slip estimates should improve and the localization errors be reduced. However, as we increase the number of neurons, there is a corresponding increase in the number of weights to $W = 60$ by equation (102). Again, in order to construct all the weights and the three rover states, we must have $\text{rank}(\mathbf{O}_{obs,k}) = 63$. The results are shown in Figures 59 - 62.

The time-varying observability is shown in Figure 59 which again shows the system is observable. Additionally, the MLFN clearly has an increased capacity to capture the unmodelled slip, see Figure 60, and apply the correction to the state estimates resulting in reduced localization errors as shown in Figure 61. An overall improved slip-adaptive capability of the EKF-MLFN navigation algorithm results, as shown by comparing the true and estimated paths of the rover in Figure 62.

Performing one more example, we set $N = 20$ and thus $W = 120$, and accounting for the three rover states we require $\text{rank}(\mathbf{O}_{obs,k}) = 123$. The results are shown in Figures 63 - 66. Once again it is demonstrated that the EKF is observable in the LTV sense; see Figure 63. As shown in Figure 64, the MLFN exhibits improved performance as well. However, as can be seen by the state estimate errors shown in Figure 65, the localization estimates are not consistent as evidenced by the significant state errors occurring outside of the 3σ bounds. The reasons for this may be attributed to errors resulting from the linearization of the nonlinear system.

For simulations performed with $N = \{1, 5, 10, 20\}$, the mean squared error averaged over five runs each was determined for each estimated rover state and is shown

Figure 59: The EKF achieved full observability for $N = 10$.

Figure 60: Good MLFN-slip approximation results for $N = 10$.

Figure 61: Improved localization errors in the rover state estimates for $N = 10$.

Figure 62: The slip-adaptive algorithm performs well with $N = 10$.

Figure 63: The EKF achieved full observability for $N = 20$.

Figure 64: Excellent MLFN-slip approximation results for $N = 20$.

Figure 65: Improved localization errors in the rover state estimates for $N = 20$.

Figure 66: The slip-adaptive algorithm performs very well with $N = 20$.

in Table 1 against the corresponding number of neurons. This demonstrates that as the number of neurons increase, the MLFN's capability to capture the rover slip improves.

Convergence and Stability

The simulations that were performed in the previous section demonstrate that the slip-adaptive rover localization algorithm is observable. Unfortunately, whereas an observable LTI system guarantees the convergence of the states and thus the stability of the eigenvalues, this is not the case for LTV systems. As shown by Khalil [26], the eigenvalues of LTV systems no longer provide a sufficient indicator of the system's stability. Therefore, establishing observability here represents a first step in demonstrating the tractability of the slip-adaptive algorithm, but state convergence is in no wise guaranteed.

For the simplest case of one neuron and six corresponding neural network weights the observability Grammian indicated the observability of the EKF. Since this is an LTV system, this result provides an indication that the states can be estimated by the EKF. If this were an LTI system, this would imply that, through application of the Kalman gain, the EKF can drive all the states to converge to the correct values. This cannot be affirmed here, but instead we perform a numerical study to provide a sense of what is happening with regards to MLFN weight convergence. Figures 67 to 72 show the time history of weight values as they are estimated over time by the EKF. Indeed, all weights clearly converge to specific values. In section 4.3.3, we investigate whether the MLFN will converge to the same values over multiple simulations.

As was previously discussed, with only one neuron the MLFN performs poorly in approximating the rover slip due to a lack of neural capacity. However, by adding more neurons to the hidden layer, MLFN slip-approximation was improved, but it also becomes more apparent that the convergence of the states is not guaranteed.

Figure 67: 1 Neuron MLFN: Weight convergence for W_1 .

Figure 68: 1 Neuron MLFN: Weight convergence for W_2 .

Figure 69: 1 Neuron MLFN: Weight convergence for W_3 .

Figure 70: 1 Neuron MLFN: Weight convergence for W_4 .

Figure 71: 1 Neuron MLFN: Weight convergence for W_5 .

Figure 72: 1 Neuron MLFN: Weight convergence for W_6 .

Appendix C presents the weight time-histories of all 60 weights of an $N = 10$ neuron MLFN. They confirm the fact that not all weights converge, but also, none seem to represent a clear divergence either. Although an analysis done using, for example, Lyapunov methods might provide a means to determine the boundedness of the state equation solution [26], no such analysis is performed in this thesis and thus forms an area for future work. However, through numerous simulations it has been observed that the slip-adaptive EKF-MLFN algorithm has never resulted in an estimated path that has diverged from the true path.

4.3.2 Proof-of-Match

Proof-of-Match (POM) was performed on each of the simulations presented above: $N = \{1, 10, 20\}$. POM provides proof that the trained MLFN can indeed approximate the rover slip by presenting the MLFN, using the final set of EKF-estimated weights, with the full range of terrain inputs values, $\mathbf{u}_{e,k} = [\phi_k \ \theta_k]^T$, where $\phi_k = \{-20 : 20\}$ deg and $\theta = \{-20 : 20\}$ deg. The slip approximation produced is then compared to the actual true slip values produced by the JPL truth slip model adopted in this thesis. POM is performed in each rover axis (Ψ -axis is omitted since the estimate was for zero slip) and the results for the 1, 10, and 20 neuron MLFNs are displayed in Figures 73-74, 75-76, and 77-78, respectively. As expected, the match improves as the number of neurons increase. The figures also confirm that the EKF is able to train the MLFN weights to values that, when held constant, provide the MLFN with rover slip-modelling capability in the 10 and 20 neuron case.

In order to demonstrate the consistent slip-modelling performance of the slip-adaptive algorithm, the POM results from five subsequent simulated traverses are presented in Figures 79 - 80. The simulations were performed over the training path using the 20 neuron MLFN slip-adaptive model. In each case, it can be seen that the algorithm provides a consistent slip approximation when compared to the truth slip

Figure 73: 1 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis.

Figure 74: 1 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis.

Figure 75: 10 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis.

Figure 76: 10 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis.

Figure 77: 20 Neuron MLFN: Proof-of-Match for rover slip in the rover x -axis.

Figure 78: 20 Neuron MLFN: Proof-of-Match for rover slip in the rover y -axis.

Figure 79: 20 Neuron MLFN: Five-Run Proof-of-Match for rover slip in the rover x -axis.

Figure 80: 20 Neuron MLFN: Five-Run Proof-of-Match for rover slip in the rover y -axis.

model.

4.3.3 Slip Model Validation

The availability of the truth slip model allowed for the “proof-of-match” of the estimated slip model produced by the slip-adaptive EKF-MLFN. In practice, an exact model of true slip is not available or easily formulated (justifying the whole purpose of this thesis!), and instead the estimated model can be validated by requiring the rover to use the final set of estimated weights over a different path to estimate the slip¹. The concept of path independence is important as it is usually desired to demonstrate that not only can the rover traverse different paths with the same set of trained weights, but that those same weights can be obtained through training on different paths. This will be explored in this section.

Validation Over a New Path

To validate the EKF-MLFN slip model produced using the $N = 20$ neuron MLFN, a different path through similar terrain is selected as shown in Figure 81. This requires a different set of control inputs to be provided to the rover and, as a consequence, the new terrain geometry will provide a different set of pitch and roll exogenous inputs to the rover with a corresponding change in the slip the rover will experience during the traverse. As the terrain properties are the same as before, the same truth model for slip is used. Figures 82 and 83 display the control input and terrain input time history for slip model validation. For comparison, Figures 53 and 54 show the control and terrain inputs received by the rover during training.

As a check, validation is first performed over the original path used for training to ensure that at least the final set of weights can approximate the same slip the rover

¹The same terrain characteristics are assumed to be present. i.e. same type of soil, same range in geometric values of roll and pitch, etc.

Figure 81: The commanded and true path of the simulated rover during validation.

experienced during training. These results are presented in Appendix D. Next, using the new control inputs, the rover traverses the new path with the trained MLFN weights held constant. The simulation results are presented in Figures 84 and 85. The trained EKF-MLFN slip-adaptive algorithm clearly captured an approximate slip model during training which it was able to use to predict slip over the new path and thus provide an accurate estimate of the true rover location.

These results were obtained with the same process noise that was used in training: $\mathbf{Q} = \text{diag}([0.04^2, 0.01^2, 0.001^2, 0.001^2], 0)$ using the MATLAB diagonal matrix command. Despite these excellent MLFN-slip approximation results, examination of the EKF estimate errors for the rover states, shown in Figure 86, indicates inconsistent EKF state estimation. This is a consequence an imperfect slip model approximated by the MLFN as shown by the POM analysis of the previous section. Thus, despite the fact that this algorithm attempts to capture the unmodelled dynamics of

Figure 82: Time history of new path validation velocity control inputs.

Figure 83: Time history of exogenous terrain inputs over the new path.

Figure 84: MLFN-slip validation results over the new path before tuning.

Figure 85: The trained (not tuned) slip-adaptive algorithm provides an excellent slip approximation over the new path.

the system (slip), the adaptive component still fails to capture all of the analytically unmodelled dynamics.

In order to improve these results, the EKF gains can be “tuned” by increasing the filter process noise during validation. Increasing the process noise introduces more uncertainty into the EKF, forcing the the algorithm to rely more heavily on sensor measurements than on the rover kinematic model or the MLFN-slip model. The Kalman gains become larger as a result, which correspondingly increases the 3σ bounds on the rover state estimates.

