1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Vous le voyez, n'est-ce pas ?

Vous le voyez, n'est-ce pas ?

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Guidance and Control of an Autonomous Planetary Rover

by

David N. Green, B.A.Sc.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of

Master of Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering

Department of Mechanical and Aerospace Engineering

Carleton University

Ottawa, Ontario

November 1993

copyright ©1993 by David N. Green

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

335 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

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

ISBN 0-315-89904-2

Canada

Name GUIDANCE AND CONTROL OF AN AUTONOMOUS PLANETARY ROVER

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

SYSTEM SCIENCE

SUBJECT TERM

0790

U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0227
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical
Business Administration 0310
General 0272
Accounting 0770
Banking 0454
Management 0338
Marketing 0385
Canadian Studies
Economics 0501
General 0503
Agricultural 0505
Commerce-Business 0508
Finance 0509
History 0510
Labor 0511
Theory 0358
Folklore 0366
Geography 0351
Gerontology
History
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
General 0615
International Law and Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
General 0626
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0329
Ecology 0353
Entomology 0369
Genetic 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics
General 0786
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0477
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
General 0566
Audiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0554
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics
Physics 0605
General 0986
Acoustics
Astronomy and Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and High Energy 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering
General 0537
Aerospace 0538
Agricultural 0539
Automotive 0540
Biomedical 0541
Chemical 0542
Civil 0543
Electronics and Electrical 0544
Heat and Thermodynamics 0348
Hydraulic 0545
Industrial 0546
Marine 0547
Materials Science 0794
Mechanical 0548
Metallurgy 0743
Mining 0551
Nuclear 0552
Packaging 0549
Petroleum 0765
Sanitary and Municipal System Science 0554
System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned recommend to the Faculty of Graduate Studies and Research
the acceptance of the thesis entitled

"Guidance and Control of an Autonomous Planetary Rover"

submitted by David N. Green, B.A.Sc.
in partial fulfilment of the requirements for
the degree of Master of Engineering

Professor J.Z. Sawadek, Thesis Advisor

Dr. G.S. Vukovich, Thesis Co-Advisor

^{for} Chair, Mechanical and Aerospace Engineering

Carleton University
January 1994

Abstract

This thesis presents a set of algorithms for piloting an autonomous planetary rover along a planned path, performing real-time obstacle avoidance and improving the dead reckoning capability. Path-tracking is accomplished through linear regulation (feedback) of position and orientation errors, measured with respect to the planned path trajectory. Obstacle avoidance is accomplished by applying the concept of an artificial potential field to data that may be acquired using a scanning rangefinder. Dead reckoning is accomplished through the algorithmic fusion of odometry and inertial navigation data. Results from computer simulation illustrate the path-tracking and obstacle-avoidance capabilities, where the system model is that of a three-axle rover, with symmetric front and rear-axle wagon steering. Experimental data, acquired using a vehicle fitted with a wheel-encoder and an accelerometer, illustrates how the fusion algorithm mitigates the effects of wheel-slippage and integration-error.

Acknowledgements

I wish to thank Professor J.Z. Sasiadel for his guidance, support and encouragement throughout this work, and for allowing me to choose my own directions.

I would also like to thank Dr. G.S. Vukovich for the best "refresher" course I've ever had (I really needed it), and the Canadian Space Agency for their financial support.

Finally, I thank Marinette. I may have put up with her, but she put up with me.

Table of Contents

Abstract	iii
Acknowledgments	iv
Table of Contents	v
List of Figures	viii
List of Tables	x
Nomenclature	xi
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Thesis Outline	2
2. AN AUTONOMOUS PLANETARY EXPLORER	4
2.1 Introduction	4
2.2 Rover Design	4
2.3 Navigation	8
3. PATH TRACKING: A Linear Regulator	13
3.1 Introduction	13
3.2 Tracking Methodology	14
3.3 System Modelling and Steering Control	17
3.4 Tracking-Control Gain: The Linear Quadratic Regulator	22
3.5 Simulation of Path Tracking	27

4. OBSTACLE AVOIDANCE: Artificial Potential Fields	35
4.1 Introduction	35
4.2 Potential Fields	36
4.3 Potential Functions and Avoidance Error	37
4.4 Simulation of Obstacle Avoidance	42
5. DEAD RECKONING: Sensor Fusion	50
5.1 Introduction	50
5.2 Odometry and Inertial Navigation	52
5.3 Experimental Data	58
5.4 Signal Processing and Fusion	63
6. CONCLUSION	71
6.1 Summary of Results	71
6.2 Future Research	72
6.3 Summary of Contributions	73
References	74
Appendix A: Motion Resistance and Drive Power	80
Appendix B: Computer Simulation Code	83
B.1 Main Program	83
B.2 Path-Tracking Error	94
B.3 Obstacle-Avoidance Error	96
B.4 Graphical Output	101

Appendix C: Dynamic Equations of Motion	105
Appendix D: Computer Code for Signal Fusion	113
D.1 Differentiation	113
D.2 Integration	115
D.3 Fusion	117
Appendix E: Kalman Filtering	120
E.1 The Kalman Filter	120
E.2 A First-Order Model	122
E.3 An Experimental Example	124

List of Figures

Figure 2.1:	Top view of planetary rover	6
Figure 2.2:	Right side view of planetary rover	6
Figure 3.1:	Path tracking with reorientation at each planned point	15
Figure 3.2:	Path tracking with smooth transfer between segments	15
Figure 3.3:	Path tracking with interception of planned points	16
Figure 3.4:	Path-tracking by the minimization of tracking errors ϵ_d and ϵ_θ	16
Figure 3.5:	Tracking-error calculation geometry	18
Figure 3.6:	Block diagram of the path-tracking control system	21
Figure 3.7:	Typical open-loop step response to steering command input, γ_c . . .	21
Figure 3.8:	Closed-loop roots for three α_d, α_θ pairs with $\beta = 1$	26
Figure 3.9:	Closed-loop roots for three β 's with $\alpha_d = \alpha_\theta = 1$	26
Figure 3.10:	Tracking with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$	29
Figure 3.11:	Response signals with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$	29
Figure 3.12:	Tracking with $K_d = 2.24$, $K_\theta = 3.3$ and $K_\gamma = 0.59$	30
Figure 3.13:	Response signals with $K_d = 2.24$, $K_\theta = 3.3$ and $K_\gamma = 0.59$	30
Figure 3.14:	Tracking with $K_d = 0.45$, $K_\theta = 2.65$ and $K_\gamma = 0.5$	31
Figure 3.15:	Response signals with $K_d = 0.45$, $K_\theta = 2.65$ and $K_\gamma = 0.5$	31
Figure 3.16:	Tracking a nominal path, with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$.	33
Figure 3.17:	Nominal signals with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$	33
Figure 4.1:	Obstacle avoidance-error calculation geometry	39
Figure 4.2:	Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 3m$	43
Figure 4.3:	Avoidance of 2m diameter obstacle with $K_A = 1$ & $\rho_0 = 3m$	44
Figure 4.4:	Avoidance of 2m diameter obstacle with $K_A = 10$ & $\rho_0 = 3m$. . .	44
Figure 4.5:	Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 2m$	46

Figure 4.6:	Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 4\text{m}$	46
Figure 4.7:	Additional obstacle leeway, with $K_A = 5$ & $\rho_0 = 3\text{m}$	47
Figure 4.8:	Additional obstacle leeway, with $K_A = 5$ & $\rho_0 = 6\text{m}$	47
Figure 4.9:	Concurrent avoidance of obstacles, with $K_A = 5$ & $\rho_0 = 3\text{m}$	48
Figure 4.10:	Concurrent avoidance of obstacles, with $K_A = 10$ & $\rho_0 = 3\text{m}$	48
Figure 5.1:	Dual-wheel odometry using left and right encoder increments	54
Figure 5.2:	Inertial navigation (planar measurement) and inclination (tilt) sensors	57
Figure 5.3:	Encoder and INS position signals	60
Figure 5.4:	Encoder and INS velocity signals	60
Figure 5.5:	Encoder and INS acceleration signals	61
Figure 5.6:	Acceleration error $\epsilon = \text{ENCacc} - \text{INSacc}$	61
Figure 5.7:	Encoder and INS signal paths	62
Figure 5.8:	Encoder, INS and fused position signals	65
Figure 5.9:	Encoder, INS and fused velocity signals	65
Figure 5.10:	Additional Kalman filtering of the encoder-based signals	67
Figure 5.11:	INS and filtered encoder acceleration signals	68
Figure 5.12:	Acceleration error (ϵ) after filtering of encoder noise	68
Figure 5.13:	Encoder, INS and <i>filtered/fused</i> position signals	70
Figure 5.14:	Encoder, INS and <i>filtered/fused</i> velocity signals	70
Figure B.1:	Sample simulation output	85
Figure B.2:	Tracking error calculation geometry	94
Figure C.1:	Two-wheel model of six-wheel rover	105
Figure C.2:	Body reaction, ground force and drive torque on the wheels	108
Figure E.1:	Raw and Kalman-filtered acceleration signals	126
Figure E.2:	Response of Kalman filter versus signal frequency	126
Figure E.3:	Spectral content of unfiltered acceleration signal	128
Figure E.4:	Spectral content of Kalman-filtered acceleration signal	128

List of Tables

Table 2.1: General specifications for the rover model	8
Table 2.2: Navigation hierarchy for an autonomous planetary rover	12
Table 3.1: LQR control-law gains, for $U = 0.1\text{m/s}$, $L = 1.5\text{m}$ and $\tau = 3.5\text{s}$	25
Table A.1: Wheel torque and power requirements	82
Table B.1: Significant events during execution of the main simulation program	84
Table B.2: Simulation parameter file <i>simparms</i>	84
Table B.3: Planned path <i>madepath</i>	85
Table B.4: Obstacle description <i>blob05m</i>	85

Nomenclature

A	state space system matrix
APF	Artificial Potential Field
B	state space input matrix
b	width of wheel
C	state output vector gain matrix
c	rolling wheel circumference
D	diameter of wheel
ENCpos(k)	encoder position signal at moment k
ENCvel(k)	encoder velocity signal at moment k
ENCacc(k)	encoder acceleration signal at moment k
F_w	weight
f	frequency
f_b	bandwidth of test vehicle system
f_s	sample frequency
h	simulation time step interval
I_r	mass moment of inertia of the two-wheel rover model
I_w	mass moment of inertia of the two-wheel rover model wheel
IN	Inertial Navigation
INS	Inertial Navigation System
INSpos(k)	accelerometer position signal at moment k
INSvel(k)	accelerometer velocity signal at moment k
INSacc(k)	accelerometer acceleration signal at moment k
J	Linear Quadratic Regulator (LQR) performance index
K_A	avoidance error feedback gain

$\mathbf{K}_f$	Kalman filter gain matrix
$\mathbf{K}_{PSPi}$	avoidance error gains for PSP point i
$\mathbf{K}_s$	optimal feedback gain matrix
K_d	displacement error feedback gain
K_θ	orientation error feedback gain
K_γ	steering angle feedback gain
k	discrete time moment (sample) number
k_c	cohesive modulus of terrain deformation
k_ϕ	frictional modulus of terrain deformation
L	wheelbase length of rover (centre to front or rear axle)
LQR	Linear Quadratic Regulator
m_r	mass of rover model
m_w	mass of rover model wheel
n	exponent of terrain deformation
$\mathbf{P}$	solution matrix to the algebraic Riccati equation used in optimal estimation
P_C	location of centre point of rover
P_p	location of closest local point on planned path
PSP	Point Subjected to Potential
$\mathbf{Q}_1$	Linear Quadratic Estimator process noise covariance matrix
$\mathbf{Q}_2$	Linear Quadratic Estimator measurement noise covariance matrix
$\mathbf{Q}$	LQR weighting parameter matrix for minimization of system states
$\mathbf{R}$	LQR weighting parameter matrix for minimization of control inputs
R	radius of curvature in turning
R_c	motion resistance due to terrain compaction
$\mathbf{S}$	solution matrix to the algebraic Riccati equation used in optimal control
s	Laplace variable
$\text{Signal}_{in}(k)$	discrete filter input signal at moment k
$\text{Signal}_{out}(k)$	discrete filter output signal at moment k

T_s	sample interval time
U	forward speed of rover
$U(z)$	input variable in the z domain
$\mathbf{u}$	system input vector
$V(\rho)$	artificial potential
$v(k)$	fused velocity
W	axle length (distance between wheels on same axle)
$\mathbf{w}_1(k)$	process noise disturbance vector at moment k
$\mathbf{w}_2(k)$	measurement noise vector at moment k
w_{1rms}	root-mean-square value of process disturbance noise signal
w_{2rms}	root-mean-square value of measurement noise signal
X_k	X position of rover at discrete moment k
$\dot{X}_k$	velocity of rover in the X direction at discrete moment k
$X(z)$	state variable in z -domain
$\mathbf{x}$	state vector
x	x position in rover coordinates
$\dot{\mathbf{x}}$	derivative of state vector
$\dot{x}$	longitudinal velocity of rover
$\ddot{x}$	longitudinal acceleration of rover
$\hat{\mathbf{x}}(k)$	state estimate at moment k
$\bar{\mathbf{x}}(k)$	predicted state based on system model
$x_m(k)$	scalar measurement at moment k
Y_k	Y position of rover at discrete moment k
$\mathbf{y}(k)$	measurement vector at moment k
y	y position in rover coordinate frame
z	discrete z domain variable
z_0	wheel sinkage
α	maximum deceleration available to vehicle

α_d	LQR minimization weighting factor for lateral displacement error
α_θ	LQR minimization weighting factor for orientation error
α_γ	LQR minimization weighting factor for steering angle
β	LQR minimization weighting factor for steering angle command
γ	steering angle
$\dot{\gamma}$	steering rate
γ_C	steering command angle
$\gamma_{\max}$	maximum steering angle limit
$\dot{\gamma}_{\max}$	maximum steering rate limit
Δs	incremental distance
Δs_L	incremental distance measured by left wheel encoder
Δs_R	incremental distance measured by right wheel encoder
Δt	discrete time interval (sample time)
ΔX	incremental change in X-position of rover
Δx	incremental change in longitudinal position of rover
ΔY	incremental change in Y-position of rover
Δy	incremental change in lateral position of rover
$\Delta \theta$	incremental change in orientation
$\Delta \phi$	incremental rotation
ε	acceleration error
$\dot{\varepsilon}$	acceleration error rate
ε_A	total avoidance error
ε_d	displacement error
ε_θ	orientation error
$\dot{\varepsilon}_d$	displacement error rate
$\dot{\varepsilon}_\theta$	orientation error rate
$\varepsilon_{\max}$	maximum acceleration error set-point
$\dot{\varepsilon}_{\max}$	maximum acceleration error rate set-point

ε_{PSP}	avoidance error at a specific Point Subjected to Potential (PSP)
η	general potential field gain
θ	orientation angle of rover
$\dot{\theta}$	orientation error rate
θ_p	orientation angle of the local path tangent
θ_k	orientation of rover at discrete moment k
λ	bearing angle of an obstacle from forward direction of rover
λ_0	bearing limit angle
ρ	shortest distance from rover to obstacle surface
ρ_0	boundary distance of artificial potential field
T_1	torque on front wheel of two-wheel rover model
T_2	torque on rear wheel of two-wheel rover model
τ	steering time constant
ω_b	bandwidth
ω_1	front-wheel rotation speed for two-wheel model
ω_2	rear-wheel rotation speed for two-wheel model

1. INTRODUCTION

1.1 Motivation

The first application of an autonomous planetary rover may occur during future exploration of Mars, in what has been termed the Sample Return Mission [1,4,47]. This mission will include an orbiter, possibly outfitted with a surface imaging system, and a landing craft that will carry a robot explorer to the planet surface. Upon deployment, the robotic rover will travel, perhaps dozens of kilometres, to previously selected areas, collect soil, atmospheric or other specimens from the planet surface, and then return with these samples to the landing site. The successful return to Earth of an escape module, with the samples intact, will represent a proof-of-concept for future missions to Mars.

In the typical exploration scenario, the rover must travel semi-autonomously during each journey from one checkpoint to the next. That is to say, the robot explorer must operate fully autonomously over short segments of a planned route. This is due primarily to the difficulty in using direct tele-operation with a return-communication delay ranging between 6 and 45 minutes. Regardless of what form the vehicle and mission take, the rover will require local guidance and control by an on-board system.

Given that the planetary rover will receive information regarding where it is expected to go, prior to each period of complete autonomy, the automatic piloting system must be capable of following a pre-planned path. Secondly, given the unstructured and essentially uncharted terrain in which it will travel, the rover will undoubtedly encounter obstacles along the way. Should this occur, it would be advantageous for the pilot to be able to modify the planned path enough to bypass smaller obstacles that are not easily

cleared or overcome, avoiding the need to stop and wait for assistance. Finally, since externally referenced position identification may not be possible during periods of autonomous motion, then dead reckoning techniques must be employed by the pilot to determine where the rover is located with respect to the planned path and detected obstacles. Three sub-functions may therefore be defined (among others) for the planetary rover pilot; namely those of *path tracking*, *obstacle avoidance* and *dead reckoning*. The development of these automatic piloting systems has been the focus of this project.

1.2 Thesis Outline

This thesis consists primarily of the investigation of what are believed to be realistic approaches to accomplishing the path tracking, obstacles avoidance and dead reckoning tasks. The general characteristics of a planetary rover vehicle and its navigation strategy are considered, in an effort to define a realistic rover model, and to understand how the automatic pilot may fit into the overall navigation scheme. Although the pilot may be considered conceptually to be a single "entity" in this work, each of the path tracking, obstacle avoidance and dead reckoning sub-systems are dealt with as separate development problems, with their own background and literature. The following is a brief outline of the work to be presented.

Chapter 2 introduces proposed specifications for a planetary exploration vehicle. These specifications are used to define a rover that is used as a model in later chapters. This chapter also discusses characteristics of a hierarchical navigation system, within which may be seen the role of the piloting sub-systems developed in subsequent chapters.

Chapter 3 develops the path tracking system, by first introducing the tracking methodology, and then developing the equations of motion required to steer the rover along a pre-planned path, using a linear feedback control law. The linear quadratic

regulator principle is applied to choose the feedback gains for the steering control system, and the characteristics of the tracking system are illustrated graphically, using results obtained from computer simulation.

Chapter 4 develops the pilot's obstacle avoidance capability, by applying the concept of an artificial potential field to data which may be acquired from a rangefinding sensor. A new scalar quantity is developed, which is referred to as the avoidance error. This new error signal is used with the steering control law developed in Chapter 3, to give the pilot the ability to avoid unplanned obstacles, while at the same time tracking a planned path. The characteristics of the combined tracking and avoidance systems are illustrated graphically, using results from computer simulation.

Chapter 5 develops a dead reckoning technique that combines data from a wheel odometer and an accelerometer, improving the estimates of the position and velocity of a wheeled rover. The general use of odometry and inertial navigation is first discussed, with consideration of the mechanics and relationships required to perform the necessary calculations. Experimental data is used to illustrate the characteristics (deficiencies) of the two techniques, using a rotary wheel encoder and a linear accelerometer. A signal processing algorithm is then developed to combine the encoder and accelerometer signals, mitigating the errors encountered when using the data from either sensor alone. The characteristics of the "fused" signals are examined by re-processing the experimental data.

Chapter 6 concludes by summarizing the results of this work, as they pertain to: 1. use of a linear regulator for path tracking control; 2. use of artificial potential fields in obstacle avoidance; 3. use of signal fusion to improve dead reckoning. Recommendations are also made with respect to future investigation of each of these piloting functions. Finally, the contributions of this work are stated.

It is noted that portions of this thesis were presented earlier in [42,43,48].

2. AN AUTONOMOUS PLANETARY EXPLORER

2.1 Introduction

The primary considerations for an autonomous planetary rover's design include: the size and weight of the vehicle assembly during transport, the reliability of the rover's on-board systems and components, and the availability of power. Additionally, it has been proposed that computation and mobility may place the most significant constraints on the design of the rover [1]. In this work, a general consideration of the above points was used to choose a basic physical vehicle design and a functional structure for the navigation system. Section 2.2, below, describes the physical design of a rover, from which parameters were taken for use in the development and simulation of techniques and systems which are described in later chapters. Section 2.3 describes a navigation hierarchy, that is supported by an automatic pilot and its internal set of controllers. This pilot represents the location, from a systems point-of-view, of the guidance and control algorithms that will be developed later.

2.2 Rover Design

There are a number of general performance specifications, stated in the literature, which were used in choosing a rover design for use as a model in this work: The rover should be capable of climbing a 60% grade on a hard-packed surface [8], and a 35% grade in loose sand [3,12]. It should be capable of maintaining a nominal speed of about 10cm/s on level ground [1,4], and a reduced speed of perhaps 6cm/s on what will likely be loose sand or otherwise rugged terrain [3]. Further parameters may be found

elsewhere [11,13,23], however those mentioned above are generally within the ranges most often quoted. It is noted that parameters quoted in the literature are often chosen with consideration of the specialized abilities of a specific vehicle design.

The drive power will be limited by current technology, which includes generation by direct solar-electric conversion or use of a Radioisotope Thermoelectric Generator (RTG) [3,4,6,7]. Either or both of these technologies may be used in combination with an energy storage system. Nominal power expectations range from 50W to 1000W [6,7], depending on the rover size and the technology used, and might be expected to peak by a factor of 4.5, remaining at this level for up to 30 seconds [6]. The drive system will likely consist of electro-mechanical actuators, which will be the primary consumers of power during motion.

The mobility of the planetary rover should generally increase with the number of degrees of freedom and size of the vehicle, at least as far as ground clearance, vertical step and crevasse crossing are concerned. The rover's speed will be constrained on the upper side by navigation and power limitations and on the lower side by exploration expectations. The required mobility, power and interplanetary transportability of the rover vehicle and related equipment will place practical constraints on the final chosen design. A realistic martian robot rover might comprise a two- or three-segment articulated vehicle frame, that rides on parallel pairs of wheels, providing support for power, navigation, communication, computation, tool, cargo and analysis systems. The vehicle model that was chosen for this work, although based on a popular consensus in the literature, is notably similar to a planetary rover design proposed in [3]. Figures 2.1 and 2.2 contain top and side views of the vehicle to be described.

The vehicle design chosen for this work is that of an articulated, six-wheel, three-segment rover, that is 4m long, 2m wide and has a mass of 1000kg. It is comprised of in-line front, centre and rear chassis. These chassis or cabs are connected to one another

FIGURE 2.1: Top view of planetary rover.

