OPTIMIZATION OF THE LOCOMOTION OF A LEGGED VEHICLE WITH RESPECT TO MANEUVERABILITY

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

by

Dominic Anthony Messuri, B.E.E.E., M.S.E.E.

* * * * *

The Ohio State University
1985

Reading Committee:

Professor C.A. Klein

Professor R.B. McGhee

Professor F. Ozguner

Approved by

C A Klein

Advisor
Department of Electrical Engineering

To my family

ACKNOWLEDGMENTS

I would like to acknowledge the advice and guidance that my advisor, Professor Charles A. Klein, has provided during this research project. His guidance and insight have been invaluable to the completion of this work. I am grateful to Professor Robert B. McGhee and Professor Fusun Ozguner for their careful review of this work. Also, I thank Professor McGhee for giving me the opportunity to conduct this research.

Thanks are due to Mr. Ronald W. Ventola for his help with hardware needs. I thank Mr. Anthony Maciejewski, Mr. Tae-Sang Chung, and Mr. Antonio I. Chirco for their thoughtful discussions and assistance in filming the videotape documentation. Special thanks are due to Ms. Debi Britton for her excellent work in the typing of this manuscript.

I sincerely express my appreciation to my friends Lillian and Sing Kwan Li for their support and encouragement throughout this research effort. I am also grateful to Ms. Susan Embert and Mr. Louis Rezanaka for their encouragement and special help.

Finally, and most importantly, I thank my parents and my brothers Anthony and Joseph for their continuous encouragement and patience throughout my studies. I am especially grateful to my brother Anthony whose constant encouragement, advice, and understanding contributed significantly to the completion of this work.

This research was supported by the Defense Advanced Research Projects Agency under Contracts MDA903-81-C-0138 and MDA903-82-K-0058.

VITA

October 26, 1953.....Born - Youngstown, Ohio

June, 1975.....B.E.E.E., Electrical Engineering
Youngstown State University
Youngstown, Ohio

1975-1976.....Graduate Teaching Assistant
Department of Electrical Engineering
Youngstown State University
Youngstown, Ohio

March, 1978.....M.S.E.E., Electrical Engineering
Youngstown State University
Youngstown, Ohio

1976-1980.....Design Engineer, Advanced Engineering
Packard Electric Division
General Motors Corporation
Warren, Ohio

1980-1984.....Graduate Research Associate
Digital Systems Laboratory
The Ohio State University
Columbus, Ohio

1981-1983.....Graduate Teaching Associate
Department of Electrical Engineering
The Ohio State University
Columbus, Ohio

FIELDS OF STUDY

Major Field: Electrical Engineering

Studies in Computer Engineering:	Professors K.J. Breeding, F. Battocletti, K.W. Olson, F. Ozguner, R. Fenton, R.B. McGhee
Studies in Control Engineering:	Professors U. Ozguner, S. Koozekanani
Studies in Computer and Information Science:	Professors D. Leinbaugh, S.H. Zweben, W. Buttelmann, N. Soundararajan, K. Proegler
Studies in Mathematics:	Professors S. Drobot, J.T. Scheick

PUBLICATIONS

"Design of an Audio Frequency Amplifier," B.E. thesis, Youngstown State University, Youngstown, Ohio, June 1975.

"Stochastic Output-Variable Feedback Control," M.S. thesis, Youngstown State Univeristy, Youngstown, Ohio, March 1978.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENT..... iii

VITA..... v

LIST OF TABLES..... xii

LIST OF FIGURES.....xiii

Chapter

1 INTRODUCTION..... 1

1.1 Background..... 1

1.2 Organization..... 3

2 SURVEY OF PREVIOUS WORK..... 4

2.1 Introduction..... 4

2.2 Existing Legged Vehicles..... 4

2.3 Gait Selection..... 8

2.3.1 Periodic Gaits..... 9

2.3.1.1 Mathematical Formulation of Gaits. 9

2.3.1.2 Wave Gaits..... 10

2.3.2 Non-Periodic Gaits..... 12

2.3.3 Previous Control Algorithms..... 13

2.4 Summary..... 18

TABLE OF CONTENTS -- Continued

Chapter		Page
3	SURVEY OF PREVIOUS WORK	20
	3.1 Introduction	20
	3.2 System Configuration	21
	3.2.1 OSU Hexapod	21
	3.2.2 Adaptive Suspension Vehicle	25
	3.2.3 Software Organization	29
	3.3 Operational Modes	32
	3.3.1 The Dual Tripod Leg Placement Algorithm	33
	3.3.2 The Manual Tripod Leg Placement Algorithm	34
	3.3.3 The Individual Foot Control Algorithm	35
	3.3.4 The Automatic Body Regulation Algorithm	35
	3.4 Human Factors Influence on Control	36
	3.4.1 Input Controller Interface	37
	3.4.2 Visual Feedback to the Operator	45
	3.5 The Control Program	46
	3.6 Summary	48
4	DERIVATION OF THE DUAL TRIPOD LEG PLACEMENT ALGORITHM	50
	4.1 Introduction	50
	4.2 Trajectory of the Body Center of Gravity	51
	4.3 Movement of Feet in the Support Phase	57
	4.4 Foot Lift and Placement Timing	63
	4.5 Where to Place a Foot Down	68

TABLE OF CONTENTS -- Continued

Chapter		Page
	4.6 Trajectory of a Foot in the Transfer Phase.....	81
	4.7 Implementation and Results.....	84
	4.7.1 Flow Chart and Explanation of the Dual Tripod Algorithm.....	84
	4.7.2 Results.....	89
	4.8 Summary.....	93
5	DEVELOPMENT OF PRECISION FOOTING LEG PLACEMENT ALGORITHM.....	97
	5.1 Introduction.....	97
	5.2 The Manual Tripod Leg Placement Algorithm.....	99
	5.2.1 Description of Operator Controls.....	99
	5.2.2 Implementation and Results.....	103
	5.2.2.1 Flow Chart.....	104
	5.2.2.2 Results.....	104
	5.3 The Individual Foot Control Algorithm.....	108
	5.3.1 Description of Operator Controls.....	108
	5.3.2 Graphics Display for Information Feedback..	111
	5.3.3 Implementation and Results.....	116
	5.3.3.1 Flow Chart.....	116
	5.3.3.2 Results.....	123

TABLE OF CONTENTS -- Continued

Chapter		Page
5.4	Automatic Body Regulation.....	126
5.4.1	Description of Operation.....	126
5.4.1.1	Body Accommodation.....	126
5.4.1.2	Body Stabilization.....	127
5.4.2	Energy Stability Margin.....	128
5.4.2.1	Previous Measure of Stability.....	129
5.4.2.2	Motivation for the Energy Stability Margin.....	130
5.4.2.2.1	Derivation of the Equations for the Energy Stability Level	137
5.4.2.3	Level Energy Curves.....	141
5.4.2.3.1	Determining the Gradient.....	143
5.4.2.3.2	Plots of Level Energy Curves.....	152
5.4.2.4	Optimally Stable Position.....	158
5.4.2.5	Comparison of Stability Curves....	164
5.4.3	Implementation and Results.....	169
5.4.3.1	Flow Chart and Explanation of Body Accommodation.....	169

Chapter | Page

5.4.3.2 Flow Chart and Explanation of Body Stabilization..... 171

5.4.3.3 Results..... 175

5.5 Summary..... 181

6 SUMMARY AND CONCLUSIONS..... 185

6.1 Research Contributions..... 185

6.2 Research Extensions..... 189

APPENDIX

COMPUTER PROGRAMS OF THE WALKING ALGORITHMS..... 194

REFERENCES..... 258

LIST OF TABLES

Table		Page
2.1	Existing Legged Vehicles.....	6

LIST OF FIGURES

Figure		Page
3.1	Photograph of the OSU Hexapod vehicle.....	22
3.2	Kinematic model of the OSU Hexapod.....	24
3.3	Photograph of a 1/10 scale model of the Adaptive Suspension Vehicle.....	26
3.4	Kinematic model of the Adaptive Suspension Vehicle....	27
3.5	Simulation model of the Adaptive Suspension Vehicle...	28
3.6	Block diagram of the computer control program partitioning.....	30
3.7	Computer graphic simulation of a vehicle maneuvering around obstacles.....	38
3.8	Computer graphic simulation of a vehicle walking on a road course.....	39
3.9	The entire road course used for the graphic simulation.....	40
3.10	The three-axis joystick.....	42
3.11	The joystick controller designed by Vertut.....	44
3.12	Flow chart of the Main Program.....	47
4.1	Assignment of the commanded velocities to the three axes of the joystick.....	52
4.2	The body coordinate system for the OSU Hexapod.....	52
4.3	Trajectory of the body center of gravity for time period Δt.....	54
4.4	Transformation of the coordinates of a fixed point P as the body coordinate system moves from A to B.....	58

LIST OF FIGURES -- Continued

Figure		Page
4.5	Top view of the vehicle, showing the circle of reachable area for each leg.....	65
4.6	As the body center of gravity moves along a circular arc, a point on the ground appears to move along a circular arc with the same center, but different radius.....	69
4.7	Consideration of possible footholds.....	71
4.8	Location of the equilibrium position of foot i, with respect to the center of rotation.....	72
4.9	Geometrical construction of intersecting circles, to find the foothold points P_1 and P_2.....	74
4.10	Trigonmetric relationships for the angles α and ς.....	76
4.11	Geometrical construction to find the foothold point P_1 for the special case when there is no rotational velocity.....	80
4.12	Desired trajectory of a transfer phase foot, in the z-direction. With respect to the Earth-fixed coordinate system.....	82
4.13	Desired trajectory of a transfer phase foot, in the z-direction, with respect to the earth-fixed coordinate system.....	82
4.14	Flow chart of the DUAL TRIPOD subroutine.....	85
4.15	Flow chart of the SUPPORT PHASE subroutine.....	87
4.16	Flow chart of the TRANSFER PHASE subroutine.....	88
4.17	Flow chart of the SPECIAL SECTION subroutine.....	90
4.18	The OSU Hexapod vehicle walking in the Dual Tripod mode.....	91
5.1	State diagram for the Manual Tripod Algorithm.....	102
5.2	Flow chart of the MANUAL TRIPOD subroutine.....	105

LIST OF FIGURES -- Continued

Figure		Page
5.3	The information presented on the graphic display terminal during operation of the OSU Hexapod vehicle. The two views at the bottom portion of the screen indicate the pitch and roll of the vehicle body.....	112
5.4	Flow chart of the LEG MOTION subroutine.....	117
5.5	Flow chart of the LINEAR MOTION subroutine.....	118
5.6	Flow chart of the ANGULAR MOTION subroutine.....	119
5.7	Flow chart of the KINEMATIC LIMIT subroutine.....	121
5.8	Flow chart of the BODY LIMIT subroutine.....	122
5.9	Computer graphics simulation of the Adaptive Suspension Vehicle (ASV), complete with graphical feedback information provided for the vehicle operator.....	124
5.10	An example of the support pattern when the vehicle is standing on an inclined plane, with the body horizontal. (a) The dotted lines show the vertical projection of the support feet. Also shown is the vertical projection of the body center of gravity.....	131
5.11	The dotted line represents the support boundary for this configuration where the vehicle is standing on an inclined plane, with the body horizontal. The support boundary is found by interconnecting the tips of the support feet that form the support pattern.....	134
5.12	Side view of the configuration in Figure 5.11, showing a geometrical comparison of the Energy Stability Level for the front and rear edges of the support boundary.....	135
5.13	A general configuration, used for the derivation of the Energy Stability Level equation. The line F_1F_2 represents one edge of a support boundary. The point CG represents the body center of gravity. The vertical distance h gives a measure of the Energy Stability Level.....	138
5.14	Level Energy Curves for the vehicle standing on level terrain, with the body horizontal.....	156

LIST OF FIGURES -- Continued

Figure		Page
5.15	Level Energy Curves for the vehicle standing on a 20 degree inclined plane, with the body horizontal.....	156
5.16	Level Energy Curves for the vehicle standing on a 20 degree inclined plane, with the body horizontal and the left front leg off the ground.....	157
5.17	Tracing the optimal paths of the vehicle center of gravity, when blending function is not included. Note the oscillations, which indicate sharp direction changes in the optimal path, due to the discrete time calaculation of the Energy Stability Margin gradient..	161
5.18	The active region of the blending function.....	163
5.19	Variation of the weight β as a function of ε where $\varepsilon = E_2 - E_1$.....	163
5.20	Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal. The dotted lines represent Level Energy Curves.....	165
5.21	Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal. The dotted lines represent Level Energy Curves.....	165
5.22	Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal and the left front leg off the ground. The dotted lines represent Level Energy Curves.....	166
5.23	Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclinded plane, with the body horizontal and the right front leg and left rear leg on rocks. The dotted lines represent Level Energy Curves.....	166
5.24	Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body pitched at 20 degrees. The dotted lines represent Level Energy Curves.....	167

LIST OF FIGURES -- Continued

Figure		Page
5.25	Optimal Paths of the vehicle center of gravity for the vehicle standing on a level terrain with a tripod support pattern. The dotted lines represent Level Energy Curves	167
5.26	Flow chart of the KINEMATIC LIMITS subroutine, with modification for Body Accommodation	170
5.27	Flow chart of the UPDOWN LIMIT subroutine	172
5.28	Flow chart of the XY LIMIT subroutine	173
5.29	Flow chart of the OPTIMIZE STABILITY subroutine	174
5.30	Photograph of the OSU Hexapod climbing onto a wooden platform. The vehicle is operating in the Individual Foot Control mode, with the Body Accommodation feature	177
5.31	The OSU Hexapod, and corresponding graphics display at the start of the Body Stabilization algorithm. The X symbol, located inside the support polygon, indicates the Optimally Stable Position to which the center of gravity should move	178
5.32	The OSU Hexapod, and corresponding graphics display, after the completion of the Body Stabilization algorithm. The body has shifted so that the center of gravity is at the Optimally Stable Position	179

CHAPTER 1

INTRODUCTION

1.1 Background

A legged vehicle possesses a tremendous potential for maneuverability over rough terrain, particularly in comparison to conventional wheeled or tracked vehicles.

Present-day wheeled vehicles provide an efficient means of transportation, capable of fairly high speeds; however, these vehicles typically can travel only on surfaces which are relatively smooth and which provide sufficient traction [1]. Specially-built "off-road" vehicles are designed to provide additional ground clearance and increased traction to enable operation on surfaces other than normal roadways, but these vehicles have definite limitations on the types of terrain conditions upon which they can operate. Tracked vehicles, such as tanks and earth-movers, are also limited by traction considerations and the steepness of slopes which can be negotiated.

On the other hand, legged animals and insects demonstrate a distinct superiority in their capabilities for rough terrain locomotion. The use of individual limbs, which can be manipulated independently or in coordinated movements [2], enables legged creatures to maneuver over a variety of terrain conditions since they have a distinct ability to adapt to varying conditions. Consider, as an example, the use of horses

or mules as a means of transportation in mountainous regions where roads have not been developed, or the use of sled-dogs in the arctic regions where snow and ice would impair the traction of wheeled vehicles.

The superiority of legged animals for rough-terrain locomotion has motivated the development of man-made legged vehicles. In general, a legged vehicle can offer more degrees of freedom for movement than conventional vehicles. Legged vehicles can provide the capabilities of stepping over obstacles or ditches, climbing over obstacles, or maneuvering within confined areas of space.