The admittedly drastically-tuned process noise was set to $\mathbf{Q} = \text{diag}([10^2, 10^2, 10^2, 10^2], 0)$ in MATLAB, which resulted in the rover state estimate errors shown in Figure 87. Clearly, these results are now consistent as 99% of the state estimates fall within the 3σ bounds although the order of the error is about the same. The MLFN-slip estates and estimated path are shown in Figures 88 and 89, respectively.

Path Independence

A final component of the validation process is to prove that the EKF-MLFN slip-adaptive algorithm can always estimate the same set of weights, and thus the same slip-approximation model, regardless of the path traversed. This is called the *path independence* of the algorithm. Unfortunately, neural networks are naturally under-determined and therefore an infinite set of weights are available to approximate the same function [45].

Consider a MLFN with N neurons in the hidden layer and M inputs. As presented in Chapter 3, the inputs are multiplied by their corresponding weights and then summed before the sigmoid function is applied. In matrix form this is represented as

Figure 86: Validation rover state errors for the slip-adaptive algorithm using training process noise (not tuned).

Figure 87: Validation rover state errors for the tuned slip-adaptive algorithm.

Figure 88: Tuned MLFN-slip validation results over the new path.

Figure 89: The trained (and tuned) slip-adaptive algorithm provides an excellent slip approximation over the new path.

$$\begin{bmatrix} w_{11}^1 & w_{12}^1 & \dots & w_{1M}^1 \\ w_{21}^1 & w_{22}^1 & \dots & w_{2M}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^1 & w_{N2}^1 & \dots & w_{NM}^1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_M \end{bmatrix} = \begin{bmatrix} v_1^1 \\ v_2^1 \\ \vdots \\ v_M^1 \end{bmatrix}. \quad (103)$$

The system has $N \times M$ weights in the hidden layer to be determined and only N equations. Thus it is underdetermined in the general case² where $N \geq 1$, $M > 1$, and $N \neq M$. For example, given a set of known inputs, e.g. assume $\mathbf{i} = [1 \ 1 \ \dots \ 1]^T$, and also given the desired output for the first element, e.g. $v_1^1 = 3$, then the first row of the above expression reduces to

$$v_1^1 = 3 = w_{11}^1 + w_{12}^1 + \dots + w_{1M}^1. \quad (104)$$

Obviously there are an infinite set of values that can be chosen in order to accomplish this summation.

This result can be verified using the slip-adaptive EKF-MLFN. Consider the final set of weights shown in Tables 2 and 3 that were produced by two subsequent simulations of the rover traversing an identical path. These weights were estimated with only a single neuron $N = 1$ MLFN. However, as shown by the estimated weight values, even for the simplest and fully observable case, a different set of weights was produced each time over the same path.

Thus, the algorithm is not path independent and an infinite set of MLFN slip models exist.

²The specific case when $M = 1$ would represent the only scenario where the MLFN would be perfectly determined since N equations in N unknowns would be available.

EKF Weight Estimates for an $N = 1$ MLFN

Table 2: First Simulation

Layer	W_1	W_2	W_3
Hidden	372.255	-342.389	-44.257
Outer	144.726	151.158	0.00191

Table 3: Second Simulation

Layer	W_1	W_2	W_3
Hidden	-251.267	338.979	-87.240
Outer	102.1573	130.698	0.00799

Chapter 5

Conclusions

The problem proposed in this thesis was to develop a slip-adaptive navigation algorithm that could autonomously learn the unmodelled slip present in the environment online while traversing a commanded path. While certain shortcomings exist in the algorithm that was developed, it nevertheless achieves the objective.

The essential building blocks of the slip-adaptive navigation algorithm are the extended Kalman filter (EKF), the Multi-Layered Feedforward Neural Network (MLFN), and the hybrid EKF-MLFN algorithm that was developed by Lobbia, Stuberud, and Owen [17]. Their work was, in turn, based on the EKF trainer for MLFNs formulated by Singhal and Wu [55]. While both sets of research present the basic algorithm, neither goes beyond the assumption that the system is observable, and no further explanation or investigation is provided concerning this essential requisite for using the EKF to estimate states.

In this thesis, the observability of both the EKF trainer and the hybrid EKF-MLFN as applied to the slip-adaptive problem was brought to the forefront and determined at every step. Since convergence of observable linear time-varying (LTV) systems is not guaranteed, an investigation into improving the convergence characteristics of the EKF trainer developed by Singhal and Wu was provided by showing

the effect of increasing the number of measurements. Concerning the hybrid EKF-MLFN, it was found that, for this implementation, the LTV system of rover states and neural network weights is in fact observable. The MLFN thus provides excellent slip-approximation capability given enough neurons in the hidden layer.

One primary result demonstrated in this thesis was that all implementations of the slip-adaptive algorithm (where the number of neurons in the MLFN hidden layer was configured with $N = \{1, 10, 20\}$ neurons) were observable in the LTV sense. However, it was also numerically demonstrated through examples that observability does not guarantee weight convergence, in this case. Time history plots of the estimated weights confirmed that while some weights converged, others seemed to wander randomly, neither converging nor diverging.

Another result found verified the requirement for “sufficient” neurons in the MLFN hidden layer in order to behave as a universal function approximator. Thus, for the simplest case of $N = 1$ neuron, it was observed that despite observability, the MLFN did not have sufficient neurons in the hidden layer to approximate the slip. As the hidden layer neurons were increased to 10, and finally 20, the EKF-MLFN approximation of slip improved drastically while the system observability was maintained.

By using Proof-of-Match (POM), the MLFN-slip approximation was compared against slip truth in order to verify that the nonlinear slip was in fact learned by the MLFN. In addition, slip model validation was performed where the trained rover was commanded to traverse a new path using only the trained weights. Both POM and model validation indicated that the slip-adaptive navigation algorithm had succeeded (in the $N = 10$ and $N = 20$ cases) in learning the slip online in all three modelled rover degrees-of-freedom: $\{X, Y, \Psi\}$. However, due to the underdetermined nature of the MLFN, an infinite set of weights can be estimated by the EKF, even for the simple case of $N = 1$. This means that the algorithm is *path dependent* since an infinite number of slip models can be estimated and validated over any path.

5.1 Summary of Contributions

The contributions of this thesis include the following:

1. The development of a slip-adaptive navigation algorithm which uses a hybrid EKF-MLFN to model slip online while estimating the rover localization states. The primary advantage of this algorithm is that complex analytic slip models can be avoided and no simplifying assumptions like linearization of highly nonlinear dynamic components need to be made. With the empirical estimation approach, all aspects of the nonlinear dynamics are included.
2. Determination that this slip-adaptive algorithm is observable in the linear time-varying sense.
3. Validation of the slip-adaptive algorithm by means of Proof-of-Match and path traverse simulations.
4. Observation of the path dependent nature of algorithm due to the underdetermined MLFN system of equations. This provides for an infinite set of possible slip model solutions.

5.2 MLFN-Slip Modelling Recommendation

Some might argue that a major shortcoming of this algorithm is that, due to the underdetermined nature of the MLFN slip-adaptive model which presents infinite possible slip model solutions (i.e. an infinite number of weight combinations), this approach may be precluded from actual implementation onboard an autonomous robotic system. This reasoning is based on the the fact that a perfectly determined system would guarantee that only a single slip-model would be learned each time

which provides for a tractable way to demonstrate that the slip model parameters, in this case the MLFN weights, converge to the correct values every time.

On the case for using MLFNs, it might be argued that the important advantage of the underdetermined case is that there are infinite solutions available, although this thesis never rigorously proves that any of these will be found. The fact that the algorithm is observable, in the LTV sense, while noting convergence is by no means guaranteed, does provide an indication that a solution can be found from the infinite ones available.

Conversely, the case could be made to discard the MLFN altogether as more tractable methods exist for constructing slip-adaptive models. The truth slip model used in this thesis, for instance, was provided by a simple linear regression algorithm that used the empirical slip data provided by JPL-Caltech [2], and produced the polynomial function that approximated the slip in the longitudinal and lateral directions. These functions required twelve coefficients combined to model slip whereas the neural network required 120 to provide a good slip approximation. Thus, there exist simpler, more straight forward algorithms that can provide function approximation capability.

However, it should be noted that the JPL-Caltech polynomial slip model was generated off-line and then used to approximate MER slip as a static model. The advantage of the EKF-MLFN slip-adaptive approach developed in this thesis is that it can potentially be used online to adapt reactively to changing slip conditions. Thus, it is the candidates view that while MLFNs look promising, other methods for generating slip-adaptive models should be investigated.

5.3 Future Research Areas

Several areas for future research have been identified below that could provide solutions and improvements to the slip-adaptive algorithm developed in this thesis. These include:

- Investigation into the stability of the system. One may be able to study the boundedness of the state estimate by, for example, employing methods from stability theory [26].
- The underdetermined nature of the MLFN precludes its use as a practical model-adaptive technique since infinite solutions exist. Investigation into other techniques to provide adaptive-learning capability should be investigated such as receptive fields. As a side benefit, whereas the MLFN learns by providing global updates to all weights in the neural network slip model - precluding it from capturing local nonlinear discontinuities that may exist - receptive fields allow for more flexibility in the behaviour of the function being learned by permitting local variations.
- Even in its present form, the slip-adaptive algorithm should be experimentally validated using a real rover on Mars-like terrain in order to determine its practical viability when integrated with hardware systems such as the rover vehicle and sensors. In this scenario the accuracy of the algorithm could be assessed in terms of rover localization and slip-prediction, and also the real-time capability of the system could be assessed.
- Application of the unscented Kalman filter (UKF) to provide the rover state estimates and neural network learning rule would eliminate the need to linearize the nonlinear rover, sensor, or slip-approximation models as is the case with the EKF. Thus the system would not be subject to the restrictions of linear

observability and detectability (although potential nonlinear analogues probably exist).