FIGURE 2.2: Side view of planetary rover.

through revolute joints. The rear articulation allows for roll, and the front articulation allows for both roll and pitch. The vehicle utilizes a symmetric front and rear wagon steering arrangement, hence there is no need to allow for yaw between the cabs. Each cab rides on one axle and pair of wheels. The centre axle is fixed, while the front and rear axles are able to pivot horizontally under their respective cabs. All six wheels are independently driven, eliminating the need for a differential transmission. This configuration also eliminates the need for steering motors, however additional actuation may be useful in difficult terrain or to facilitate steering in the event of the failure of a wheel motor. The pitch joint should have an actuator or brake to facilitate crevasse crossing. This requirement may also hold for the front and rear roll joints as well, depending on the mass distribution within each cab. Table 2.1 contains a summary of the joint and actuator requirements, as well as a set of general performance specifications.

To put the above design and performance specifications into perspective, it is noted that the rover will have a maximum power-to-weight ratio of 0.5W/kg , and that the ratio for a typical road car is on the order of 100W/kg . Furthermore, it is noted that at a speed of 10cm/s (0.36km/h), and a maximum steering angle of 30° (turning radius = 2.6m), the total lateral thrust that must be supported by the combined traction of the six rover wheels would be less than 4N . Comparing this value to an estimated minimum longitudinal motion resistance of about 438N , it seems that the rover should not be significantly affected by inertial forces. Appendix A contains a brief analysis of the motion resistance forces that may be encountered by the rover, as well as estimations of the required drive power and motor sizes. Finally, although the effect of inertial-force-induced skidding should be negligible, there will still be (static-force-induced) wheel slippage in loose sand or otherwise rugged terrain. The effects of such slippage will be discussed in Chapter 5, where data from inertial navigation sensors is shown to mitigate the effects of significant intermittent wheel slippage on position and velocity estimation.

TABLE 2.1: General specifications for the rover model.

Length:	4m
Width:	2m
Mass:	1000kg
Wheel diameter:	1m
Turning radius:	2.6m
Step height crossing:	1m
Crevasse crossing:	1.5m
Hard packed grade:	60% (30° slope)
Loose sand grade:	35% (20° slope)
Speed in loose sand:	10cm/s
35% grade in sand:	5cm/s
Acceleration:	10cm/s ²
Total power:	250-500W
Wheels:	6 motors
Front pitch joint:	1 actuator or brake, $\pm 30^\circ$
Front/rear axel steering:	2 actuators, $\pm 30^\circ$ (optional)
Front/rear roll joints:	2 actuators or brakes, $\pm 30^\circ$ (optional)

2.3 Navigation

The ability of a robot to sense and navigate its surrounding environment is to some extent a measure of its intelligence, or at the very least, its autonomy. In this work, the rover has been considered to be autonomous in the same sense as one might consider a large sea, air or spaceship. While such vessels are "intelligent" and autonomous, relative to the external environment, internally they usually consist of a hierarchy of dependent operations, from the captain to the engine room. Each intermediate level in this hierarchy receives an instruction from the immediately higher level, performs some form of reasoning or processing using established knowledge or by acquiring it through (sensor) analysis, and finally defines a more precise set of sub-goals for the next lower level.

This hierarchical structure facilitates the tractable combining of sophisticated systems, each operating in different spacial and temporal regimes, into a single cohesive system. One such hierarchy, that has been described for autonomous mobile robots [5,14], consists of four levels; the *Route Planner*, *Path Navigator*, *Pilot* and *Controller*. The following brief discussion of these levels is intended to put into perspective the development (in later chapters) of an automatic piloting system and its control systems.

Route Planner

The planner is primarily concerned with choosing a general route to be followed over a distance of say 100 vehicle body lengths (400m). The planner may receive instructions from humans or through decisions taken by other mission systems. The choice of route would account for the general terrain and vehicle characteristics as well as various mission goals. Factors affecting the planning of the route may include the avoidance of steep hills or large rock formations, a preference for higher ground that would ensure good communications or optimum solar exposure, or the necessity to use the shortest and most efficient route to ensure a specific energy supply safety margin. This type of planning requires relatively long-range information, which might be supplied by topographical (satellite) maps and possibly long-range ($> 400\text{m}$) sensors, such as radar. The chosen route is subdivided by the planner into smaller segments, which are individually passed to the path navigator for further analysis.

Path Navigator

The navigator's primary function is to plot a detailed path, along each route segment received from the planner, that will be on the order of 10 vehicle bodies in length (40m). The navigation process requires the greatest amount of local area sensing,

analysis and decision making. The planned path must show consideration for specific terrain and vehicle characteristics, and the path planning algorithm may be quite similar to that used by the route planner. Factors affecting path selection may include hazard avoidance, mobility, stability, power, efficiency, reliability, communications quality and traversal time. Costing functions might be applied to statistically based information, such as grid maps, which indicate probable occupancy or level of hazard. These grid maps are essentially a composite of sufficiently detailed topographic maps and data acquired through a detailed sensor analysis of the area. This analysis would utilize both vision and targetable rangefinding techniques. The physical sensors may include stereo camera, laser and acoustic devices. The sensor analysis must finally result in a description of the terrain that can be used by a path-selection algorithm.

When the path-selection process has been completed, a simulation of the path traversal may be performed to determine the expected power consumption and stability, and to schedule landmarks or objects to be verified (checkpoints) along the way [2]. If the initial analysis and simulation is acceptable, the navigator must pass appropriate segments of the selected path to the pilot for execution. These path segments should require only limited external sensing, within the real-time capabilities of the pilot. It should be noted that the operations of the planner and navigator are not expected to be performed while the rover is in motion. This state is likely to be required to meet limitations in power and computational speed, however it should also facilitate improved sensing (with regard to targeting and noise), external communication and calibration of guidance systems.

Pilot and Controller(s)

The pilot "drives" the rover along each path segment received from the navigator. Each of these segments, perhaps only a few body-lengths long (10m), would consist of

either position and heading data, describing a continuous (relatively unobstructed) path, or the specifications for overcoming a specific obstacle. The navigator should have identified and accounted for (avoided) significant obstacles in its definition of a continuous path segment, however this may not always be the case. The pilot should utilize as much real-time external and internal sensing as possible during motion. It may stop to signal the navigator when an unexpected hazard is encountered, or it may avoid and later report the encounter. Deviations from the path specified by the navigator may result from vehicle-terrain interactions, pilot error, or through effort by the pilot to avoid close-proximity hazards on its own. (This last idea will be developed further in Chapter 4.)

Controllers directly operate or control the wheel motors and steering actuators, using the error between the desired speed and heading received from the pilot, and the data received from sensors. The controller may operate the actuators in a speed or force/torque control mode, and would be responsible for maintaining traction (avoiding wheel slip). Depending on the sophistication of the controller, control inputs applied by the pilot may range in nature from specific wheel torques, speeds and steering angles, to a simple definition of the required speed and heading. More sophisticated controllers may have direct access to some of the guidance sensors, reducing the load on the pilot. In light of this "overlapping" of pilot and controller operations, the term "pilot" is used in later chapters to refer to the combined operations of the pilot and controller levels of the navigation hierarchy.

Table 2.2 lists some of the significant characteristics of the navigation hierarchy structure, including the use of various resources. It should be noted that the same sensor or analysis data may be shared by different levels of the hierarchy. Each sensor is listed under a level according to the function of that level and consideration of the spacial or temporal constraints of the sensor or its analysis. To summarize the above-described navigation hierarchy, the planner and navigator represent functional system blocks that are involved with planning the motion of the rover. The collective actions of these two

TABLE 2.2: Navigation hierarchy for an autonomous planetary rover.

	MOTION PLANNING		MOTION CONTROL	
	<i>Route Planner</i>	<i>Path Navigator</i>	<i>Pilot</i>	<i>Controller</i>
Perception Range	100 body lengths (400m)	10 body lengths (40m)	A few body lengths (10m)	internal / tactile
Functions & Operations	- mission interface - plan basic route	- detailed survey - plan safe path - (simulation)	- drive rover - hazard avoidance - obstacle crossing	- motor control - steering control - inverse kin/dyn
Factors & Parameters Considered	- mission goals - general terrain - vehicle mobility - communications - power supply	- terrain detail - vehicle mobility - hazards - efficiency	- path error - hazard proximity - stability - obstacle specs.	- path error - heading error - wheel traction
Resources & Sensors	- satellite maps - radar	- satellite maps - stereo camera - laser scanner - acoustic ranger	- accelerometers - inertial compass - acoustic ranger - infrared edge det.	- accelerometers - inertial compass - steer angle - wheel torque
Computation Load	medium / high (off-line)	high (off-line)	low / medium (real-time)	low (real-time)

upper levels are often referred to as navigating, path planning, or perhaps more clearly, *motion planning*. The operations of the pilot and controller(s) are also often referred to as navigating, and alternatively, piloting, trajectory control, or perhaps again more clearly, *motion control*. The techniques and systems which will be presented in later chapters are all within the motion control realm of the hierarchy. The term *pilot*, as used in later chapters, refers to the general idea of a motion control system. The term *navigator*, on the other hand, will be used to refer to the motion planning system.

3. PATH TRACKING: A Linear Regulator

3.1 Introduction

Consider the hierarchical navigation scheme, described in Chapter 2, within which the navigator presents the pilot with each desired path trajectory, possibly formulated in terms of a velocity schedule (of heading and speed). Given that the path trajectory is continuous and relatively smooth, then the desired velocity profile may be described by regularly spaced points, and a desired traversal speed. Although not discussed here, a continuous and smooth path may be generated using techniques such as those presented in [27,28], with which "line-arc" type of paths may be modified to ensure continuous path curvature and smooth path-segment transfers. If one considers that the rover will "crawl" relatively slowly (10cm/s, with low inertial coupling), then the tracking problem can be divided into two parts, the first being more concerned with steering-control and remaining on the path, and the second with wheel-drive control and maintaining the desired speed.

In this chapter, we will discuss the choice of a path tracking method (Sec. 3.2), develop the steering kinematics and a feedback control law (Sec. 3.3), adjust the tracking performance according to the principles of the Linear Quadratic Regulator (Sec. 3.4) and examine the tracking characteristics through use of computer simulation (Sec. 3.5). Path-tracking in this chapter is considered primarily in terms of a steering control system, while the wheel-drive control problem, not specifically addressed in this work, is discussed in terms of wheel slippage and position estimation in Chapter 5.

3.2 Tracking Methodology

It is assumed in this work that the piloting system will receive and attempt to track a discretized path, that is updated, possibly in real time, by the navigator or other navigation subsystem. The simplest tracking technique would seem to consist of straight-line, point-to-point moves, with reorientation at each point, as illustrated in Figure 3.1. For some applications this technique appears to be quite practical [17], however it does not lend itself to continuous, smooth travel, and omnidirectionality is a requirement. (Omnidirectional robots are not, however, limited to this tracking technique.)

A variation of the point-to-point technique can be implemented using a non-omnidirectional vehicle. In this case the rover also drives straight toward the target point, but at some intermediate point it executes a pre-programmed path transfer routine [16]. Using knowledge of the next planned path segment, the pilot steers a smooth turn onto the that segment, and then continues straight toward the next target point (Fig. 3.2) This technique is intended for more traditional Automatic Guided Vehicle (AGV) plant applications, where the pilot can utilize some form of guide-way on the straightaway, or perhaps even rely on initial vehicle alignment during unguided motion. In rugged terrain, this technique would require the use of yet another tracking system to ensure that the rover is actually on each straight segment.

Another variation of the above technique that also facilitates smooth motion between points is illustrated in Figure 3.3. In this technique, a profile for the modulation of the orientation (steering) is calculated as the rover approaches each point, such that execution of this profile results in the rover continuing smoothly through the point, and on toward the next [22]. Use of this technique assumes that the rover is not required to travel in straight lines between the planned points.

In an effort to reduce the effect of inevitable wheel slippage in "off-road"

FIGURE 3.1: Path tracking with reorientation at each planned point.

FIGURE 3.2: Path tracking with smooth transfer between segments.

FIGURE 3.3: Path tracking with interception of planned points.

FIGURE 3.4: Path-tracking by the minimization of tracking errors ϵ_d and ϵ_θ .

conditions, and to attempt to keep both position and orientation close to that of the planned path, a "control theory" type of approach may be applied to facilitate continuous minimization of tracking errors. This approach to the solution of the steering-control problem has been examined by others in [15,18,19,32-36,38], and will be pursued in this work. In brief, the steering control system will effectively track a given path if it can minimize the errors in both the lateral displacement, ϵ_d , and orientation, ϵ_θ , of the rover, relative to the planned path, as illustrated in Figure 3.4. A more complete development of this system is presented next.

3.3 System Modelling and Steering Control

Considering the minimal effect of inertia on the current rover design, as mentioned in Chapter 2, the system model description and the equations of motion are based primarily on kinematic relationships. Dynamic (force) equations of motion for a related rover model are, however, presented in Appendix C. The equation development used there is quite similar to one followed in [20], while the development to be presented here, although similar to that used in many of the references mentioned above, is probably most similar to that described in [15,32].

The following development results in a state-space description of the steered rover system, where the state variables are the tracking errors, ϵ_d and ϵ_θ , and the steering angle, γ , and the system input is the steering angle command, γ_C . Figure 3.5 illustrates the significant parameters used in this development, where the displacement error, ϵ_d , is defined as the distance from the centre of the rover, P_C , to the closest local point on the planned path, P_P . The orientation error, ϵ_θ , is defined as the difference between the rover orientation, θ , and the slope of the tangent-line through the local area of the planned path, θ_P , as can also be seen in Figure 3.5.

The orientation error and its rate of change are given by the equations:

$$\begin{aligned}\epsilon_{\theta} &= \theta - \theta_p \\ \dot{\epsilon}_{\theta} &= \dot{\theta}\end{aligned}\tag{3.1}$$

The rate of change of orientation, or yaw rate, $\dot{\theta}$, is given by

$$\dot{\theta} = \frac{U}{R}\tag{3.2}$$

where U is the relatively constant forward velocity of the rover, and R is the turning radius. The turning radius is a function of the steering angle, γ , and is given by

$$R = \frac{L}{\tan \gamma}\tag{3.3}$$

where L is the wheelbase length, measured from the front axle to the centre axle. Expressions 3.1, 3.2 and 3.3 can be combined to yield the following state equation:

$$\dot{\epsilon}_{\theta} = \frac{U}{L} \tan \gamma\tag{3.4}$$

The lateral displacement error, ϵ_d , is measured from the local path tangent point, P_p , to the centre of the rover, P_c , along a line that is normal to the path tangent, as can be seen in Figure 3.5. The rate of change in this error, $\dot{\epsilon}_d$, is equal to the velocity component of the rover in the direction of this line. This reasoning, combined with inspection of Figure 3.5, leads directly to the following state equation:

$$\dot{\epsilon}_d = U \sin \epsilon_{\theta}\tag{3.5}$$

The steering dynamics are approximated by the first-order system,

$$\tau \dot{\gamma} + \gamma = \gamma_C \quad (3.6)$$

where γ_C is the steering command and τ is the time-constant. The primary purpose for including steering dynamics in the system model was to ensure a realistic steering rate, and hence a more realistic path-tracking performance during simulation. It should be noted that the steering dynamics remain independent of any inertia effects, since it is assumed that, in addition to the slow rover motion, the maximum steering rate is rather low ($2.9^\circ/\text{s}$), and the steering actuation can be modelled by a first order plant.

During nominal path tracking, the orientation error and steering angle are expected to be relatively small, such that $\epsilon_\theta \approx \sin \epsilon_\theta$ and $\gamma \approx \tan \gamma$. The state equations (3.4,3.5,3.6) and the linear feedback control law are, respectively, in matrix form,

$$\begin{bmatrix} \dot{\epsilon}_d \\ \dot{\epsilon}_\theta \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 0 & U & 0 \\ 0 & 0 & \frac{U}{L} \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} \epsilon_d \\ \epsilon_\theta \\ \gamma \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\tau} \end{bmatrix} \gamma_C \quad (3.7)$$

$$\gamma_C = - \begin{bmatrix} K_d & K_\theta & K_\gamma \end{bmatrix} \begin{bmatrix} \epsilon_d \\ \epsilon_\theta \\ \gamma \end{bmatrix} \quad (3.8)$$

where the K 's are feedback gains that may be chosen from the results of experiment, simulation or theoretical optimization. (Note that expression 3.8 is a control law.)

A block diagram of the system described by (3.7) and (3.8) is contained in Figure 3.6, where the signal flow is more clearly illustrated. It can be seen in this diagram that feedback of orientation error, ϵ_θ , not only decreases the error itself, but

FIGURE 3.6: Block diagram of the path-tracking control system.

FIGURE 3.7: Typical open-loop step response to steering command input, γ_C .

serves a similar function to derivative feedback in displacement control, damping the oscillations caused by the displacement control loop.

The steering actuation system can be seen (Fig. 3.6) to control the steering angle, γ , using an inner feedback loop that is hidden from the outer tracking-error control system. The term $1/\tau$ represents the input gain on the actuation speed, where the steering rate, $\dot{\gamma}$, is proportional to the difference between the commanded and current steering angles ($\gamma_c - \gamma$). A nominal time-constant of $\tau = 3.5\text{s}$ may be calculated by assuming that the steering angle changes at a maximum rate of 0.05s^{-1} ($\approx 2.9^\circ/\text{s}$) when the steering error is $\gamma_c - \gamma = 0.175$ ($\approx 10^\circ$). In the computer simulations to be described in later sections, the steering angle and rate each have hard limits built into their models, at $\gamma_{\max} = \pm 30^\circ$ and $\dot{\gamma}_{\max} = \pm 2.9^\circ/\text{s}$. The extra feedback loop for the steering angle (γ via K_γ) has been included primarily to yield a more complete canonical model. Although this signal will indeed affect the system operation, the feedback gain, K_γ , may be chosen without specific regard for minimization of the steering angle, as may be observed next.

3.4 Tracking-Control Gain: the Linear Quadratic Regulator

To choose a set of practical gains (K 's) for use in the control law in (3.8), the characteristic response of the open-loop system is first studied under a set of nominal conditions. Figure 3.7 illustrates the open-loop response of the system described by (3.7) and illustrated in the block diagram in Figure 3.6. This response was simulated using MATLAB, a commercial software package [49]. A step input of $\gamma_c = 0.1$ ($\approx 5.7^\circ$) was applied for 10s, and then reversed for a further 10s, to simulate steering operations that were found to occur during typical closed-loop operation. It should be noted that this motion represents small tracking oscillations of a few meters in length. The peak magnitudes of the signals can be seen to be similar to one another. The peak tracking-error signal magnitudes, $\epsilon_d \approx 0.07$ (7cm) and $\epsilon_\theta \approx 0.05$ (3°), are representative of

nominal tracking errors encountered in preliminary closed-loop simulations.

Although the relationship between these response signals will change for different time-scales, the similarities between magnitudes implies that similarly scaled feedback-gains might be used in the control law. The setting of $K_d = K_\theta$ was indeed found to yield a good balance in response for a three-wheeled tricycle-type vehicle used in [32], where the steering control law was similar to the one proposed here in (3.8). As mentioned earlier, the feedback gains may be chosen from experiment, simulation or through a more theoretical process. In this work, we will choose practical feedback gains from among sets of optimal gains, that have been determined according to the principles of the Linear Quadratic Regulator (LQR) problem. The LQR problem formulation facilitates choosing large numbers of gains in an intuitive and logical manner.

Consider the general system described by

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (3.9)$$

where, with respect to (3.7), the state vector $\mathbf{x} = [\varepsilon_d \ \varepsilon_\theta \ \gamma]^T$, the control vector $\mathbf{u} = \gamma_C$, and $\mathbf{A}$ and $\mathbf{B}$ are the corresponding constant parameter matrices. The LQR gains are determined through minimization of the performance index, J , given by

$$J = \int_0^\infty (\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}) dt \quad (3.10)$$

where $\mathbf{Q}$ and $\mathbf{R}$ are matrices of parameters that must be chosen by the designer [40]. For the current system described by (3.7) and (3.8), these matrices are defined as

$$\mathbf{Q} = \begin{bmatrix} \alpha_d & 0 & 0 \\ 0 & \alpha_\theta & 0 \\ 0 & 0 & \alpha_\gamma \end{bmatrix} \quad \mathbf{R} = \beta \quad (3.11)$$

where the parameters α_d , α_θ , α_γ and β are relative weighting factors that determine the

effort that the resulting LQR control law should expend to minimize ϵ_d^2 , ϵ_θ^2 , γ^2 and γ_c^2 , respectively. Once a set of minimization weights (α 's and β) is chosen the corresponding feedback gains, $[K_d K_\theta K_\gamma]$, can be determined from the solution, matrix S , to the algebraic (matrix) Riccati equation [40], which is given by

$$S A + A^T S - S B Q^{-1} B^T S + R = 0 \quad (3.12)$$

where the optimal feedback gain matrix, $K_S (= [K_d K_\theta K_\gamma])$, is given by

$$K_S = Q^{-1} B^T S \quad (3.13)$$

In this work, equations 3.12 and 3.13 are solved using MATLAB [49], with which the LQR feedback gain matrix, K_S , is determined directly from input of the minimization weight matrices, Q and R , from (3.11).

Table 3.1 contains various sets of optimal gains (K_S 's), determined for the above system in (3.7) and (3.8), according to the principles of LQR design described by equations (3.9-3.13). In all cases, the system parameters were set to $U = 0.1\text{m/s}$, $L = 1.5\text{m}$ and $\tau = 3.5\text{s}$, and the steering-angle minimization weight was fixed at $\alpha_\gamma = 0.0001$. Minimization of the actual steering angle, γ , was not considered significant to this optimization study. The remaining weights are varied about a common value of $\alpha_d = \alpha_\theta = \beta = 1$. The LQR gains, calculated at this common operating point, should result in similar emphasis being placed on the minimization of each of the system variables. It should be noted, with reference to Figure 3.7, that the nominal, open-loop responses of the system variables are of the same magnitude. If this had not been the case, it would have been necessary to normalize the system variables (or system matrices A and B) prior to application of the LQR design procedure. This "scaling" procedure ensures that the ratios between the chosen minimization weights will correspond well (more intuitively) with the relative amount of effort that the controller will expend to minimize the respective system variables.

TABLE 3.1: LQR control-law gains, for $U = 0.1\text{m/s}$, $L = 1.5\text{m}$ and $\tau = 3.5\text{s}$.