However, the coordination of the movements of the various leg joints in such a way as to produce the desired locomotion of the vehicle is an extremely complex task. Physiological analysis of limb movements and coordination in both walking animals and insects have led to the development of a variety of viewpoints which attempt to explain how these movements are initiated and controlled [3,4]. The fact that none of these viewpoints fully explains the movements inherent in walking is an indication of the natural complexity involved. Early studies of a mechanical walking machine have shown that if the leg coordination is left entirely to the human operator, even a relatively simple walking machine presents such a highly complex task that the operator becomes exhausted after only a short period of operation [5]. Therefore, it is essential to relieve the operator of as much of this complex task as possible.

The work documented in this report is aimed at improving the maneuverability of a legged vehicle and reducing some of the operator's burden of manipulating the body and limbs.

1.2 Organization

The information presented in this dissertation is organized as follows:

Previous work performed in the area of legged locomotion is surveyed in Chapter 2. A brief historical outline shows the evolution of walking machines, and some of the major developments are discussed.

Two walking vehicles, both of which are currently under development at The Ohio State University, are described in Chapter 3. Also included is a description of the various operational modes which are under consideration for these vehicles.

The derivation of a highly maneuverable walking algorithm, referred to as the Dual Tripod Algorithm for leg placement, is detailed in Chapter 4.

A particular operational mode, the Precision Footing mode, is addressed in Chapter 5. Various control algorithms are developed for use in this mode. Particularly important is the derivation of a new measure of vehicle stability. This quantitative stability measure leads to the development of an important Automatic Body Regulation scheme.

The contributions of this research are summarized in Chapter 6, and suggestions for possible future research work are discussed. A listing of the computer programs developed during the course of this research are contained in the appendix of this dissertation.

CHAPTER 2

SURVEY OF PREVIOUS WORK

2.1 Introduction

This chapter gives an overview of previous work performed in the area of legged locomotion and the development of computer control algorithms for walking vehicles.

A brief historical outline is given in Section 2.2, showing the evolution of walking machines, with an emphasis on the more recent developments.

The concept of gait selection is reviewed in Section 2.3. The background and terminology for periodic gaits is presented, along with a discussion of free gaits. Previous computer control algorithms which are based upon these gait concepts are discussed.

2.2 Existing Legged Vehicles

Over the years, a variety of ideas have been conceived for legged vehicles. As the technology has improved, it has become feasible to actually construct walking vehicles. In particular, tremendous recent advances in the areas of computer architecture and VLSI (Very Large Scale Integration) capabilities now make it possible to construct self-contained vehicles with on-board computers capable of performing complex computations and logical decisions at amazingly fast speeds.

Table 2.1 provides a brief historical outline listing major developments in the area of walking machines and indicating their relative time frame. More detailed information on most of these vehicles can be found in [6,7,8,9]. Of particular interest in relation to this dissertation are the developments of the current decade.

A quadruped vehicle developed at the Tokyo Institute of Technology utilizes a pantograph leg design which allows for simpler calculations when determining the leg positions [10]. The vehicle is equipped with contact sensors on each foot to detect when a foot comes in contact with surrounding objects. Whenever a foot is being moved through the air and comes into contact with an obstacle, the body height of the vehicle is automatically adjusted. This adjustment allows that foot and the vehicle body to continue traveling along the desired path over the obstacle.

At Carnegie Mellon University, a six-legged vehicle has been built using hydraulic actuators to move the legs [11]. This vehicle is fully self-contained, using a gasoline engine to provide power for the hydraulic system. The human operator rides inside the vehicle and controls the speed and direction of motion. The operator can also set the tilt of the vehicle body and the ground clearance. The hydraulic actuator system is designed to achieve leg coordination and to accomplish the complex leg motions needed for locomotion, without requiring an excessive amount of computations. An on-board microprocessor monitors the operator's commands and the position of each leg. The microprocessor uses this information to determine leg placement locations and to assist in the control of vehicle steering.

Table 2.1

Existing Legged Vehicles

Year	Researcher	Vehicle Features
1960	J.E. Shigley	Quadruped vehicle with mechanical linkages. Impractical because of noncircular gears and complex linkages.
1960	Space General Corporation	A six-legged and an eight-legged machine. Both used mechanical linkages. Had limited adaptability to terrain conditions.
1966	McGhee and Frank at University of Southern California	The first legged vehicle to walk autonomously under computer control. A four-legged machine referred to as the "Phoney Pony."
1968	General Electric Corporation	A four-legged machine, weighing 3,000 pounds. The operator was required to use hands and feet to manually control the twelve joints for movements of the vehicle legs.
1969	Bucyrus-Erie Company	A four-legged machine, weighing 13,500 tons, used for coal mining operations. The largest walking machine ever built.
1972	Vukobratovic at Institute Mihailo Pupin, in Yugoslavia	Powered biped exoskeletons for use in locomotion of paraplegics.
1972	Waseda University in Tokyo, Japan	A series of biped robots; pneumatic or hydraulic controlled. Used pressure sensors at front and rear of feet to detect ground reaction.

Table 2.1 (continued)

Year	Researcher	Vehicle Features
1972	University of Rome, in Italy	Six-legged vehicle, two degrees of freedom per leg. Uses interactive computer control.
1976	University of Wisconsin	Exoskeleton for use by paraplegics. Controlled by analog computer.
1977	Ohio State University	Six-legged vehicle, referred to as the OSU hexapod. Three degrees of freedom per leg. Uses interactive computer control.
1977	Moscow Physio-Technical Institute	Six-legged vehicle, controlled by a hybrid analog-digital computer.
1980	Tokyo Institute of Technology	Quadruped vehicle, using pantograph leg geometry, and interactive computer control.
1982	Carnegie-Mellon University	Six-legged vehicle, hydraulically actuated. Fully self-contained.
1983	Odetics, Inc.	Six-legged vehicle, referred to as ODEX. Seven on board computers, controlled by the external operator.

Odetics, Inc. has developed a six-legged walking vehicle referred to as Odex I [12]. The six legs of this vehicle are arranged in a circular pattern, as opposed to most machines which have three legs on either side of a rectangular body. Each of the legs is manipulated by three electric motors. The vehicle has a self-contained power system, using a 24 volt aircraft battery. Leg coordination and motion control is performed by seven on-board computers. Each leg is controlled by a "level-1" computer. These six computers, in turn, are coordinated by a "level-2" computer which interprets the operator's commands and computes the required leg motions. Operator commands are sent to the vehicle through a radio transceiver link.

One other vehicle, which is currently being developed at the Ohio State University, is a six-legged, fully self-contained walking machine referred to as the Adaptive Suspension Vehicle (ASV). This vehicle will utilize a pantograph leg design and will be hydraulically powered. More details on this new vehicle are presented in Chapter 3.

2.3 Gait Selection

Research efforts on the various legged vehicles described in Section 2.2 have led to the development of a number of different control schemes. These control methods range from purely mechanical control to complex computer control by a dedicated computer network. Common to all of these control schemes is the concept of gait selection - choosing a particular sequence for the lifting and placement of the legs. Gait selection involves consideration of factors such as stability, terrain adaptability, limb coordination, and timing of limb movements [6].

The first scientific study of gaits in animals was conducted by Eadweard Muybridge in 1872 [13]. A series of photographs of a galloping horse were obtained by electrically triggering a row of cameras in sequential order. These photographs clearly displayed the sequence of leg movements of a horse while walking, trotting, and galloping. For the first time it was clearly demonstrated that while the horse is trotting or galloping, there are instances when all four of the horse's feet are off the ground. Muybridge went on to apply this technique to photograph the leg movements of a variety of animals and likewise later applied it to human locomotion. These photographs were published in two classic works entitled Animals in Motion [13] and The Human Figure in Motion [14].

2.3.1 Periodic Gaits

2.3.1.1 Mathematical Formulation of Gaits

The empirical study of gaits was quantified to some degree by Hildebrand [15]. Later, McGhee generalized the concept of gaits to a more mathematical formulation and developed the following definitions.

DEFINITION 1: A gait is periodic if every limb of the vehicle operates with the same cycle time [16].

The cycle time for leg i is the time from when leg i is lifted until the next time leg i is lifted.

DEFINITION 2: The period, τ, of a periodic gait is the time required for one complete locomotion cycle [16].

DEFINITION 3: The stride length, λ, of a gait is the distance by which the center of gravity of the system is translated during one complete locomotion cycle [16]. This assumes straight line locomotion.

DEFINITION 4: The duty factor, β_i, is the fraction of a locomotion cycle during which leg i is in contact with the supporting surface [16].

DEFINITION 5: The leg phase, ϕ_i, is the fraction of a locomotion cycle by which the contact of leg i with the supporting surface lags the contact of leg 1 [16].

The total number of distinct gaits in a legged system is extremely large, and this number increases dramatically as the number of legs increases [17]. In order to determine which of these gaits is most desirable to use, McGhee and Frank [18] introduced the following concepts dealing with stability:

DEFINITION 6: The support pattern associated with a given support state is the convex polygon, in a horizontal plane, which contains the vertical projection of all of the supporting feet [19].

DEFINITION 7: The longitudinal stability margin is measured in the direction of travel, over an entire cycle of locomotion. It is the shortest distance from an edge of the support pattern to the vertical projection of the vehicle center of gravity onto the horizontal plane [19].

2.3.1.2 Wave Gaits

A criterion for an optimum gait would be that it maximize the minimum stability margin. In order to find nontrivial solutions to this

minimax problem, McGhee and Frank [18] assumed that all legs of the machine operate with the same duty factor, β. Also, they assumed that the vehicle legs are evenly spaced in right-left pairs, symmetric about the center axis. The optimization is for straight line locomotion over level terrain and at a constant speed. McGhee and Frank [18] proved that, for a quadruped vehicle, there is a unique optimum gait, referred to as the quadruped crawl.

For a hexapod vehicle, Bessonov and Umnov [20] have shown that for a given duty factor, β, the set of optimal gaits are members of the set referred to as wave gaits [21]. Wave gaits are characterized by a wave motion of stepping events from the rear to the front, on either side of the vehicle. There is a constant phase interval between successive leg motions on either side of the body, and laterally opposing legs are one-half cycle out of phase. The phase relationships for optimal hexapod wave gaits has been shown to be [17]:

$$\phi_3 = \beta, \qquad \phi_5 = 2\beta - 1, \qquad 0.5 \leq \beta < 1$$

where ϕ_3 is the phase delay of the left middle leg, and ϕ_5 is the phase delay of the left rear leg. Both ϕ_3 and ϕ_5 are measured as a fraction of a total locomotion cycle and are relative to the placing of leg 1, the left front leg.

While the quadruped has a single optimal gait, the hexapod actually has six optimal gaits, depending upon the value of β [2].

In studying the leg movements of insects it has been found that many natural quadrupeds use the wave gait for very low speed locomotion, and all hexapods (such as insects) use wave gaits in almost all speed

ranges. The wave gait is particularly obvious in multi-legged animals such as millipedes and centipedes where the back to front movement of the legs appears as a wave motion [3].

2.3.2 Non-Periodic Gaits

Periodic gaits are characterized by the fact that each leg operates with the same cycle time. Therefore, within one locomotion cycle, every leg is lifted and placed exactly one time. Wave gaits represent a subset of periodic gaits, with the additional contraints of a constant duty factor, β, and a constant phase interval between successive leg motions. Wave gaits have been shown to be the optimal gait for ideal legged machines, particularly for straight line locomotion at a constant speed over level terrain [18,20].

As the vehicle operating conditions deviate from the ideal conditions of constant speed straight line locomotion, wave gaits may no longer provide an optimal gait. Furthermore, when the terrain conditions cannot be represented by a level surface, a periodic gait may not be optimal. Under certain terrain and operating conditions it may become advantageous to utilize a non-periodic, or free, gait.

Preliminary work in the area of non-periodic gaits was performed by Kugushev and Jaroshevskij [22], who suggested that the mathematical formalizations developed for the study of periodic gaits could be extended to include the more general case of non-periodic gaits. They presented a partial problem formalization for the situation where the trajectory of the center of gravity is specified in advance. This

trajectory covers a predefined terrain containing certain designated regions which have been determined to be unsuitable for support. This work was extended by McGhee and Iswandhi [19], who completed the formalization of this problem and developed a heuristic algorithm for its solution. Recently, Kwak [23] incorporated some refinements to this algorithm and implemented them in a computer simulation.

The concept of a free gait walking algorithm was also applied by Klein and Patterson [24] in their study of a three-armed manipulator for use in space assembly. Free gaits become necessary when considering locomotion on a structure in space, such as a space station, since gravity is no longer a consideration and footholds must be grasped. Consequently, the vehicle is limited to discrete footholds and regular periodic gaits generally cannot be used. Furthermore, the absence of gravity eliminates stability considerations.

However, to date, no algorithm incorporating a free gait has been implemented on an actual walking machine. All the work in this area has consisted solely of computer simulations.

2.3.3 Previous Control Algorithms

A great deal of research work has been performed at The Ohio State University in the development of control algorithms for walking vehicles. The research has included a wide range of topics dealing with different gait implementations, selection and evaluation of various sensors, investigation of different computer architectures, development of servo-control schemes, and other related topics. This section will

briefly outline the major developments in this area which are related to the present work.

One of the earliest computer investigations of walking control algorithms was performed by Orin [9]. This work dealt with an open-loop computer simulation of the OSU Hexapod vehicle on undulating terrain. The control program was structured as a supervisory control system in which the human operator supplies the higher level commands such as speed, steering, mode, etc., to the control computer. The control computer then provides the necessary control signals to the leg actuators in order to produce the leg movements required to move the vehicle in accordance with the operator's input. (It should be noted that, during this same period of time, a computer simulation of a legged vehicle was under study in Russia [25].)

In Orin's control program, the leg movements were accomplished by implementing the family of wave gaits. Therefore the phase relationships of all the legs was predefined. Although the wave gait has only been shown to optimize the static stability for straight line locomotion over level terrain, the algorithm used the wave gait formulation for sidestepping and turning in place.

Orin also implemented automatic body height, pitch, and roll regulation. The control algorithm relied on the vehicle support points to calculate a plane surface, which was then used as an estimate of the terrain. The height, pitch, and roll of the body are adjusted to maintain the vehicle parallel to this estimated plane. The body must also be maintained at a constant height above this plane.

The physical design and construction of the OSU Hexapod vehicle was performed by Jaswa [26]. In addition, Jaswa implemented the first computer software to control the walking motion of this vehicle. The walking motion utilized a basic wave gait formulation. Because of limitations in execution speed of the computer programs, the vehicle did not actually operate in a real-time mode. Instead, the motion planning phase produced a matrix of coordinates of desired joint angles and rates; the motion execution phase then produced the desired movement, based upon the stored matrix of coordinates.

Chao [27] revised a portion of the motion planning algorithm of Orin's work [9] and, using a multiprocessor configuration, implemented it for real time control of the OSU Hexapod vehicle. The motion planning algorithm was simplified to account for the vehicle moving only on level terrain. Furthermore, Orin's control solution required large amounts of computation time and was therefore not directly suitable for real time vehicle control. Chao applied more efficient computation methods which resulted in a suboptimal solution, but with only minimal decrease in system performance.