- The development of a more general adaptive algorithm to capture general dynamic system changes such as rover wheel malfunctions or misalignments (as well as slip) would represent the ultimate form of adaptive autonomy. Such a system would require autonomous input-output correlation assessment in order to determine the nature of the general, analytically-unmodelled dynamics, that is empirically being modelled autonomously by the rover.

5.4 Final Remarks

The Mars Exploration Rovers have provided an invaluable example of the importance of slip-adaptive capability for rover navigation algorithms. As engineers and scientists continue to demand more autonomous capability from these robotic explorers, it is vital that such autonomous systems include the ability to adapt the entire system model to learn unmodelled dynamics arising from the environment, sensors, and the rover itself. This thesis has devoted its discussion primarily to a slip-adaptive navigation technique using the synergistic capabilities of an EKF rover localization algorithm and MLFN. However, there is no reason that this approach and others could not be developed in order to provide a broader range of adaptive behaviour. For example, while Visual Odometry had been developed and used primarily for slip measurement, as discussed in Chapter 1, it provided the mission-saving solution to correct for a malfunctioning wheel that otherwise would have terminated what has proven to be one of the most scientifically fruitful and inspiring examples of human endeavor.

References

- [1] J. A. Farrell and M. Barth, *The Global Positioning System & Inertial Navigation*. New York: McGraw-Hill, 1999.
- [2] R. A. Lindermann and C. J. Voorhees, “Mars exploration rover mobility assembly design, test and performance,” vol. 1, pp. 450–455, IEEE International Conference on Systems, Man and Cybernetics, October 2005.
- [3] E. Tunstel, “Operational performance metrics for mars exploration rovers,” *Journal of Field Robotics*, vol. 24, pp. 651–670, August/September 2007.
- [4] M. Maimone, Y. Cheng, and L. Matthies, “Two years of visual odometry on the mars exploration rovers,” *Journal of Field Robotics*, vol. 24, pp. 169–186, March 2007.
- [5] W. Travis and D. M. Bevy, “Navigation errors introduced by ground vehicle dynamics,” vol. 2005, (Long Beach, CA, USA), pp. 302–310, Proceedings of the 18th International Technical Meeting of the Satellite Division of the Institute of Navigation, ION GNSS 2005, September 2005.
- [6] J. J. Biesiadecki and M. W. Maimone, “The mars exploration rover surface mobility flight software: Driving ambition,” (Big Sky, Montana, USA), 2006 IEEE Aerospace Conference, March 2006.
- [7] M. Maimone, P. C. Ledger, and J. J. Biesiadecki, “Overview of the mars exploration rovers’ autonomous mobility and vision capabilities,” (Rome, Italy), IEEE International Conference on Robotics and Automation (ICRA) Space Robotics Workshop, April 2007.
- [8] “Mars exploration rover mission.” Website, Accessed March 2008. http://marsrovers.jpl.nasa.gov/technology/is_autonomous_mobility.html.

- [9] J. Hutton and B. Dowling, "Computer vision demonstration website: Corner detection." Website, Accessed July 2008. http://users.ecs.soton.ac.uk/msn/book/new_demo/corners/.
- [10] M. V. Winnendaal, P. Baglioni, and J. Vago, "Development of the ESA exomars rover," No. 603, (European Space Agency), Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS 2005, 2005. Special Publication.
- [11] R. Volpe, "Rover functional autonomy development for the mars mobile science laboratory," (Big Sky, Montana, USA), IEEE Aerospace Conference, March 2003.
- [12] D. Helmick, A. Angelova, M. Livianu, and L. Matthies, "Terrain adaptive navigation for mars rovers," pp. 1–11, IEEE Aerospace Conference, March 2007.
- [13] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Learning slip behavior using automatic mechanical supervision," IEEE International Conference on Robotics and Automation (ICRA), April 2007.
- [14] A. Angelova, L. Matthies, D. Helmick, and P. Perona, "Learning and prediction of slip using visual information," *Journal of Field Robotics*, vol. 24, pp. 205–231, March 2007.
- [15] A. Angelova, *Visual Prediction of Rover Slip: Learning Algorithms and Field Experiments*. PhD thesis, California Institute of Technology, 2008.
- [16] X. Lu, A. K. Jain, and D. Colbry, "Matching 2.5D face scans to 3D models," vol. 28(1), pp. 31–42, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006.
- [17] R. N. Lobbia, S. C. Stubberud, and M. W. Owen, "An adaptive extended kalman filter using artificial neural networks," *International Journal of Smart Engineering System Design*, vol. 1, no. 3, pp. 207–221, 1998.
- [18] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge: MIT Press, 2005.
- [19] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge: MIT Press, 2005.

- [20] L. A. McGee and S. F. Schmidt, "Discovery of the kalman filter as a practical tool for aerospace and industry." Website, Accessed July 2008. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860003843_1986003843.pdf.
- [21] R. F. Stengel, *Optimal Control and Estimation*. New York: Dover, 1994.
- [22] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering: A Practical Approach*. Reston: AIAA, 2005.
- [23] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximations," vol. 17, pp. 439–448, IEEE Transactions on Automatic Control, August 1972.
- [24] Q. Chen, *Approximate Kalman Filtering*. New Jersey: World Scientific, 1993.
- [25] K. Ogata, *Modern Control Engineering*. New Jersey, USA: Prentice Hall, fourth ed., 2001.
- [26] H. K. Khalil, *Nonlinear Systems*. New Jersey, USA: Prentice Hall, third ed., 2002.
- [27] I. Rusnak, A. Guez, and I. BarKana, "State observability and parameters identifiability of stochastic linear systems," pp. 846–850, First IEEE Regional Conference on Aerospace Control Systems, May 1993.
- [28] D. G. Lowe, "Object recognition from local scale-invariant features," vol. 2, pp. 1150–1157, Proceedings of the IEEE International Conference on Computer Vision, 1999.
- [29] J. Marshall, *MECH 5504 Guidance, Navigation, and Control for Mobile Robots*. MECH 5504 Course Notes, Carleton University, 2008.
- [30] A. Ellery, *An Introduction to Space Robotics*. Chichester, UK: Springer-Praxis, 2000.
- [31] I. Rekleitis, J.-L. Bedwani, S. Gemme, T. Lamarche, and E. Dupuis, "Terrain modelling for planetary exploration," pp. 243–249, IEEE Fourth Canadian Conference on Computer and Robot Vision, May 2007.
- [32] A. Deslauriers, C. English, C. Bennett, P. Iles, R. Taylor, and A. Montpool, "3D inspection for the shuttle return to flight." Website, Accessed July 2008. http://www.neptec.com/pdf/2006SPIE_3D_LCS_STS114.pdf.

- [33] I. Smith, R. Hudson, and L. Cournoyer, "Design of laser optics used in Neptec 3D laser camera system for low earth orbit environment." Website, Accessed July 2008. http://www.neptec.com/pdf/2003CANCOM_LCS.pdf.
- [34] S. Ruel, C. English, L. Melo, A. Berube, D. Aikman, A. Deslauriers, P. Church, and J. Meheux, "Field testing of a 3D automatic target recognition and pose estimation algorithm," vol. 5426, pp. 102–111, Proceedings of the SPIE - The International Society for Optical Engineering, Automatic Target Recognition XIV, 2004.
- [35] J. Y. Wong, *Theory of Ground Vehicles*. Toronto: John Wiley and Sons, 2001.
- [36] G. Andrade, F. B. Amar, P. Bidaud, and R. Chatila, "Modeling robot-soil interaction for planetary rover motion control," vol. 1, pp. 576–581, IEEE International Conference on Intelligent Robots and Systems, 1998.
- [37] J. Matijevic and et al., "Characterization of martian surface deposits by the mars pathfinder rover, sojourner," *Science*, vol. 278, pp. 1765–1768, December 1997.
- [38] R. Arvidson and et al., "Localization and physical property experiments conducted by opportunity at meridiani planum," *Science*, vol. 306, pp. 1730–1733, December 2004.
- [39] M. G. Bekker, *Introduction to Terrain-Vehicle Systems*. Michigan: University of Michigan Press, 1969.
- [40] C. C. Ward and K. Iagnemma, "Model-based wheel slip detection for outdoor mobile robots," (Rome, Italy), pp. 2724 – 2729, IEEE International Conference on Robotics and Automation, April 2007.
- [41] C. C. Ward and K. Iagnemma, "A dynamic model-based wheel slip detector for mobile robots on outdoor terrain," vol. 24, pp. 821–831, IEEE Transactions on Robotics, August 2008.
- [42] C. A. Brooks, K. D. Iagnemma, and S. Dubowsky, "Visual wheel sinkage measurement for planetary rover mobility characterization," *Autonomous Robots*, vol. 21, pp. 55–64, August 2006.
- [43] S. Schaal and C. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, pp. 2047–2084, November 1998.
- [44] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New Jersey: Prentice Hall, 1999.