Q ($\alpha_\gamma = 0.0001$)		R	K_S		
α_d	α_θ	β (γ_C)	K_d	K_θ	K_γ
10	0.1	1	3.16	4.02	0.70
5	0.2	1	2.24	3.30	0.59
2	0.5	1	1.41	2.61	0.49
1	1	1	1.00	2.31	0.44
0.5	2	1	0.71	2.24	0.43
0.2	5	1	0.45	2.65	0.50
0.1	10	1	0.32	3.39	0.61
1	1	0.1	3.16	5.26	0.86
1	1	10	0.32	1.13	0.24

Although an analysis of the characteristic roots of the closed-loop system is not required for the LQR design procedure, such a study provides some insight into the nature of the resulting LQR system. Figures 3.8 and 3.9 contain characteristic-root plots for the current closed-loop system where the feedback gains, K_S , are selected from those listed in Table 3.1. Figure 3.8 contains a plot of the roots resulting from three pairs of tracking-error minimization weights, α_d and α_θ , for a fixed control-input minimization weight of $\beta = 1$. It can be observed in this plot that significant relative emphasis on the minimization of either tracking error (large α_d or large α_θ) should result in some oscillation of the system. The plot in Figure 3.9 illustrates the effect of changing the control-input minimization weight, β , with fixed tracking-error minimization weights of $\alpha_d = \alpha_\theta = 1$. This plot indicates that decreasing the control-input minimization weight, from $\beta = 10$ to $\beta = 0.1$, should generally decrease the response time of the system (poles move left), as less value is placed on the minimization of the control-input.

FIGURE 3.8: Closed-loop roots for three α_d, α_θ pairs with $\beta = 1$.

FIGURE 3.9: Closed-loop roots for three β 's with $\alpha_d = \alpha_\theta = 1$.

It should be noted that the system under consideration is not actually linear, and also, that there are practical (hard) limits on the variable values, such as steering angle and steering rate. In the next section, computer simulation will be used to illustrate the actual expected tracking response of the rover steering system, using the control law in (3.8), and various sets of feedback gains from Table 3.1.

3.5 Simulation of Path Tracking

In the early stages of this project, it was decided that a simulator should be developed to study the operation and performance of the non-linear and non-holonomic system. A computer program was therefore developed to simulate the kinematic operation of the rover under various operational scenarios. A simulation consists primarily of an iterated solution to the equations of motion, within which practical operational constraints and the non-linear characteristics of the system are easily handled. In this section we will use the results of simulation to illustrate some of the characteristics of the steering control system developed above. A detailed description of the simulation program code is contained in Appendix B.

Each planned path and its derivative should (ideally) be smooth (ie. 2nd derivative of the path should be continuous), and should have a discernable radius of curvature that is larger than the minimum turning radius of the rover (2.6m). In this work, continuous path plans were generated graphically (by "eye"), and then discretized into a series of points in the plane, where the point interval ranged from about 0.5m to 1m for a nominal path radius of about 4m. Two to three times this point spacing density was found to be adequate for a practical road-following application described in [19], where the vehicle was similar in scale to the proposed rover. In that work, a point spacing of about 0.5m was found to yield acceptable tracking performance for a minimum turning radius of 7m. It was further pointed out that the required point spacing interval is inversely proportional

to the "sharpness" of the turn. Using the findings in [19], this proportionality principle suggests a smaller point interval of about 0.3m for the above-mentioned simulation path radius of 4m, and a spacing of about 0.2m for the rover's minimum turning radius of 2.6m. The use of a larger point spacing, while still yielding reasonable results in simulation, exposes some noteworthy tracking characteristics associated with a sparsely described path, as will be pointed-out in a later simulation example.

Figures 3.10-3.15 illustrate the results of simulations, where the target path consisted of a straight, 15m-long segment, defined by two planned path points. In all cases the rover was situated with initial tracking errors of $\epsilon_d = -2.5\text{m}$ and $\epsilon_\theta = 45^\circ$. Although unacceptable in normal unimpeded path tracking, these errors are typically encountered when the rover is recovering from circumvention of an obstacle in the path, as may be observed in the next chapter on obstacle avoidance. The control-law gains used in these examples are taken from the LQR gains in Table 3.1. In all cases the steering command minimization weight was fixed at a nominal value of $\beta = 1$, and the relative sizes of α_d and α_θ were varied to examine the tradeoff between minimization of the two tracking errors, and the overall performance. All objects and paths in the plan views are drawn to scale.

The simulation results illustrated in Figures 3.10 and 3.11 were performed with control law gains of $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$, which correspond to what are considered to be nominal tracking-error minimization weights of $\alpha_d = \alpha_\theta = 1$, as can be seen in Table 3.1. These weights, or more correctly their corresponding feedback gains, cause the controller to make similar effort in the minimization of each of the lateral displacement and orientation errors. The numerically larger displacement error, ϵ_d (Fig. 3.11), results in an initial large positive steering command, γ_C . The combination of this command signal and the initial rover orientation (toward the path) results in a fairly rapid approach to the planned path. The rapid decrease in ϵ_d results in a quick reverse of the command angle as the magnitude of the orientation error, ϵ_θ , becomes more significant.

FIGURE 3.10: Tracking with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$ ($\alpha_d = \alpha_\theta = \beta = 1$).

FIGURE 3.11: Response signals with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$.

FIGURE 3.12: Tracking with $K_d = 2.24$, $K_\theta = 3.3$ and $K_\gamma = 0.59$ ($\alpha_d = 5$, $\alpha_\theta = 0.2$, $\beta = 1$).

FIGURE 3.13: Response signals with $K_d = 2.24$, $K_\theta = 3.3$ & $K_\gamma = 0.59$.

FIGURE 3.14: Tracking with $K_d = 0.45$, $K_\theta = 2.65$ & $K_\gamma = 0.5$
 $(\alpha_d = 0.2, \alpha_\theta = 5, \beta = 1)$.

FIGURE 3.15: Response signals with $K_d = 0.45$, $K_\theta = 2.65$ & $K_\gamma = 0.5$.

The approach to the path appears to be nearly critically damped, with negligible overshoot of the target trajectory.

In the simulation illustrated by Figures 3.12 and 3.13, the control gains were set to $K_d = 2.24$, $K_\theta = 3.3$ and $K_\gamma = 0.59$, which correspond to tracking-error minimization weights of $\alpha_d = 5$ and $\alpha_\theta = 0.2$ in Table 3.1. These gains tend to overemphasize the need to minimize the displacement error. The most significant points to observe in this example are the rather large tracking overshoot, and the presence of steering-angle and steering-rate limiting. The larger feedback gains promote a large oscillating steering command (γ_C). The steering angle itself (γ) can be seen to rise and fall between the hard steering limits ($\pm 30^\circ$), in a linear, actuation-rate limited ($\pm 2.9^\circ/\text{s}$) motion.

The response illustrated by Figures 3.14 and 3.15 was simulated with feedback-control gains of $K_d = 0.45$, $K_\theta = 2.65$ and $K_\gamma = 0.5$, which correspond to tracking-error minimization weights of $\alpha_d = 0.2$ and $\alpha_\theta = 5$. These gains tend to overemphasize the need for minimization of the orientation error, at the expense of the tracking displacement error, which can be seen to decrease much more slowly than in the previous examples. Feedback-induced oscillation resulting from large orientation error gain, K_θ , has been observed to become significant only at very high values of this gain.

Figures 3.16 and 3.17 illustrate the simulated tracking performance for what may be considered to be a nominal planned path of about 17m in length. In this case, the gains are reset to the "nominal" values used for the earlier example shown in Figures 3.10 and 3.11, where $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$ ($\alpha_d = \alpha_\theta = 1$). Aside from the fact that the target path curvature is continuously changing, the notable difference between this and earlier sample responses is the presence of the rapidly oscillating steering angle and tracking error signals. These oscillations result from the effect of the discrete definition of the planned path in Figure 3.16 (also see Figure 3.4) on the

FIGURE 3.16: Tracking a nominal path; $K_d = 1$, $K_\theta = 2.31$ & $K_\gamma = 0.44$ ($\alpha_d = \alpha_\theta = \beta = 1$).

FIGURE 3.17: Nominal signals with $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$.

algorithm used to calculate the tracking errors (see Appendix B for details of the code). The short (inter-point) segments in Figure 3.16 define a path that changes direction abruptly at each segment intersection. This results in small, abrupt changes in the direction of the target path, which in turn result in abrupt changes in the calculated tracking errors and the corresponding steering command signal (γ_C). It should be noted that the effects of these oscillations are more significant to the *calculated* error and command signals, than they are to the physical rover. However, if such signal oscillations are found to be of significance, say to the actuator or power systems, they may be reduced by defining more planned points along the target path (perhaps two to three times as many, as suggested earlier), or by fitting a continuous curve to the points supplied in the path plan.

Qualitatively, the simulated responses illustrated by Figures 3.10, 3.11, 3.16 and 3.17, where $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$ ($\alpha_d = \alpha_\theta = \beta = 1$), appear to be reasonably well proportioned. These examples, and their feedback gains, will serve to represent a nominal tuning for the path-tracking control system. This nominal system will be used in the development of the obstacle avoidance algorithm in the next chapter.

4. OBSTACLE AVOIDANCE: Artificial Potential Fields

4.1 Introduction

In Chapter 2, a hierarchical navigation scheme was described, in which the tasks of motion planning and motion control could be separated by function, using the titles of navigator and pilot, respectively. If it is assumed that the navigator will normally determine a safe, practical path, then the pilot can use the control law in (3.8) to track the given trajectory. However, if the pilot encountered an unplanned obstacle during travel, it would have no choice but to stop, notify the navigator, and wait for a new path plan. For large obstacles and terrain features, say larger than the rover ($> 4\text{m}$), this may be the best action. When the rover makes an unexpected encounter with obstacles of this size, the resources and decision-making abilities of the longer-range navigation systems should be consulted before simply "driving" around the obstacle.

In the case of encounters with small obstacles, say smaller than the rover ($< 4\text{m}$) but greater in size than that which can be easily overcome, it may be more expedient for the pilot itself to modify the target trajectory, avoiding the need to stop and consult with the navigator. This idea would indeed seem to be possible, if the pilot had real-time proximity detection capability and a method to modify the steering command. Acoustic, infrared and laser rangefinding sensors would seem to meet the first requirement, and use of an Artificial Potential Field (APF) will be shown to satisfy the second.

Section 4.2, below, introduces the Artificial Potential Field (APF) technique and mentions some of the difficulties with the technique, which can be mitigated by utilizing *a priori* information from the navigator. Section 4.3 discusses APF functions and an

algorithm to determine the proximity of the rover to a set of obstacle surfaces. This proximity is shown to represent what is in effect an error in obstacle avoidance, which can then be minimized by appending a new term to the steering control law (3.8) developed earlier. Section 4.4 illustrates the use of the new control law using simulation, where the pilot is shown to simultaneously avoid obstacles and track a planned path.

4.2 Potential Fields

The application of an Artificial Potential Field (APF) has been described as the binding together of a robot's plan and action into one description of the task [24]. In a typical application, repulsive force fields are assumed to exist around obstacles and an attractive field is assumed at the goal position. The summation of the collection of artificial potential force fields is transformed into an acceleration away from obstacles and toward the goal. The APF is therefore used to facilitate a type of non-contact force control, where the acceleration signal is used as a command to the pilot to drive the rover. Knowledge of the goal position and a real-time ability to detect the proximity of obstacles are all that is required to complete a journey, thus avoiding the need to generate a detailed path plan prior to moving. The APF concept has been developed and applied successfully to mobile robotics in [25,26,29,37,39,41], however there are a couple of problems associated with using such a technique by itself, in this work.

Perhaps one of the most common problems with using an APF approach for motion control, especially for autonomous mobile robots, is that the robot tends to get caught in spurious potential minima of the total APF [24]. This can be understood and pictured by considering a rover that is travelling toward an attractive goal, while at the same time approaching two repulsive obstacles, where one obstacle is to the left and other to the right of the rover's heading. It is possible that the summation of the attractive and repulsive force fields will form a local minimum, into which the rover will enter and then

stop, prior to passing between the two obstacles. The rover must be encouraged to avoid or become less influenced by this local (position) minimum, using other information.

A second difficulty with using an APF-only method, perhaps more significant in the current application, is that real-time proximity sensors cannot easily be used in the analysis of the practicality or safeness of a particular trajectory. A typical APF-only motion control system utilizes only data that is acquired in real-time. The need to achieve near 100% reliability for an autonomous planetary rover mission would seem to suggest that if data from "higher" planning systems is available, it should be utilized by the pilot, to ensure greater probability of mission success.

It is noted that the APF technique could also be used at the path planning stage by the navigator. The resulting plan would then represent *a priori* knowledge to the pilot, and would not contribute to the solution of the motion control problem. In this work, it is desired to combine real-time obstacle avoidance with the previously defined path plan provided by the navigator, regardless of how that plan was generated. Despite the above issues, the APF technique does indeed offer a useful technique to quantify the presence of obstacles during motion control. It will be seen that, using an APF method, rather sparse ranging data (distance and bearing) can be used to create what is in effect a description of the error in avoiding an obstacle. The new error information can then be input to the path tracking system, giving pilot the ability to avoid unplanned obstacles as well as track a planned path.

4.3 Potential Functions and Avoidance Error

In what is perhaps a more common application of the APF concept, discussed above, repulsive potential force fields are assumed to exist around obstacles, and an attractive field is assumed at a goal position. In this work, only repulsive fields need be

considered, since the path tracking system, developed in the last chapter, will be used to "direct" the rover toward its goal, and only unplanned obstacles must be avoided.

To determine the repulsive potential, or proximity, the pilot will require a function, computable in real time, to determine the distance or significance of obstacles near the rover and planned path. For dynamic systems, a repulsive potential force, such as that encountered in systems of like-charged particles (inverse-square law), offers one possibility. A practical function for this law, suggested in [25], may take the form

$$V(\rho) = \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right)^2 \quad \text{if } \rho \leq \rho_0, \quad (4.1)$$

$$V(\rho) = 0 \quad \text{if } \rho > \rho_0$$

where $V(\rho)$ is the repulsive potential, ρ is the shortest distance to the obstacle, η is a gain, and ρ_0 is a boundary distance outside of which the obstacle is of no consequence and should be ignored (see Figure 4.1). In this case the potential increases from zero at the outer boundary ($\rho = \rho_0$), and approaches infinity at the obstacle surface ($\rho = 0$), ensuring that obstacle contact will not take place (in theory).

In situations where inertia (acceleration) is a factor, the rate of change of the proximity may be of concern. A function for this case, suggested in [39], may be written

$$V(\rho, \dot{\rho}) = \frac{\dot{\rho}}{\dot{\rho}^2 - 2\alpha\rho} \quad (4.2)$$

where $V(\rho, \dot{\rho})$ is an impedance-like repulsive potential, $\dot{\rho}$ is the rate of change of the distance to the obstacle, and α is the maximum deceleration that the vehicle can use to stop. The potential can be seen to increase with decreasing distance to the obstacle as well as with increasing velocity toward the obstacle. This formulation results in the

non-threatening obstacles. For example, obstacles that are detected to the side of the rover are probably of less consequence than obstacles directly in the path, and obstacles that lie behind the rover should be of little or no consequence to the control of forward motion. A simple function that uses direction information to compute a modified avoidance error, ϵ_{PSP} , is described by

$$\begin{aligned} \epsilon_{PSP} &= (\rho_0 - \rho)(\lambda_0 - \lambda) && \text{if } \rho \leq \rho_0 \text{ and } 0 \leq \lambda \leq \lambda_0, \\ \epsilon_{PSP} &= 0 && \text{if } \rho > \rho_0 \text{ or } \lambda < 0 \text{ or } \lambda > \lambda_0 \end{aligned} \quad (4.4)$$

where λ is the bearing of the obstacle from the forward direction (Figure 4.1), and λ_0 is a bearing limit, past which the obstacle is of no consequence and can be ignored.

The notation ϵ_{PSP} , in (4.4), refers to an error term calculated at a specific Point Subjected to Potential (PSP). This idea stems from work on collision avoidance of robot manipulators [25], where it was found that a few well placed PSP's could keep a whole manipulator (or mobile robot) from contacting surrounding obstacles. Figure 4.1 illustrates the meaning of the various parameters in equation (4.4), as well as the location of the PSP's used on the rover in this work.

In writing the obstacle avoidance error calculation routine used for the simulations in the next section, a slightly modified version of equations (4.4) was coded for each PSP, to result in positive avoidance errors for obstacles to the left of the rover and negative errors for obstacles to the right (refer to Appendix B for details). In this routine, all obstacles are checked to find the closest surface, or reflecting "facet", to each PSP. The closest facet to a PSP is then used in the calculation of the avoidance error at that specific PSP. The sum of the PSP avoidance errors is then calculated according to

$$\varepsilon_A = \sum_i K_{PSPi} \varepsilon_{PSPi} \quad (4.5)$$

where K_{PSPi} and ε_{PSPi} are the gain and calculated error, respectively, at each PSP location ($i = 1$ to 7), and ε_A is the total effective error in obstacle avoidance.

The steering control law from (3.8) can now be reformulated to include the obstacle avoidance error term, ε_A , in a new control law given by

$$\gamma_C = -K_d \varepsilon_d - K_\theta \varepsilon_\theta - K_\gamma \gamma - K_A \varepsilon_A \quad (4.6)$$

where the gains (K 's) may be optimized or otherwise practically chosen. The first three terms on the right side of this new law influence the steering command in the same manner as demonstrated in the simulation examples in the last chapter. The last term in the law operates in a similar manner to the tracking error terms. For example: if the front-left corner of the rover is found to have crossed into the safe boundary region of an obstacle detected to the front and left of the rover (see Figure 4.1), a positive avoidance error (ε_A) will be generated. This will result in a negative contribution to the steering command in (4.6), causing the rover to turn more to the right of the obstacle.

The algorithms in (4.4) and (4.5) are intended to process raw data from a directional rangefinding sensor. No distinction is made as to whether the individual avoidance errors, calculated for each PSP, have come from the same obstacle body or from separate, stand-alone obstacles. A more detailed analysis of object features could be used, at greater computational expense, to describe continuous obstacle surfaces or to group multiple objects into single impassable regions. Data from such an analysis would make the current algorithm considerably more robust during execution. Simulation will be used in the next section to illustrate the operation of the obstacle avoidance algorithm described by (4.4) and (4.5), and the new steering control law in (4.6).

4.4 Simulation of Obstacle Avoidance

Simulation of the obstacle avoidance capability was accomplished using a modified version of the computer program mentioned in the last chapter. The modifications consisted of the addition of a routine to determine avoidance error, according to expressions 4.4 and 4.5, as well as a change in the steering control law from expression 3.8 to 4.6. The obstacles are defined by a series of points that describe the surfaces. These points represent reflecting sub-surfaces or "facets", that may be detected by a directional rangefinder. The proximity routine chooses the closest facet to each PSP for use in calculating the individual PSP avoidance errors, as discussed in the last section. Detailed descriptions of the simulator and avoidance routine are contained in Appendix B.

In Chapter 2, it was pointed out that the pilot should be expected to make only minor modifications to the planned path. However, many of the following simulation examples use head-on or very close proximity encounters to emphasize the characteristics of the avoidance algorithm and steering control law. In the following examples, the tracking gains were set to $K_d = 1$, $K_\theta = 2.31$ and $K_\gamma = 0.44$, which correspond to a nominal set of LQR gains, determined with $\alpha_d = \alpha_\theta = \beta = 1$, as listed in Table 3.1. The individual PSP gains, K_{PSP_i} in (4.5), were set to unity for all simulations. The total avoidance-error feedback gain, K_A , and potential field boundary distance, ρ_0 , are the significant parameters to be observed in the following examples.

Figure 4.2 illustrates the simulation results for an encounter with a nominal obstacle (2m diameter vertical cylinder). The avoidance gain is set to $K_A = 5$ and the potential boundary is set to $\rho_0 = 3m$. The rover begins turning when the forward-centre PSP first crosses the potential boundary limit, at $\rho = 3m$. The rover's path can be seen to oscillate about the boundary for most of the detour, except where the rover is just leaving the vicinity of the obstacle. The contributions of the rear PSP's to the total avoidance error result in a small delay in the rover's return to the planned path.

FIGURE 4.2: Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 3\text{m}$.

Figures 4.3 and 4.4 illustrate the results of simulations, where the avoidance gains were set to $K_A = 1$ and 10, respectively, and the boundary distance was fixed at $\rho_0 = 3\text{m}$. These examples may be compared directly with the "nominal" simulation illustrated in Figure 4.2, where the avoidance gain was set to $K_A = 5$. In Figure 4.3, the rover can be seen to pass well into the potential field, narrowly missing the obstacle. In the central portion of the detour, the rover is well immersed in the field, and its path tends to oscillate much less than in the nominal case. In Figure 4.4, the rover can be seen to maintain its initial sharp turn for considerably longer than in the nominal case, resulting in an excursion well out of the influence of the potential field. It should be noted that the minimum distance or leeway between the rover and obstacle is about the same here (Fig. 4.4) as it was in the nominal case (Fig. 4.2). These examples illustrate that, while smaller avoidance gains allow the rover to pass more closely to the obstacle, larger gains are of little value beyond a certain point; the rover must always enter the

FIGURE 4.3: Avoidance of 2m diameter obstacle with $K_A = 1$ & $\rho_0 = 3\text{m}$.

FIGURE 4.4: Avoidance of 2m diam. obstacle with $K_A = 10$ & $\rho_0 = 3\text{m}$.

potential field, and it cannot turn away at a greater rate than that dictated by the hard steering limits.

Figures 4.5 and 4.6 illustrate simulation results, where the boundary distances were set to $\rho_0 = 2\text{m}$ and 4m , respectively, and the avoidance error gain was fixed at $K_A = 5$. These examples may also be compared directly with the nominal simulation case illustrated in Figure 4.2, where the boundary was set to $\rho_0 = 3\text{m}$. In Figure 4.5, the rover begins turning soon after entering the potential field, but only narrowly misses (perhaps contacting) the obstacle due to the very short lead time. In Figure 4.6, the rover easily circumvents the obstacle, oscillating about the perimeter of the field in a similar fashion to that observed in the nominal case (Fig. 4.2), but at a greater distance. These last two examples illustrate how the minimum leeway given to a single obstacle is dictated primarily by the potential field boundary distance, ρ_0 . Figures 4.7 and 4.8 contain further simulation examples, that illustrate the increased leeway afforded to single objects, for boundary settings of $\rho_0 = 3\text{m}$ and 6m , respectively, and a fixed gain of $K_A = 5$.

Figures 4.9 and 4.10 illustrate the simulation results for concurrent avoidance of two obstacles that are relatively close together. In Figure 4.9, the avoidance parameters were set to nominal values of $K_A = 5$ and $\rho_0 = 3\text{m}$. In this case the rover can be seen to "squeeze" between the two obstacles, in an effort to return to the planned path after having circumvented the first obstacle. For the simulation in Figure 4.10, the avoidance gain was increased to $K_A = 10$, and the rover can be seen to avoid the obstacles as if they were an impassable group. In this case, the increased avoidance gain did not allow the rover to turn toward the obstacle passageway. If it had, it would have passed between the obstacles in a similar manner to that shown in Figure 4.9. This group avoidance phenomenon also results when the potential field boundary distance, ρ_0 , is increased.