A linear servo system was incorporated which included two feedback loops: a joint-rate servo loop and a foot-position servo loop. With this closed-loop servo control system implemented on a PDP-11/45 computer, and the modified motion planning algorithm implemented on an interconnected PDP-11/03 computer, Chao demonstrated successful real-time operation of the OSU Hexapod.

Force sensing was first demonstrated on the OSU Hexapod by Briggs [7]. A force sensing unit was designed, constructed, and installed on one leg of the vehicle. The sensing unit provided a vector force sensing capability for the one leg, and this feedback information was utilized to demonstrate that the vehicle leg could adapt to terrain conditions through the use of active compliance. The active compliance feature was incorporated into the walking control algorithm of Chao [27]. It was demonstrated that the OSU Hexapod could negotiate obstacles placed in the path of the one force-sensing leg while the vehicle continued to walk using a periodic wave gait.

The incorporation of vector force sensing on all six legs of the OSU Hexapod was accomplished by Pugh [28]. In addition, a vertical gyroscope and a pair of gravitational pendulums were installed on the vehicle for attitude sensing. The force sensing capabilities were utilized in an active compliance algorithm which included a closed-form solution for the force setpoints. The attitude sensing allowed the development of a closed loop attitude regulation scheme. Both of these features were incorporated into a motion planning algorithm, based upon the algorithms developed by Orin [9] and Chao [27], to implement the periodic wave gait. With these modifications, it was demonstrated that the OSU Hexapod is capable of walking over irregular terrain while maintaining body orientation.

A binocular vision system utilizing two solid-state TV cameras was installed on the OSU Hexapod by Tsai [29,30]. Using a laser beam

designator to indicate a selected foothold, a Follow-The-Leader control mode was incorporated. For this particular gait, the operator selects footholds for the front foot on each side of the body. As the vehicle moves forward, each of the remaining legs follows in the foothold of the leg immediately in front of it. The Follow-The-Leader algorithm developed by Tsai allows each leg to spend a fixed amount of time in the transfer phase, and allows the vehicle body to be moved for a specified period of time. In effect, this Follow-The-Leader algorithm implements a backward wave gait since the "wave" of leg movements is from front to back, rather than from back to front as in the normal wave gait.

An interesting feature of the Follow-The-Leader algorithm is that the operator specifies the footholds to be used by the front two legs, and the motion of the vehicle body must accommodate the motion of the legs. This is in contrast to all previous locomotion algorithms for the OSU Hexapod which allow the operator to control the motion of the center of the body, and the leg movements must accommodate the body motion. To determine where to move the center of the body for Follow-The-Leader locomotion, Tsai introduced the concept of the Optimally Stable Point. He suggested that, if the position of the supporting legs is known, it might be possible to determine a location to which the center of the body could be moved so that the vehicle's stability is optimal. Tsai proposed a heuristic algorithm to determine an Optimally Stable Point [29]. This algorithm was based upon the concept of "shrinking" the support polygon until the Optimally Stable Point is found. However, this algorithm could not be implemented as part of the Follow-The-Leader

algorithm because it required an excessive amount of computation time. Furthermore, the Optimally Stable Point could not be determined for all situations and only considered the case for level terrain.

An algorithm for omnidirectional control of locomotion over rough terrain has been developed by Lee [31], and was simulated using a computer graphics terminal. The algorithm is based on previous wave gait concepts, with one major exception. Rather than using a fixed cycle time and stride length, as had been done in previous algorithms, this new algorithm allows the stride length to vary. The variable stride length is based on the concept of a constrained working volume. The optimal cycle period is the time period which allows at least one leg to fully utilize its constrained working volume. The simulation algorithm also includes automatic control of body altitude and attitude. Previous control algorithms were not capable of omnidirectional control; instead, they required range limits on velocities in various directions. In Lee's simulations, the use of a constrained working volume allowed him to demonstrate that a walking vehicle could achieve omnidirectional control over rough terrain.

2.4 Summary

This chapter has given an overview of previous work performed in the area of legged locomotion, particularly those areas which relate to the present work. A brief description was given of the major contributions in control algorithms developed for a walking vehicle at The Ohio State University. A common concept in all of the algorithms

which have been implemented on the OSU Hexapod is that of a wave gait, where the relative movements, or phasing, of the legs is determined by time. Some studies have looked at using other concepts, such as the free gait, but these have only been implemented in computer simulations.

The remainder of this dissertation will describe the development of new leg placement algorithms in which the phasing of the legs is independent of time. These new algorithms have been successfully implemented on the OSU Hexapod, in real-time control, as well as in computer simulation studies on a new walking vehicle presently being constructed.

CHAPTER 3

PROBLEM DESCRIPTION

3.1 Introduction

The development of a set of computer control algorithms which provide highly maneuverable locomotion of a legged vehicle is one of the chief objectives of the research work presented in this dissertation. The computer control algorithms are implemented and tested on two vehicle configurations: (1) the OSU Hexapod vehicle and (2) a computer simulation model of a new six-legged vehicle. This hydraulically powered vehicle incorporates a unique pantograph leg design and is currently under construction at The Ohio State University.

The system configuration of these two vehicles is described in Section 3.2. In addition, a discussion of the hierarchical design philosophy of the computer software is included.

A description of the various operational modes which have been defined for these legged vehicles is given in Section 3.3. A brief overview is presented of the particular algorithms which have been developed in this research. The derivation and specific details of these algorithms comprise Chapters 4 and 5.

Consideration of the human-machine interface was influential in the selection of command inputs and cockpit controls. These ergonomic factors are discussed in Section 3.4.

3.2 System Configuration

3.2.1 OSU Hexapod

A major development of the research work being conducted at The Ohio State University in the area of legged locomotion is the OSU Hexapod vehicle (Figure 3.1). This six-legged vehicle is an experimental prototype which is being used to develop various control schemes and leg placement algorithms. It serves as a test-bed for the development and evaluation of new sensors and sensing systems.

The vehicle frame and leg segments are constructed of aluminum. Each of the six legs is comprised of three independent angular joints arranged in an arthropod configuration. The joints are powered by industrial-grade electric drill motors connected to a gear reduction unit [32].

Each of the joints is equipped with a potentiometer to measure the angular position and a tachometer to measure the angular rate. The lower segment of each leg is equipped with two semiconductor strain gauges to measure lateral forces, and a piezoelectric load cell is mounted in each foot to measure the axial force [7,28].

The vehicle is interfaced to a PDP-11/70 computer via an optically isolated digital data link [7]. The feedforward path of the data link, from the computer to the vehicle, is used to transmit the eighteen input voltages to control the joint actuator motors. Each data word consists of eight data bits and five address bits. Digital-to-analog converters, located on the vehicle, are used to convert the eight data bits to an analog voltage. The feedback path of the data link, from the vehicle to the computer, is capable of 72 channels of data [28]. Present

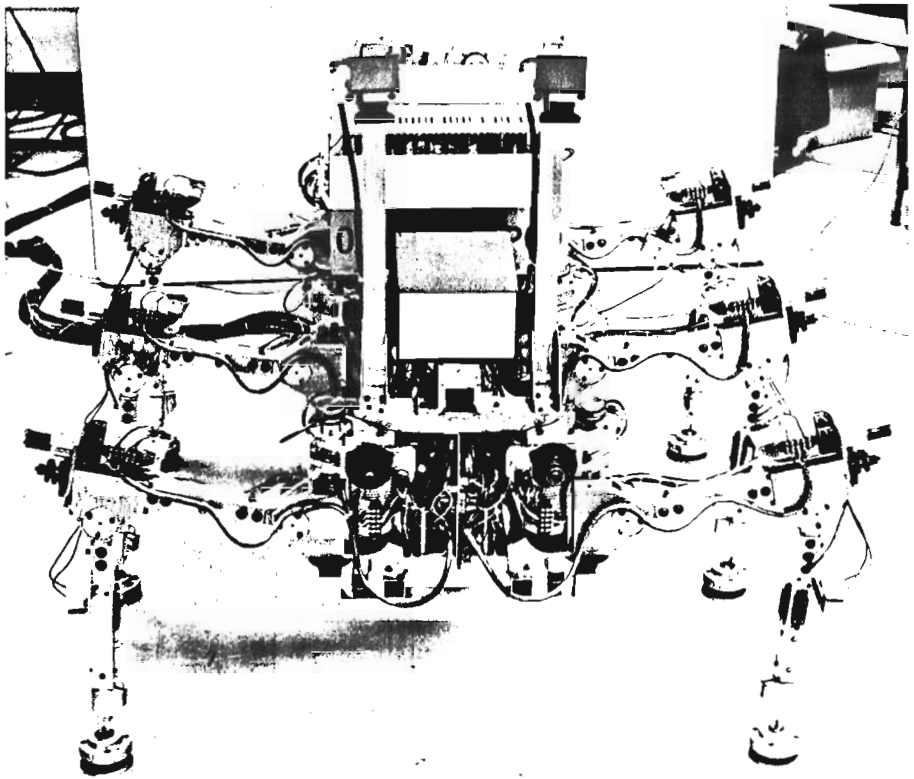


Figure 3.1 Photograph of the OSU Hexapod vehicle.

allocation of these channels is: (1) eighteen channels of angular joint positions, (2) eighteen channels of angular joint rates, (3) eighteen channels of leg forces, (4) four channels of attitude sensor information, and (5) the remaining fourteen channels are used for status and diagnostic information. All sensor voltages are passed through an analog-to-digital converter to transform them into digital form. These discrete digital values can then be easily manipulated by the computer. The feedback data consists of ten data bits and six address bits. Also included in the digital data link is the arbitration and control circuitry necessary to monitor and control the flow of data on this link [7,28].

The kinematic model for the OSU Hexapod vehicle is shown in Figure 3.2. The details of this model and the associated kinematic equations can be found in [9] and [33]. Notice from Figure 3.2 that the vehicle legs are numbered 1 through 6, with leg 1 being the left front leg, leg 2 being the right front leg, and the remaining legs numbered in sequence with odd numbers on the left side and even numbers on the right side of the body. Two coordinate systems are defined in order to facilitate the description of the vehicle's movement over the terrain surface. The earth-fixed coordinate system (X_E, Y_E, Z_E) has the X_E coordinate axis in the initial direction of travel, the Z_E coordinate axis in the downward direction of gravitational acceleration, and the Y_E coordinate axis in the direction to form a right-handed coordinate system. The body-fixed coordinate system (X_B, Y_B, Z_B) has its origin fixed at the center of the vehicle body, with the X_B coordinate axis directed toward the front of the vehicle, the Z_B coordinate axis directed downward perpendicular to

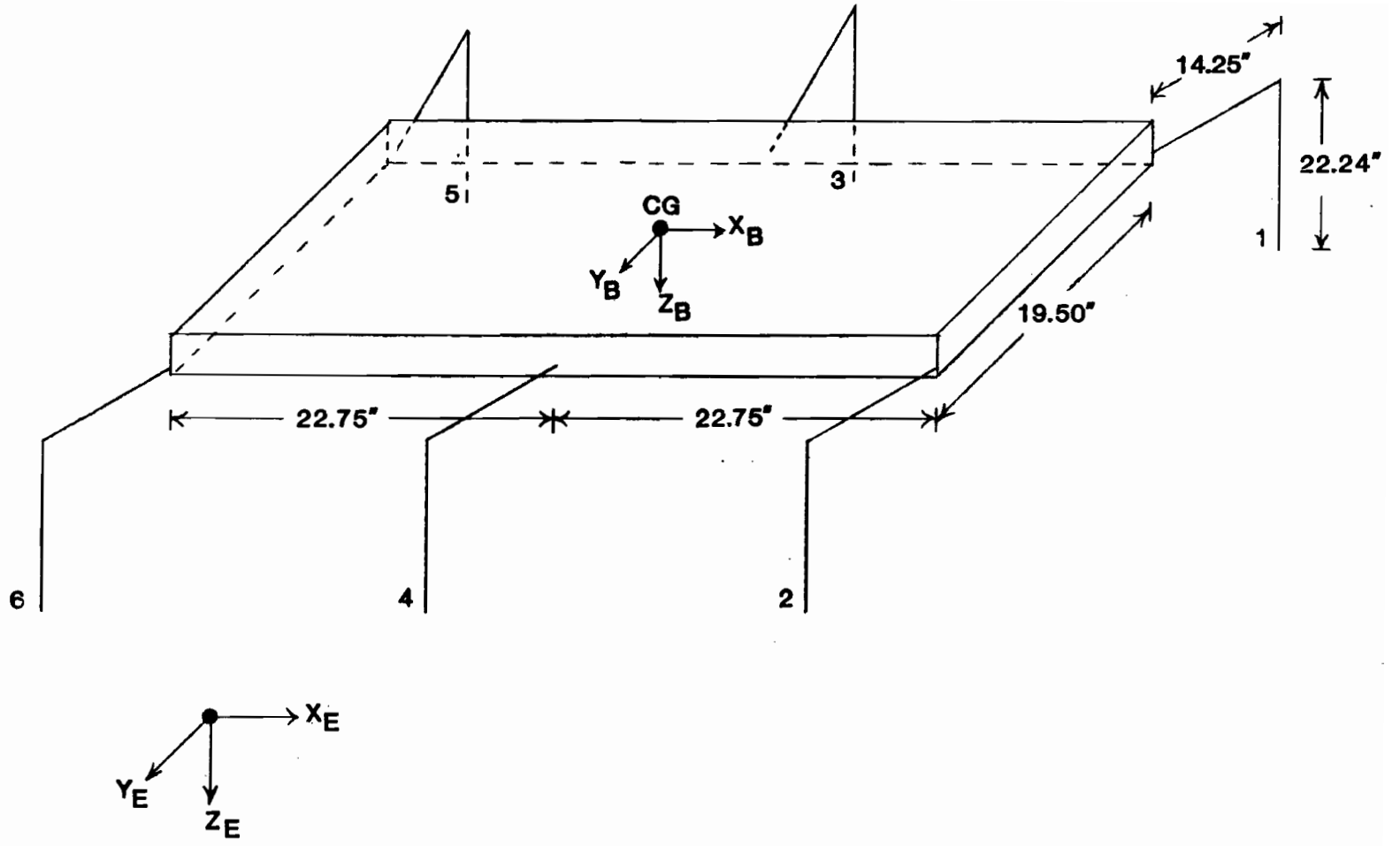


Figure 3.2 Kinematic model of the OSU Hexapod.

the body plane, and the Y_B coordinate axis in the direction to form a right-handed coordinate system.

3.2.2 Adaptive Suspension Vehicle

Presently under construction at The Ohio State University is a new six-legged vehicle referred to as the Adaptive Suspension Vehicle (ASV). A preliminary model of this vehicle is shown in Figure 3.3. This vehicle will be a full-scale, self-contained walking machine. Each of the six legs of the ASV will have a planar pantograph geometry in a rotatable plane, and will be controlled by three independent actuators [34]. This special leg configuration will provide a more energy-efficient system. All of the actuators will be hydraulically controlled.

The ASV will be completely outfitted with sensing equipment which will provide feedback information relating to the state of each leg, as well as providing information concerning the environmental surroundings. The vehicle will contain a network of on-board computers, which have the responsibility of performing the various data-processing and control tasks. Included in this computer network will be a Coordination computer, Guidance computer, Cockpit computer, and six Leg Control computers [35].