- [45] K. Mehrotra, C. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Cambridge: MIT Press, 2000.
- [46] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [47] S. Gallant and H. White, "There exists a neural network that does not make avoidable mistakes," vol. 1, pp. 657–664, Proceedings of the IEEE International Conference on Neural Networks, July 1988.
- [48] R. Hecht-Nielsen, "Theory of the backpropagation neural network," (San Diego, California, USA), pp. 593–605, Proceedings of the International Joint Conference on Neural Networks, 1989.
- [49] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. New York: Spartan, 1961.
- [50] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.
- [51] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pp. 318–362, MIT Press, 1986.
- [52] S. Deng and Y. Hwang, "Solution of inverse heat conduction problems using kalman filter-enhanced bayesian back-propagation neural network data fusion," *International Journal of Heat and Mass Transfer*, vol. 50, pp. 2089–2100, June 2007.
- [53] J. Z. Sasiadek, "Sensor fusion," *Annual Reviews in Control*, vol. 26, pp. 203–228, 2002.
- [54] M. R. Mosavi, M. Mirzaeepour, and H. Nabavi, "Extrapolative model of DGPS corrections using a multilayered neural network based on the extended kalman filter," pp. 1–5, IEEE Conference on Cybernetics and Intelligent Systems, June 2006.
- [55] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended kalman algorithm," *Advances in Neural Information Processing System I*, pp. 133–140, 1989.

- [56] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended kalman filter," vol. 40, pp. 959–996, IEEE Transactions on Signal Processing, 1992.
- [57] D. W. Ruck, S. K. Rogers, M. Kabrisky, P. S. Maybeck, and M. E. Oxley, "Comparative analysis of backpropagation and the extended kalman filter for training multilayer perceptrons," vol. 14, pp. 686–691, IEEE Transactions on Pattern Analysis and Machine Intelligence, June 1992.
- [58] F. Heimes, "Extended kalman filter neural network training: Experimental results and algorithm improvements," vol. 2, pp. 1639–1644, IEEE International Conference on Systems, Man, and Cybernetics, 1998.
- [59] G. V. Puskorius and L. A. Feldkamp, "Practical considerations for kalman filter training of recurrent neural networks," vol. 3, pp. 1189–1195, IEEE International Conference on Neural Networks, 1993.
- [60] Y. Zhang, Y. Wu, W. Zhang, and Y. Zheng, "Using unscented kalman filter for training the minimal resource allocation neural network," vol. 3610 (Part I), pp. 8–14, Proceedings of ICNC First International Conference for Advances in Natural Computation, 2005.
- [61] R. Zhan and J. Wan, "Neural network-aided adaptive unscented kalman filter for nonlinear state estimation," vol. 13, pp. 445–448, IEEE Signal Processing Letters, 2006.
- [62] T. P. Crain, R. H. Bishop, and T. A. Ely, "Event detection and characterization during autonomous interplanetary navigation," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 2, pp. 394–403, 2002.
- [63] D.-J. Jwo, C.-S. Chang, and C.-H. Lin, "Neural network aided adaptive kalman filtering for gps applications," vol. 4, pp. 3686–3691, IEEE International Conference on Systems, Man and Cybernetics, 2004.
- [64] J. J. Wang, J. Wang, D. Sinclair, and L. Watts, "A neural network and kalman filter hybrid approach for gps/ins integration." Website, Accessed November 2008. http://www.gmat.unsw.edu.au/snap/publications/wangja_etal2006c.pdf.
- [65] J. Cao, J. Fang, and W. Sheng, "Neural network aided adaptive kalman filtering for gps applications," vol. 4, pp. 3686–3691, IEEE International Conference on Systems, Man and Cybernetics, 2006.

- [66] K. Kramer and S. C. Stubberud, "Tracking of multiple target types with a single neural extended kalman filter," pp. 463–468, Third International IEEE Conference on Intelligent Systems, 2006.
- [67] W. A. Fisher and H. E. Rauch, "Augmentation of an extended kalman filter with a neural network," vol. 2, pp. 1191–1196, IEEE International Conference on Neural Networks. IEEE World Congress on Computational Intelligence, 1994.
- [68] B. Francis, *ECE1619 Linear Systems*. ECE1619 Course Notes, University of Toronto, 2001.
- [69] C. C. Paige, "Properties of numerical algorithms related to computing controllability," vol. 26, pp. 130–138, IEEE Transactions on Automatic Control, February 1981.

Appendix A

Slip-Adaptive Navigation Algorithm: MATLAB Implementation

```

% roverNAV_slipAdaptive
% Created by Mark Swartz (100298379)
% DATE: 25 August 2008
%
% Generates a rover traverse simulation over terrain in the
% presence of unmodeled slip. The slip truth was taken from
% the MER slip models developed by Lindemann and Voorhees of
% JPL-Caltech.

clear all
clc
close all

testName = 'ch4_ex2_10N_60W_sin'; % test figure file name
saveFig = false; % flag to save a copy of generated figures
%% Simulation parameters

tic          % track program execution time
T = 0.1;     % time step

% INITIALIZE MULTILAYER FEEDFORWARD NEURAL NETWORK *****
% *****
x_j = 3;     % x_j = num_Inputs to NN
O = 3;       % num_Outputs (State corrections)
N = 1;       % num_Neurons

```

```

weightsInner = randn(N,x_j); % Hidden layer weight matrix
weightsOuter = randn(O,N); % Output layer weight matrix

num_Weights = N*x_j + O*N; % Total number of weights

% INITIALIZE EXTENDED KALMAN FILTER *****
% *****

% initial states
x_r0 = 0; y_r0 = 0; psi_r0 = 0;

% Noise statistics: standard deviation
n1 = 0.04; % Process noise on velocity signal
n2 = 0.01; % Process noise on angular velocity signal
n3 = 0.001; % Process noise on pitch (phi) signal
n4 = 0.001; % Process noise on roll (theta) signal

r1 = 0.5; % Measurement noise on range measurement
r2 = 0.009; % Measurement noise on bearing measurement
r3 = 0.001; % Measurement noise on psi (yaw) measurement

% Process noise error covariance
Q = diag([n1^2,n2^2,n3^2,n4^2],0);

% Initial EKF estimate error covariance
Po = diag([2^2,2^2,(10*pi/180)^2],0); % rover states
P_k_k = 500*eye(num_Weights+3); % NN weights

```

```

P_k_k(1:3,1:3) = Po;                                % combined cov

%% Control Sequence

% Forward and angular velocity control inputs
vel = 10;                                             % [m/s]
omega = 10*pi/180;                                   % [rad/s]

% Input map way-points, (x,y) coordinates, where the
% first set are the starting point and the last set
% the ending point. The "velocityProfile" m-function
% computes the n-1 paths to execute a forward drive
% manoeuvre at velocity "vel" and the n turning
% manoeuvres at angular velocity "omega" that target
% each way-point. n is the number of elements in the
% way-point vectors "xwayPoints" and "ywayPoints".

xwayPoints = [x_r0 100 100 200 200 300 600 1000 1250...
1500 1700 1800];
ywayPoints = [y_r0 0 100 100 200 300 300 450 450...
200 200 200];

[velArray,omegaArray] = velocityProfile(vel, omega, ...
xwayPoints, ywayPoints, T);

u1 = velArray;                                       % [m/s]   time-history
u2 = omegaArray;                                    % [rad/s]  time-history

```

```

t = 0:T:length(velArray)*T; % total simulation time [s]
t = t(1:end-1);

%% TERRAIN ENVIRONMENT MAP

mapSize = [2000;500]; % [m^2]
numFeatures = 500; % number of navigation beacons
arrayLength = length(velArray); % how long is the traverse

% OPTION A: Terrain inputs via step
phiSteps = [0 5 10 15 20 15 10 5 0 -5 -10 -15 -20 -15 -10 ...
-5 0 10 -5 15 5 -5 20 -15 -20 -5 10]*pi/180; % pitch angle
thetaSteps = [0 5 10 15 20 15 10 5 0 -5 -10 -15 -20 -15 -10 ...
-5 0 10 -5 15 5 -5 20 -15 -20 -5 10]*pi/180; % roll angle
[Features, phiArray, thetaArray, SLIP] = terrainProfile...
(mapSize,numFeatures, arrayLength, phiSteps, thetaSteps);

% OPTION B: Terrain inputs via sine
phiArray = 20*sin(0.07*T*(1:length(t)))*pi/180;
thetaArray = 20*sin(0.07*T*(1:length(t)))*pi/180;
[Features, SLIP] = terrainProfile2(mapSize, numFeatures, ...
phiArray, thetaArray);

X_f = Features(1,:); % map feature x-coordinates
Y_f = Features(2,:); % map feature y-coordinates