It should be observed that the "stalling" problem associated with the use of APF's (from spurious local minima), mentioned earlier, should be mitigated in practical

FIGURE 4.5: Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 2\text{m}$.

FIGURE 4.6: Avoidance of 2m diameter obstacle with $K_A = 5$ & $\rho_0 = 4\text{m}$.

FIGURE 4.7: Additional obstacle leeway, with $K_A = 5$ and $\rho_0 = 3\text{m}$.

FIGURE 4.8: Additional obstacle leeway, with $K_A = 5$ and $\rho_0 = 6\text{m}$.

FIGURE 4.9: Concurrent avoidance of obstacles, with $K_A = 5$ & $\rho_0 = 3\text{m}$.

FIGURE 4.10: Concurrent obstacle avoidance with $K_A = 10$ & $\rho_0 = 3\text{m}$.

applications by the tracking of the planned path, as suggested by the simulations illustrated in Figures 4.9 and 4.10. It should also be noted, however, that the rover can still become trapped during execution of the current algorithm (assuming that the rover stops when proximity exceeds safe limits). This may be understood by considering a scenario, similar to that described earlier, where the planned path lies between two obstacles (or detected facets of a single obstacle) that the rover cannot actually pass between. During approach, there will be no change in the steering command by the control law in (4.6), if the proximity errors are balanced. When collision becomes imminent, the pilot will have no choice but to stop the rover, and notify the navigator of the error in path execution. To alleviate this problem, a more sophisticated data analysis algorithm may be used (for instance, curve fitting) to describe smooth obstacle surfaces from multiple facets, or to group multiple obstacles into single impassable regions, as was mentioned in the last section. Such procedures were not implemented in this work, since the phenomenon posed no problem for the generation of practical examples used to illustrate the avoidance algorithm's general characteristics.

5. DEAD RECKONING: Sensor Fusion

5.1 Introduction

The most common dead reckoning technique used for ground vehicles is that of odometry. Errors in odometry usually result from wheel slippage and ground or wheel deformation. Indoor mobile systems seem to suffer the least from these wheel/ground interaction phenomena, and linear position accuracies of $\pm 0.5\%$ are not uncommon [17,21]. In contrast to indoor systems, an autonomous planetary rover will likely suffer from all of the above problems, especially that of wheel slippage. It is perhaps useful to point-out a difference between the terms skidding and slipping [50]. For example, a typical road car is propelled by a single-axel pair of differentially driven wheels. Poor traction, by such a car, usually occurs in the form of skidding, where inertial forces or drive-torques overcome the wheel-to-ground contact (static friction) forces. During motion of the slow-moving planetary rover, the inertial forces will be of much less significance, however, kinematic constraints of the independently driven wheels, as well as climbing and wheel-ploughing forces, will likely result in what is referred to as wheel slippage. Irregular changes in wheel/ground interaction make it difficult to compensate for slippage (or skidding). Odometry errors tend to sum to an increasing total error in position estimation as the travelled distance increases. This problem may be thought of as a distance-dependent integration error.

The most common dead reckoning technique used for marine and aerospace vehicles is that of inertial navigation. It has been suggested that an Inertial Navigation System (INS) might suit the navigation needs of an autonomous planetary explorer and its mission [1]. Such a system would be able to avoid problems with wheel slippage, and

should be able to maintain a position accuracy (over short, pilot-level path segments) similar to that of an indoor odometry system [1]. The most significant drawback to an INS is that it suffers from time-dependent integration error or "drift". In a typical mobile robot application, double integration of the signal from an inexpensive linear accelerometer may result in a position error rate of 1-8cm/s [44]. This problem is usually addressed by using precision (costly) mechanisms and/or sophisticated error correction techniques. Despite the superiority of inertial navigation, there are still a number of advantages to using odometry, including: high physical robustness and reliability of the encoders, low weight, power consumption and cost, and a zero time-dependent drift.

In more rugged planetary terrains, such as dense rock fields, it is unlikely that odometry or inertial navigation will be practical over great distances, due to extreme wheel slippage or deformation, and to large amplitude random vibrations. Realistic navigation scenarios should include frequent stops for system recalibration. A scenario described in [1] suggests that a planetary rover should stop about every 10m (100s) to recalibrate its INS. This intermittent recalibration requirement aside, consideration of the complementary nature of the odometry and INS errors (distance- versus time-dependent) suggests that the two systems may be combined to address the problems encountered when using either system alone. The appeal of such a hybrid system becomes stronger if one considers using inexpensive sensors, such as rotary position encoders for odometry and off-the-shelf accelerometers for inertial navigation.

Section 5.2, below, discusses practical implementations of odometry and inertial navigation for the planetary rover. Section 5.3 describes a test vehicle, outfitted with an optical wheel encoder and a linear accelerometer, and introduces experimental data that illustrates the nature of the encoder (odometry) and accelerometer (INS) errors. This data was used to develop and analyze a signal-fusion technique to mitigate the position and velocity errors generated by each of the sensors. Section 5.4 presents the fusion and related processing algorithms, and discusses the performance of the signal-fusion process.

5.2 Odometry and Inertial Navigation

Odometry, in this application, involves the measurement of relative changes in position, from which one can differentiate to find velocity and acceleration. Conversely, Inertial Navigation (IN) usually involves the measurement of accelerations, which may be integrated to find relative changes in velocity and position. The following discussion describes a configuration and set of equations that should give complementary odometry and inertial navigation capabilities to the planetary rover under consideration. This discussion is intended to highlight the computational characteristics of the two techniques, prior to the consideration of the experimental data, signal processing and fusion algorithms presented in later sections.

Odometry

The incremental distance, Δs , travelled by the hub of a rolling wheel, is given by

$$\Delta s = c\Delta\phi \quad (5.1)$$

where c is the rolling circumference of the wheel, and $\Delta\phi$ is the incremental rotation of the wheel about its axle. For the six-wheel rover under consideration, the measurements of Δs , at multiple wheel hubs, may be used to determine the changes in position and orientation of the rover. The use of rotary position encoders at multiple wheel locations can improve the system reliability through redundancy, and in the case where only a single wheel encoder is operational, the rover's position and orientation may still be determined through measurement of the steering angle γ .

The following development, similar to a "dual odometer system" presented in [16], is intended to illustrate how incremental changes in rover position, ΔX and ΔY , and orientation, $\Delta\theta$, may be determined from measurement of Δs at each of the two wheels

on the central axle of the vehicle considered in this work. This particular approach offers some redundancy, since it does not require the steering angle for normal computation, but can easily utilize it in the event of an encoder failure.

The incremental distances measured by the central left and right wheel-encoders, Δs_L and Δs_R respectively, are given by

$$\Delta s_L = \left(R - \frac{W}{2} \right) \Delta \theta \quad (5.2)$$

$$\Delta s_R = \left(R + \frac{W}{2} \right) \Delta \theta \quad (5.3)$$

where R is the turning radius and W is the fixed distance between the encoder wheels, as illustrated in Figure 5.1. Combining equations 5.2 and 5.3 together to eliminate R directly yields an equation to compute the incremental change in rover orientation, $\Delta \theta$:

$$\Delta \theta = \left(\frac{\Delta s_R - \Delta s_L}{W} \right) \quad (5.4)$$

Combining equations 5.2 and 5.3 together to eliminate $\Delta \theta$, yields an equation to compute the radius of curvature of the rover's path:

$$K = \frac{W}{2} \left(\frac{\Delta s_L + \Delta s_R}{\Delta s_R - \Delta s_L} \right) \quad (5.5)$$

In the instantaneous local coordinate frame, the incremental changes in longitudinal and lateral position, Δx and Δy respectively, are given by:

FIGURE 5.1: Dual-wheel odometry using left (Δs_L) and right (Δs_R) encoder increments.

$$\begin{aligned}\Delta x &= R \sin \Delta\theta \\ \Delta y &= R(1 - \cos \Delta\theta)\end{aligned}\tag{5.6}$$

When the rover's motion approaches a straight line ($\Delta s_L \approx \Delta s_R$), R approaches infinity, and the incremental changes in position may be set to $\Delta x = \Delta s_L$ and $\Delta y = 0$.

In the event of an encoder failure, the measured steering angle γ , may be used to calculate R using equation 3.3. This "measured" R may then be substituted into either equation 5.2 or 5.3 to determine $\Delta\theta$, and into equations 5.6 to determine Δx and Δy . It should be noted that the concept of incremental changes (Δ 's) has been introduced, in part, to emphasize the discrete nature of the data that is received from a wheel encoder system. It is assumed that each signal will be updated and then immediately processed

by digital means, after each sample, at a discrete time instant referred to as moment k .

In world coordinates, at moment k , the orientation of the rover, θ_k , is given by

$$\theta_k = \theta_{k-1} + \Delta\theta \quad (5.7)$$

where $\Delta\theta$ is the most recent incremental change in the orientation, and θ_{k-1} is the previously determined orientation of the rover. The incremental global changes in position, ΔX and ΔY , may then be found using the formulae

$$\begin{aligned} \Delta X &= \Delta x \cos \theta - \Delta y \sin \theta \\ \Delta Y &= \Delta x \sin \theta + \Delta y \cos \theta \end{aligned} \quad (5.8)$$

where the orientation may be set to $\theta = (\theta_k + \theta_{k-1})/2$. The global position at moment k , X_k and Y_k , is then given by

$$\begin{aligned} X_k &= X_{k-1} + \Delta X \\ Y_k &= Y_{k-1} + \Delta Y \end{aligned} \quad (5.9)$$

where ΔX and ΔY define the most recent incremental change in absolute position, and X_{k-1} and Y_{k-1} describe the last known position of the rover (centre point).

The velocities and accelerations may be determined through differentiation of the position data streams. For example, in absolute coordinates, at moment k , the rate of change of X_k may be approximated using the backward difference formula

$$\dot{X}_k = \frac{X_k - X_{k-1}}{\Delta t} \quad (5.10)$$

where Δt is the time interval since the previous position estimate, X_{k-1} .

When wheel slippage occurs, it should be noted that Δx , Δy and $\Delta\theta$ will be in error. While a single occurrence of wheel slippage will be retained as an "offset" error in X_k , Y_k and θ_k , repeated slippage usually results in increasing position and orientation errors within the global reference frame. It should also be noted that while velocity and acceleration are affected by slippage and differentiation "noise", the resulting errors are not integrated, and so do not increase over the distance travelled. These slippage-induced errors will be illustrated by the experimental results presented later.

Inertial Navigation

The most common system components of an INS are gyroscopes and linear accelerometers. Linear accelerometers are used to measure translational acceleration, while a gyro's rotational inertia is usually used as a reference for the measurement of either rotational position or rate (rate being more common). Figure 5.2 illustrates the location of INS sensors that would measure the planar accelerations, $\ddot{x}$ and $\ddot{y}$, and yaw rate, $\dot{\theta}$, of the rover, in vehicle coordinates.

Integration of the INS-based acceleration ($\ddot{x}, \ddot{y}$) or velocity ($\dot{x}, \dot{y}, \dot{\theta}$) signals can be performed using a formula (Runge-Kutta mid-point algorithm) of the form,

$$\dot{x}_k = \Delta t \frac{\ddot{x}_k + \ddot{x}_{k-1}}{2} + \dot{x}_{k-1} \quad (5.11)$$

where in this particular case, $\ddot{x}_k$ and $\ddot{x}_{k-1}$ represent the current (moment k) and previously measured longitudinal accelerations of the rover, respectively, and $\dot{x}_k$ and $\dot{x}_{k-1}$ represent the current and previously calculated longitudinal velocities, respectively. The absolute velocity, $\dot{X}$ and $\dot{Y}$, can be determined using

FIGURE 5.2: Inertial navigation (planar measurement) and inclination (tilt) sensors.

$$\begin{aligned}\dot{X} &= \dot{x} \cos \theta - \dot{y} \sin \theta \\ \dot{Y} &= \dot{x} \sin \theta + \dot{y} \cos \theta\end{aligned}\tag{5.12}$$

where, as in equations 5.8, $\theta = (\theta_k + \theta_{k-1})/2$ at sample moment k .

The time-dependent drift associated with inertial navigation, mentioned earlier, may be explained through examination of equation 5.11. Consider that any bias or otherwise unbalanced error in the acceleration signal, $\ddot{x}$, will be integrated into the calculated velocity, $\dot{x}$, to result in an increasing velocity error over time. If the rover has stopped, the velocity estimation error will, at best (with well-zeroed sensors), remain constant. Any error in velocity will be continuously (discretely but regularly) integrated, resulting in a changing position estimate, x . This time-dependent drift phenomenon will be observed in the experimental results presented in the next section.

Dynamic variables, such as angular acceleration, have been omitted from this

discussion, and only planar motion has been considered throughout this work. While slower moving vehicles, such as the planetary rover, should not require measurement of rotational accelerations, such as $\ddot{\theta}$, other gyroscopes or tilt sensors may be used to measure pitch and roll attitude [19,44,45], in an effort to ensure that proper coordinate transformations can be made between the local and absolute reference frames. (ex. The distance specified between two points on a map of unexplored mountainous terrain will be shorter than the actual distance the rover must drive, to go over the hills, from one point to the other.)

Finally, it should be noted that the use of a rate-gyro for inertial navigation, or a tachometer for odometry, will yield a signal that is at neither the acceleration nor position "level". Use of these rate signals will of course improve both odometry and INS performance, since they would reduce the required amount of signal processing. In this work, however, inertial navigation is represented only by acceleration, and odometry by displacement. It is believed that this greater disparity between odometry and INS signal "levels" may lead to the development of a more complete (and perhaps more robust) fusion technique. From a practical standpoint, it should also be noted that rotary position encoders and linear accelerometers are widely available at relatively low cost.

5.3 Experimental Data

The raw data used for development and testing of the fusion algorithm was collected by Neculescu and Kim in [43], using a customized laboratory test vehicle with a tricycle configuration. An optical position encoder was used to measure the rotational displacement of the forward drive wheel, and a linear accelerometer was mounted near the drive wheel to measure the longitudinal acceleration of the vehicle. A counter-weight was extended from the rear of the vehicle to decrease the load on the drive wheel, and encourage slippage. (Loose paper was also placed on the floor to increase the probability

of wheel slippage.) The total mass of the vehicle assembly was about 11kg, and the maximum torque available at the drive wheel (radius = 7cm) was about 2Nm.

The linear displacement (drive motor) was closed-loop controlled using Proportional-Derivative (PD) feedback of the wheel-encoder signal. For the experiment, the controller was commanded to drive the vehicle, from rest, to a point 2m distant. The bandwidth of the vehicle system was determined, from experiment, to be $f_b \approx 0.35\text{Hz}$. The signals from the encoder and accelerometer were each sampled, simultaneously, at a rate $f_s = 100\text{Hz}$, yielding encoder-based position and INS-based acceleration data streams. A sampling rate as low as 7Hz may have been sufficient to capture the accelerometer response, assuming $f_s/f_b \geq 20$ is prudent [31], however the encoder velocity (wheel oscillation) was observed to be near this rate during slippage, necessitating a higher sampling rate. The two resulting discrete data sets were used in the development and testing of the data-fusion algorithms that will be discussed in the next section.

Figures 5.3-5.6 illustrate experimental results, where the encoder-based velocity (ENCvel) and acceleration (ENCacc) signals, and INS-based velocity (INSvel) and position (INSpos) signals, were derived as illustrated in the flow-chart in Figure 5.7. The differentiation and integration steps in the chart were performed using expressions similar to equations 5.10 and 5.11, respectively. The filtering step illustrated in the flow-chart, used to remove sensor vibration noise from the accelerometer signal, was performed using a discrete, second-order, Butterworth low-pass filter, with a cut-off frequency of 10Hz. No analog pre-filtering was performed on the sensor signals. A description of the signal processing algorithms (with exception of the Butterworth filter) may be found in the documented computer code in Appendix D.

In Figure 5.3, the encoder position signal can be seen to overshoot and then reverse to settle just short of the 2m set-point. (Steady-state error is attributed to mechanical friction and play.) The INS position signal, however, indicates that the

FIGURE 5.3: Encoder (x) and INS (o) position signals.

FIGURE 5.4: Encoder (x) and INS (o) velocity signals.

FIGURE 5.5: Encoder (*) and INS (⊗) acceleration signals.

FIGURE 5.6: Acceleration error $\varepsilon = \text{ENCacc} - \text{INSacc}$.

FIGURE 5.7: Encoder and INS signal paths.

vehicle had moved more slowly, overshooting and oscillating at a closer point, and then continuing to move forward without apparent bound. In Figure 5.4, the encoder velocity signal can be seen to increase and then oscillate in the region of about 1.3s to 3.3s and then again from about 4.3s to 5.5s. These oscillations occurred during periods of observed wheel slippage. The INS signal, although not subject to the effects of slip, suffered from the problem of integration error, which is quite evident after about 5.5s, where the INS position and velocity signals indicate that the rover continued to move forward at a speed of about 13cm/s, when in fact it had stopped. (Comparable drift-rates of 1-8cm/s were found in [44], as mentioned earlier.) Finally, looking at the acceleration signal (Fig. 5.5), it is noted that deviation of the encoder from the INS acceleration signal also appears to correspond well with the occurrence of wheel slippage, and that neither of these signals is affected in the long term by previous motion. (Neither signal suffers from time-based drift.) This last observation is especially significant to the development of a method for detecting the occurrence of wheel slippage.

5.4 Signal Processing and Fusion

In review of the experimental results presented above, it is noted that when there is no apparent wheel slippage, the encoder estimates of position and velocity will likely be more accurate (over time) than the continually "drifting" INS-based estimates. However, at the onset of wheel slippage, the INS signals should maintain more reliable baselines from which one could estimate changes in the true position and velocity. The problem then, is to determine the onset and presence of wheel slippage, and to update the position and velocity estimates using the appropriate signal source. Neither the encoder nor INS acceleration signal suffers from an accumulating (integration) error, and so it is at this signal level that one can reliably (at any time) detect the presence of wheel slippage. Figure 5.6 contains a plot of the difference between the encoder and INS acceleration signals, ϵ . The determination of this "error" signal is depicted in the flow chart in Figure 5.7. Observation of the magnitude $|\epsilon|$ should facilitate detection of the occurrence of significant wheel slippage.

In addition to the differentiation and integration routines mentioned earlier, a C language routine, detailed in Appendix D, was written to perform the signal fusion process. The following code describes the subroutine:

$$\begin{aligned}
 &\text{if } |\epsilon| \leq \epsilon_{\max} \\
 &\quad \text{then: } v(k) = \text{ENCvel}(k) \\
 &\quad \quad p(k) = p(k-1) + \text{ENCpos}(k) - \text{ENCpos}(k-1) \\
 &\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (5.13) \\
 &\quad \text{else } v(k) = v(k-1) + \text{INSvel}(k) - \text{INSvel}(k-1) \\
 &\quad \quad p(k) = p(k-1) + \text{INSpos}(k) - \text{INSpos}(k-1)
 \end{aligned}$$

where $\epsilon_{\max}$ is a maximum error set-point, the ENCxxx and INSxxx terms represent signals taken from correspondingly labelled points in the signal paths in Figure 5.7, and

$v(k)$ and $p(k)$ are the current fusion estimates of velocity and position, respectively. The operation of this routine is as follows: If the magnitude of the acceleration error signal (ϵ) is within the maximum set-point magnitude limits ($|\epsilon| \leq \epsilon_{\max}$), then the velocity and position estimates are updated using encoder data. If the error signal is outside of these limits, then the fusion estimates are updated with INS data.

The results of fusion of the encoder and INS data streams are illustrated in the position and velocity plots in Figures 5.8 and 5.9. For these examples, the error limit was set to $\epsilon_{\max} = 3\text{m/s}^2$, as illustrated in Figure 5.6. During the first 1.5s, the fused signals can be seen to follow those of the encoder. During the first episode of wheel slippage, between about 1.5s and 3.3s, the fused signals can be seen to update alternately between the encoder and INS signals. This is more clearly visible in the velocity plot in Figure 5.9, where the fused signal appears to randomly "jump" to the encoder signal, and then "flatten-out" for brief periods while it parallels the INS signal. After about 3.3s, traction is regained, and the fused velocity follows the encoder signal, while the fused position signal continues to parallel that of the encoder. At about 4.5s, slippage begins again, and a similar process is repeated. After about 5.5s, the vehicle comes to a stop, and the fusion estimates continue to update using the encoder data-stream alone. The fused estimate of the final position remains constant, at somewhat less than the encoder value, while the INS position signal continues to increase due to integration error.

The most obvious deficiency of the current fusion algorithm is that it fails to detect slippage in the zero-crossing regions of the acceleration error signal, where slippage (oscillations) may be occurring, but where $|\epsilon| < \epsilon_{\max}$. These regions may, however, be detected by monitoring the error derivative, $\dot{\epsilon}$, for relatively large values. (see flow-chart in Figure 5.7). The fusion algorithm could be modified, such that $|\epsilon| \leq \epsilon_{\max}$ and $|\dot{\epsilon}| \leq \dot{\epsilon}_{\max}$ must both be satisfied before it may be assumed that traction has been regained. Ideally, ϵ and $\dot{\epsilon}$ would be about 90° out-of-phase, such that small settings of $\epsilon_{\max}$ and $\dot{\epsilon}_{\max}$ would ensure detection of any part of an oscillating slippage signal.

FIGURE 5.8: Encoder (x), INS (o) and fused (—) position signals.

FIGURE 5.9: Encoder (x), INS (o) and fused (—) velocity signals.

During the course of this work, it was found that this technique does indeed reduce the tendency for erratic triggering or "jumps" in the fused signal, however the presence of noise in the original signals precludes taking a "clean" derivative, and false triggering still tends to occur, albeit less frequently. Alternatively, it was found that use of an estimator (predictive filter) can improve the fusion performance by "filling-in:" with more reasonable data, when slippage detection fails. This approach was finally adopted, and will be discussed later. It is noted that simultaneous use of the error-rate signal ($\hat{\epsilon}$) and the predictive filter was found to be only marginally better than use of the filter alone.