The kinematic model of the ASV, as used in this dissertation, is shown in Figures 3.4 and 3.5. The dimensions shown in this kinematic model are the values which were used in the computer simulation. It may be noted that the center legs of the vehicle scale model are mounted opposite to those of the kinematic model. This vehicle leg mounting

Figure 3.3 Photograph of a 1/10 scale model of the Adaptive Suspension Vehicle.

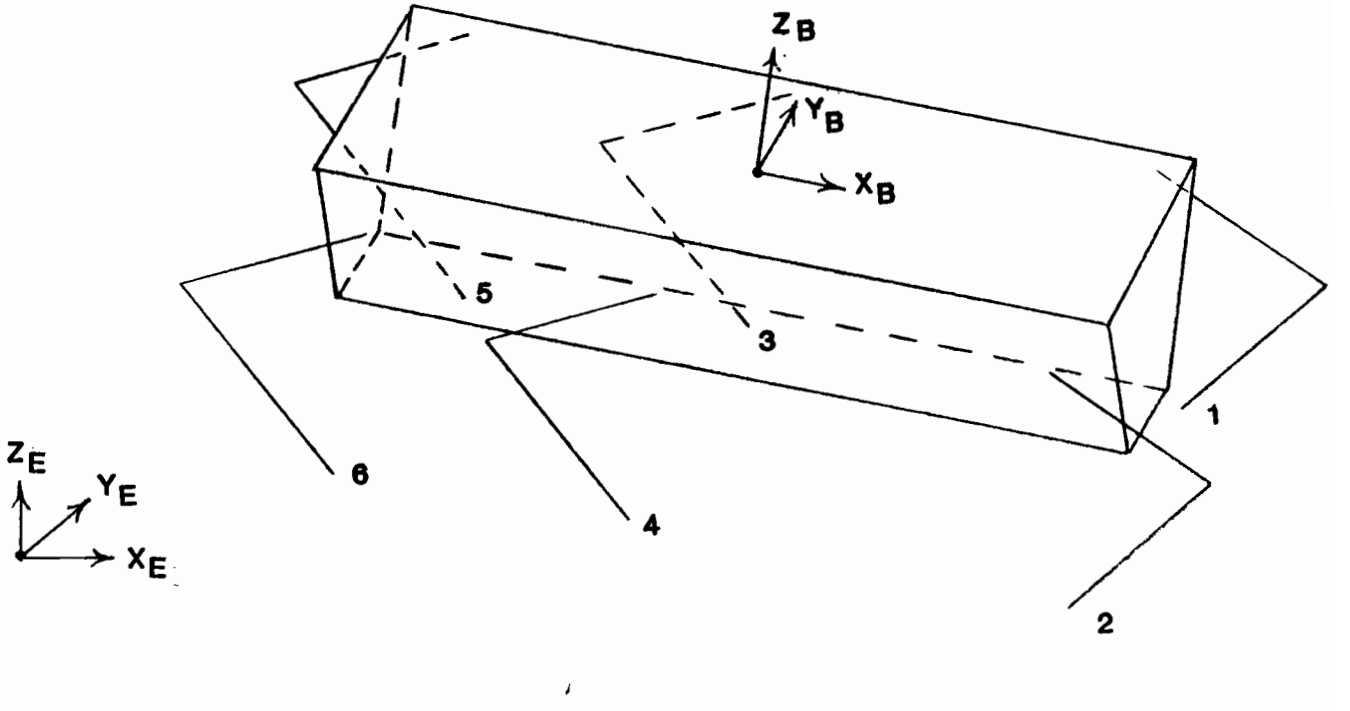
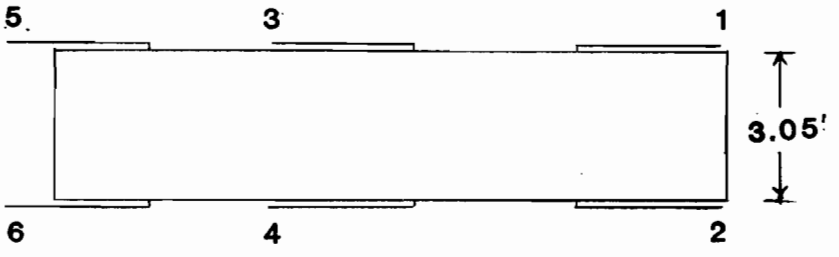
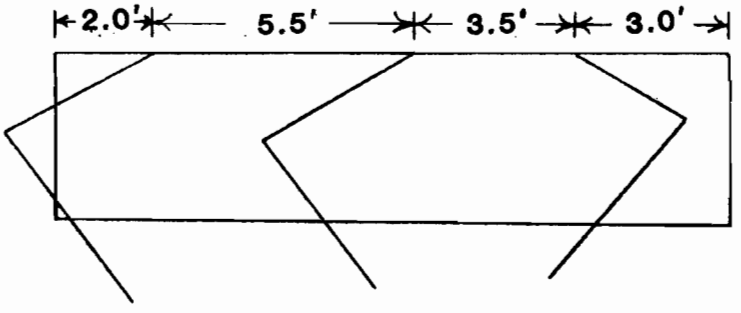


Figure 3.4 Kinematic model of the Adaptive Suspension Vehicle.





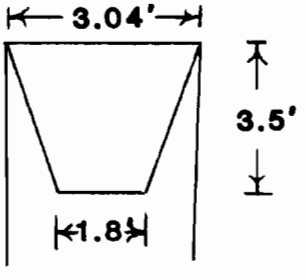


Figure 3.5 Simulation model of the Adaptive Suspension Vehicle.

was revised; the present configuration is represented by the scale mode of Figure 3.3. The operation of the computer control algorithms are the same, regardless of this feature. The details of the kinematic equations for this model can be found in [36] and [31].

Notice from Figure 3.4 that the vehicle legs are numbered 1 through 6, with the same pattern as used for the OSU Hexapod vehicle. In order to facilitate the description of the vehicle's movement over the terrain surface, two coordinate systems are defined: an earth-fixed coordinate system (X_E, Y_E, Z_E) and a body-fixed coordinate system (X_B, Y_B, Z_B). These coordinate systems differ from those defined for the OSU Hexapod vehicle in that each of these Z coordinate axes are now defined with an upward positive direction. This redefining of the Z axis, in turn, changes the positive direction of each Y coordinate axis. This is necessary if a right-handed coordinate system is to be maintained.

3.2.3 Software Organization

The computer software, which controls the operation and movements of the vehicle, is based upon a hierarchical design that partitions the overall task into various subtasks. The general partitioning, as shown in Figure 3.6 [37], is chosen such that communication between subtasks is minimal. This program structure is motivated by the philosophy of an automatic supervisory control system [38]. In this system the human operator supplies the high level commands, such as speed and direction, and the control computer implements the appropriate algorithms to perform the desired movements.

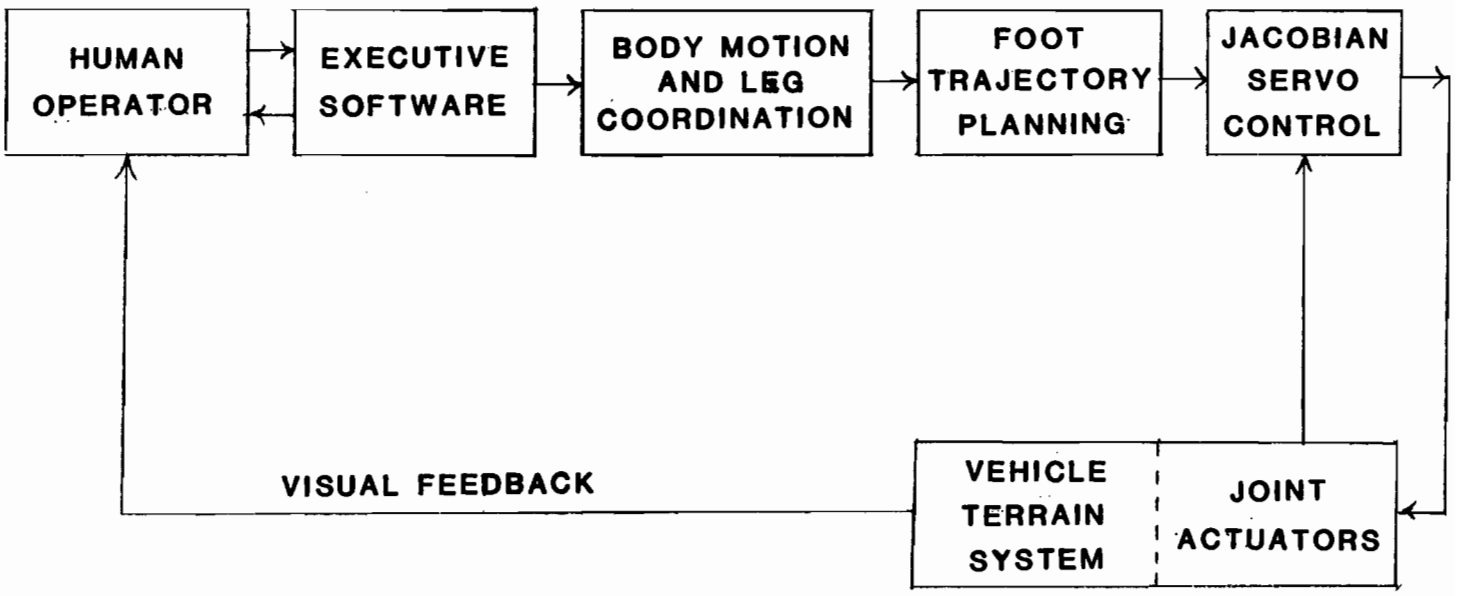


Figure 3.6 Block diagram of the computer control program partitioning. [37]

The highest level in the program structure is the "Executive Software" section, which provides the interface between the human operator and the vehicle control programs. For the OSU Hexapod vehicle, the operator can issue commands to the control programs by using a three-axis joystick and a selected set of keys on the computer terminal keyboard. When construction of the ASV is completed, the operator control mechanism will consist of a custom-designed arm controller and a set of function-select switches [39]. These interface devices will be discussed further in the next section of this dissertation. Regardless of the hardware interface, the computer control functions are the same.

The "Body Motion and Leg Coordination" section determines the leg-lifting and leg-placing sequence and the relative motion of the body required to satisfy the operator commands.

The third level, "Foot Trajectory Planning," calculates the desired position and velocity of each foot in order to achieve the commanded body and leg motions.

The "Jacobian Servo Control" section converts the rectilinear foot velocities to angular joint rates. These calculated joint rates are converted to voltage levels and are used to control the joint actuators.

Feedback from the joint actuators is used to control the joint movements so that the actual joint rates follow the commanded joint rates.

3.3 Operational Modes

The operation of the vehicle has been partitioned into a set of operational modes which allow the vehicle to function in a variety of terrain conditions and which require varying degrees of operator control. When the terrain is relatively smooth, the vehicle should be able to operate with minimal input from the operator. As the terrain conditions become more complex, it becomes necessary for the operator to provide additional input. Following is a list of the major operational modes [35], arranged according to suitability for relatively simple terrain to progressively more complex terrain:

a) Cruise - This mode is intended for locomotion over reasonably smooth terrain. The body crab angle, which is defined as the angle between the heading of the vehicle and the instantaneous body velocity vector [31], as well as the minimum turning radius may be limited. The control algorithm should not require full use of all the sensors.

b) Terrain Following - A terrain scanner will be used to provide terrain preview data. This information can then be used by the control computer to predict foothold locations and also to determine the average slope and elevation of the terrain. Using this slope and elevation information, the body attitude and altitude may be adjusted, if necessary. Proximity sensors may be used for local control of foot elevation [40], while force sensors may be used to provide active compliance [41].

c) Close Maneuvering - This mode is intended for relatively rough terrain, where the operator desires a high degree of

maneuverability in arbitrary directions. The operator makes use of a controller, such as a joystick, to command arbitrary combinations of forward velocity, lateral velocity, and body rotation rates.

d) Precision Footing - This mode is useful for maneuvering over extremely rough terrain, as well as to climb over large obstacles. The operator can control individual legs by means of a joystick, keyboard, or through the use of other commands. The control of the body motion could be either automatic or manual.

The algorithms which have been developed in this particular research work concentrate on the Close Maneuvering and Precision Footing operating modes. In particular, four algorithms have been implemented: (1) The Dual Tripod Leg Placement Algorithm, (2) The Manual Tripod Leg Placement Algorithm, (3) The Individual Foot Control Algorithm, and (4) The Automatic Body Regulation Algorithm.

3.3.1 The Dual Tripod Leg Placement Algorithm

The Dual Tripod Leg Placement Algorithm treats the six legs of the vehicle as two independent sets of tripods. At all times, at least one of these tripods is supporting the vehicle. This algorithm is particularly well-suited for the Close Maneuvering mode. Leg motion is limited only by geometrical considerations; there are no time-sequencing constraints as in the typical wave-gait formulation.

A predominant problem in the Close Maneuvering mode is the gait transition as the direction of body motion is changed. Previous

algorithms required a tradeoff of speed versus agility; the velocity input commands required a filter with a long time-constant, in order to avoid sudden changes in command. The Dual Tripod algorithm has no gait transition problem, so it does not require filtering of the input commands. The result is an extremely agile walking algorithm, which has been demonstrated to be especially well-suited for the Close Maneuvering mode.

3.3.2 The Manual Tripod Leg Placement Algorithm

Similar to the Dual Tripod Algorithm, the Manual Tripod Leg Placement Algorithm also treats the six legs of the vehicle as two independent sets of tripods. However, in the Manual Tripod mode, the vehicle operator directly controls the motion of the tripods. The operator may lift a tripod off the ground, position this tripod so that all three feet are above solid footholds, then lower this tripod to the ground to support the vehicle. The operator may also control the motion of the vehicle body with respect to the feet which remained on the ground.

The Manual Tripod mode provides the vehicle operator with maneuvering capabilities which are a hybrid of the capabilities of the Close Maneuvering and the Precision Footing operational modes. The vehicle is highly maneuverable since the operator may control the movements of each of the the two tripods independently. However, in order to reduce the burden on the operator, the vehicle's legs may be controlled as tripod sets rather than treating each leg individually. This Manual Tripod mode is particularly useful for high maneuverability

over moderately rough terrain, where the full capabilities of the Precision Footing mode may not be required.

3.3.3 The Individual Foot Control Algorithm

The Individual Foot Control Algorithm allows the vehicle operator to directly control the motion of each leg individually. The operator can also control the motion of the vehicle body. The ability to specifically control each leg provides a significant increase in vehicle maneuverability. Support legs can be placed on firm footholds to provide solid support for the vehicle.

Included in this algorithm are various automatic monitoring features to assure the safety of the operator and vehicle. The positions of all legs are monitored to insure that they do not exceed their kinematic limits. The operator is inhibited from lifting a foot which would cause the vehicle to become unstable. Also, the position of the body center of gravity is monitored to insure that the body is not moved to a statically unstable position.

With all of these features, the Individual Foot Control Algorithm is essential for maneuvering over extremely rough terrain or for climbing over large obstacles.