```

```
%% DYNAMIC MODEL - True Rover States
```

```
% Matrix Preallocation
```

```
x_r      = zeros(1,length(t)); % x-position  [m]
```

```
y_r      = zeros(1,length(t)); % y-position  [m]
```

```
psi_r    = zeros(1,length(t)); % orientation [rad]
```

```
X_globalLongSLIP = zeros(1,length(t));
```

```
Y_globalTransSLIP = zeros(1,length(t));
```

```
Psi_globalYawSLIP = zeros(1,length(t));
```

```
% commanded state time history
```

```
q_k_comd   = zeros(3,length(t));
```

```
% Initial States
```

```
x_r(1) = x_r0;
```

```
y_r(1) = y_r0;
```

```
psi_r(1) = psi_r0;
```

```
q_k_comd(:,1) = [x_r0; y_r0; psi_r0];
```

```
% True Rover States
```

```
for i = 2:length(t)
```

```
    phi = phiArray(i);
```

```
    theta = thetaArray(i);
```

```
% [m] slip in Global X-axis
```

```
X_globalLongSLIP(i) = -(u1(i)*T/100)*(SLIP(1,i)*...
```



```

cos(psi_r(i-1))*cos(phi) + SLIP(2,i)*(sin(theta)*...
sin(phi)*cos(psi_r(i-1)) - cos(theta)*sin(psi_r(i-1))) );

% [m] slip in Global Y-axis
Y_globalTransSLIP(i) = -(u1(i)*T/100)*(SLIP(1,i)*...
sin(psi_r(i-1))*cos(phi) + SLIP(2,i)*(sin(theta)*...
sin(phi)*sin(psi_r(i-1)) + cos(theta)*cos(psi_r(i-1))) );

% [rad] slip in global Psi-axis
Psi_globalYawSLIP(i) = -SLIP(3,i)*T*cos(phi)*cos(theta);

x_r(i) = u1(i)*T*cos(psi_r(i-1))*cos(phi) + x_r(i-1) + ...
X_globalLongSLIP(i);      % X-position [m]

y_r(i) = u1(i)*T*sin(psi_r(i-1))*cos(phi) + y_r(i-1) + ...
Y_globalTransSLIP(i);      % Y-position [m]

psi_r(i) = u2(i)*T*cos(phi)*cos(theta) + psi_r(i-1) + ...
Psi_globalYawSLIP(i);      % Orientation [rads]

q_k_comd(:,i) = [T*cos(q_k_comd(3,i-1))*cos(phi) 0; ...
                 T*sin(q_k_comd(3,i-1))*cos(phi) 0; ...
                 0 T*cos(phi)*cos(theta)]*[u1(i);u2(i)] ...
+ q_k_comd(:,i-1);

end

```

```

% Control Signals with Noise
vel      = u1 + randn(1, length(u1))*n1; % [m/s]
omega    = u2 + randn(1, length(u2))*n2; % [rad/s]
phi      = phiArray + randn(1,length(phiArray))*n3; % [rad]
theta    = thetaArray + randn(1,length(thetaArray))*n4; % [rad]

%% Extended Kalman Filter

% STEP 1: INITIALIZATION *****
% *****

% Initialize States
x_r_k_k = x_r0; % initial x-location
y_r_k_k = y_r0; % initial y-location
psi_r_k_k = psi_r0; % initial orientation

% Storage Variables for Plotting
q_r_plot = zeros(3+num_Weights,length(t));
SLIP_plot = zeros(3,length(t));
G = zeros(3,3+num_Weights,length(t));
B = zeros(3+num_Weights,2);

SP1 = zeros(1,length(t)); % 3-sigma error bound on x-locations
SP2 = zeros(1,length(t)); % 3-sigma error bound on y-locations
SP3 = zeros(1,length(t)); % 3-sigma error bound on orientation

```

```

% Initialize Observability Check Parameters
PHI = eye(num_Weights+3); % initial state transition matrix
N_obs = zeros(num_Weights+3); % initial observability Grammian
obs = zeros(1,length(t)); % Initialize observability vector
% *****

% SIMULATION
for i = 1:length(t)

% RUN Multilayer Feedforward Nerual Netwrok *****
% *****

    NN_input = [phi(i); theta(i); 1]; % Neural Network Input
    other_input = [psi_r_k_k; vel(i); T];

% Run the Mulit-layer Feedforward Neural Network
% OUTPUTS:  SLIP_rover =    [%] slip in rover frame
%           SLIP_global_w = [m] distance slipped
%           G =             derivs NN wrt weights
%           dg_dn =         derivs NN wrt inputs noise
    [SLIP_rover, q_w, G(:, :, i), dg_dn] = ...
    MLFN(NN_input, other_input, weightsInner, weightsOuter, N, 0);

% STEP 2: JACOBIANS *****
% *****

% Derivative of rover dynamics wrt yaw angle state (psi)

```

```

% *****

df1_dpsi = -vel(i)*T*sin(psi_r_k_k)*cos(phi(i)) + ...
(vel(i)*T/100)*( SLIP_rover(1)*cos(phi(i))*sin(psi_r_k_k) + ...
SLIP_rover(2)*( sin(theta(i))*sin(phi(i))*sin(psi_r_k_k) + ...
cos(theta(i))*cos(psi_r_k_k)) );

df2_dpsi =  vel(i)*T*cos(psi_r_k_k)*cos(phi(i)) - ...
(vel(i)*T/100)*( SLIP_rover(1)*cos(phi(i))*cos(psi_r_k_k) + ...
SLIP_rover(2)*( sin(theta(i))*sin(phi(i))*cos(psi_r_k_k) - ...
cos(theta(i))*sin(psi_r_k_k)) );

% Linearized dynamics
F = [1 0 df1_dpsi;
      0 1 df2_dpsi;
      0 0 1];

% Derivatives of rover dynamics wrt control/terrain input noise
% *****

% derivatives wrt noise on forward velocity (vel)
df1_dn1 = T*cos(psi_r_k_k)*cos(phi(i)) - ...
(T/100)*(SLIP_rover(1)*cos(psi_r_k_k)*cos(phi(i)) + ...
SLIP_rover(2)*(sin(theta(i))*sin(phi(i))*cos(psi_r_k_k) - ...
cos(theta(i))*sin(psi_r_k_k)) );

df2_dn1 = T*sin(psi_r_k_k)*cos(phi(i)) - ...
(T/100)*(SLIP_rover(1)*sin(psi_r_k_k)*cos(phi(i)) + ...

```

```
SLIP_rover(2)*(sin(theta(i))*sin(phi(i))*sin(psi_r_k_k) + ...
cos(theta(i))*cos(psi_r_k_k)) );
```

```
% derivatives wrt noise on angular velocity (omega)
df3_dn2 = T*cos(phi(i))*cos(theta(i));
```

```
% derivatives wrt noise on slope pitch angle (phi)
df1_dn3 = -vel(i)*T*cos(psi_r_k_k)*sin(phi(i)) - ...
(vel(i)*T/100)*(dg_dn(1,3)*cos(psi_r_k_k)*cos(phi(i)) - ...
SLIP_rover(1)*cos(psi_r_k_k)*sin(phi(i)) - ...
dg_dn(2,3)*sin(psi_r_k_k)*cos(theta(i)) + ...
dg_dn(2,3)*sin(theta(i))*sin(phi(i))*cos(psi_r_k_k) + ...
SLIP_rover(2)*sin(theta(i))*cos(phi(i))*cos(psi_r_k_k) );
```

```
df2_dn3 = -vel(i)*T*sin(psi_r_k_k)*sin(phi(i)) - ...
(vel(i)*T/100)*(dg_dn(1,3)*sin(psi_r_k_k)*cos(phi(i)) - ...
SLIP_rover(1)*sin(psi_r_k_k)*sin(phi(i)) + ...
dg_dn(2,3)*cos(psi_r_k_k)*cos(theta(i)) + ...
dg_dn(2,3)*sin(theta(i))*sin(phi(i))*sin(psi_r_k_k) + ...
SLIP_rover(2)*sin(theta(i))*cos(phi(i))*sin(psi_r_k_k) );
```

```
df3_dn3 = -omega(i)*T*cos(theta(i))*sin(phi(i)) - ...
T*dg_dn(3,3)*cos(theta(i))*cos(phi(i)) + ...
T*SLIP_rover(3)*cos(theta(i))*sin(phi(i));
```

```
% derivatives wrt noise on slope roll angle (theta)
df1_dn4 = -(vel(i)*T/100)*(dg_dn(1,4)*cos(phi(i)))...
```

```

*cos(psi_r_k_k) - dg_dn(2,4)*cos(theta(i))*...
sin(psi_r_k_k) + SLIP_rover(2)*sin(theta(i))*...
sin(psi_r_k_k) + dg_dn(2,4)*sin(theta(i))*...
sin(phi(i))*cos(psi_r_k_k) + SLIP_rover(2)*...
cos(theta(i))*sin(phi(i))*cos(psi_r_k_k) );

df2_dn4 = -(vel(i)*T/100)*(dg_dn(1,4)*...
cos(phi(i))*sin(psi_r_k_k) + dg_dn(2,4)*...
cos(theta(i))*cos(psi_r_k_k) - SLIP_rover(2)*...
sin(theta(i))*cos(psi_r_k_k) + dg_dn(2,4)*...
sin(theta(i))*sin(phi(i))*sin(psi_r_k_k) + ...
SLIP_rover(2)*cos(theta(i))*sin(phi(i))*sin(psi_r_k_k) );

df3_dn4 = -omega(i)*T*cos(phi(i))*sin(theta(i)) - ...
T*dg_dn(3,4)*cos(phi(i))*cos(theta(i)) + ...
T*SLIP_rover(3)*cos(phi(i))*sin(theta(i));

L = [df1_dn1      0      df1_dn3  df1_dn4;
      df2_dn1      0      df2_dn3  df2_dn4;
      0      df3_dn2      df3_dn3  df3_dn4 ];

% STEP 3: A PRIORI ESTIMATE *****
% *****

% [m] slip in Global X-axis