To first address the noise problem, it is noted that the encoder acceleration data, illustrated in Figure 5.5, appears to contain a quantized "noise" signal, of about 2m/s^2 in amplitude, that is most clearly evident during periods where slippage does not take place. (For instance, during the first 1.3s in Figure 5.5). This noise seems to have been introduced during (and via) the differentiation processes. A pre-fusion filtration process may be used to reduce this noise and the resulting erratic "triggering" of the fusion algorithm. While various types of low-pass filters may be used to reduce this noise, a Kalman filter design was chosen for this work. The Kalman filter design technique offers a formal method for choosing the filter parameters using knowledge (or estimation) of the amplitudes of the sensor noise and the process disturbance. In this case, the quantized component in the signal, mentioned above, represents the sensor noise, and the slippage oscillations represent the process disturbance. Appendix E describes the design of a first-order Kalman filter that is based on analysis of the encoder acceleration signal. The resulting design is a simple, real-time executable, recursive linear filter, described by

$$\text{Signal}_{\text{out}}(k) = 0.1165 \text{Signal}_{\text{in}}(k) + 0.8641 \text{Signal}_{\text{out}}(k-1) \quad (5.14)$$

where the output at moment k , $\text{Signal}_{\text{out}}(k)$, is a linear combination of the current input, $\text{Signal}_{\text{in}}(k)$, and previous output, $\text{Signal}_{\text{out}}(k-1)$. The frequency response of this low-pass filter has a decay rate of about 2.8dB/octave, beyond a cut-off frequency of about 4Hz.

FIGURE 5.10: Additional Kalman filtering of the encoder-based signals.

Figure 5.10 contains a flow-chart illustrating the location, within the encoder signal path, where the Kalman filter in (5.14) was placed to reduce the noise in the encoder acceleration signal. While the need is to filter the error signal, ϵ , prior to use by the fusion algorithm, the filter was placed in the indicated location to expose the underlying nature of the encoder signal, so that it may be studied and compared more easily with that of the INS signal. Figure 5.11 contains a plot of the INS and filtered encoder acceleration signals. (It should be noted that the **ENCacc** signal is not used directly by the fusion algorithm.) The Kalman filter used here is quite similar to that which would have been designed if the filter was to be placed immediately after the subtraction process in Figure 5.10, since the magnitude and nature of the slippage and noise signals is nearly identical at both points in the signal paths. (Compare Figure 5.5 with 5.6.) Figure 5.12 contains a plot of the acceleration error, ϵ , obtained with the above filter in place. This signal may be compared with that illustrated in Figure 5.6, to observe how the higher-frequency quantized noise has been reduced, to more clearly

FIGURE 5.11: INS (o) and filtered encoder (x) acceleration signals.

FIGURE 5.12: Acceleration error (ϵ) after filtering of encoder noise.

reveal the difference between the encoder and INS acceleration characteristics.

It was pointed-out above that false triggering within the fusion algorithm cannot be eliminated entirely, using the current slippage detection method. The "jumps" in the encoder-based fusion updates, however, may be reduced by passing the fused signals through an additional estimator or filter. A Kalman filter design may be used once again, however the characteristics of the "noise" in the fused signals are somewhat more difficult to quantify in this case. The design of an optimum state estimator (Kalman filter), while desirable, is not considered to be particularly illustrative at this point. (The general procedure is described in Appendix E.) Nevertheless, in an effort to reduce the obvious error when the fusion algorithm "misses", and the fused velocity signal attempts to "jump" to the value of the encoder velocity, the Kalman filter in 5.14 was inserted, without modification, into the encoder-based velocity update portion of the fusion algorithm in 5.13. The location of the new filter is illustrated pictorially in Figure 5.10.

The results of fusion of the encoder and INS data streams, with the new filters in place, are illustrated in the plots in Figures 5.13 and 5.14. For these examples, the error limit was set to $\varepsilon_{\max} = 0.5\text{m/s}^2$, as illustrated in Figure 5.12. On close inspection, it can be observed (especially in Figure 5.14) that an irregular noise signal is superimposed on the fused signal. This noise is in fact the filtered "jumps" that occurred (similarly to those shown in Figure 5.9), where the fusion algorithm missed detecting slippage. On the whole, the results are similar to those obtained without the additional filters (see Figures 5.8-5.9), except that the fused velocity signal is now less noisy, and the position estimate contains correspondingly less integration error.

FIGURE 5.13: Encoder (x), INS (o) and *filtered/fused* (—) position signals.

FIGURE 5.14: Encoder (x), INS (o) and *filtered/fused* (—) velocity signals.

6. CONCLUSION

6.1 Summary of Results

1. The linear regulation ("control-theory") technique has been shown to be useful in creating a path tracking system for the proposed planetary rover. The Linear Quadratic Regulator (LQR) design principle facilitated rather "intuitive" tuning of the feedback control law and the resulting path tracking characteristics. It should be possible to extend the design, using the LQR technique, to include the control or minimization of other tracking parameters deemed significant in the future.
2. The use of an artificial potential field has been shown to facilitate the analysis of obstacle proximity data that could be obtained from a directional rangefinder. In addition, the output signal of the avoidance algorithm is easily utilized by the linear steering control law already developed for path tracking. With the addition of this new error signal, the pilot acquired the ability to circumvent unplanned obstacles detected during traversal of a pre-planned path.
3. An algorithm has been developed to fuse the signals from a rotary wheel-position encoder and a linear accelerometer. This fusion process mitigates the time-dependent drift-errors caused by integration of the accelerometer signal, and displacement-dependent drift-error cause by wheel-slippage. In addition, the use of simple linear recursive filters has been shown to be capable of removing a substantial amount of signal processing noise. The resulting fused estimates of both position and velocity appear to be better than those obtained using either of the odometry or INS (accelerometer) signals alone.

6.2 Future Research

1. The next apparent step in the development of the path tracking system is to implement the steering control law (linear regulator) on a real system. The test platform should have sensors capable of determining the actual position and orientation of the vehicle in real time. (This work could of course be carried-out in conjunction with development of the dead reckoning system.)
2. Once again, the most apparent next step in the development of the obstacle avoidance capability is to implement the avoidance error algorithm on a real rover, that has on-board rangefinding instruments such as an acoustic array or laser scanner. To test the robustness of the avoidance (and path tracking) capability, it must be possible to vary the general environment and obstacle configuration. In addition, it may be useful to try other potential functions in the avoidance error routine, such as an "impedance function", that utilizes velocity information. Lastly, the LQR design principle may be applied to choosing an optimal feedback gain for the avoidance error signal, once typical obstacle encounters have been defined and studied.
3. The development and testing of the signal fusion algorithms for dead reckoning measurements was performed "off-line" on experimental data. The algorithms should next be written into the code of the digital controller used to control the test vehicle. Closing the feedback-loop with the fusion algorithm will facilitate testing of the actual estimation improvement as well as the stability of the complete system. The use of a tachometer should also be investigated, in an effort to reduce the amount of signal processing and resultant noise, thereby improving the accuracy and stability of the overall system. It is further noted that a calibrated, non-slipping encoder wheel, or other reliable reference, should be installed "parallel" to vehicle control system, to properly characterize the accuracy of signal fusion system.

6.3 Summary of Contributions

1. It has been shown that path-tracking by a slow-moving vehicle, such as an autonomous planetary rover, can be handled using a simple linear regulation technique that has been discussed in recent literature. In addition, the feedback gains of the steering control law, and hence the tracking characteristics, have been shown to be "intuitively" adjustable using the Linear Quadratic Regulator design principle.
2. The Artificial Potential Field technique, most often applied to "reactive" control in the robotics literature, has been combined with a somewhat more traditional pre-planned path tracking methodology that uses *a priori* information. More specifically, a system has been developed to facilitate simultaneous path tracking and obstacle avoidance by an autonomous rover, using the combined techniques.
3. An algorithmic technique has been developed to fuse signals from a rotary wheel encoder (odometer) and a linear accelerometer, into velocity and position estimation signals that appear to be better than those obtained using either sensor alone.
4. A set of simple but functional algorithms has been developed to pilot an autonomous, wheeled planetary rover. The attempt to use practical model parameters and limitations, or actual experimental data, throughout system development, establishes confidence in these preliminary results, and encourages further investigation.

References

- [1] D.P. Miller, A.H. Mishkin, K.E. Lambert, D. Bickler and D.P. Bernard, "Autonomous Navigation and Mobility for a Planetary Rover," *27th Aerospace Sciences Meeting*, AIAA, #89-0859, 1989.
- [2] D.P. Miller, D.J. Atkinson, B.H. Wilcox and A.H. Mishkin, "Autonomous Navigation and Control of a Mars Rover", *Automatic Control In Aerospace - Selected Papers from the IFAC Symposium*, pages 111-114, 1990.
- [3] L.S. McTamaney, B.D. Douglas and S.Y. Harmon, "Mars Rover Concept Development", *Mobile Robots III*, Proc. SPIE, vol. 1007, pp. 85-94, 1989.
- [4] B.H. Wilcox, D.B. Gennery and A.H. Mishkin, "Mars Rover Local Navigation and Hazard Avoidance", *Mobile Robots III*, Proc. SPIE, vol. 1007, pp. 72-6, 1989.
- [5] V.R. Kumar and K.J. Waldron, "A Review of Research on Walking Vehicles", *The Robotics Review I*, pp. 243-266, The MIT Press, Cambridge, MA, 1989.
- [6] P.M. O'Donnell et al., "Energy Storage Considerations for a Robotic Mars Surface Sampler", *23rd Intersociety Energy Conversion Engineering Conference*, vol. 2, pp. 385-9, 1988.
- [7] A. Chmielewski, "RTG's for Space Exploration at the End of the 20th Century", *24th Intersociety Energy Conversion Engineering Conference*, vol. 2, pp. 715-20, 1989.
- [8] J. Bares, M. Herbert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons and W. Whittaker, "Ambler: An Autonomous Rover for Planetary Exploration", *Computer*, pp. 18-26, June, 1989.

- [9] J.Y. Wong, *Theory of Ground Vehicles*, Second Edition, John Wiley & Sons, Toronto, 1993.
- [10] D.D. Wright, "Power and Propulsion Systems for Mobile Robotic Applications", *Mobile Robots*, Proc. SPIE, vol. 727, pp. 10-16, 1987.
- [11] A. Speissbach, B. Clark, S. Larimer, B. Tobey, B. Lindauer, R. Koenig and T. Lisec, "Issues and Options for a Mars Rover", *Mobile Robots II*, Proc. SPIE, vol. 852, pp. 164-171, 1988.
- [12] D.D. Wright and R.E. Watson, "Comparison of Mobility System Concepts for a Mars Rover", *Mobile Robots II*, Proc. SPIE, vol. 852, pp. 180-187, 1988.
- [13] W. Chun, "Ground Vehicle Options", *Mobile Robots II*, Proc. SPIE, vol. 852, pp. 188-194, 1988.
- [14] C. Istik and A.M. Meystel, "Pilot Level of a Hierarchical Controller for an Unmanned Mobile Robot", *IEEE Journal of Robotics and Automation*, Vol. 4, No. 3, pages 241-255, June 1988.
- [15] A. Hemami and S. Hodai, "Comments on Steering and Control of Mobile Robots," *International Conference on Engineering Applications of Mechanics*, pp. 315-322, Tehran, June 1992.
- [16] B. Steer, "Trajectory Planning for a Mobile Robot," *The International Journal of Robotics Research*, vol. 8, no. 5, pp. 3-14, October 1989.
- [17] J. Borenstein and Y. Koren, "Motion Control Analysis of a Mobile Robot," *Journal of Dynamic Systems, Measurement, and Control*, vol. 109, pp. 73-79, June 1987.
- [18] A.Y. Lee, "A Preview Steering Autopilot Control Algorithm for Four-Wheel-Steering Passenger Vehicles." *Journal of Dynamic Systems, Measurement, and*

- Control*, vol. 114, pp. 401-408, September 1992.
- [19] O. Amidi and C. Thorpe, "Integrated Mobile Robot Control," *Mobile Robots V*, Proc. SPIE, vol. 1388, pp. 504-523, 1991.
 - [20] T.J. Graettinger and B.H. Krogh, "Evaluation and Time-Scaling of Trajectories for Wheeled Mobile Robots", *Journal of Dynamic Systems, Measurement, and Control*, vol. 111, pp. 222-231, June 1989.
 - [21] J. Holland, H.R. Everett and G.A. Gilbreath, "Hybrid Navigational Control Scheme for Autonomous Platforms", *Mobile Robots V*, Proc. SPIE, vol. 1388, pp. 291-298, 1991.
 - [22] E. Malotiaux, R. Alimenti, M. Bogaert and P. Gaspart, "Intelligent Piloting Tools for Control of an Autonomous Mobile Robot." *Mobile Robots V*, Proc. SPIE, vol. 1388, pp. 372-383, 1991.
 - [23] S. Price, W. Chun, M. Hammond and A. Hubert, "Wheeled Planetary Rover Testbed", *Mobile Robots V*, Proc. SPIE, vol. 1388, pp. 550-559, 1991.
 - [24] D.E. Koditshek, "Robot Planning and Control via Potential Functions," *The Robotics Review 1*, MIT Press, 1989.
 - [25] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90-98, Spring 1986.
 - [26] D.S. Neculescu, B. Kim and S. Kalaycioglu, "Free Motion, Collision Avoidance and Contact Motion Control for Mobile Robots", *7th IFAC/IFIP/IFORS/IMACS/ISPE Symposium on Information Control Problems in Manufacturing Technology*, 1992.
 - [27] W. Neison, "Continuous Curvature Paths for Autonomous Vehicles," *1989 IEEE*

International Conference on Robotics and Automation, pp. 1260-1264.

- [28] Y. Kanayama and B.I. Hartman, "Smooth Local Path Planning for Autonomous Vehicles," *1989 IEEE International Conference on Robotics and Automation*, pp. 1265-1270.
- [29] R.C. Arkin, "Motor Schema-Based Mobile Robot Navigation," *The International Journal of Robotics Research*, pp. 92-112, August 1989.
- [30] J.Z Sasiadek and P.J. Wojcik, "Design of a Discrete Kalman Filter for a Tactile Sensor in a Feedback Control System", *International Journal of Control*, vol. 45, no. 4, pp.1211-1228, 1987.
- [31] G.F Franklin, J.D. Powell and M.L. Workman, *Digital Control of Dynamic Systems*, Second Edition, Addison-Wesley, 1990.
- [32] A. Hemami, M.G. Mehrabi and R.M.H. Cheng, "A New Control Strategy for Tracking in Mobile Robots and AGV's", *1990 IEEE International Conference on Robotics and Automation*, pp. 1122-1127.
- [33] C.C. de Wit and R. Roskam, "Path Following of a 2-DOF Wheeled Mobile Robot under Path and Input Torque Constraints," *1991 IEEE International Conference on Robotics and Automation*, pp. 1142-1147.
- [34] C. Samson and K. Ait-Abderrahim, "Feedback Control of a Nonholonomic Wheeled Cart in Cartesian Space," *1991 IEEE International Conference on Robotics and Automation*, pp. 1136-1141.
- [35] B. d'Andrea-Novet, G. Bastin and G. Campion, "Modelling and Control of Non Holonomic Wheeled Mobile Robots," *1991 IEEE International Conference on Robotics and Automation*, pp. 1130-1135.
- [36] W.L. Nelson and I.J. Cox, "Local Path Control for an Autonomous Vehicle."

- 1988 IEEE International Conference on Robotics and Automation*, pp. 1504-1510.
- [37] R.B. Tilove, "Local Obstacle Avoidance for Mobile Robots, Based on the Method of Artificial Potentials", *1990 IEEE International Conference on Robotics and Automation*, pp. 566-571.
 - [38] Y. Kanayama, Y. Kimura, F. Miyazaki and T. Noguchi, "A Stable Tracking Control Method for an Autonomous Mobile Robot," *1990 IEEE International Conference on Robotics and Automation*, pp. 1142-1147.
 - [39] B.H. Krogh and C.E. Thorpe, "Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles," *1986 IEEE International Conference on Robotics and Automation*, pp. 1664-1669.
 - [40] K. Ogata, *Modern Control Engineering*, Second Edition, Prentice Hall, 1990.
 - [41] B.H. Krogh, "A Generalized Potential Field Approach to Obstacle Avoidance Control," *Robotics Research: The Next Five Years and Beyond*, SME Conference Proceedings, Bethlehem, PA, August 1984.
 - [42] D.N. Green, J.Z. Sasiadek and G.S. Vukovich, "Guidance and Control of an Autonomous Planetary Rover", *Proc. IEE - IEEE Vehicle Navigation and Information Systems Conference - VNIS '93*, pp. 539-542, Ottawa, October 1993.
 - [43] D.S. Neculescu, J.Z. Sasiadek, B. Kim and D.N. Green, "Fusion of Inertial and Kinematic Navigation Systems for Autonomous Vehicles", *Proc. IEE - IEEE Vehicle Navigation and Information Systems Conference - VNIS '93*, pp. 462-465, Ottawa, October 1993.
 - [44] B. Barshan and H.F. Durrant-Whyte, "An Inertial Navigation System for a Mobile Robot", *IFAC Intelligent Autonomous Vehicles Workshop*, Southampton, UK, April 1993.

- [45] J. Vaganay and M.J. Aldon, "Attitude Estimation for a Vehicle using Inertial Navigation", *IFAC Intelligent Autonomous Vehicles Workshop*, Southampton, UK, April 1993.
- [46] H.W. Sorenson, "Kalman Filtering Techniques", *Advances in Control Systems Theory and Applications*, vol. 3, pp. 219-292, C.T. Leondes, editor, Academic Press, 1966.
- [47] R.T. Gamber and G.L. Adams, "Mars Sample Return Mission Options", *Mars: Past, Present and Future*, Chapter 19, ed. E.B. Pritchard, American Institute of Aeronautics and Astronautics, (AIAA), 1992.
- [48] D.N. Green, J.Z. Sasiadek, D.S. Neculescu and B. Kim, "Dead Reckoning for Mobile Robots via Sensor Fusion", *Department of National Defence Knowledge Based Systems and Robotics Workshop*, in press, Ottawa, November 1993.
- [49] "MATLAB ... a high-performance interactive software package for scientific and engineering numeric computation." and the "Control System Toolbox for MATLAB ... a collection of algorithms ... that implement common control system design, analysis, and modelling techniques.", The MathWorks, Inc., South Natick, MA, USA.
- [50] J.C. Alexander and J.H. Maddocks, "On the Kinematics of Wheeled Mobile Robots", *International Journal of Robotics Research*, vol. 8, no. 5, pp. 15-27, October 1989.

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

Appendix A: Motion Resistance and Drive Power

The following is an analysis of motion resistance and the required drive power for the 1000kg, six-wheel rover design. It has been assumed here that the weight of the rover is equally distributed, such that each wheel bears a weight of about 627N on the surface of Mars, where gravitational acceleration is approximately 3.76m/s^2 .

The sinkage, z_0 , of a rigid wheel can be estimated using the empirical formula,

$$z_0 = \left[\frac{3F_w}{(3-n)(k_c + bk_\phi)D^{0.5}} \right]^{2n+1} \quad (\text{A } 1)$$

where F_w , b and D are the weight, width and diameter of the wheel, respectively, and n , k_c and k_ϕ are referred to as pressure-sinkage parameters for the terrain [9]. One empirically determined set of terrain parameters for loose dry sand is given in [9] as $n = 1.1$, $k_c = 950\text{N/m}^{n+1}$ and $k_\phi = 1526\text{kN/m}^{n+2}$. If it is assumed that the 1m-diameter rover wheels are 25cm wide, then expression A.1 can be used to estimate a single wheel sinkage of about $z_0 = 2.4\text{cm}$. This is of course only an estimate of what might be expected in a similar environment to that in which the pressure-sinkage parameters were originally determined. Also, slip-induced-sinkage, or simply slip-sinkage [9], will likely result in the rover wheels sinking even further into the terrain.

In the case of moderate sinkage, where $z_0 \leq D/6 \approx 16.7\text{cm}$, the motion or compaction resistance of a wheel may be approximated by the formula [9],

$$R_c = \frac{(3 F_w)^{\frac{2n+2}{2n+1}}}{(3-n)^{\frac{2n+2}{2n+1}}(n+1)(k_c + b k_\phi)^{\frac{1}{2n+1}} D^{\frac{n+1}{2n+1}}} \quad (\text{A.2})$$

which, by substitution of the known parameters from above, results in compaction resistance of $R_c \approx 73\text{N}$ for each wheel. The rover would therefore face a total motion resistance about 438N in loose dry sand.

The above analysis yields similar results for dry sand on a 35% grade (19.3° slope), where normal to the terrain surface, $F_w = 589\text{N}$ and $z_0 = 2.3\text{cm}$, and the compaction resistance is $R_c \approx 67\text{N}$ at each wheel. Considering that there is an additional climbing force of about 1243N , acting on the rover body, each drive wheel would therefore have to overcome a force of about $1243/6 + 67 = 274\text{N}$. Sinkage and traction on a 60% grade (31° slope) of hard packed soil or rock may vary considerably, depending on the surface conditions, however the rover must at least overcome the gravitational climbing force of about 1937N , or about $1937/6 = 323\text{N}$ at each wheel.

The above analysis has assumed certain vehicle characteristics (or lack of, such as wheel grousers or lugs), it does not account for the actual martian terrain conditions and is itself based only on specific experimental data. Table A.1, however, summarizes the torque and power requirements, based on the forces calculated, in an effort to characterize the drive motors which may be required.

According to the data listed in Table A.1, a minimum torque of about 161Nm must be available at each wheel, and each drive motor should be capable of delivering at least 16.1W . The likelihood of considerable wheel slippage occurring in normal operation suggests that a stepper motor or backlash-free type of motor (ex. harmonic drive motor) may be of little value. Considering the requirement of a low power/torque

TABLE A.1: Wheel torque and power requirements.

Terrain Condition	Wheel Torque (Nm)	Wheel Speed (rad/s)	Wheel Power (Watts)	6-Wheel Power (Watts)
0% grade, sand, 10cm/s	36.5	0.2	7.3	43.8
35% grade, sand, 5cm/s	137	0.1	13.7	82.2
60% grade, hard, 5cm/s	161	0.1	16.1	96.6

ratio, one should be able to apply suitable wheel torque using a low-power dc motor with high gear-reduction transmission.

Wright [10] offers a useful set of tools that can be used to scale electric motors to the wheel diameter of the rover. Using these tools, with the current rover specifications, the required 16.1W motor would have a diameter of about 7cm. Considering that slippage, increased sinkage and transmission losses have not been taken into account, let the total drive power be increased to 375W, which is the midpoint of the range of expected available power (250-500W). A drive power of about $375/6 = 62.5\text{W}$, which would then be available to each wheel, could therefore be delivered by a motor that is about 10cm in diameter.

Alternatively, a direct-drive type of motor, capable of producing torque in the range required, could be used with minimal or no gear reduction. Such a motor could be more easily integrated directly into the wheel, reducing space, transmission loss and increasing reliability. It should be noted that without significant gear reduction, a braking system may be required to conserve power when the rover is still. Finally, it may be possible to adapt any of the above motors to meet the drive requirements, as long as reliability, weight, compactness and power requirements are met.