3.3.4 The Automatic Body Regulation Algorithm

The Automatic Body Regulation Algorithm is based upon the Individual Foot Control Algorithm with two important extensions. In addition to all of the features of the Individual Foot Control Algorithm, the Automatic Body Regulation Algorithm enables the operator to make use of the computer control software to automatically adjust the

position of the vehicle body. These adjustments are made in accordance with certain predefined criteria. The Automatic Body Regulation Algorithm includes two major features: (1) the vehicle body is automatically adjusted to compensate for movements of the vehicle legs; (2) the operator can request that the computer control algorithm move the vehicle body so that vehicle stability is maximized. To implement this second feature, it is necessary to determine the Optimally Stable Point to which the body center of gravity should be moved in such a way as to maximize the vehicle stability. In order to accomplish this, a new measure of vehicle stability, referred to as the Energy Stability Margin, has been developed. The Energy Stability Margin provides a more accurate measure of stability than previous measures and is therefore especially important for use on rough terrain.

3.4 Human Factors Influence on Control

While developing and implementing the previously mentioned computer control algorithms, careful consideration has been given to the ergonomic aspects of the control interface. The physical interface between the human operator and the computer control algorithm must be designed to reduce the operator burden as much as possible. The actual hardware used for the interface can significantly affect the ease with which an operator can control the vehicle movements. Furthermore, the assignment of vehicle control parameters to the hardware interface has a significant effect on the operator's coordination ability in regards to maneuvering the vehicle. While some investigation of controller hardware configurations has been performed by Beringer [39], that

investigation concentrated primarily on the Cruise mode of operation. The examination of the controls for the Precision Footing mode had not been investigated until this present dissertation work.

3.4.1 Input Controller Interface

As part of this dissertation research work, computer graphic simulations were used to evaluate various hardware interface controls, as well as to evaluate some different assignments of vehicle control parameters. Photographs of the two computer graphic simulation arrangements are shown in Figure 3.7 and Figure 3.8. The graphic display in Figure 3.7 depicts an outline of the vehicle body, which the operator must maneuver around obstacles. These obstacles are represented in the figure by squares. The graphic display in Figure 3.8 depicts an outline of the vehicle body including the vehicle's legs. The operator must maneuver the vehicle along a predetermined road course consisting of a variety of twists and turns. At any one point in time, only a single portion of the road course appears on the graphics display. The entire road course is shown in Figure 3.9.

The first control mode was implemented by making use of various keys on the computer keyboard to input commands to the software algorithm. By pressing selected keys, the operator can control the direction of motion of the vehicle body. The magnitude of velocity in a particular direction is determined by the number of times that the corresponding key is pressed. For example, pressing the F key increments the forward velocity by a fixed increment; pressing the F key a second time will cause the forward velocity to be increased by the same

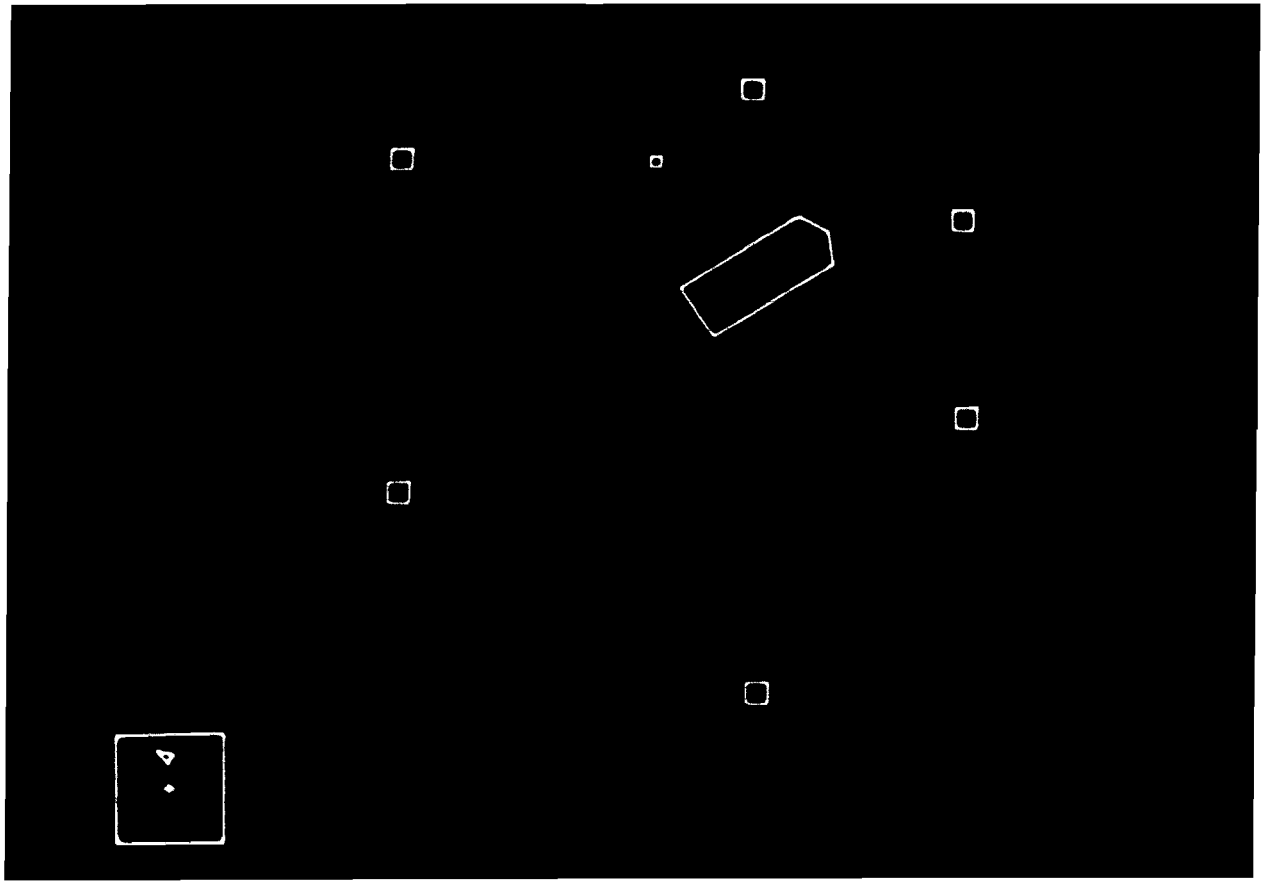


Figure 3.7 Computer graphic simulation of a vehicle maneuvering around obstacles.

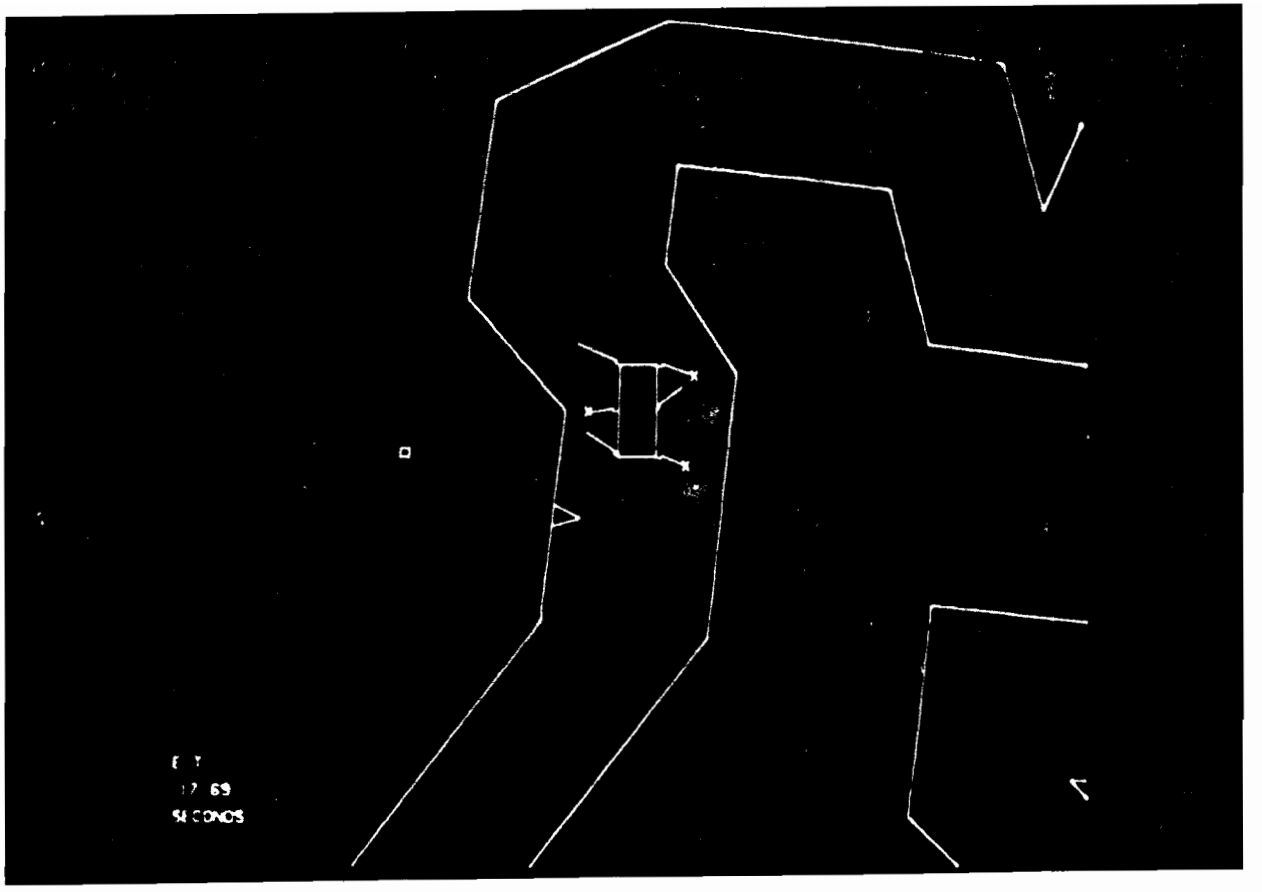


Figure 3.8 Computer graphic simulation of a vehicle walking on a road course.

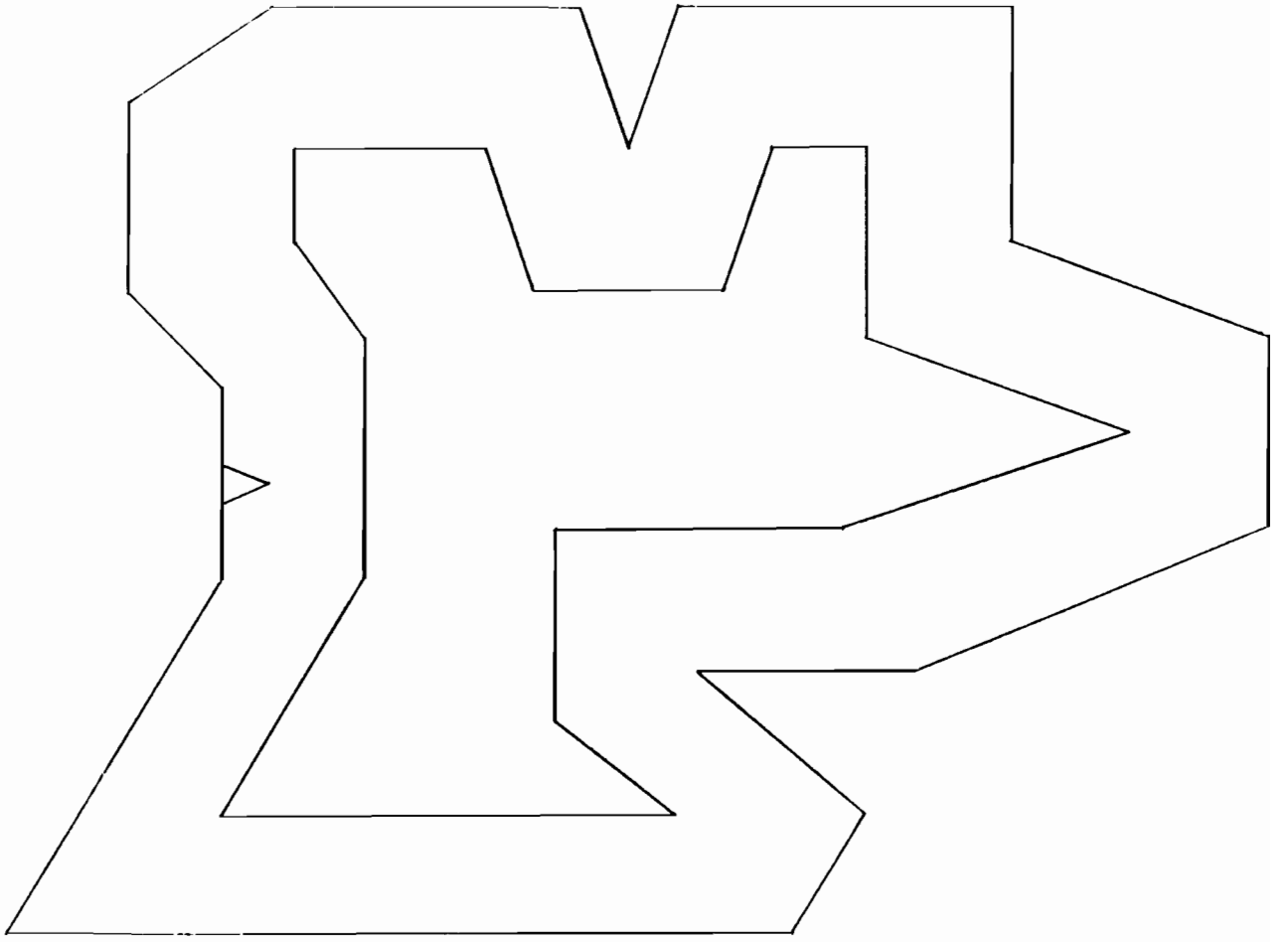


Figure 3.9 The entire road course used for the graphic simulation.

fixed increment. In a similar manner, pressing the L key increments the left velocity, indicating a sidestep motion towards the left side of the vehicle. This type of control mode enables the operator to control the vehicle direction and velocity; however, it is cumbersome when attempting to perform an extensive amount of vehicle maneuvering.

An improvement in this keyboard command interface has been achieved by the incorporation of speech recognition. This voice command interface involved the use of a Votran speech recognition system and a Votrax speech synthesizer. Using this speech recognition system, the operator now has the added capability of providing commands to the computer control algorithm merely by pronouncing certain key words such as "Forward", "Left", "Right". As in the keyboard input system, these commands result in the corresponding vehicle velocity being incremented by a fixed amount. Repeated activations would cause an incremental increase of the appropriate velocity. The Votrax speech synthesizer module has the function of echoing back the operator's commands. In this way the operator may verify that the appropriate command has been recognized. This type of operator/vehicle interface can be desirable in certain circumstances since the operator is not required to perform any manual activations. However, as in the keyboard interface, this type of control can be quite cumbersome when attempting to perform an extensive amount of vehicle maneuvering.

A three-axis joystick, as shown in Figure 3.10, provides a versatile control interface between the operator and the vehicle. The computer control algorithm software is used to assign the relationship of the joystick inputs to the vehicle control parameters. In this way,

Figure 3.10 The three-axis joystick.

the joystick manipulations enable the operator to control an assortment of vehicle movements as determined by the interpretation in the control algorithm.

One of the assignments used for the joystick variables is: (1) the fore/aft motion of the joystick corresponds to the forward/backward velocity of the vehicle body; (2) the left/right motion of the joystick corresponds to the left/right lateral velocity of the vehicle body; and (3) the rotational motion of the joystick corresponds to the rotational velocity of the vehicle body. The use of the three-axis joystick enables the vehicle operator to control any or all of these vehicle parameters in a continuous fashion. The parameters can be varied independently or simultaneously. This type of input device provides great ease of control.