```

```

SLIP_MLFN_X    = -(vel(i)*T/100)*(SLIP_rover(1)*...
cos(psi_r_k_k)*cos(phi(i)) + SLIP_rover(2)*...
(sin(theta(i))*sin(phi(i))*cos(psi_r_k_k) - ...
cos(theta(i))*sin(psi_r_k_k)) );

% [m] slip in Global Y-axis
SLIP_MLFN_Y    = -(vel(i)*T/100)*(SLIP_rover(1)*...
sin(psi_r_k_k)*cos(phi(i)) + SLIP_rover(2)*...
(sin(theta(i))*sin(phi(i))*sin(psi_r_k_k) + ...
cos(theta(i))*cos(psi_r_k_k)) );

SLIP_MLFN_PSI = -SLIP_rover(3)*T*cos(phi(i))*cos(theta(i));

% Nonlinear dynamics model
q_k1_k = [vel(i)*T*cos(psi_r_k_k)*cos(phi(i)) ...
+ x_r_k_k + SLIP_MLFN_X;
          vel(i)*T*sin(psi_r_k_k)*cos(phi(i)) + ...
          y_r_k_k + SLIP_MLFN_Y;
          omega(i)*T*cos(phi(i))*cos(theta(i)) + ...
          psi_r_k_k + SLIP_MLFN_PSI];

% Combine MLFN SLIP approx into the EKF state estimate
% *****

% Augment states and linearized dynamics to include
% MLFN weights
[q_k, F_w, G_w, L_w] = ...

```

```

augment(q_k1_k, F, G(:, :, i), L, num_Weights, 0);

% Add the MLFN error approximation to the EKF estimates
F = F_w + G_w;
q_k1_k = q_k + q_w;
L = L_w;
% *****

P_k1_k = F*P_k_k*F' + L*Q*L';

x_r_hat = q_k1_k(1);
y_r_hat = q_k1_k(2);
psi_r_hat = q_k1_k(3);

% STEP 4: Get New Measurement *****
% *****
x_true = x_r(i); y_true = y_r(i); psi_true = psi_r(i);
Terrain = Features; % set coordinates of features
numFeatures = length(X_f); % total number of features
z = zeros(2*numFeatures,1); % initialize measurement signal
index = 1;
for j = 1:numFeatures
    z(2*j-1:2*j,1) = [sqrt( (X_f(j) - x_true)^2 + ...
        (Y_f(j) - y_true)^2 ) + randn*r1; ...
        atan2( (Y_f(j) - y_true), (X_f(j) - x_true) ) ...
        - psi_true + randn*r2];

```



```

% Check if features in sensor range/field of view
sensorRange = 150; % max range of laser
bearingRange = 90; % +/- viewable range sweep
if z(2*j-1,1) > sensorRange || abs(z(2*j,1)*180/pi) ...
> bearingRange
    removeMeasurement(2*index-1:2*index,1) ...
    = [2*j-1; 2*j];
    removeFeature(index) = (2*j)/2;
    index = index+1;
end
end

% Remove out of range measurements
z(removeMeasurement) = [];
if isempty(z)
    toc % end sim time-tracking
    error('No features in range/field of view for ...
    localization.\n Change driving commands or ...
    terrain model so features within %5.2f m of ...
    rover and within +/- %5.2f deg viewable range.', ...
    sensorRange, bearingRange)
end

% Psi (yaw - orientation measurement)
z(end+1) = psi_r(i) + randn*r3;

```

```

% Remove corresponding features for EKF estimation
Terrain(:,removeFeature) = []; % remove features
X_fr = Terrain(1,:);           % set of x-coord
Y_fr = Terrain(2,:);           % set of y-coord
numFeatures = length(X_fr);     % set new num

% Measurement error covariance
R = zeros(2*numFeatures+1);
for j = 1:numFeatures
    R(j*2-1:j*2,j*2-1:j*2) = [r1^2 0; 0 r2^2];
end
R(end,end) = r3^2;

% STEP 5: JACOBIANS *****
% *****

% H = Linearized sensor model
H = zeros(2*numFeatures+1,3);
for j = 1:numFeatures
    H(j*2-1:j*2,1:3) = [(x_r_hat - X_fr(j))/...
        sqrt( (X_fr(j) - x_r_hat)^2 + (Y_fr(j) - ...
        y_r_hat)^2 ), (y_r_hat - Y_fr(j))/...
        sqrt( (X_fr(j) - x_r_hat)^2 + ...
        (Y_fr(j) - y_r_hat)^2 ), 0;...
        (Y_fr(j) - y_r_hat)/((X_fr(j) - ...
        x_r_hat)^2+(Y_fr(j) - y_r_hat)^2 ),...
        -(X_fr(j) - x_r_hat)/...

```

```

        ( (X_fr(j) - x_r_hat)^2 + ...
        (Y_fr(j) - y_r_hat)^2 ) , -1 ];

end

H(end,1:end) = [0 0 1];

% Pad H with zeros - dim increase due to MLFN weights
H = [H zeros(size(H,1),num_Weights)];

M = eye(2*numFeatures+1);

% STEP 6: POSTERIORI ESTIMATE *****
% *****

% h = Nonlinear sensor model
h = zeros(2*numFeatures+1,1);
for j = 1:numFeatures
    h(2*j-1:2*j,1) = [sqrt( (X_fr(j) - ...
        x_r_hat)^2 + (Y_fr(j) - y_r_hat)^2 );...
        atan2( (Y_fr(j) - y_r_hat), ...
        (X_fr(j) - x_r_hat) ) - psi_r_hat];
end

h(end,end) = psi_r_hat;

v = z - h;

K = P_k1_k*H'*inv(H*P_k1_k*H' + M*R*M');

q_k1_k1 = q_k1_k + K*v;

```

```

P_k1_k1 = P_k1_k - K*H*P_k1_k;

% STEP 7: GET READY FOR NEXT ITERATION *****
% *****

% Pass for Next Iteration
x_r_k_k = q_k1_k1(1);
y_r_k_k = q_k1_k1(2);
psi_r_k_k = q_k1_k1(3);
P_k_k = P_k1_k1;

% Save for plotting
q_r_plot(:,i) = q_k1_k1;
SLIP_plot(:,i) = SLIP_rover;

% Extract the weight matrices from the state vector
[weightsInner,weightsOuter] = weightMatrix(q_k1_k1,N,0,x_j);

% Extract 3-sigma error bounds
P(:, :, i) = P_k1_k1(1:3,1:3);
SP1(i) = 3*sqrt( P_k1_k1(1,1) );
SP2(i) = 3*sqrt( P_k1_k1(2,2) );
SP3(i) = 3*sqrt( P_k1_k1(3,3) );

% LTV OBSERVABILITY Check of System
PHI = PHI*F;
N_obs = N_obs + (PHI'*H'*H*PHI); % Observability Grammian

```

```

tol = eps; % singular value (SV) < eps -> zero
obs(i) = rank(N_obs,tol); % uses SV to assess rank

% Positive Definiteness Check of P
[SS,posDef(i)] = chol(P_k1_k1);

end

toc % display execution time

% PLOT OUTPUT *****
% *****

% check positive definiteness of P
figure
plot(t, posDef, 'k')
title('Positive Definite Error Covariance...
      (posDef = 0)', 'FontSize',12)
xlabel('Time [s]', 'FontSize',12)
ylabel('posDef', 'FontSize',12)
axis([0 t(end) min(posDef)-1 max(posDef)+10])

% OBSERVABILITY MATRIX //////////////////////////////////////
% //////////////////////////////////////

% Required number of observable states
reqObsv = (3+num_Weights)*ones(1,length(t));

```

```

figure
plot(t, reqObsv, 'k', t, obs, 'r--')
title('Number of Observable States','FontSize',12)
xlabel('Time [s]','FontSize',12)
ylabel('Observability','FontSize',12)
axis([0 t(end) 0 max(reqObsv)+10])
legend('Required Observability',...
'Achieved Observability', 'location', 'Best')

if saveFig == true
    saveas(gcf, sprintf('%s_obs.fig',testName))
    print(gcf, sprintf('%s_obs.jpg',testName),'-djpeg')
end

% //////////////////////////////////////
% //////////////////////////////////////

% Neural Network Slip Prediction //////////////////////////////////
% //////////////////////////////////////

figure
subplot(3,1,1)
plot(t, SLIP(1,:), 'k', t, SLIP_plot(1,:), '-.r')
title('Neural Network Slip Estimate','FontSize',12)
ylabel('Rover x-Axis Slip [%]','FontSize',12)
axis([0 t(end) min(SLIP(1,:))-10 max(SLIP(1,:))+10])

subplot(3,1,2)