Appendix B: Computer Simulation Code

The path tracking and obstacle avoidance systems, developed in Chapters 3 and 4, respectively, were tested by computer simulation. The computer code was written in the C language and compiled using Microsoft QuickC. An 80386/387 microcomputer was used during compilation and execution of the code. The following sub-sections contain annotated listings of the code itself. Section B.1 contains the main simulation program, from which the tracking-error (Section B.2), avoidance-error (Section B.3) and graphics output (Section B.4) routines are called.

B.1 Main Program: *simtrack.c*

Table B.1 contains a point-form synopsis of the most significant events in the execution schedule of the main simulation program. The main program reads three files before beginning the simulation. The first file, named "simparms" contains general and miscellaneous simulation parameters. The second and third files, named in the parameter file, contain sequential lists of x,y-point pairs that describe the planned path and the obstacle surfaces, respectively. Sample listings of these files are illustrated in Tables B.2, B.3 and B.4, respectively. Figure B.1 contains the graphics output from simulation with the data contained in the sample files. The complete listing of the main program is contained at the end of this section.

TABLE B.1: Significant events during execution of main simulation program.

READ SIMULATION PARAMETER FILE
READ PLANNED PATH FILE
READ OBSTACLE DESCRIPTION FILE
INITIALIZE PARAMETERS AND VARIABLES
ITERATION LOOP
Update steering angle: γ
Update rover position and orientation: x, y, θ
Find new path-tracking errors: $\epsilon_d, \epsilon_\theta$
Find new obstacle avoidance error: ϵ_A
Issue new steering command: γ_C
Store data
WRITE OUTPUT FILE
CALL PLOTTING ROUTINE

TABLE B.2: Simulation parameter file *simparms*.

madepath	:	name of file containing planned path data	
blob05m	:	name of file containing obstacle data ('xxxxxxxx')	none)
0.0	:	X - initial X coordinate of rover (m)	
0.0	:	Y - initial Y coordinate of rover (m)	
36	:	theta - initial orientation of rover (deg)	
0	:	delta - initial steering angle (deg)	
10.0	:	rover speed (cm/s)	
0.2	:	simulation time step (sec)	
5	:	iteration skip-factor for data output	
1.	:	k1, displacement error feedback gain	
2.31	:	k2, orientation error feedback gain	
0.44	:	k3, steering angle feedback gain	
5	:	k4, obstacle proximity feedback gain	
20	:	number of rovers to be drawn along the path	
0.0	:	horizontal dimension (x) of plot space (m), 0.0 auto	
0	:	draw border around the plot? (1=yes)	
1.5	:	wheelbase length (m)	
30.0	:	maximum steering angle (deg)	
2.9	:	maximum steering rate (deg/sec)	
3.5	:	steering actuation time constant (sec)	
3.0	:	safe proximity dist. (m) (ignore obstacles at this dist.)	
1	:	output data file? (1 = yes)	

TABLE B.3: Planned path *madepath*.

25	
0.0000E+000	0.0000E+000
6.0000E-001	3.9806E-001
1.2000E+000	7.7460E-001
1.8000E+000	1.1487E+000
2.4000E+000	1.4808E+000
3.0000E+000	1.7729E+000
3.6000E+000	2.0196E+000
4.2000E+000	2.2179E+000
4.8000E+000	2.3678E+000
5.4000E+000	2.4726E+000
6.0000E+000	2.5386E+000
6.6000E+000	2.5753E+000
7.2000E+000	2.5951E+000
7.8000E+000	2.6128E+000
8.4000E+000	2.6457E+000
9.0000E+000	2.7122E+000
9.6000E+000	2.8324E+000
1.0200E+001	3.0263E+000
1.0800E+001	3.3140E+000
1.1400E+001	3.7148E+000
1.2000E+001	4.2464E+000
1.2600E+001	4.9248E+000
1.3200E+001	5.7634E+000
1.3800E+001	6.7729E+000
1.4400E+001	7.9610E+000

TABLE B.4: Obstacle description *blob05m*.

12	
7.5	5.0
7.567	4.75
7.75	4.567
8	4.5
8.25	4.567
8.433	4.75
8.5	5
8.433	5.25
8.25	5.433
8	5.5
7.75	5.433
7.567	5.25

FIGURE B.1: Sample simulation output, using above input files.

Main simulation program *simtrack*

```

#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>

#include "qc25\src\recipes\nrutil.c" /* utility functions */

#include "qc25\dngfiles\patherr.c" /* path tracking-error calculation routine */
#include "qc25\dngfiles\avoiderr.c" /* obstacle avoidance-error calculation routine */
#include "qc25\dngfiles\plotpath.c" /* graphics output routine */

main() {
    int i,num_plan_points,max_num_out,point_number,num_rovers,output_data_file,border;
    int sim_counter,point_skip, num_obstacle_points;
    float *Xout,*Yout,*Tout, *Error_Dout, *Error_Tout,*gammaout,*Timeout,*Error_Obout;
    double *xplan,*yplan,error_d,error_t,travel,xplotsize,last_leg,dist_last,dist_2nd_last;
    double k1,k2,k3,gamma,max_steer,theta_start,t,s,dt,ds,plan_path_length;
    double speed,curvature,Lf,theta,dx,dy,d_theta,avg_t,dX,dY,X,Y,steer_time_const;
    double max_steer_rate,gamma_command,gamma_rate,gamma_old,gamma_rate_old,
           gamma_avg,gamma_error;
    double *obstacle_x,*obstacle_y,error_ob,k4,ignore_obstacle_dist,DGR;
    char dummy[200], plan_path_file[13], obstacle_file[13];
    FILE *fp;

    /* allocate memory for output data using "nrutil" memory allocation routines */
    max_num_out = 1400;
    Xout = vector(1,max_num_out);
    Yout = vector(1,max_num_out);
    Tout = vector(1,max_num_out);
    Error_Dout = vector(1,max_num_out);
    Error_Tout = vector(1,max_num_out);
    gammaout = vector(1,max_num_out);
    Timeout = vector(1,max_num_out);
    Error_Obout = vector(1,max_num_out);

    _clearscreen(_GCLEARSCREEN);
    _settextrows(_MAXTEXTROWS);

    DGR = 57.29577951; /* degrees per radian */

```

```

/* READ PARAMETER FILE *****/
printf("\n\n\n Reading parameter file 'simparms' ... \n");
fp = fopen("simparms", "r");
fscanf(fp, "%12s %[^\\n\\t]", plan_path_file, dummy); /* planned path data file name*/
fscanf(fp, "%12s %[^\\n\\t]", obstacle_file, dummy); /* obstacle point data file name*/
fscanf(fp, "%12f %[^\\n\\t]", &X, dummy); /* initial X coord */
fscanf(fp, "%12f %[^\\n\\t]", &Y, dummy); /* initial Y coord */
fscanf(fp, "%12f %[^\\n\\t]", &theta, dummy); /* initial orientaion */
fscanf(fp, "%12f %[^\\n\\t]", &gamma, dummy); /* initial steering angle */
fscanf(fp, "%12f %[^\\n\\t]", &speed, dummy); /* speed of rover */
fscanf(fp, "%12f %[^\\n\\t]", &dt, dummy); /* time increment */
fscanf(fp, "%12i %[^\\n\\t]", &point_skip, dummy); /* skip factor for data output */
fscanf(fp, "%12f %[^\\n\\t]", &k1, dummy); /* disp. error feedback gain */
fscanf(fp, "%12f %[^\\n\\t]", &k2, dummy); /* orient. error feedback gain */
fscanf(fp, "%12f %[^\\n\\t]", &k3, dummy); /* steering angle feedback gain */
fscanf(fp, "%12f %[^\\n\\t]", &k4, dummy); /* obstacle proximity feedback gain */
fscanf(fp, "%12i %[^\\n\\t]", &num_rovers, dummy); /* number of rovers to draw */
fscanf(fp, "%12f %[^\\n\\t]", &xplotsize, dummy); /* horizontal size of plot space */
fscanf(fp, "%12i %[^\\n\\t]", &border, dummy); /* draw a border around the plot? */
fscanf(fp, "%12f %[^\\n\\t]", &Lf, dummy); /* rover wheelbase length */
fscanf(fp, "%12f %[^\\n\\t]", &max_steer, dummy); /* maximum steering angle */
fscanf(fp, "%12f %[^\\n\\t]", &max_steer_rate, dummy); /* maximum steering rate */
fscanf(fp, "%12f %[^\\n\\t]", &steer_time_const, dummy); /* steering actuator time constant */
fscanf(fp, "%12f %[^\\n\\t]", &ignore_obstacle_dist, dummy); /* ignore obstacles at this distance */
fscanf(fp, "%12i %[^\\n\\t]", &output_data_file, dummy); /* data output? */

fclose(fp);

max_steer = max_steer/DGR; /* convert to rads */
max_steer_rate = max_steer_rate/DGR; /* convert to rad/sec */
theta = theta/DGR; /* convert initial theta to rads */
gamma = gamma/DGR; /* convert initial gamma to rads */
speed = speed / 100; /* convert to m/s */

/* READ PLANNED-PATH FILE *****/
printf("\n\n\n Reading planned path file '%s' ... \n", plan_path_file);
fp = fopen(plan_path_file, "r");
fscanf(fp, "%4i", &num_plan_points);

/* allocate memory for the data */
xplan = dvector(1, num_plan_points);
yplan = dvector(1, num_plan_points);

for (i = 1; i <= num_plan_points; ++i)
    fscanf(fp, "%12f %12f", &xplan[i], &yplan[i]);
fclose(fp);

```

```

/* READ OBSTACLE FILE ***** */
num_obstacle_points = 0;
if (strcmp(obstacle_file, "xxxxxxx") != 0) {
    printf("\n\n\n Reading obstacle file '%s' ... \n", obstacle_file);
    fp = fopen(obstacle_file, "r");
    fscanf(fp, "%4i", &num_obstacle_points);

    /* allocate memory for the data */
    obstacle_x = dvector(1, num_obstacle_points);
    obstacle_y = dvector(1, num_obstacle_points);

    for (i = 1; i <= num_obstacle_points; ++i)
        fscanf(fp, "%12lf %12lf", &obstacle_x[i], &obstacle_y[i]);
    fclose(fp);
}

/* INITIALIZATION ***** */
sim_counter = 0; /* iteration counter */

t = 0.0; /* time */
s = 0.0; /* total rover travel (m) */

gamma_rate = 0; /* rate of change in steering angle */

plan_path_length = 0.0; /* find length of planned path */
for (i = 2; i <= num_plan_points; ++i)
    plan_path_length += pow(pow(xplan[i] - xplan[i-1], 2) +
                             pow(yplan[i] - yplan[i-1], 2), .5);

dist_last = 1000000; /* variables and parameters describing distance to */
dist_2nd_last = 0.0; /* end of path; used to terminate simulation */
last_leg = pow(pow(xplan[num_plan_points] - xplan[num_plan_points-1], 2) +
               pow(yplan[num_plan_points] - yplan[num_plan_points-1], 2), .5);

/* record first data point at start of path */
point_number = 1;
Xout[1] = X;
Yout[1] = Y;
Tout[1] = theta;

gammaout[1] = gamma; /* initial steering angle */

Timeout[1] = t;

/* initial path-tracking errors */
find_error(xplan, yplan, num_plan_points, X, Y, theta, &error_d, &error_t);
Error_Dout[1] = error_d; /* initial lateral displacement error */
Error_Tout[1] = error_t; /* initial orientation error */

```

```

/* initial proximity error signal */
if (num_obstacle_points != 0)
    find_avoid_error(obstacle_x,obstacle_y,num_obstacle_points,
        X,Y,theta,ignore_obstacle_dist,&error_ob);
else
    error_ob = 0.0;
Error_Obout[1] = error_ob;

/* initial steering command */
gamma_command = -k1 * error_d - k2 * error_t - k3 * gamma - k4 * error_ob;

/* ITERATION ***** */
printf("\n\n\n Running simulation ...");
while ((dist_last > last_leg) || (dist_2nd_last <= last_leg)) { /*cycle until rover nears goal*/

    ++ sim_counter;

    /* STEERING ACTUATOR DYNAMICS (find new gamma) ----- */
    gamma_old = gamma;
    gamma_rate_old = gamma_rate;

    /* gamma error signal from feedback */
    gamma_error = gamma_command - gamma_old;

    /* gamma rate resulting from steering actuator gain */
    gamma_rate = gamma_error / steer_time_const;
    if (gamma_rate < -1 * max_steer_rate) gamma_rate = -1 * max_steer_rate;
    if (gamma_rate > max_steer_rate) gamma_rate = max_steer_rate;

    /* integration of gamma ODE to new gamma using Runge-Kutta mid-point algrm. */
    gamma = gamma_old + dt*(.5)*(gamma_rate + gamma_rate_old);
    if (gamma < -1 * max_steer) gamma = -1 * max_steer;
    if (gamma > max_steer) gamma = max_steer;

    /* ROVER TRAVEL UPDATE of position and orientation ----- */
    /* Steering Radius (1/R) during travel increment */
    gamma_avg = (gamma + gamma_old) / 2;
    curvature = tan(gamma_avg) / Lf;

    ds = speed * dt; /* distance travelled during increment */

    d_theta = curvature * ds; /* change in orientation during inc */

    /* change in position, in rover coords */
    if (curvature < .0001) { /* very small gamma */
        dx = ds;

```

```

        dy = 0.0;
    }
    else {
        dx = sin(d_theta) / curvature;
        dy = (1 - cos(d_theta)) / curvature;
    }

    /* change in position, in world coordinates */
    avg_t = theta + d_theta / 2; /* avg theta during increment, theta is the old value */
    dX = dx * cos(avg_t) - dy * sin(avg_t);
    dY = dy * cos(avg_t) + dx * sin(avg_t);

    theta = theta + d_theta; /* new orientation after increment */

    X = X + dX; /* new position after increment */
    Y = Y + dY;

    t = t + dt; /* total time, after increment */

    s = s + ds; /* total distance travelled, after increment */

    /* FIND ERRORS and ISSUE NEW STEERING COMMAND ----- */
    /* tracking errors */
    find_error(xplan,yplan,num_plan_points,X,Y,theta,&error_d,&error_t);

    /* obstacle proximity error */
    if (num_obstacle_points != 0)
        find_avoid_error(obstacle_x,obstacle_y,num_obstacle_points,
                        X,Y,theta,ignore_obstacle_dist,&error_ob);
    else
        error_ob = 0.0;

    gamma_command = -k1 * error_d - k2 * error_t - k3 * gamma - k4 * error_ob;

    /* STORE INFO FOR OUTPUT AND PLOTTING ROUTINES ----- */
    if (sim_counter % point_skip == 0) {
        if (point_number < max_num_out) {
            ++point_number; /* increment point number and store */
            Xout[point_number] = (float) X;
            Yout[point_number] = (float) Y;
            Tout[point_number] = (float) theta;
            Error_Dout[point_number] = error_d;
            Error_Tout[point_number] = error_t;
            gammaout[point_number] = gamma;
            Timeout[point_number] = t;
            Error_Obout[point_number] = error_ob;
            /* Error_Obout[point_number] = DGR * gamma_command; */
        }
    }

```



```

    getch();

    /* deallocate array memory */
    free_dvector(xplan,1,num_plan_points);
    free_dvector(yplan,1,num_plan_points);
    free_vector(Xout,1,max_num_out);
    free_vector(Yout,1,max_num_out);
    free_vector(Tout,1,max_num_out);
    free_vector(Error_Dout,1,max_num_out);
    free_vector(Error_Tout,1,max_num_out);
    free_vector(gammaout,1,max_num_out);
    free_vector(Timeout,1,max_num_out);

    _setvideomode( _DEFAULTMODE );
}

```

NOTE: The following code, used for memory allocation and array (vector) definition, is taken from *Numerical Recipes in C: The Art of Scientific Computing*, Second Edition, by W.H. Press, S.A Teukolsky, W.T. Vetterling and B.P. Flannery, Cambridge University Press, 1992. This code is referenced using the files *nrutil.c* in the main program, and *nrutil.h* in the sub-programs listed in later sections.

```

#include <malloc.h>
#include <stdio.h>

void nrerror(error_text)
char error_text[];
{
    void exit();

    fprintf(stderr,"Numerical Recipes run-time error...\n");
    fprintf(stderr,"%s\n",error_text);
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

```

```

float *vector(nl,nh)
int nl,nh;
{
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

```



```

int *ivector(nl,nh)
int nl,nh;
{
    int *v;

    v = (int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) perror("allocation failure in ivector()");
    return v-nl;
}

```

```

double *dvector(nl,nh)
int nl,nh;
{
    double *v;

    v = (double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) perror("allocation failure in dvector()");
    return v - nl;
}

```

```

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
    free((char*) (v + nl));
}

```

```

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
    free((char*) (v + nl));
}

```

```

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
    free((char*) (v + nl));
}

```

B.2 Path-Tracking Error: *patherr.c*

The following code is used to calculate the lateral displacement and orientation errors, ϵ_d (error_d) and ϵ_θ (error_t), respectively, when called by the main simulation program. Figure B.2 illustrates the intermediate calculation parameters of interest.

FIGURE B.2: Tracking error calculation geometry

```

#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>

#include "\\qc25\\nrc\\recipes\\nrutil.h"

void find_error(double xplan[], double yplan[], int num_plan_points, double xnow, double ynow,
double theta_now, double *error_d, double *error_t)
{
    int i,j,p1,p2;
    double length,closest,b,rover_to_last,rover_to_next,phi_a,phi_b;

    /* FIND CLOSEST POINT TO ROVER CENTRE */
    j = 1; /* assume first planned point is closest */
    closest = pow (pow(xnow-xplan[1],2) + pow(ynow-yplan[1],2),.5);

    for(i= 1;i<=num_plan_points; ++i) { /* check each point */
        length = pow (pow(xnow-xplan[i],2) + pow(ynow-yplan[i],2),.5);

        if (length < closest) { /* check if current point is closer */
            closest = length;
            j = i;
        }
    }

    /* FIND NEXT CLOSEST POINT and determine p1, p2 and b */
    if (j==1 || j==num_plan_points) {
        if (j==1) {
            p1 = 1;
            p2 = 2;
            b = closest;
        }
        else {
            p1 = num_plan_points - 1;
            p2 = num_plan_points;
            b = pow (pow(xnow-xplan[p1],2) + pow(ynow-yplan[p1],2),.5);
        }
    }

    else {
        rover_to_last = pow (pow(xnow-xplan[j-1],2) + pow(ynow-yplan[j-1],2),.5);
        rover_to_next = pow (pow(xnow-xplan[j+1],2) + pow(ynow-yplan[j+1],2),.5);
    }
}

```

```

        if (rover_to_last < rover_to_next) {
            p1 = j-1;
            p2 = j;
            b = rover_to_last;
        }
        else {
            p1 = j;
            p2 = j+1;
            b = closest;
        }
    }

    /* FIND SLOPE ANGLES phi_a & phi_b and calculate *error_d & *error_t */
    phi_a = atan2(yplan[p2]-yplan[p1],xplan[p2]-xplan[p1]);

    if (b == 0.0) /* ie. rover is directly over p1 */
        *error_d = 0.0;
    else {
        phi_b = atan2(ynow-yplan[p1],xnow-xplan[p1]);
        *error_d = b * sin(phi_b - phi_a);
    }
    *error_t = theta_now - phi_a;
}

```

B.3 Obstacle-Avoidance Error: *avoiderr.c*

The following code is used to calculate the individual error (potential) at each of the seven Points Subject to Potential (PSP's), as discussed in Chapter 4 and illustrated in Figure 4.1. The total avoidance error, ϵ_A (error_ob), which is the sum of the individual PSP errors, is the only quantity returned to the main program.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>
```

```
#include "qc25\src\recipes\nrutil.h"
```

```
void find_avoid_error(double obstacle_x[], double obstacle_y[], int num_obstacle_points, double xnow,
double ynow, double theta_now, double safe_obstacle_dist, double *error_ob)
```

```
{
    int i,j,p1,p2;
    double length,closest,obstacle_direction,lambda,obstacle_dist,temp;
    double boundary_error,error_centre,Xpop,Ypop,error_front_left,max_lambda;
    double error_centre_left,error_front_right,error_centre_right,DGR;
    double error_front_centre,error_rear_left,error_rear_right;

    DGR = 57.29577951; /* degrees per radian */

    obstacle_direction = 0.0;
    lambda = 0.0;
    max_lambda = 90 / DGR;

    *error_ob = 0;

    /* FIND ERROR SIGNAL error_front_left */
    error_front_left = 0;
    Xpop = 2 * cos(theta_now) - sin(theta_now) + xnow;
    Ypop = 2 * sin(theta_now) + cos(theta_now) + ynow;

    j = 1; /* assume first obstacle point is closest */
    closest = pow (pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

    for(i= 1;i<=num_obstacle_points;++i) { /* check each point */
        length = pow (pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

        if (length < closest) { /* check if current point is closer */
            closest = length;
            j = i;
        }
    }

    boundary_error = safe_obstacle_dist - closest;
    obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
    lambda = obstacle_direction - theta_now;
```

```

if ((boundary_error > 0) && (lambda > 0) && (lambda < max_lambda))
    error_front_left = boundary_error * (max_lambda - lambda);

/* FIND ERROR SIGNAL error_centre_left */
error_centre_left = 0;
Xpop = - sin(theta_now) + xnow;
Ypop = cos(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow (pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i= 1;i<=num_obstacle_points; ++i) { /* check each point */
    length = pow (pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
lambda = obstacle_direction - theta_now;

if ((boundary_error > 0) && (lambda > 0) && (lambda < max_lambda))
    error_centre_left = boundary_error * (max_lambda - lambda);

/* FIND ERROR SIGNAL error_front_right */
error_front_right = 0;
Xpop = 2 * cos(theta_now) + sin(theta_now) + xnow;
Ypop = 2 * sin(theta_now) - cos(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow (pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i= 1;i<=num_obstacle_points; ++i) { /* check each point */
    length = pow (pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
lambda = obstacle_direction - theta_now;

```

```

if ((boundary_error > 0) && (lambda < 0) && (lambda > -max_lambda))
    error_front_right = boundary_error * (-max_lambda - lambda);

/* FIND ERROR SIGNAL error_centre_right */
error_centre_right = 0;
Xpop = sin(theta_now) + xnow;
Ypop = -cos(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow(pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i = 1;i <= num_obstacle_points; ++i) { /* check each point */
    length = pow(pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
lambda = obstacle_direction - theta_now;

if ((boundary_error > 0) && (lambda < 0) && (lambda > -max_lambda))
    error_centre_right = boundary_error * (-max_lambda - lambda);

/* FIND ERROR SIGNAL error_front_centre */
error_front_centre = 0;
Xpop = 2 * cos(theta_now) + xnow;
Ypop = 2 * sin(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow(pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i = 1;i <= num_obstacle_points; ++i) { /* check each point */
    length = pow(pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
lambda = obstacle_direction - theta_now;

```

```

if ((boundary_error > 0) && (lambda > 0) && (lambda < max_lambda))
    error_front_centre = boundary_error * (max_lambda - lambda);
if ((boundary_error > 0) && (lambda <= 0) && (lambda > -max_lambda))
    error_front_centre = boundary_error * (-max_lambda - lambda);

/* FIND ERROR SIGNAL error_rear_left */
error_rear_left = 0;
Xpop = -2 * cos(theta_now) - sin(theta_now) + xnow;
Ypop = -2 * sin(theta_now) + cos(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow(pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i= 1;i<=num_obstacle_points; ++i) { /* check each point */
    lengthi = pow(pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);
lambda = obstacle_direction - theta_now;

if ((boundary_error > 0) && (lambda > 0) && (lambda < max_lambda))
    error_rear_left = boundary_error * (max_lambda - lambda);

/* FIND ERROR SIGNAL error_rear_right */
error_rear_right = 0;
Xpop = -2 * cos(theta_now) + sin(theta_now) + xnow;
Ypop = -2 * sin(theta_now) - cos(theta_now) + ynow;

j = 1; /* assume first obstacle point is closest */
closest = pow(pow(Xpop-obstacle_x[1],2) + pow(Ypop-obstacle_y[1],2),.5);

for(i= 1;i<=num_obstacle_points; ++i) { /* check each point */
    length = pow(pow(Xpop-obstacle_x[i],2) + pow(Ypop-obstacle_y[i],2),.5);

    if (length < closest) { /* check if current point is closer */
        closest = length;
        j = i;
    }
}

boundary_error = safe_obstacle_dist - closest;
obstacle_direction = atan2(obstacle_y[j]-Ypop,obstacle_x[j]-Xpop);

```



```

lambda = obstacle_direction * theta_now;

if ((boundary_error > 0) && (lambda < 0) && (lambda > -max_lambda))
    error_rear_right = boundary_error * (-max_lambda - lambda);

*error_ob = error_front_centre + error_front_left + error_centre_left + error_rear_left
            + error_front_right + error_centre_right + error_rear_right;
}

```

B.4 Graphical Output: *plotpath.c*

The following sub-program is called from the main program to plot the planned path, defined obstacles, and rover path, once the iterative simulation process has been completed. It should be noted that the plotting routines (statements) used in this sub-program are dependent on the computing platform and output hardware.