Another assignment used for the joystick variables is based upon the concept of controlling the location of the turning center for the vehicle motion. In this assignment, the fore/aft motion of the joystick corresponds to the location of the turning center at some position along the longitudinal axis of the vehicle. The rotational motion of the joystick controls the rotational velocity of the vehicle about the turning center. This type of control is instrumental in performing close maneuvering of the vehicle around obstacles.

The concept of controlling the location of the turning center was explicitly designed into a special controller by Vertut [42]. A photograph of this controller is shown in Figure 3.11. In this configuration: (1) the rotational axis corresponds to the angular position of the vehicle heading; (2) the left/right movement of the

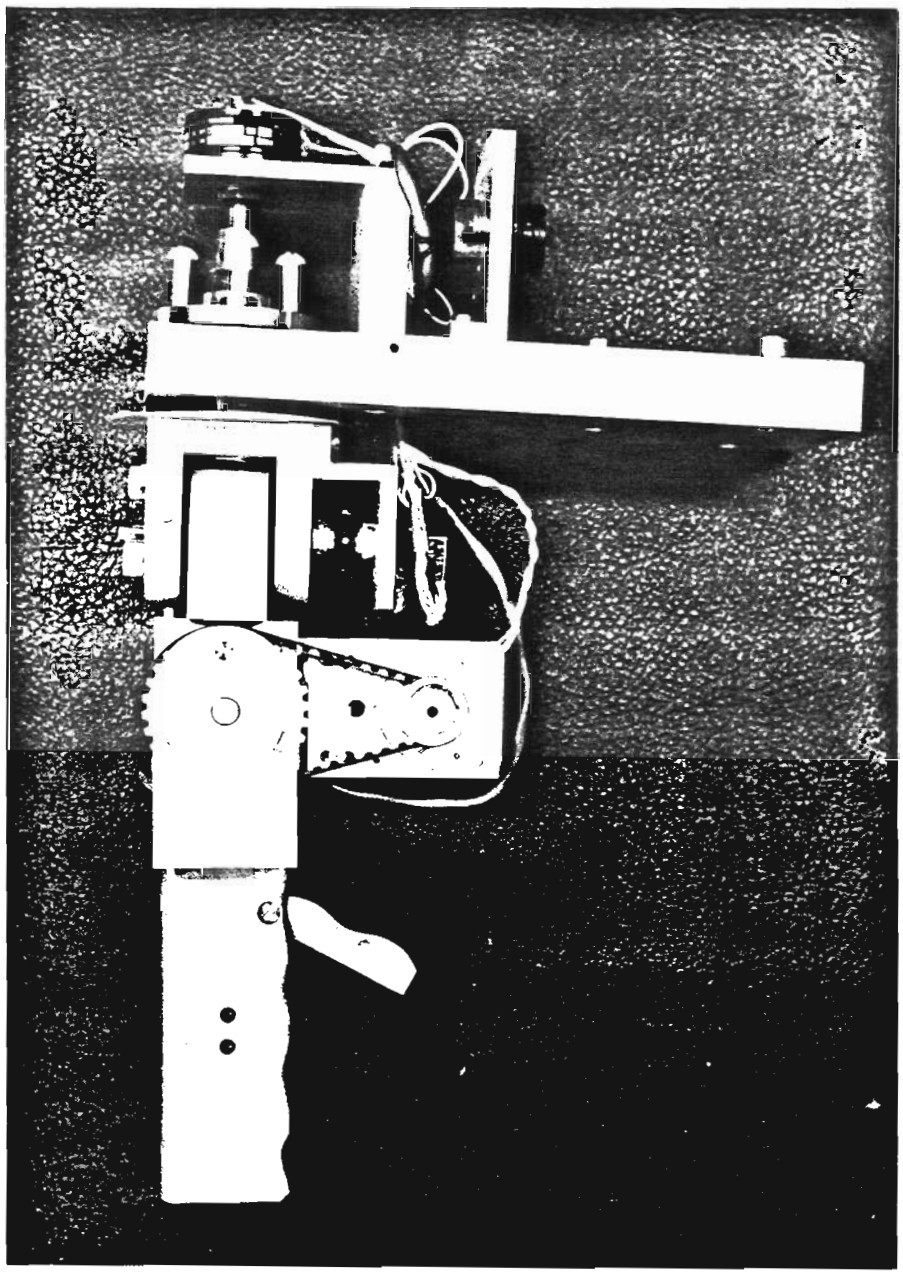


Figure 3.11 The joystick controller designed by Vertut [36].

controller grip determines the distance of the turning center from the center of the body; and (3) the fore/aft movement of the controller grip controls the vehicle speed. An interesting feature of this controller concept is that it allows the operator to control the trajectory of the vehicle motion independently of the speed of the vehicle [39]. As in the previous case, this controller may be helpful in performing close maneuvering of the vehicle.

3.4.2 Visual Feedback to the Operator

The operator's ability to maneuver the vehicle is greatly enhanced by providing feedback information regarding the present state of the vehicle. In this research work, feedback information is supplied to the operator via a computer graphics display terminal. The computer control algorithms are designed to supply the operator graphic information concerning the location of the vehicle feet, the position of the vehicle body, the vehicle stability, and other essential information. Because the information is presented in a graphical format, the operator can quickly assimilate the necessary information.

Feedback information is particularly crucial when using the Precision Footing operational mode. The operator must perform a significant portion of the decision-making during this control mode. The graphical feedback provides the operator with the necessary information required to make reasonable decisions regarding the movement of the vehicle legs and body.

3.5 The Control Program

Figure 3.12 shows a flow chart of the main control program which was used to implement the various leg placement algorithms developed in this research. The format of this control program follows the hierarchical structure of an automatic supervisory control system. Operator commands, which are input via the select-switches, are used to determine which of the various subroutine procedures will be utilized for vehicle control. These subroutine procedures perform the "Body Motion and Leg Coordination" and the "Foot Trajectory Planning" functions in the supervisory control structure of Figure 3.6. The development and implementation of these procedures represents a significant portion of this present research work.

In Figure 3.12, the block labeled "Servo Routine" corresponds to the "Jacobian Servo Control" section of the supervisory control structure of Figure 3.6. This portion of the control program includes servo control routines used by Pugh [28].

The block labeled "Graphics Display Routine" in Figure 3.12 represents another major contribution of the present research work. This section is comprised of subroutines which control the computer graphics display terminal. The feedback information provided to the vehicle operator via the graphics terminal significantly improves the operator's ability to control the vehicle. This graphics display information expands the "Visual Feedback" path depicted in Figure 3.6. In previous vehicle control programs, this feedback consisted solely of information which the operator could visually perceive directly from the

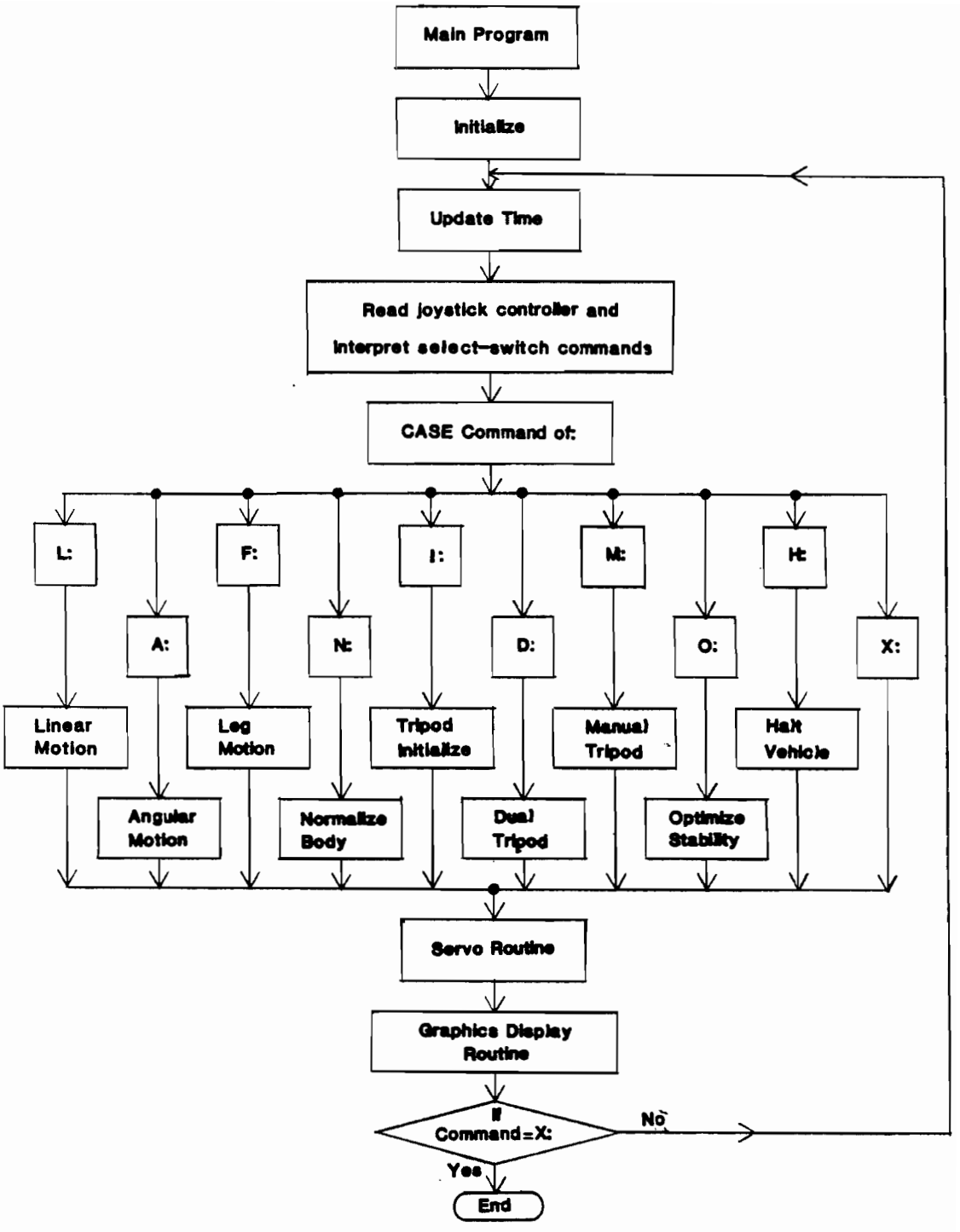


Figure 3.12 Flow chart of the Main Program.

surroundings. In the present research work, the graphics display terminal is incorporated as a means of providing the operator quantitative feedback information in a visual format. This quantitative information, which was previously unavailable, can now be readily interpreted by the vehicle operator.

3.6 Summary

This chapter presented certain vital background information regarding the system configurations which are considered in this research work. Two system configurations were used: (1) the OSU Hexapod vehicle and (2) the Adaptive Suspension Vehicle. All the computer control algorithms developed in this research are completely compatible with either of these machines. The computer software utilizes a hierarchical structure based upon automatic supervisory control.

A set of distinct operational modes has been described in this chapter. These operational modes allow the vehicle to function in various terrain conditions and require different degrees of complexity with respect to operator control. Of these operational modes, the present research work is primarily concerned with the Close Maneuvering and Precision Footing operating modes, since these two modes are required if the vehicle is to be capable of traversing obstacles and rough terrain. With these two modes, the vehicle can now be used for practical applications.

A brief overview describing the computer algorithms developed in this research has also been included in this chapter. Various

approaches for the physical interface between the human operator and the control computer have been discussed. Of the different hardware controllers, the three-axis joystick provides a significant amount of versatility and has therefore been used as the major input controller interface. Feedback information is provided for the operator via a graphics display terminal.

The overall goal of this research is to develop a set of computer control algorithms which provide high maneuverability and which can assist the vehicle operator by simplifying the complexity of the control task. The remainder of this dissertation presents the detailed development of the various control algorithms resulting from this research.

CHAPTER 4

DERIVATION OF THE DUAL TRIPOD LEG PLACEMENT ALGORITHM

4.1 Introduction

The Close Maneuvering operational mode is intended for relatively rough terrain where the operator desires a high degree of vehicle maneuverability in arbitrary directions. In this mode, the operator specifies control commands consisting of combinations of forward velocity, lateral velocity, and yaw rotational rate through the use of a controller, such as a joystick. The movements of the various leg joints are coordinated by a computer control algorithm which converts the supervisory control commands of the operator into appropriate joint commands in order to achieve the desired motion.

The derivation of a particular computer control algorithm, referred to as the Dual Tripod Algorithm, is presented in this chapter. The Dual Tripod Algorithm is a highly maneuverable walking algorithm which is particularly well-suited for the Close Maneuvering mode. In this algorithm, the six legs of the vehicle are treated as two independent sets of tripods. At least one of these tripods is supporting the vehicle at all times. Leg motion is limited only by geometrical considerations; there are no time-sequencing constraints as in the typical wave-gait formulations. The tripod support pattern provides a stable stance, even on rough terrain. Also, with the legs arranged as two tripod sets, the leg sequencing problem is simplified.

In Section 4.2, the necessary equations are derived to describe the trajectory of the body center of gravity in response to the joystick commands. The equations describing the movement of the feet in the support phase are developed in Section 4.3. Section 4.4 discusses the considerations which determine when to lift a foot off the ground and when to place a foot down. The equations for determining where to place a foot on the ground are derived in Section 4.5. The desired trajectory for feet in the transfer phase is discussed in Section 4.6.

The Dual Tripod Algorithm was implemented on both the OSU Hexapod and a computer simulation of the Adaptive Suspension Vehicle (ASV). The implementation of this algorithm is described in Section 4.7.

4.2 Trajectory of the Body Center of Gravity

In the Dual Tripod Algorithm the operator can control the vehicle movement by means of a three-axis joystick. The displacement of the joystick from its neutral position corresponds to commanded velocities with respect to the vehicle's body coordinate system. Figure 4.1 shows a three-axis joystick with the associated three degrees of freedom labeled as u, v, and r. Figure 4.2 shows the body coordinate system for the OSU Hexapod configuration. (All derivations will be with respect to the OSU Hexapod configuration. Only minor changes are needed when results are used on the ASV configuration.) The assignment of the joystick axes is as follows:

(1) u represents the commanded velocity along the x-axis of the body coordinate system (longitudinal velocity).

(2) v represents the commanded velocity along the y-axis of the body coordinate system (lateral velocity).

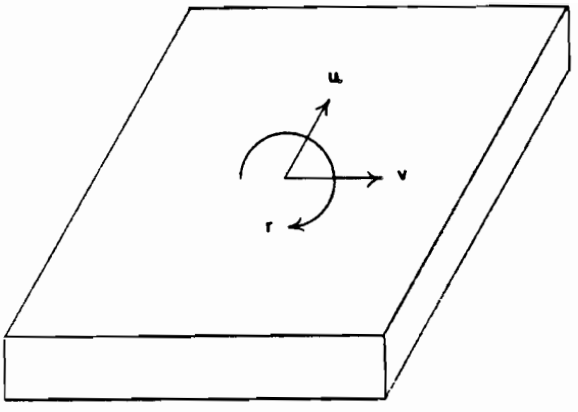


Figure 4.1 Assignment of the commanded velocities to the three axes of the joystick.