```

```

plot(t, SLIP(2,:), 'k', t, SLIP_plot(2,:), '-.r')
ylabel('Rover y-Axis Slip [%]','FontSize',12)
axis([0 t(end) min(SLIP(2,:))-10 max(SLIP(2,:))+10])

subplot(3,1,3)
plot(t, SLIP(3,:), 'k', t, SLIP_plot(3,:), '-.r')
ylabel('Rover \psi-Axis Slip [%]','FontSize',12)
xlabel('Time [s]','FontSize',12)
axis([0 t(end) min(SLIP(3,:))-1 max(SLIP(3,:))+1])
legend('Actual Rover x/y/\psi Slip',...
'Neural Network Slip Estimate', 'location', 'Best')

if saveFig == true
    saveas(gcf, sprintf('%s_NN.fig',testName))
    print(gcf, sprintf('%s_NN.jpg',testName),'-djpeg')
end

% ACTUAL vs. ESTIMATED STATES //////////////////////////////////
% //////////////////////////////////////
% STATE 1

figure
hold on

subplot(3,1,1);
plot(t, x_r, 'k', t, q_r_plot(1,:), '-.r')
ylabel('X [m]','FontSize',12)
title('Actual State vs. Estimated State','FontSize',12)

```

```

axis([0 t(end) min(q_r_plot(1,:)) max(q_r_plot(1,:))])

% STATE 2
subplot(3,1,2);
plot(t, y_r,'k', t, q_r_plot(2,:), '-.r')
ylabel('Y [m]', 'FontSize', 12)
axis([0 t(end) min(q_r_plot(2,:)) max(q_r_plot(2,:))])

% STATE 3
subplot(3,1,3);
plot(t, psi_r*180/pi,'k', t, q_r_plot(3,:)*180/pi, '-.r')
ylabel('\Psi [deg]', 'FontSize', 12)
xlabel('Time [s]', 'FontSize', 12)
axis([0 t(end) min(q_r_plot(3,:)*180/pi) ...
max(q_r_plot(3,:)*180/pi)])
legend('Actual', 'Estimate', 'location', 'Best')

if saveFig == true
    saveas(gcf, sprintf('%s_states.fig', testName))
    print(gcf, sprintf('%s_states.jpg', testName), '-djpeg')
end

% ESTIMATION ERRORS //////////////////////////////////
% //////////////////////////////////

% ERRORS - between actual state and EKF estimated state
x_r_error = x_r - q_r_plot(1,:);

```



```

y_r_error = y_r - q_r_plot(2,:);
psi_r_error = psi_r - q_r_plot(3,:);

% Mean Squared Error
mseX = mean(x_r_error.^2);
mseY = mean(y_r_error.^2);
msePsi = mean(psi_r_error.^2);
disp(sprintf('MSE X: %10.5f, MSE Y: %10.5f, ...
MSE Psi: %10.5f.',mseX,mseY,msePsi) );

% ERROR in state 1
figure
subplot(3,1,1)
plot(t, x_r_error, 'k', t, SP1, '--r', t, -SP1, '--r')
title('State Estimate Errors','FontSize',12)
ylabel('X Error [m]','FontSize',12)
axis([0 t(end) min(x_r_error) max(x_r_error)])

% ERROR in state 2
subplot(3,1,2)
plot(t, y_r_error, 'k', t, SP2, '--r', t, -SP2, '--r')
ylabel('Y Error [m]','FontSize',12)
axis([0 t(end) min(y_r_error) max(y_r_error)])

% ERROR in state 3
subplot(3,1,3)
plot(t, psi_r_error*180/pi, 'k', t, SP3*180/pi, ...

```

```

'--r', t, -SP3*180/pi, '--r')
xlabel('Time [s]', 'FontSize', 12)
ylabel('\Psi Error [deg]', 'FontSize', 12)
axis([0 t(end) min(psi_r_error*180/pi) ...
max(psi_r_error*180/pi)])
legend('Model Error', 'Predicted 3\sigma', ...
'location', 'Best')

if saveFig == true
    saveas(gcf, sprintf('%s_errors.fig', testName))
    print(gcf, sprintf('%s_errors.jpg', testName), '-djpeg')
end

% MAP //////////////////////////////////////
% //////////////////////////////////////

figure
hold on
plot(q_k_comd(1,:), q_k_comd(2,:), 'k')
plot(x_r, y_r, 'b:', q_r_plot(1,:), q_r_plot(2,:), ...
'r-.', X_f, Y_f, 'square', 'MarkerEdgeColor', 'k', ...
'MarkerFaceColor', 'g', 'MarkerSize', 5)

title('Simulated 2.25D Terrain', 'FontSize', 12)
xlabel('X Position [m]', 'FontSize', 12)
ylabel('Y Position [m]', 'FontSize', 12)
axis([0 mapSize(1) -mapSize(2) mapSize(2)])
legend('Commanded Path', 'True Path', 'Estimated Path', ...

```

```

'Map Features', 'location', 'Best')
hold off

if saveFig == true
    saveas(gcf, sprintf('%s_map.fig',testName))
    print(gcf, sprintf('%s_map.jpg',testName),'-djpeg')
end

% PLOT WEIGHT TIME HISTORY
for w = 4:size(q_r_plot,1)
    figure('Position',[440    343    685    449])
    axes('FontSize',20)
    plot(t, q_r_plot(w,:), 'k')

    title(sprintf('Weight Time History: W_{%1.0f}', ...
w-3),'FontSize',20);
    xlabel('Time [s]','FontSize',20)
    ylabel('Weight Value','FontSize',20)
    axis([0 t(end) min(q_r_plot(w,:)) max(q_r_plot(w,:))])
    axis 'auto y'

    if saveFig == true
        saveas(gcf, sprintf('%s_W%1.0f.fig',testName,w-3))
        print(gcf, sprintf('%s_W%1.0f.jpg',testName,w-3),...
'-djpeg')
    end
end

```

```
pause  
close(gcf);  
end
```

Appendix B

To Observe and Detect: Policing the Kalman Filter

The following presentation was adapted in its entirety from the lecture notes of Professor B. Francis of the University of Toronto [68].

Consider the system for which we desire to construct an observer:

$$\dot{x} = Fx + Gu, \quad x(0) = x_0 \quad (105)$$

$$y = Hx. \quad (106)$$

Before the Kalman filter can be constructed, it must be determined if the states, x , can be constructed from the available measurements, y . Another way to state this problem is to ask, does knowing $y(t)$ for $0 \leq t \leq t_f$, where t_f is the final time a measurement is taken, allow the construction of the initial state, x_0 [68]? This is known as the "observability" of the system.

B.1 Observability and Detectability

For any time, t , the output is

$$y(t) = He^{tF}x_0, \quad (107)$$

which defines a linear transform, $\mathbf{O}_{\text{obsv}}$, such that

$$y(t) = (\mathbf{O}_{\text{obsv}}x_0)(t). \quad (108)$$

The previous question becomes, does $\mathbf{O}_{\text{obsv}}$ uniquely determine x_0 ? There are a few ways in which one could answer this question, such as determining the rank of $\mathbf{O}_{\text{obsv}}$ (are the rows and columns linearly independent?), or alternatively one could determine if the equation

$$\mathbf{O}_{\text{obsv}}x_0 = 0 \quad (109)$$

has only the trivial solution, in which case we are looking for the null space of $\mathbf{O}_{\text{obsv}}$, or the set of x_0 that satisfy equation (109). We will use the latter approach here where the following expressions will be useful:

$$e^A := I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots, \quad (110)$$

$$A^n + c_{n-1}A^{n-1} + \dots + c_0I = 0, \quad (111)$$

where A is an square matrix, I is the identity matrix, and c are general coefficients. Equation (110) is the Taylor series expansion of the exponential of a matrix, while equation (111) is the Cayley-Hamilton theorem.

$$\begin{aligned} & \Rightarrow He^{tF}x_0 = 0, \quad 0 \leq t \leq t_f \\ \Rightarrow H \left[I + tF + \frac{t^2}{2!}F^2 + \frac{t^3}{3!}F^3 + \dots \right] x_0 &= 0 \end{aligned}$$

and thus

$$Hx_0 = 0, \quad HFx_0 = 0, \dots, HF^{n-1}x_0 = 0.$$

By Cayley-Hamilton $HF^n x_0 = 0$ and therefore equation (109) is true for all $t \geq 0$. The linear transform can now be explicitly formulated in terms of the known dynamics and measurement matrices as

$$\begin{bmatrix} H \\ HF \\ \vdots \\ HF^{n-1} \end{bmatrix} x_0 = 0, \quad (112)$$

and is called the observability matrix. If this matrix has full rank (i.e. $\text{rank}(\mathbf{O}_{\text{obsv}}) = n$) the initial state can be uniquely determined from knowledge of the output. However, if $\text{rank}(\mathbf{O}_{\text{obsv}}) < n$, then the initial state is not completely observable through knowledge of the output. In this case, the convergence of the Kalman filter will depend upon the stability of the unobservable part. i.e. where are the eigenvalues in the complex plane? This is known as the system detectability.

If the eigenvalues are in the left half of the complex plane, then the system is detectable (and thus stable) and the state estimates will converge automatically. But

since the eigenvalues cannot be placed where we want, the speed of convergence will be completely dependent on the dynamics of the unobservable component. If the eigenvalues fall anywhere else, the system is not detectable (and therefore unstable) and the Kalman filter state estimates cannot converge.