```

#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>

#include "qc25\nrc\recipes\nrutil.h"

void plot_path(int num_rovers, int border, double xplotsize, double xplan[], double yplan[], int
num_plan_points, float xloc[], float yloc[], float theta[], int num_rover_points, double obstacle_x[], double
obstacle_y[], int num_obstacle_points)
{
    int i, j, rover_interval;
    double largest_x, largest_y;
    struct xycoord view;
    struct wxycoord window;
    FILE *fp;

    _clearscreen( _GCLEARSCREEN );

```

```

if( !_setvideomode( _MAXRESMODE ) ) /* Find a valid graphics mode */
exit( 1 ); /* No graphics available */
_settextrows(_MAXTEXTROWS);

if (xplotsize < 9.0) { /* set horizontal dimension of graphics window */
    largest_x = 0;
    largest_y = 0;
    for (i = 1; i <= num_plan_points; ++i) {
        if (xplan[i] > largest_x)
            largest_x = xplan[i];
        if (yplan[i] > largest_y)
            largest_y = yplan[i];
    }
    if (largest_y > largest_x * 3/4)
        xplotsize = (largest_y + 5) * 4/3;
    else
        xplotsize = largest_x + 5;
}

_setwindow(1,-2.5,-2.5, xplotsize - 2.5, xplotsize * 3/4 - 2.5);

if (border == 1) { /* draw a border */
    _moveto _w(-2.5,-2.5);
    _lineto _w(-2.5,xplotsize*3/4-2.5);
    _lineto _w(xplotsize-2.5,xplotsize*3/4-2.5);
    _lineto _w(xplotsize-2.5,-2.5);
    _lineto _w(-2.5,-2.5);
}

/* PLOT PLANNED PATH POINTS */
for(i=1;i<=num_plan_points;++i) {
    view = _getviewcoord_w(xplan[i],yplan[i]);
    _setpixel(view.xcoord,view.ycoord);
    _setpixel(view.xcoord+1,view.ycoord);
    _setpixel(view.xcoord-1,view.ycoord);
    _setpixel(view.xcoord,view.ycoord+1);
    _setpixel(view.xcoord,view.ycoord-1);
    _setpixel(view.xcoord+1,view.ycoord+1);
    _setpixel(view.xcoord-1,view.ycoord-1);
    _setpixel(view.xcoord+1,view.ycoord-1);
    _setpixel(view.xcoord-1,view.ycoord+1);
}

/* connect planned points with lines */
_moveto _w(xplan[1],yplan[1]);
for(i=2;i<=num_plan_points;++i)
    _lineto _w(xplan[i],yplan[i]);

```

```

/* PLOT OBSTACLE POINTS */
if (num_obstacle_points != 0) {
    for(i=1;i <= num_obstacle_points; ++i) {
        view = _getviewcoord_w(obstacle_x[i],obstacle_y[i]);
        _setpixel(view.xcoord,view.ycoord);
        _setpixel(view.xcoord+1,view.ycoord);
        _setpixel(view.xcoord-1,view.ycoord);
        _setpixel(view.xcoord,view.ycoord+1);
        _setpixel(view.xcoord,view.ycoord-1);
    }
}

/* PLOT ROVER CENTRE POINTS */
for(i=1;i <= num_rover_points; ++i) _setpixel_w(xloc[i],yloc[i]);

view = _getviewcoord_w(xloc[1],yloc[1]);
_setpixel(view.xcoord,view.ycoord);
_setpixel(view.xcoord+1,view.ycoord);
_setpixel(view.xcoord-1,view.ycoord);
_setpixel(view.xcoord,view.ycoord+1);
_setpixel(view.xcoord,view.ycoord-1);

view = _getviewcoord_w(xloc[num_rover_points],yloc[num_rover_points]);
_setpixel(view.xcoord,view.ycoord);
_setpixel(view.xcoord+1,view.ycoord);
_setpixel(view.xcoord-1,view.ycoord);
_setpixel(view.xcoord,view.ycoord+1);
_setpixel(view.xcoord,view.ycoord-1);

/* DRAW ROVERS */
if (num_rovers >= 2) {

    rover_interval= (int) ((float) (num_rover_points - 1) / (float) (num_rovers - 1) + .5);
    if (rover_interval < 1) rover_interval = 1;

    for(i=0;i <= (num_rovers - 2); ++i) {
        j = 1 + i * rover_interval;

        moveto_w(xloc[j]+2*cos(theta[j])-1*sin(theta[j]),yloc[j]+2*sin(theta[j])+1*cos(theta[j]));

        _lineto_w(xloc[j]+2*cos(theta[j])+1*sin(theta[j]),yloc[j]+2*sin(theta[j])-1*cos(theta[j]));

        _lineto_w(xloc[j]-2*cos(theta[j])+1*sin(theta[j]),yloc[j]-2*sin(theta[j])-1*cos(theta[j]));

        lineto_w(xloc[j]-2*cos(theta[j])-1*sin(theta[j]),yloc[j]-2*sin(theta[j])+1*cos(theta[j]));

        lineto_w(xloc[j]+2*cos(theta[j])-1*sin(theta[j]),yloc[j]+2*sin(theta[j])+1*cos(theta[j]));
    }
}

```

```

        j = num_rover_points; /* draw rover in the final position */
        _moveto_w(xloc[j] + 2*cos(theta[j]) - 1*sin(theta[j]), yloc[j] + 2*sin(theta[j]) + 1*cos(theta[j]));
        _lineto_w(xloc[j] + 2*cos(theta[j]) + 1*sin(theta[j]), yloc[j] + 2*sin(theta[j]) - 1*cos(theta[j]));
        _lineto_w(xloc[j] - 2*cos(theta[j]) + 1*sin(theta[j]), yloc[j] - 2*sin(theta[j]) - 1*cos(theta[j]));
        _lineto_w(xloc[j] - 2*cos(theta[j]) - 1*sin(theta[j]), yloc[j] - 2*sin(theta[j]) + 1*cos(theta[j]));
        _lineto_w(xloc[j] + 2*cos(theta[j]) - 1*sin(theta[j]), yloc[j] + 2*sin(theta[j]) + 1*cos(theta[j]));
    }

    getch();
}

```

Appendix C: Dynamic Equations of Motion

The following is a development of the equations of motion for a two-wheel rover (bicycle), illustrated in Figure C.1, that is in fact a simplification of a four-wheel, symmetric-wagon-steering design. This design is in turn similar to the six-wheel vehicle design adopted and used for simulation in Chapters 3 and 4. The procedure used here is quite similar to that presented in [20] for a tricycle-type rover model. It is intended that the developed equations may be used as a reference or starting point, possibly in an extension of the current work, for the study of the characteristics of dynamic forces as they relate to the skidding of wheels, or perhaps for inverse kinematics and the determination of required drive torques for a given acceleration profile.

FIGURE C.1: Two-wheel model of six-wheel rover.

Following the approach used in [20], all forces, accelerations and velocities will be rotated, by rover orientation θ , from the world reference coordinate (X,Y) frame, to a fixed coordinate (x,y) frame that is instantaneously coincident (IC) with the rover body. Figure C.1 illustrates the planar geometry considered. It is desired that all expressions should contain absolute vector values of the variables, expressed in the IC coordinate system of the rover. Note that subscripts 1 and 2 refer the front and rear wheels, respectively, while unless otherwise stated, all other variables refer to the centre of gravity of the rover, P_C . Also note that for the forces, F, subscripts A and B refer to directions that are parallel and normal to the rolling wheels, respectively.

Kinematic Relationships

The velocities (v) and accelerations (a) of the front (1) and rear (2) wheel points may be expressed in terms of the velocities and accelerations of the centre-point, P_C , by

$$\begin{aligned} v_{1x} &= v_x \\ v_{1y} &= v_y + L_1 \dot{\theta} \end{aligned} \quad (C.1)$$

$$\begin{aligned} a_{1x} &= a_x - L_1 \dot{\theta}^2 \\ a_{1y} &= a_y + L_1 \ddot{\theta} \end{aligned} \quad (C.2)$$

$$\begin{aligned} v_{2x} &= v_x \\ v_{2y} &= v_y - L_2 \dot{\theta} \end{aligned} \quad (C.3)$$

$$\begin{aligned} a_{2x} &= a_x + L_2 \dot{\theta}^2 \\ a_{2y} &= a_y - L_2 \ddot{\theta} \end{aligned} \quad (C.4)$$

where $\dot{\theta}$ and $\ddot{\theta}$ are the rotational rate and rotational acceleration of the vehicle body,

respectively, and L_1 and L_2 are the fixed wheel-base lengths illustrated in Figure C.1.

Kinetic Relationships

Equations of motion may be found through application of a force balance on the vehicle body, as well as (in this case) on each of the rover's wheels. The translational equations of motion in the IC reference frame are given by

$$\begin{aligned} m_r a_x &= F_{1A} \cos \gamma_1 - F_{1B} \sin \gamma_1 + F_{2A} \cos \gamma_2 + F_{2B} \sin \gamma_2 \\ m_r a_y &= F_{1A} \sin \gamma_1 + F_{1B} \cos \gamma_1 - F_{2A} \sin \gamma_2 + F_{2B} \cos \gamma_2 \end{aligned} \quad (C.5)$$

where m_r is the mass of the rover body, and γ is the steering angle. Balancing the moments acting about the centre-point, P_C , yields the rotational motion equation

$$I_r \ddot{\theta} = F_{1A} L_1 \sin \gamma_1 + F_{1B} L_1 \cos \gamma_1 + F_{2A} L_2 \sin \gamma_2 - F_{2B} L_2 \cos \gamma_2 \quad (C.6)$$

where I_r is the rotational inertia of the vehicle body.

The translational equations of motion for the front wheel centre-point (1) are

$$\begin{aligned} m_w a_{1x} &= G_{1A} \cos \gamma_1 - F_{1A} \cos \gamma_1 + F_{1B} \sin \gamma_1 - G_{1B} \sin \gamma_1 \\ m_w a_{1y} &= G_{1B} \cos \gamma_1 - F_{1B} \cos \gamma_1 + G_{1A} \sin \gamma_1 - F_{1A} \sin \gamma_1 \end{aligned} \quad (C.7)$$

where m_w is the mass of the wheel, and F and G are the vehicle and ground reaction forces, respectively, as illustrated in Figure C.2. The applied torque is related to the ground reaction force through balancing of moments in the rolling plane, as given by

$$I_w \dot{\omega}_1 = T_1 - G_{1A} r \quad (C.8)$$

FIGURE C.2: Body-reaction, ground force and drive torque on the wheels.

where I_w is the rotational inertia, $\dot{\omega}_1$ is the rotational acceleration, r is the wheel radius and T_1 is the drive torque applied by the front wheel motor.

The motion equations for the rear wheel may be (similarly) found to be given by

$$m_w a_{2x} = G_{2A} \cos \gamma_2 - F_{2A} \cos \gamma_2 - F_{2B} \sin \gamma_2 + G_{2B} \sin \gamma_2 \quad (C 9)$$

$$m_w a_{2y} = G_{2B} \cos \gamma_2 - F_{2B} \cos \gamma_2 - G_{2A} \sin \gamma_2 + F_{2A} \sin \gamma_2$$

$$I_w \dot{\omega}_2 = T_2 - G_{2A} r \quad (C 10)$$

where $\dot{\omega}_2$ is the rotational acceleration and T_2 is the torque applied to the rear wheel.

Perfect Rolling Constraints

To facilitate further study of the inertial forces involved in wheel skidding, one may first assume that perfect rolling (no skidding) takes place. This allows for the development of a set of constraints that may be used to solve for the reaction (F) and

ground (G) traction (frictional) forces, as functions of the state and inputs.

The velocity of front wheel centroid, in world coordinates, is given by

$$\begin{aligned} v_{1X} &= r\omega_1 \cos(\theta + \gamma_1) \\ v_{1Y} &= r\omega_1 \sin(\theta + \gamma_1) \end{aligned} \quad (C.11)$$

With respect to IC coordinate frame, this velocity is given by:

$$\begin{aligned} v_{1x} &= r\omega_1 \cos \gamma_1 \\ v_{1y} &= r\omega_1 \sin \gamma_1 \end{aligned} \quad (C.12)$$

Substitution of expression C.1 into C.12 moves the constraint from the front wheel to the centre of the rover, yielding

$$\begin{aligned} v_x &= r\omega_1 \cos \gamma_1 \\ v_y + L_1 \dot{\theta} &= r\omega_1 \sin \gamma_1 \end{aligned} \quad (C.13)$$

or alternatively, upon rearrangement of C.13:

$$\begin{aligned} v_x \tan \gamma_1 &= v_y + L_1 \dot{\theta} \\ r\omega_1 &= \sqrt{v_x^2 + (v_y + L_1 \dot{\theta})^2} \end{aligned} \quad (C.14)$$

Differentiation of C.11 yields the acceleration of front wheel in world coordinates:

$$\begin{aligned} a_{1X} &= r\dot{\omega}_1 \cos(\theta + \gamma_1) - r\omega_1(\dot{\theta} + \dot{\gamma}_1) \sin(\theta + \gamma_1) \\ a_{1Y} &= r\dot{\omega}_1 \sin(\theta + \gamma_1) + r\omega_1(\dot{\theta} + \dot{\gamma}_1) \cos(\theta + \gamma_1) \end{aligned} \quad (C.15)$$

With respect to the IC coordinate frame, the acceleration is given by:

$$\begin{aligned} a_{1x} &= r\dot{\omega}_1 \cos \gamma_1 - r\omega_1(\dot{\theta} + \dot{\gamma}_1) \sin \gamma_1 \\ a_{1y} &= r\dot{\omega}_1 \sin \gamma_1 + r\omega_1(\dot{\theta} + \dot{\gamma}_1) \cos \gamma_1 \end{aligned} \quad (C 16)$$

Substitution of expressions C.2 into C.16 moves this constraint from the front wheel to the centre of the rover, yielding

$$\begin{aligned} a_x - L_1 \dot{\theta}^2 &= r\dot{\omega}_1 \cos \gamma_1 - r\omega_1(\dot{\theta} + \dot{\gamma}_1) \sin \gamma_1 \\ a_y + L_1 \ddot{\theta} &= r\dot{\omega}_1 \sin \gamma_1 + r\omega_1(\dot{\theta} + \dot{\gamma}_1) \cos \gamma_1 \end{aligned} \quad (C 17)$$

or alternatively, upon rearrangement of C.17:

$$\begin{aligned} r\dot{\omega}_1 &= (a_x - L_1 \dot{\theta}^2) \cos \gamma_1 + (a_y + L_1 \ddot{\theta}) \sin \gamma_1 \\ r\omega_1(\dot{\theta} + \dot{\gamma}_1) &= -(a_x - L_1 \dot{\theta}^2) \sin \gamma_1 + (a_y + L_1 \ddot{\theta}) \cos \gamma_1 \end{aligned} \quad (C 18)$$

Following a similar set of steps as those used above, the following constraints may be determined for the rear wheel under perfect rolling conditions:

$$\begin{aligned} v_{2x} &= r\omega_2 \cos(\theta - \gamma_2) \\ v_{2y} &= r\omega_2 \sin(\theta - \gamma_2) \end{aligned} \quad (C 19)$$

$$\begin{aligned} v_{2x} &= r\omega_2 \cos \gamma_2 \\ v_{2y} &= -r\omega_2 \sin \gamma_2 \end{aligned} \quad (C 20)$$

$$\begin{aligned} v_x &= r\omega_2 \cos \gamma_2 \\ v_y - L_2 \dot{\theta} &= -r\omega_2 \sin \gamma_2 \end{aligned} \quad (C 21)$$

$$\begin{aligned} v_x \tan \gamma_2 &= -v_y + L_2 \dot{\theta} \\ r\omega_2 &= \sqrt{v_x^2 + (v_y - L_2 \dot{\theta})^2} \end{aligned} \quad (C.22)$$

$$\begin{aligned} a_{2X} &= r\dot{\omega}_2 \cos(\theta - \gamma_2) - r\omega_2(\dot{\theta} - \dot{\gamma}_2) \sin(\theta - \gamma_2) \\ a_{2Y} &= r\dot{\omega}_2 \sin(\theta - \gamma_2) + r\omega_2(\dot{\theta} - \dot{\gamma}_2) \cos(\theta - \gamma_2) \end{aligned} \quad (C.23)$$

$$\begin{aligned} a_{2x} &= r\dot{\omega}_2 \cos \gamma_2 + r\omega_2(\dot{\theta} - \dot{\gamma}_2) \sin \gamma_2 \\ a_{2y} &= -r\dot{\omega}_2 \sin \gamma_2 + r\omega_2(\dot{\theta} - \dot{\gamma}_2) \cos \gamma_2 \end{aligned} \quad (C.24)$$

$$\begin{aligned} a_x + L_2 \dot{\theta}^2 &= r\dot{\omega}_2 \cos \gamma_2 + r\omega_2(\dot{\theta} - \dot{\gamma}_2) \sin \gamma_2 \\ a_y - L_2 \ddot{\theta} &= -r\dot{\omega}_2 \sin \gamma_2 + r\omega_2(\dot{\theta} - \dot{\gamma}_2) \cos \gamma_2 \end{aligned} \quad (C.25)$$

$$\begin{aligned} r\dot{\omega}_2 &= (a_x + L_2 \dot{\theta}^2) \cos \gamma_2 - (a_y - L_2 \ddot{\theta}) \sin \gamma_2 \\ r\omega_2(\dot{\theta} - \dot{\gamma}_2) &= (a_x + L_2 \dot{\theta}^2) \sin \gamma_2 + (a_y - L_2 \ddot{\theta}) \cos \gamma_2 \end{aligned} \quad (C.26)$$

Application of the Equations

The constraint equations, C.11-C.26, may be combined with the equations of motion, C.5-C.10, to determine the reaction forces, F , ground forces, G , or required torques, T , given the known state and inputs of the two-wheel system. Although investigation of dynamics was not pursued in this work, consider as an example the system in C.27, which was developed during the initial stages of this project, using the above relationships for the two-wheel model. In this case, measured values of T_1 , T_2 , γ_1 , γ_2 , $\dot{\gamma}_1$, $\dot{\gamma}_2$, ω_1 , ω_2 and $\dot{\theta}$ may be substituted into the system (C.27) to solve for the rover reaction forces, F , which may then be used, with the equations presented above, to determine the ground forces, G . These forces may be calculated at each state

measurement update, and then compared to observations of the points at which wheel slippage is detected or otherwise observed during experiment. Other formulations and applications of the above equations are possible, as mentioned, however this system illustrates the general form that one might expect for some applications. (It is noted that this particular system is simplified considerably by assuming that $L_1 = L_2$ and $\gamma_1 = \gamma_2$.)

$$\begin{bmatrix} -k_8 c_1 s_1 + k_9 c_2 s_1 & -k_9 s_1 s_2 - k_8 c_1 s_2 & -k_{10} c_1 s_2 & k_{10} c_1^2 - k_8 - k_{10} - k_{11} \\ k_{14} c_1 c_2 + k_{15} s_1 s_2 & -k_{14} c_1 s_2 + k_{15} s_1 c_2 & -k_{16} c_1^2 - k_{15} s_1^2 & -k_3 c_1 s_1 \\ -k_{17} c_2 s_2 & -k_{17} c_2^2 + k_8 + k_{11} + k_{17} & -k_8 c_2 s_1 + k_9 c_1 s_2 & k_8 c_1 c_2 + k_9 s_1 s_2 \\ k_{21} c_2^2 + k_{15} s_2^2 & -k_{19} c_2 s_2 & -k_{14} c_1 c_2 - k_{15} s_1 s_2 & -k_{14} c_2 s_1 + k_{15} s_2 c_1 \end{bmatrix} \begin{bmatrix} F_{2B} \\ F_{2A} \\ F_{1B} \\ F_{1A} \end{bmatrix} = \begin{bmatrix} -k_{13} T_1 - k_{12} \dot{\theta}^2 c_1 \\ -k_4 (\dot{\theta} + \dot{\gamma}) r \omega_1 + k_4 k_7 \dot{\theta}^2 s_1 \\ k_{13} T_2 - k_{18} \dot{\theta}^2 c_2 \\ k_4 (\dot{\theta} - \dot{\gamma}) r \omega_2 - k_4 k_{20} \dot{\theta}^2 s_2 \end{bmatrix} \quad (C.27)$$

where: $k_1 = \frac{1}{m_r}$, $k_2 = I_w + m_w r^2$, $k_3 = L_1^2$, $k_4 = I_r$, $k_5 = r$, $k_6 = r^2$, $k_7 = L_1$,

$$k_8 = k_1 k_2 k_4, \quad k_9 = L_1 L_2 k_2 - k_8, \quad k_{10} = L_1^2 k_2, \quad k_{11} = k_4 k_6, \quad k_{12} = L_1 k_2 k_4,$$

$$k_{13} = k_4 k_5, \quad k_{14} = L_1 L_2 - k_1 k_4, \quad k_{15} = k_1 k_4, \quad k_{16} = L_1^2 + k_1 k_4, \quad k_{17} = L_2^2 k_2,$$

$$k_{18} = L_2 k_2 k_4, \quad k_{19} = L_2^2, \quad k_{20} = L_2, \quad k_{21} = L_2^2 + k_1 k_4$$

$$c_1 = \cos \gamma_1, \quad c_2 = \cos \gamma_2, \quad s_1 = \sin \gamma_1, \quad s_2 = \sin \gamma_2$$

Appendix D: Computer Code for Signal Fusion

The following routines were used to post-process the experimental data presented in Chapter 5, as part of the development and testing of the signal fusion technique discussed in that chapter. Although these routines have not been optimized for memory or speed, they are quite simple, and can be adapted to real-time processing for the sensor fusion application discussed in Chapter 5. The computer code itself is written in the C language, and was compiled using Microsoft QuickC. An 80386/387 microcomputer was used during compilation and execution of the code.

D.1 Differentiation: *derive.c*

The following routine calculates the derivative of a signal (data stream) using a simple backwards difference, described by

$$\dot{x}_k = \frac{x_k - x_{k-1}}{h} \quad (4.1)$$

where, at sample moment k , x_k and x_{k-1} are the current and previous data points, $\dot{x}_k$ is the current derivative and h is the sample interval time.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>
```

```

#include "\qc25\nrc\recipes\nrutil.c" /* utility funtions */

main() {
    double *y_sig, *y_derivative;
    double h;
    int i,k, num_ins_points, max_data_out;
    char dummy[200];
    FILE *fp;

    _clearscreen(_GCLEARSCREEN);
    _settextrows(_MAXTEXTROWS);

    h = 0.01; /* time interval */

    /* READ DATA FILE ***** */
    printf("\n");
    fp=fopen("encpos","r");

    num_ins_points = 1000;

    /* allocate memory for the data */
    y_sig = dvector(1,num_ins_points);
    y_derivative = dvector(1,num_ins_points);

    for (i=1; i<=num_ins_points; ++i)
        fscanf(fp, "%22lf ", &y_sig[i]);
    fclose(fp);

    /* TAKE DERIVATIVE ***** */
    y_derivative[1] = (y_sig[2] - y_sig[1]) / h; /* forward difference at first point */

    for (k=2; k<=num_ins_points; ++k)
        y_derivative[k] = (y_sig[k] - y_sig[k-1]) / h; /* backward difference */

    /* WRITE OUTPUT to file ***** */
    fp=fopen("encvel","w");
    for (i=1; i<=num_ins_points; ++i) {
        fprintf(fp, "%9.4lf \n", y_derivative[i]);
        printf    ("%9.4lf \n", y_derivative[i]);
    }
    fclose(fp);

    getch();

    _setvideomode(_DEFAULTMODE );
}