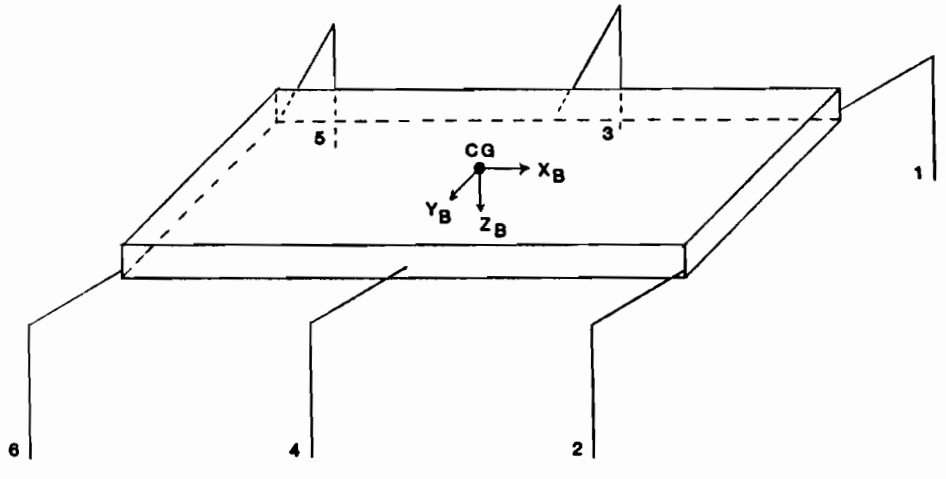


Figure 4.2 The body coordinate system for the OSU Hexapod.

(3) r represents the commanded turning rate about the z-axis of the body coordinate system (rotational velocity).

Since the three joystick axes are completely independent, the operator can command any combination of these three body velocity components.

The derivation of the equations of motion in this leg placement algorithm is based upon the observation that, in general, as the vehicle's body travels in the xy plane, the center of gravity of the body can be considered to be moving along a circular arc. The radius and center of this circular arc can be expressed as a function of the commanded velocities u, v, and r. The concept of moving along a circular arc may not be completely obvious. In the following discussion the equation's will be derived in order to describe the motion of the vehicle along this circular arc.

The commanded velocity components u, v, and r can be assumed to remain constant for the period of time Δt between sampling intervals, since the joystick inputs are sampled by a digital computer.

Consider the situation depicted in Figure 4.3. At time t, the body center of gravity is at some location (X_1,Y_1) in the earth-fixed coordinate system, and the body-fixed coordinate system (X_B,Y_B) is arbitrarily oriented, as shown. At time $t + \Delta t$, the body center of gravity has moved to location (X_2,Y_2) in the earth-fixed coordinate system while the body-fixed coordinate system (X_B,Y_B) has rotated by an angle ψ from its original orientation. Assuming that the commanded velocity components u, v, and r remain constant over the time interval Δt, then the translational velocity $\vec{v}_T$ also remains constant. This constant velocity has a magnitude v_T:

$$v_T = (u^2 + v^2)^{1/2} \qquad (4.1)$$

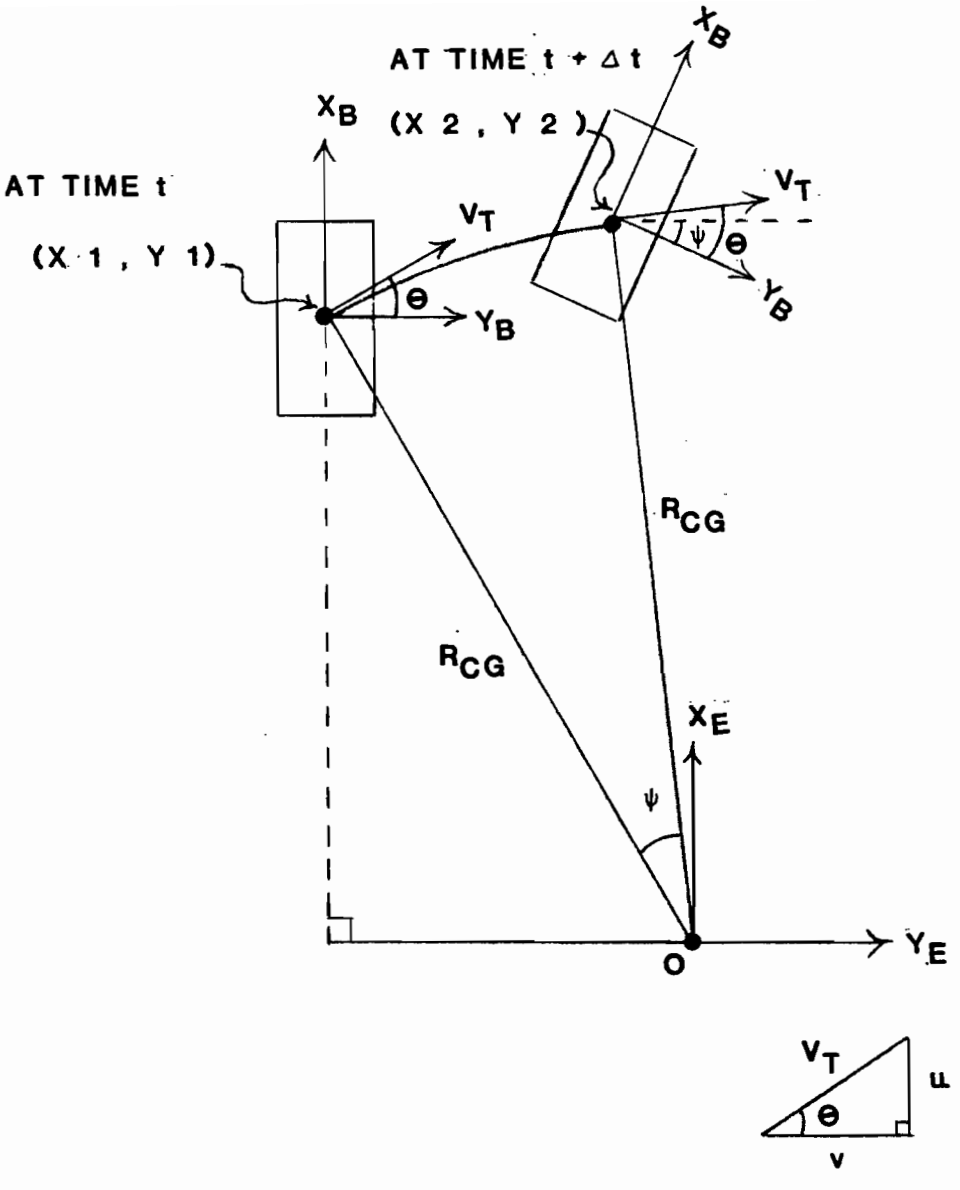


Figure 4.3 Trajectory of the body center of gravity for time period Δt.

and direction θ, with respect to the body-fixed coordinate system:

$$\theta = \arctan (u/v) \quad . \tag{4.2}$$

Likewise, the angle ψ through which the body-fixed coordinate system has rotated is given by

$$\psi = r \, \Delta t \quad . \tag{4.3}$$

Furthermore, the trajectory of the body center of gravity traces a circular arc, where v_T is the velocity component tangent to this arc. The radius of the circular arc, R_{CG}, is perpendicular to the vector $\vec{v}_T$, and its length remains constant over the time internal Δt. The point O represents the center of the circular arc.

For motion along a circular path, the tangential velocity is equal to the product of the rotational velocity and the radius of the circular path; hence

$$v_T = R_{CG} \, r \quad . \tag{4.4}$$

Therefore the length of the radius of the circular arc can be expressed as

$$R_{CG} = \left| \frac{v_T}{r} \right| \tag{4.5}$$

$$R_{CG} = \frac{(u^2 + v^2)^{1/2}}{|r|} \quad . \tag{4.6}$$

Arbitrarily selecting the point O as the origin of the earth-fixed coordinate system, the initial location of the body center of gravity with respect to the center of the circular arc is

$$X_1 = R_{CG} \sin (90° + \theta)$$
$$X_1 = R_{CG} \cos \theta \tag{4.7}$$

$$Y_1 = R_{CG} \cos (90° + \theta)$$
$$Y_1 = -R_{CG} \sin \theta \quad . \tag{4.8}$$

The location of the body center of gravity after the time interval Δt is

$$X_2 = R_{CG} \sin (90° + \theta - \psi)$$
$$X_2 = R_{CG} \cos (\theta - \psi) \qquad (4.9)$$
$$Y_2 = R_{CG} \cos (90° + \theta - \psi)$$
$$Y_2 = -R_{CG} \sin (\theta - \psi) \quad . \qquad (4.10)$$

Hence, the change in position of the body center of gravity, after time Δt, is

$$\Delta x = X_2 - X_1 = R_{CG} \cos (\theta - \psi) - R_{CG} \cos \theta$$
$$\Delta x = R_{CG} [\cos (\theta - \psi) - \cos \theta]$$
$$\Delta x = R_{CG} [\cos \theta \cos \psi + \sin \theta \sin \psi - \cos \theta] \qquad (4.11)$$

$$\Delta y = Y_2 - Y_1 = -R_{CG} \sin (\theta - \psi) - (-R_{CG} \sin \theta)$$
$$\Delta y = -R_{CG} [\sin (\theta - \psi) - \sin \theta]$$
$$\Delta y = -R_{CG} [\sin \theta \cos \psi - \cos \theta \sin \psi - \sin \theta] \quad . \qquad (4.12)$$

But, as seen in Figure 4.3,

$$\cos \theta = \frac{v}{(u^2 + v^2)^{1/2}} \qquad (4.13)$$

and

$$\sin \theta = \frac{u}{(u^2 + v^2)^{1/2}} \quad . \qquad (4.14)$$

Substituting equations (4.3), (4.6), (4.13) and (4.14) into (4.11) and (4.12), and simplifying:

$$\Delta x = \frac{1}{|r|} [v \cos (r\Delta t) + u \sin (r\Delta t) - v] \qquad (4.15)$$

$$\Delta y = \frac{1}{|r|} [v \sin (r\Delta t) - u \cos (r\Delta t) + u] \qquad (4.16)$$

Notice that in the case where there is no rotational velocity, that is r=0, the expressions for Δx and Δy must be evaluated by taking the limit as r approaches zero.

$$\Delta x\Big|_{r=0} = \lim_{r \to 0} \left\{ \frac{1}{|r|} [v \cos (r\Delta t) + u \sin (r\Delta t) - v] \right\} \qquad (4.17)$$

Applying L'Hospital's Rule,

$$\Delta x\Big|_{r=0} = \lim_{r\to 0} \{ \frac{1}{1} [-v\Delta t \sin (r\Delta t) + u\Delta t \cos (r\Delta t) - 0]\} \quad (4.18)$$

$$\Delta x\Big|_{r=0} = u\ \Delta t \quad (4.19)$$

Similarly,

$$\Delta y\Big|_{r=0} = \lim_{r\to 0} \{ \frac{1}{|r|} [v \sin (r\Delta t) - u \cos (r\Delta t) + u]\} \quad (4.20)$$

Applying L'Hospital's Rule,

$$\Delta y\Big|_{r=0} = \lim_{r\to 0} \{ \frac{1}{1} [v\Delta t \cos (r\Delta t) - u\Delta t (-\sin (r\Delta t) + 0]\} \quad (4.21)$$

$$\Delta y\Big|_{r=0} = v\ \Delta t \quad (4.22)$$

Therefore, equations (4.15) and (4.16) describe the desired position of the body center of gravity at time t + Δt with respect to its position at time t, based upon the commanded velocity components u, v, and r. For the case where there is no rotational velocity, that is r=0, the equations for Δx and Δy reduce to equations (4.19) and (4.22).

4.3 Movement of Feet in the Support Phase

In order to propel the body center of gravity along the desired trajectory, it is necessary to determine the desired position and velocity for each of the support feet with respect to the body coordinate system. Assuming that there is negligible slippage of the support feet, the location of each support foot will remain fixed with respect to the earth-based coordinate system. In Figure 4.4, the point P is located at some fixed position in the earth-based coordinate system. Consider that the body center of gravity is to move from point A to point B in the time interval Δt. The location of the point P, in

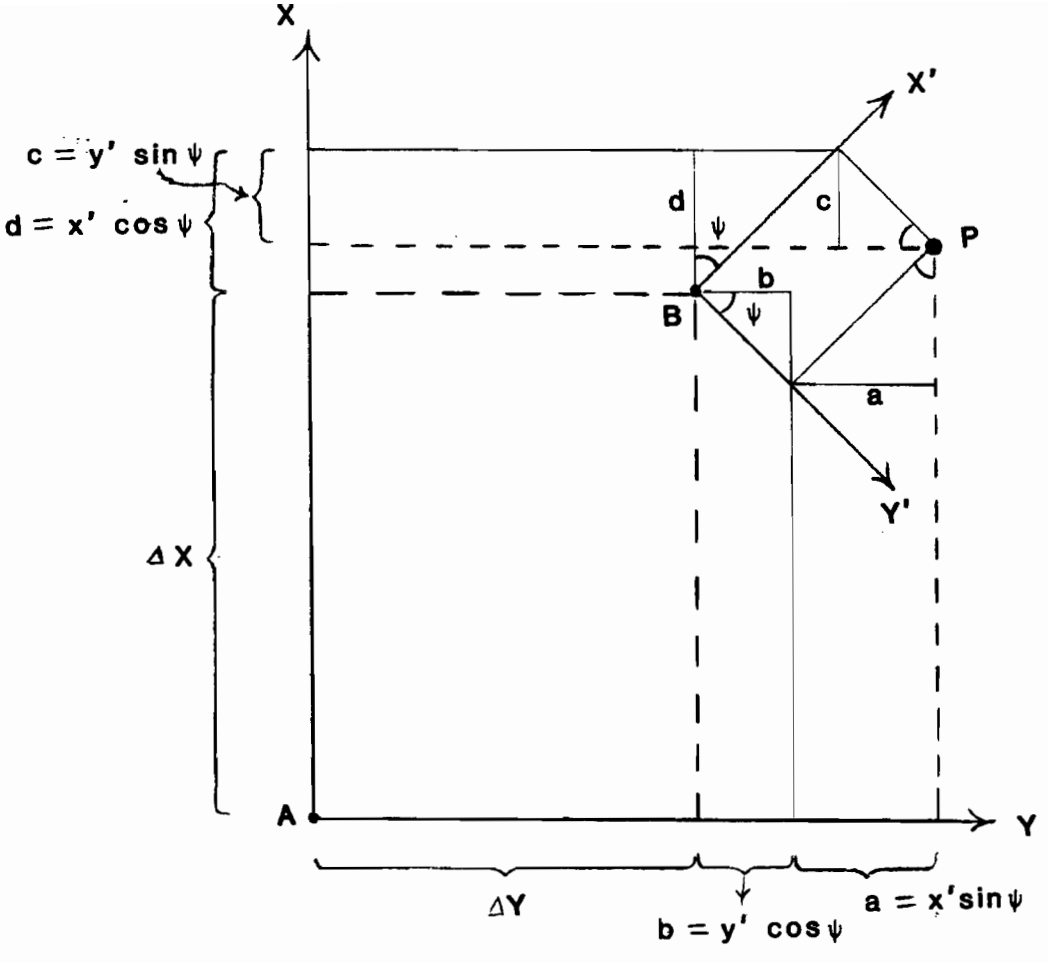


Figure 4.4 Transformation of the coordinates of a fixed point P as the body coordinate system moves from A to B.

the body coordinate system, is initially given by the body coordinates (x,y). After the time interval Δt, the location of the point P in the body coordinate system is (x',y'). From Figure 4.4 it can be seen that the relationships defining the transformation of the point P from the x'y' body-coordinate system to the xy body-coordinate system are

$$x = x' \cos \psi - y' \sin \psi + \Delta x \tag{4.23}$$

$$y = x' \sin \psi + y' \cos \psi + \Delta y \tag{4.24}$$

In these equations, Δx and Δy represent the change in position of the body center of gravity and are found from equations (4.15) and (4.16). Expressing (4.23) and (4.24) in matrix form:

$$\begin{bmatrix} x \\ y \\ \hline 1 \end{bmatrix} = \left[\begin{array}{ccc} \cos\psi & -\sin\psi & \Delta x \\ \sin\psi & \cos\psi & \Delta y \\ \hline 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x' \\ y' \\ \hline 1 \end{bmatrix} \tag{4.25}$$

By taking the inverse of this transformation matrix, the final coordinates (x',y') can be expressed in terms of the initial coordinates (x,y):

$$\begin{bmatrix} x' \\ y' \\ \hline 1 \end{bmatrix} = \left[\begin{array}{ccc} \cos\psi & \sin\psi & -\Delta x \cos\psi - \Delta y \sin\psi \\ -\sin\psi & \cos\psi & \Delta x \sin\psi - \Delta y \cos\psi \\ \hline 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x \\ y \\ \hline 1 \end{bmatrix} \tag{4.26}$$

This can be written in simpler form as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} x - \Delta x \\ y - \Delta y \end{bmatrix} \tag{4.27}$$

Therefore, if at time t a support foot is on the ground at position (x,y) in body coordinates, then at time $t + \Delta t$ that foot must be at

position (x',y') in body coordinates in order to move the body center of gravity to (Δx,Δy) with respect to its initial position at time t.