B.2 Decomposition of Observable and Unobservable Components

In order to determine the system detectability, the system must be decomposed into observable and unobservable parts. To accomplish this, we need to perform a change of basis for the system so that these parts become explicit. The change of basis will be completed using the linear time invariant transform, T , that operates on the state as follows

$$\bar{x} = T^{-1}x. \quad (113)$$

Then the system becomes

$$\begin{aligned} \Rightarrow \dot{x} &= Fx + Gu \\ \Rightarrow T\dot{\bar{x}} &= FT\bar{x} + Gu \\ \Rightarrow \dot{\bar{x}} &= T^{-1}FT\bar{x} + T^{-1}Gu \end{aligned} \quad (114)$$

To construct T , we use the $p < n$ vectors that form the basis of the null space of N - which is the observability matrix with $\text{rank}(O_{obsv}) = p$. This constitutes the unobservable part. Since the system (and thus the transformed system in equation (114)), is in $\mathbf{R}^n$, we are required to choose any $n - p$ linearly independent vectors to

complete the basis so that $\text{rank}(T) = n$.

As an example, suppose we have the following system

$$F = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & -2 & 0 \end{bmatrix}, \quad H = \begin{bmatrix} -1 & 0 & 1 & 0 \end{bmatrix},$$

where the observability matrix is

$$O_{obsv} = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 3 & 0 & -3 & 0 \\ 0 & 3 & 0 & -3 \end{bmatrix},$$

which has $\text{rank} = 2$. Then the basis for the null space of O_{obsv} can be expressed as $\{b_1, b_2\}$, where

$$b_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Then, choosing two more vectors, $\{b_3, b_4\}$ such that the entire set is linearly independent and thus forms a basis in $\mathbf{R}^n$, the transformation matrix can be constructed

as $T = [b_1 \ b_2 \ b_3 \ b_4]$, or

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}. \quad (115)$$

Applying the matrix as shown in equation (114), results in the decomposition of the system into the observable and unobservable parts:

$$T^{-1}FT = \begin{bmatrix} F_{\bar{o}} & F_{12} \\ 0 & F_o \end{bmatrix}. \quad (116)$$

In the new basis the system becomes decomposed into the form

$$\begin{bmatrix} \dot{\bar{x}}_1 \\ \dot{\bar{x}}_2 \end{bmatrix} = \begin{bmatrix} F_{\bar{o}} & F_{12} \\ 0 & F_o \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} + \begin{bmatrix} \bar{G}_{11} \\ \bar{G}_{21} \end{bmatrix} u(t), \quad (117)$$

$$y = \begin{bmatrix} 0 & \bar{H}_{12} \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix}, \quad (118)$$

where F_o forms the unobservable part. Constructing an observer for this system results in

$$\begin{bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} = \begin{bmatrix} F_{\bar{o}} & F_{12} \\ 0 & F_o \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} + \begin{bmatrix} \bar{G}_{11} \\ \bar{G}_{21} \end{bmatrix} u(t) + K \left[y - \begin{bmatrix} 0 & \bar{H}_{12} \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \end{bmatrix} \right] \quad (119)$$

for which the error dynamics, formed by taking the difference $\bar{e} = \hat{x} - \bar{x}$, are constructed below.

$$\dot{\bar{e}}_1 = F_{\bar{o}}e_1 + F_{12}e_2 \quad (120)$$

$$\dot{\bar{e}}_2 = F_o e_2 - K \bar{H}_2 e_2. \quad (121)$$

While (121) provides for $e_2 \rightarrow 0$ at $t \rightarrow t_f$ by means of the gain matrix, equation (120) shows that the e_1 errors will only converge provided the eigenvalues of $F_{\bar{o}}$ lie in the left half of the complex plain. That is, provided the dynamics of $F_{\bar{o}}$ are naturally stable. If this is the case, then the system, while not (completely) observable, is detectable.

Appendix C

Weight Convergence: 10 Neuron MLFN

The following MLFN weight time-histories of a 10 neuron - 60 weight MLFN slip-adaptive algorithm. Despite that the EKF is observable, it can be seen how some weights converge, while others seem to “wander”. Thus, for LTV systems, observability does not guarantee state convergence.

Figure 90: Weight convergence for W_1 .

Figure 93: Weight convergence for W_4 .

Figure 91: Weight convergence for W_2 .

Figure 94: Weight convergence for W_5 .

Figure 92: Weight convergence for W_3 .

Figure 95: Weight convergence for W_6 .

Figure 96: Weight convergence for W_7 .

Figure 99: Weight convergence for W_{10} .

Figure 97: Weight convergence for W_8 .

Figure 100: Weight convergence for W_{11} .

Figure 98: Weight convergence for W_9 .

Figure 101: Weight convergence for W_{12} .

Figure 102: Weight convergence for W_{13} . Figure 105: Weight convergence for W_{16} . Figure 103: Weight convergence for W_{14} . Figure 106: Weight convergence for W_{17} . Figure 104: Weight convergence for W_{15} . Figure 107: Weight convergence for W_{18} .

Figure 108: Weight convergence for W_{19} .

Figure 111: Weight convergence for W_{22} .

Figure 109: Weight convergence for W_{20} .

Figure 112: Weight convergence for W_{23} .

Figure 110: Weight convergence for W_{21} .

Figure 113: Weight convergence for W_{24} .

Figure 114: Weight convergence for W_{25} . Figure 117: Weight convergence for W_{28} . Figure 115: Weight convergence for W_{26} . Figure 118: Weight convergence for W_{29} . Figure 116: Weight convergence for W_{27} . Figure 119: Weight convergence for W_{30} .

Figure 120: Weight convergence for W_{31} . Figure 123: Weight convergence for W_{34} . Figure 121: Weight convergence for W_{32} . Figure 124: Weight convergence for W_{35} . Figure 122: Weight convergence for W_{33} . Figure 125: Weight convergence for W_{36} .

Figure 126: Weight convergence for W_{37} . Figure 129: Weight convergence for W_{40} . Figure 127: Weight convergence for W_{38} . Figure 130: Weight convergence for W_{41} . Figure 128: Weight convergence for W_{39} . Figure 131: Weight convergence for W_{42} .

Figure 132: Weight convergence for W_{43} .

Figure 135: Weight convergence for W_{46} .

Figure 133: Weight convergence for W_{44} .

Figure 136: Weight convergence for W_{47} .

Figure 134: Weight convergence for W_{45} .

Figure 137: Weight convergence for W_{48} .

Figure 138: Weight convergence for W_{49} .

Figure 141: Weight convergence for W_{52} .

Figure 139: Weight convergence for W_{50} .

Figure 142: Weight convergence for W_{53} .

Figure 140: Weight convergence for W_{51} .

Figure 143: Weight convergence for W_{54} .

Figure 144: Weight convergence for W_{55} .

Figure 147: Weight convergence for W_{58} .

Figure 145: Weight convergence for W_{56} .

Figure 148: Weight convergence for W_{59} .

Figure 146: Weight convergence for W_{57} .

Figure 149: Weight convergence for W_{60} .

Appendix D

Slip-Model Validation Results Over the Training Path

The following validation results were obtained by having the rover use the trained MLFN (with fixed weights) and follow the original training path in order to ensure that the MLFN-slip model could in fact approximate the slip that was used in training.

Figure 150: MLFN-slip validation results over the original training path.

Figure 151: The trained slip-adaptive algorithm provides an excellent slip approximation over the original path.

Appendix E

Computing Observability in MATLAB

For linear time-varying (LTV) discrete systems, the observability Grammian matrix can be computed from:

$$\mathbf{O}_{obs,k} = \sum_{j=0}^k \Phi_j^T \mathbf{H}_j^T \mathbf{H}_j \Phi_j,$$

where $\Phi_j = \Pi_{i=0}^j \mathbf{F}_i$. A system is observable if the initial state, $\mathbf{q}_0$, can be reconstructed from knowledge of the measurements, $\mathbf{y}_k$, for all time $k \in \{0 : k_f\}$. Since the Grammian is positive definite, once the rank the Grammian reaches the maximum possible for the system, it cannot lose rank.

The most reliable computational algorithm for computing the rank of a matrix uses singular value decomposition (SVD) as the singular values are insensitive to small perturbations [69]. The number of linearly independent rows or columns can be determined by the number of non-zero singular values (SVs) that result by performing SVD. Since the algorithm looks for “effectively” non-zero SVs — those with a value greater than the specified tolerance, any SV that falls below the tolerance is treated effectively as zero. This can result (and has resulted in this thesis) in an abrupt loss of rank.

Thus, while the Grammian cannot drop rank once it has been established, due to

the nature of the SVD rank-determination approach, the Grammian can lose rank if the tolerance is not set accordingly. The default tolerance used by MATLAB's rank function is computed according to

$$tol = \max(size(\mathbf{O}_{obs,k})) * eps(norm(\mathbf{O}_{obs,k})).$$

By using this tolerance algorithm in the slip-adaptive rover localization algorithm, the tolerance that was computed was found to be as high as 7.6875 at the final time step while the smallest SV was determined to be on the order of 10^{-4} . This resulted in a loss of rank of the observability Grammian as shown in Figure 152.

Fortunately, MATLAB provides the ability provide a user-defined tolerance to the rank function. In this thesis, the obvious choice was the smallest non-zero number that can be represented on a computer: in MATLAB this is denoted as $eps = 2^{-52}$ (the distance from 1.0 to the next largest double precision number). By using this tolerance, the numerical issue with the rank function discarding non-zero SVs is resolved and the Grammian can be determined as shown in Figure 153.

Figure 152: Erroneous loss of rank of the observability Grammian for an observable system.

Figure 153: Expected rank development of the observability Grammian for an observable system.