```

D.2 Integration: *integ.c*

The following routine integrates a signal (data stream) using the simple Runge-Kutta mid-point algorithm, described by

$$x_k = h \frac{\dot{x}_k + \dot{x}_{k-1}}{2} + x_{k-1} \quad (4.2)$$

where, at sample moment k , $\dot{x}_k$ and $\dot{x}_{k-1}$ are the current and previous samples from the input signal, respectively, x_k and x_{k-1} are the current and previous discrete-time values of the integrated signal, respectively, and h the sample interval time.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>

#include "qc25\nrc\recipes\nrutil.c" /* utility funtions */

main() {
    double *y_sig, *y_int;
    double h;
    int i,k, num_ins_points, max_data_out;
    char dummy[200];
    FILE *fp;

    _clearscreen(_GCLEARSCREEN);
    _settextrows(_MAXTEXTROWS);

    h = 0.01;

    /* READ DATA FILE ***** */
    printf("\n");
    fp = fopen("insvel", "r");
```

```

num_ins_points = 1000;

/* allocate memory for the data */
y_sig = dvector(1,num_ins_points);
y_int = dvector(1,num_ins_points);

for (i=1; i<=num_ins_points; ++i) {
    fscanf(fp, "%22lf ", &y_sig[i]);
}
fclose(fp);

/* INTEGRATE ***** */
y_int[1] = h * (0.0 + y_sig[1]) / 2; /* first input point occurs at t = h, no data at t = 0.0 */

for (k=2; k<=num_ins_points; ++k)
    y_int[k] = h * (y_sig[k] + y_sig[k-1]) / 2 + y_int[k-1];

/* WRITE OUTPUT to file ***** */
fp=fopen("inspos","w");
for (i=1; i<=num_ins_points; ++i) {
    fprintf(fp, "%9.4lf \n", y_int[i]);
    printf    ("%9.4lf \n", y_int[i]);
}
fclose(fp);

getch();

_setvideomode( _DEFAULTMODE );
}

```


D.3 Fusion: *fuse.c*

The following program is used to fuse wheel-encoder and accelerometer (INS) signals (data streams), according to the flow-chart shown in Figure 5.10, and the algorithm in expressions 5.13. With reference to this flow-chart and algorithm, it should be noted that the data-streams (files) ENCpos, ENCvel, INSacc and DIFFacc are loaded prior to fusion, where $\text{DIFFacc} = \varepsilon = \text{ENCacc} - \text{INSacc}$. It should also be noted that INSacc is integrated "on-the-fly"; neither INSvel nor INSpos are loaded prior to fusion. Finally, this code was used to post-process the experimental test data. A real-time implementation would carry-out all signal processing operations, beginning with only the input data-streams labelled ENCpos and INSacc, shown in the flow-chart in Figure 5.10.

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <malloc.h>

#include "\qc25\nrc\recipes\nrutil.c" /* utility funtions */

main() {
    float *p,*v,*encpos,*insacc,diffacc_max,delta_encpos;
    float *encvel,encvel_old,insvel_old,encpos_old,insacc_old,insvel,h;
    float delta_insvel,delta_inspos,A,L,encvel_hat_old,encvel_bar,encvel_hat;
    float vel_percent_error_max,v_bar,p_bar;
    float diffacc1,diffacc2,diffacc3,diffacc4,diffacc_old;
    float delta_pos,*diffacc;
    float B,C,W_sigma,V_sigma;
    int i,k, num_ins_points, max_data_out, ins_used;
    char dummy[200];
    FILE *fp;

    clrscr(_GCLEARSCREEN);
    _settextrows(_MAXTEXTROWS);
```

```

/* READ PARAMETER FILE ***** */
fp = fopen("filparms", "r"); /* not all loaded parameters are used in this code */
fscanf(fp, "%12f %[^\\n\\t]", &A, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &B, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &C, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &L, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &W_sigma, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &V_sigma, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &diffacc_max, dummy);
fscanf(fp, "%12f %[^\\n\\t]", &diffaccr_max, dummy);
fclose(fp);

h = 0.01; /* sample interval time */
num_ins_points = 1000;

/* allocate memory for the data */
p = vector(0, num_ins_points);
v = vector(0, num_ins_points);
encpos = vector(0, num_ins_points);
encvel = vector(0, num_ins_points);
insacc = vector(0, num_ins_points);
diffacc = vector(0, num_ins_points);

/* READ DATA FILES ***** */
printf("\\n");

fp = fopen("encpos", "r");
for (i = 1; i <= num_ins_points; ++i)
    fscanf(fp, "%22f ", &encpos[i]);
fclose(fp);

fp = fopen("encvel", "r");
for (i = 1; i <= num_ins_points; ++i)
    fscanf(fp, "%22f ", &encvel[i]);
fclose(fp);

fp = fopen("insacc", "r");
for (i = 1; i <= num_ins_points; ++i)
    fscanf(fp, "%22f ", &insacc[i]);
fclose(fp);

fp = fopen("diffacc", "r"); /* epsilon */
for (i = 1; i <= num_ins_points; ++i)
    fscanf(fp, "%22f ", &diffacc[i]);
fclose(fp);

/* INITIALIZE */
encvel_hat_old = 0.0;
p[0] = 0.0;

```

```

v[0] = 0.0;
encpos[0] = 0.0;
insacc[0] = 0.0;
encvel_old = 0.0;
insvel_old = 0.0;
encpos_old = 0.0;
insacc_old = 0.0;
ditracc_old = 0.0;
ins_used = 0;
diffacc1 = 0.0;
diffacc2 = 0.0;

```

```

/* FUSION ALGORITHM with additional KALMAN FILTERING of v[k] = encvel[k] */
for (k = 1; k <= num_ins_points; ++k) {
    if (fabs(diffacc[k]) <= diffacc_max) {

        v[k] = encvel[k]; /* encoder-based velocity update */

        v_bar = A * v[k-1]; /* kalman filter on encoder-based velocity update */
        v[k] = v_bar + L * (v[k] - v_bar);

        delta_pos = encpos[k] - encpos[k-1];
        p[k] = p[k-1] + delta_pos; /* encoder-based position update */
    }
    else {
        delta_insvel = h * (insacc[k] + insacc[k-1]) / 2;
        v[k] = v[k-1] + delta_insvel; /* INS-based velocity update */

        delta_pos = h * (v[k] + v[k-1]) / 2;
        p[k] = p[k-1] + delta_pos; /* INS-based position update */
    }
}

```

```

/* WRITE OUTPUT to file ***** */
fp = fopen("fuseout", "w");
for (i = 1; i <= num_ins_points; ++i) {
    fprintf(fp, "%9.4f %9.4f\n", p[i], v[i]);
    printf    ("%9.4f %9.4f\n", p[i], v[i]);
}
fclose(fp);
getch();
_setvideomode( _DEFAULTMODE );
}

```

Appendix E: Kalman Filtering

The following discussion is intended to illustrate the development of a Kalman filter that may be used to reduce the relative quantity of noise in a discrete signal. A design example will be presented, that is based on the encounter of a practical noise problem. The resulting filter is used in the sensor fusion work presented in Chapter 5.

E.1 The Kalman Filter

Following similar theoretical developments presented in [30,31,46], consider a system (vehicle motion) described by a discrete-time model and the state-space equations,

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A} \mathbf{x}(k) + \mathbf{B} \mathbf{w}_1(k) \\ \mathbf{y}(k) &= \mathbf{C} \mathbf{x}(k) + \mathbf{w}_2(k) \end{aligned} \quad (\text{E } 1)$$

where, at sample moment k , vectors $\mathbf{x}(k)$ and $\mathbf{y}(k)$ are the true state and the measured output, respectively, $\mathbf{w}_1(k)$ and $\mathbf{w}_2(k)$ represent process disturbance and measurement noise vectors, respectively, and $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are matrices describing the system.

The noise vectors, $\mathbf{w}_1(k)$ and $\mathbf{w}_2(k)$, are defined as random, white-noise sequences, with zero means, and magnitudes described by expected values of covariance, given by,

$$E \{ \mathbf{w}_1(k) \mathbf{w}_1^T(k) \} = \mathbf{Q}_1, \quad E \{ \mathbf{w}_2(k) \mathbf{w}_2^T(k) \} = \mathbf{Q}_2 \quad (\text{E } 2)$$

where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are diagonal matrices, containing the expected values of the variances,

for each of the respective elements in the noise vectors.

The Linear Quadratic Estimator (LQE) is a state estimator that uses minimization of the mean-square estimation error as an optimality criterion, as expressed by,

$$E \left\{ \left[\hat{y}(k) - y_T(k) \right]^2 \right\} = \min \quad (E.3)$$

where $\hat{y}(k)$ is the estimate of the true value of the measured system: output, $y_T(k) = Cx(k)$ is that true value, and the expected value of the square-difference between these signals is to be minimized. The LQE itself may be described by,

$$\begin{aligned} \hat{x}(k) &= \bar{x}(k) + K_f [y(k) - C\bar{x}(k)] \\ \bar{x}(k) &= A \hat{x}(k-1) \\ \hat{y}(k) &= C \hat{x}(k) \end{aligned} \quad (E.4)$$

where $\hat{x}(k)$ is the current estimate of the state vector, $\bar{x}(k)$ is a predicted state that is based on the system model, $y(k)$ is the noisy measurement signal, $\hat{y}(k)$ is the estimate of the true value of that signal, and K_f is a gain matrix. The basic principle of operation of expression E.4 is simply that, with smaller gains (K_f), the estimate, $\hat{x}$ (and hence $\hat{y}$), will be based more on the model prediction, $\bar{x}$, whereas with larger gains, the estimate will be based more on the sensor measurements, y .

The solution to the LQE design problem satisfies expression E.3, by considering the characteristics of the known (often estimated) noise matrices, Q_1 and Q_2 , in E.2. The result of such a design procedure is the *optimal* gain matrix, K_f , which yields an estimator that allows for detection of the effects of process disturbances, w_1 , while minimizing the effects of measurement noise, w_2 . The resulting noise-optimized estimator is most often referred to as a Kalman filter.

To design a Kalman filter, one must know the system matrices **A**, **B** and **C**, and the variances, diagonal elements of **Q**₁ and **Q**₂, of the process disturbance noise, **w**₁, and measurement (sensor) noise, **w**₂. Once **A**, **B**, **C**, **Q**₁ and **Q**₂ have been determined, the optimal or Kalman gain matrix, **K**_f, can be determined from the solution, matrix **P**, to the algebraic Riccati equation [31], given by

$$\mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T - \mathbf{P}\mathbf{C}^T\mathbf{Q}_2^{-1}\mathbf{C}\mathbf{P} + \mathbf{B}\mathbf{Q}_1\mathbf{B}^T = \mathbf{0} \quad (\text{E } 5)$$

where the Kalman gain itself is given by

$$\mathbf{K}_f = \mathbf{P}\mathbf{C}^T\mathbf{Q}_2^{-1} \quad (\text{E } 6)$$

In the example presented below, equations E.5 and E.6 are solved using MATLAB [49], where the Kalman gain, **K**_f, is determined directly from input of **A**, **B**, **C**, **Q**₁ and **Q**₂.

E.2 A First-Order Model

In many practical applications, a first-order low-pass filter (transfer function) can be used to approximate how a system will process an input [30]. Therefore, such a system may be described in the Laplace (s or frequency) domain by the expression,

$$G(s) = \frac{\omega_b}{s + \omega_b} \quad (\text{E } 7)$$

where the transfer function is defined as $G(s) = X(s)/U(s)$ (output/input), and ω_b is the bandwidth (-3dB cut-off frequency) of the filter. The continuous s-domain may be related to the discrete z-domain by the Forward Rectangular Rule [31], given by

$$s = \frac{z - 1}{T_s} \quad (\text{E.8})$$

where T_s is the discrete sampling interval time. Substitution of E.8 into E.7 yields the discrete transfer function of the system (low-pass filter),

$$G(z) = \frac{\omega_b T_s}{z - 1 + \omega_b T_s} \quad (\text{E.9})$$

where, if $G(z) = X(z)/U(z)$, then E.9 may be rearranged to yield:

$$z X(z) = (1 - \omega_b T_s) X(z) + \omega_b T_s U(z) \quad (\text{E.10})$$

Transformation of this last expression into the discrete time domain, yields:

$$x(k+1) = (1 - \omega_b T_s)x(k) + \omega_b T_s u(k) \quad (\text{E.11})$$

Comparison of expression E.11 with E.1 reveals that, for the first-order system:

$$A = 1 - \omega_b T_s, \quad B = \omega_b T_s \quad (\text{E.12})$$

To simplify the LQE formulation, it is noted that if $C = 1$ in E.1 and E.4, then the expression in E.4 can be rewritten as,

$$\begin{aligned} \hat{x}(k) &= \bar{x}(k) + K_f [x_m(k) - \bar{x}(k)] \\ \bar{x}(k) &= A \hat{x}(k-1) \end{aligned} \quad (\text{E.13})$$

where x_m represents a scalar stream of noisy measurement data, and $\hat{x}$ is the filtered output. Furthermore, the covariance matrices, Q_1 and Q_2 , may be given simply by,

$$Q_1 = w_{1rms}^2, \quad Q_2 = w_{2rms}^2 \quad (E.14)$$

where w_{1rms} and w_{2rms} are root-mean-square (rms) values of the scalar process input disturbance, w_1 , and measurement noise, w_2 , respectively.

To summarize the design procedure, the known (or estimated) characteristics of the system are used to find the now scalar values of A , B , Q_1 and Q_2 (with $C = 1$) using expressions E.12 and E.14. Equations E.5 and E.6 are then solved to find the optimum gain, K_f , and the Kalman filter for the system is then defined by expression E.13.

E.3 An Experimental Example

Figure E.1 contains a plot of the time history of an acceleration signal, that was obtained through the double-differentiation of linear position data acquired using a rotary encoder that was mounted to the drive wheel of an experimental vehicle. This "raw" acceleration signal appears to contain a quantized signal that was generated or at least enhanced by the differentiation process. This noise is most visible during the time intervals of 0.2s-1.3s and 3.3s-4.3s, and after 5.5s. While it is desirable to filter this noise, it is also desirable to leave alone the process disturbance "noise" resulting from wheel slippage, which can be seen as large magnitude oscillations in the acceleration signal during the periods of 1.3s-3.3s and 4.3s-5.5s. In general terms, a discrete low-pass filter is required, that can minimize the effect of the higher-frequency ($> 20\text{Hz}$) measurement (differentiation) noise on the observation of the process disturbance.

A spectral analysis of the raw acceleration signal in Figure E.1 (see Figure I.3) could be used to identify an effective cut-off point for a high-order ("sharp") low-pass filter, that could be used to remove the measurement (differentiation) noise. This

approach would require a higher-order filter with many terms in the discrete approximation, and the cut-off frequency would have to be very carefully chosen, such that changes in the vehicle and traction system would not result in dramatic changes in the effectiveness of the filter. Alternatively, a lower-order (and less frequency sensitive) filter can be implemented, using the Kalman filter design technique to choose filter parameters that will balance the desire to observe the process disturbance signal, with the desire to eliminate the measurement (differentiation) noise. It is noted that study of the spectral content of the process disturbance and measurement noise signals (using Figures E.1 and E.3) indicates that, although they are "coloured", they are each sufficiently "white" to merit application of the Kalman design procedure. The resulting filter may not be truly optimal in the Kalman sense, but the procedure is illustrative, and the filter will still be functional.

The discrete sampling interval, used to acquire the raw data, was $T_s = 0.01s$. The bandwidth of the system (vehicle) has been estimated, based on step-response (Fig. 5.13) and spectral analysis (Fig. E.4), to be $\omega_b \approx 2.2$ (0.35Hz). Assuming that the system (vehicle) can be modelled as a first-order low-pass filter, with cut-off frequency ω_b , the system parameters may be determined, using expression E.12, to be $A = 0.978$ and $B = 0.022$. The magnitudes of each of the process disturbance (w_1) and measurement (w_2) noise signals may be characterized by their root-mean-square (rms) values, which are $w_{1rms} = 10.4$ and $w_{2rms} = 1.6$, respectively, when averaged over the corresponding time intervals mentioned above. The filter design matrices (scalars here) can therefore be determined, using expressions E.14, to be $Q_1 = (10.4)^2$ and $Q_2 = (1.6)^2$.

Parameters A , B , Q_1 and Q_2 may now be substituted into expressions E.5 and E.6, to solve for the "optimum" filter gain, K_f , which then fully determines the characteristics of the Kalman filter described by expression E.13. Using MATLAB, the solution to expressions E.5 and E.6 is $K_f = 0.1165$, and the Kalman filter may now be written as

FIGURE E.1: Raw (fine —) and Kalman-filtered (heavy —) acceleration signals.

FIGURE E.2: Response of Kalman filter (dB) versus signal frequency (Hz).

$$\hat{a}_{out}(k) = \bar{a}(k) + 0.1165 [a_{in}(k) - \bar{a}(k)] \quad (E.15)$$

$$\bar{a}(k) = 0.978 \hat{a}(k-1)$$

where, at moment k , $a_{in}(k)$ is the raw acceleration data point, $\bar{a}(k)$ is the acceleration predicted by the system model alone, and $\hat{a}_{out}(k)$ is the Kalman filter estimate of the true acceleration. The expression in E.15 is simply a recursive linear filter, which for practical purposes, may be re-expressed as:

$$\hat{a}_{out}(k) = 0.1165 a_{in}(k) + 0.8641 \hat{a}_{out}(k-1) \quad (E.16)$$

The decay rate of this filter is about 2.8dB/octave, beyond a the cut-off frequency of about 4Hz, as may be observed in the frequency response of the filter, plotted in Figure E.2. It is noted that at $f = 0\text{Hz}$, response of the filter is less than unity ($< 0\text{dB}$).

Figure E.1 compares the resulting filtered time response of the acceleration data stream with the original unfiltered (raw) signal. The amount of higher-frequency quantized "measurement" noise can be seen to have been greatly reduced, relative to the prominent lower-frequency components of the process disturbance noise (wheel slippage). This increase in the ratio of low- to high-frequency content is perhaps more clearly illustrated in the frequency domain. Figures E.3 and E.4 contain the results of spectral analysis of each of the raw and filtered data streams, respectively. These plots were normalized to one another using the most prominent peak in the process disturbance (wheel slippage) frequency band ($\approx 6.5\text{Hz}$). The effect of filtering is most clearly evident when comparing the vehicle acceleration signal, below about 1Hz, to the (apparently) random noise signal, above about 17Hz. (The power density of this noise signal appears to be fairly constant up to the Nyquist frequency of 50Hz.)

In summary, a Kalman filter has been designed and implemented to remove

FIGURE E.3: Spectral content of unfiltered acceleration signal.

FIGURE E.4: Spectral content of Kalman-filtered acceleration signal.

measurement noise from a vehicle acceleration signal. The filtration process reduced higher frequency noise, producing a considerably "cleaner" process disturbance (wheel-slippage) indicator signal ($\approx 6.5\text{Hz}$), while leaving the lower-frequency acceleration signal ($< 1\text{Hz}$) nearly unchanged. The simplicity of this filter should facilitate real-time implementation.

END

21-07-94

FIN