The relative velocity required to move the support foot in the body coordinate system, from position (x,y) to (x',y'), can be determined by differentiating equation (4.27) with respect to time. Recalling that $\psi = r\,\Delta t$,

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = r \begin{bmatrix} -\sin\psi & \cos\psi \\ -\cos\psi & -\sin\psi \end{bmatrix} \begin{bmatrix} x - \Delta x \\ y - \Delta y \end{bmatrix} + \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \frac{d}{dt}(x - \Delta x) \\ \frac{d}{dt}(y - \Delta y) \end{bmatrix} \qquad (4.28)$$

Since the position (x,y) represents the initial location of the support foot, this coordinate is a constant; as a result

$$\frac{dx}{dt} = 0 \quad \text{and} \quad \frac{dy}{dt} = 0 \quad . \qquad (4.29)$$

The coordinate (Δx,Δy) is given by equations (4.15) and (4.16). Therefore

$$\frac{d}{dt}(\Delta x) = \frac{d}{dt}\left\{\frac{1}{|r|}\left[v\cos(r\Delta t) + u\sin(r\Delta t) - v\right]\right\}$$

$$\frac{d}{dt}(\Delta x) = -v\sin(r\Delta t) + u\cos(r\Delta t) \qquad (4.30)$$

and

$$\frac{d}{dt}(\Delta y) = \frac{d}{dt}\left\{\frac{1}{|r|}\left[v\sin(r\Delta t) - u\cos(r\Delta t) + u\right]\right\}$$

$$\frac{d}{dt}(\Delta y) = v\cos(r\Delta t) + u\sin(r\Delta t) \quad . \qquad (4.31)$$

In matrix form:

$$\begin{bmatrix} \frac{d}{dt}(\Delta x) \\ \frac{d}{dt}(\Delta y) \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.32}$$

Substituting (4.29) and (4.32) into (4.28):

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = r \begin{bmatrix} -\sin\psi & \cos\psi \\ -\cos\psi & -\sin\psi \end{bmatrix} \begin{bmatrix} x - \Delta x \\ y - \Delta y \end{bmatrix} -$$

$$\begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.33}$$

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = r \begin{bmatrix} -\sin\psi & \cos\psi \\ -\cos\psi & -\sin\psi \end{bmatrix} \begin{bmatrix} x - \Delta x \\ y - \Delta y \end{bmatrix} -$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.34}$$

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = r \begin{bmatrix} -\sin\psi & \cos\psi \\ -\cos\psi & -\sin\psi \end{bmatrix} \begin{bmatrix} x - \Delta x \\ y - \Delta y \end{bmatrix} - \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.35}$$

Comparing the first term of (4.35) to equation (4.27),

$$\begin{bmatrix} \dot{x}' \\ \dot{y}' \end{bmatrix} = r \begin{bmatrix} y' \\ -x' \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.36}$$

Equation (4.36) produces the velocity of the support foot, relative to the body center of gravity, which is necessary to move the body center of gravity along the desired trajectory. Notice that the coordinates (x',y') are given by equation (4.27), and $(\Delta x,\Delta y)$ are given by equation (4.15) and (4.16). Therefore, the desired position and velocity for each of the support feet, with respect to the body coordinate system, is expressed in terms of the joystick commanded velocity components u, v, and r.

It should be noted that the equations which have been derived to describe the position and velocity for each of the support feet are given in rectilinear coordinates. The servo control system used to control the motion of the leg joints in the vehicle requires angular joint rates. The rectilinear velocities, $\dot{\chi}$, of the foot are related to the angular joint rates, $\dot{\Phi}$, by the Jacobian matrix J:

$$\dot{\chi} = J\,\dot{\Phi} \qquad (4.37)$$

The Jacobian, J, is a matrix of partial derivatives of the rectilinear coordinates (Px, Py, Pz) with respect to the angular joint coordinates $(P\phi, P\theta, P\psi)$

$$J = \begin{bmatrix} \partial Px/\partial P\phi & \partial Px/\partial P\theta & \partial Px/\partial P\psi \\ \partial Py/\partial P\phi & \partial Py/\partial P\theta & \partial Py/\partial P\psi \\ \partial Pz/\partial P\phi & \partial Pz/\partial P\theta & \partial Pz/\partial P\psi \end{bmatrix} \qquad (4.38)$$

Note that the relationship of (Px, Py, Pz) to $(P\phi, P\theta, P\psi)$ is determined by the geometry of the vehicle legs. Knowing the rectilinear velocities, the angular joint rates can be found using the inverse Jacobian matrix:

$$\dot{\Phi} = J^{-1}\,\dot{\chi} \qquad (4.39)$$

Thus, equations (4.15), (4.16), (4.27) and (4.36) can be used to calculate the desired position and velocity of each support foot in order to move the vehicle body in accordance with the commanded velocities u, v, and r. The calculated velocities can be converted to angular rates by equation (4.39). These desired angular joint rates are then used by the servo control system to obtain the necessary leg motions.

4.4 Foot Lift and Placement Timing

As described in Section 2.3, a periodic gait is one in which every limb of the vehicle operates with the same cycle time. Therefore, every leg is lifted and placed exactly one time within one locomotion cycle. A special subset of periodic gaits, referred to as wave gaits, has the additional constraints of a constant duty factor, β, and a constant phase interval between successive leg motions.

The Dual Tripod Algorithm can be classified as a periodic gait since every leg is lifted and placed exactly one time during each locomotion cycle. However, when designing the Dual Tripod Algorithm, it was decided that the leg motions should not be limited by the time-sequencing constraints of the wave gait formulation. In terms of stability, wave gaits have been shown to be the theoretically optimal gait, for ideal legged machines during constant speed, straight line locomotion over level terrain [18,20]. Since the major design objective of the Dual Tripod Algorithm is maneuverability, the vehicle movements cannot be restricted to straight line locomotion or constant speed. Rather than using the time-sequencing constraint of the wave gait, the Dual Tripod Algorithm is based upon geometrical considerations.

By considering the physical geometry of each of the vehicle's legs, a reachable volume can be determined for each of the vehicle's feet. The reachable volume of a foot is the locus of all points to which that foot can be moved. The leg lifting and placing motions in the Dual Tripod Algorithm are based upon the concept that, in general, the three legs forming the support tripod can remain in the support phase as long as all three of these feet are within their reachable volume. Whenever a support foot arrives at the boundary of its reachable volume, the vehicle cannot continue to move with the present support tripod. As a result, the new tripod set should be placed on the ground and the old tripod set should be lifted.

The total reachable volume of a foot is entirely dependent upon the leg and vehicle geometries. The boundaries of the reachable volume may have rather complex configurations [16,19,34]. To simplify the implementation of this concept, the reachable volume of each foot is considered to be a cylinder with a radius R_A and height H. The center axis of the cylinder is located at the equilibrium position of each foot. Figure 4.5 shows a top view of the vehicle and indicates the reachable boundary for each foot. While the reachable volume is considered to be a cylinder, the criteria used in the implementation of the Dual Tripod Algorithm is that the foot must remain within the circle of reachable area. This circle of reachable area is simply a cross-section of the cylindrical reachable volume. Notice that, as a safety feature, the radius of the reachable circle is chosen so that none of the reachable areas overlap. This feature insures that the vehicle legs never collide with adjacent legs.

In the body-fixed coordinate system, let (E_{xi},E_{yi}) represent the xy coordinates of the equilibrium position for foot i. Then the equation

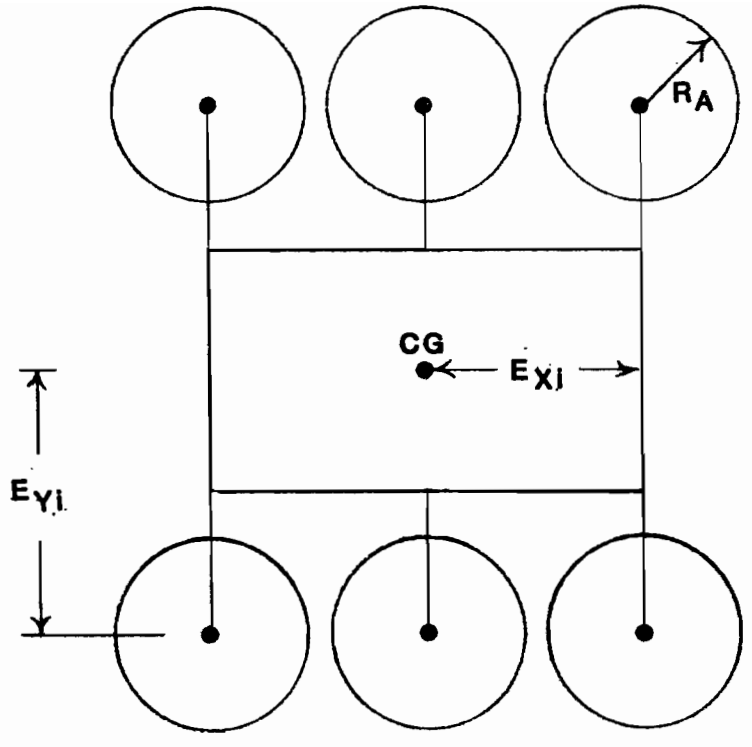


Figure 4.5 Top view of the vehicle, showing the circle of reachable area for each leg.

defining the circle of reachable area for foot i, in the body-fixed coordinate system is

$$(x-E_{xi})^2 + (y-E_{yi})^2 = R_A^2 \tag{4.40}$$

If foot i is located at position (x_i,y_i) in the body-fixed coordinate system, then that foot will be within the circle of reachable area when

$$(x_i-E_{xi})^2 + (y_i-E_{yi})^2 < R_A^2 \tag{4.41}$$

The Dual Tripod Algorithm is intended for use on a physical vehicle such as the OSU Hexapod, as opposed to being used solely on an ideal computer graphics model. While an ideal model might allow the legs to be lifted in the air or placed on the ground in zero time, which would imply an infinite rate, the actual physical system requires a finite amount of time due to the limited allowable joint rates that can be obtained. As a means of allowing sufficient time for a new tripod to be placed down before the current tripod arrives at its reachable limits, the new tripod is commanded to begin descending as soon as any foot of the current support tripod is outside the circle of radius 0.56 R_A. The size of this "switchpoint circle" is determined by considering the allowable joint rate limits and the size of the circle of reachable area. Furthermore, the selection of this radius takes into account the full geometry of the vehicle legs. This feature insures that no part of the vehicle legs will collide with adjacent legs.

Since the Dual Tripod Algorithm is intended to have a high degree of maneuverability, the joystick commanded velocity components u, v, and r are not filtered. Any change in joystick position will cause immediate changes in vehicle and leg movements. With these changes in speed and direction, it is possible that a support foot may still arrive at a reachable limit before the new tripod can be placed on the ground. For safety protection in this type of situation, the support feet are

monitored to insure that they never extend beyond the circle of radius 0.95 R_A. Whenever a support foot moves outside of the 0.95 R_A circle, all vehicle and leg movements are stopped; then the new support tripod is placed on the ground and the old support tripod is lifted. After this has been completed, vehicle and leg movements continue in the normal mode.

In addition to checking if a support foot is within the circle of radius 0.56 R_A or of 0.95 R_A, it is also necessary to determine if the foot is moving towards the boundary of the reachable area, or away from that boundary. When a foot is first placed down in the support phase, it could be very close to the edge of the reachable area; however it will be moving away from this boundary, towards the inside of the circle. As a result, the necessary conditions which are used to determine when the new tripod should begin descending into the support phase are: (1) one or more of the present support feet is outside of the circle of radius 0.56 R_A, and (2) that foot is moving towards the boundary of the reachable area. Likewise, when a support foot is outside of the circle of radius 0.95 R_A, the vehicle and body motion should only be stopped if the foot is moving towards the boundary of the reachable area. The determination of whether or not a support foot is moving towards the edge of its reachable area is made by comparing the present location of the foot with its previous location. By calculating the radial distance from the center of the reachable area, (i.e., the equilibrium position of the foot) to the present location of the foot, a comparison can be made between this present radial distance and the previously calculated value of radial distance. If the comparison

$$R_{PRESENT} > R_{PAST} \tag{4.42}$$

is true, then the foot is moving towards the boundary of its reachable area. As a consequence, the new tripod set should begin descending into the support phase as discussed previously.

4.5 Where to Place a Foot Down

The selection of a foothold while placing a foot down in the support phase can be performed in various ways. One simple but relatively effective method is to place the foot down at the center of its reachable area. This requires very little calculation and allows the foot to move in any direction. However, the stride length is limited to half of the maximum possible stride length, so the amount of leg sequencing between the support phase and transfer phase is greatly increased [36]. A more effective foothold selection is one which would allow better use of the full stride length.

Consider the illustration of Figure 4.6. As the body center of gravity moves along a circular arc, a point on the ground appears to move in a circular arc around the same center, but with a different radius. The location of a support foot is fixed at its foothold location on the ground as the body center of gravity moves along the circular arc of radius R_{CG}. The support foot i will then appear to move along a circular arc of radius R_{Fi} and in the opposite direction to the desired body motion. The intersection of the arc of radius R_{Fi} with the circle of reachable area of foot i determines the usable stride length of foot i. It is desired to choose the foothold location of foot i so as to maximize its usable stride length.

To simplify the calculation of the foothold location, a constraint is added so that the desired path of each foot will pass through the equilibrium position of that foot. This constraint may lead to a

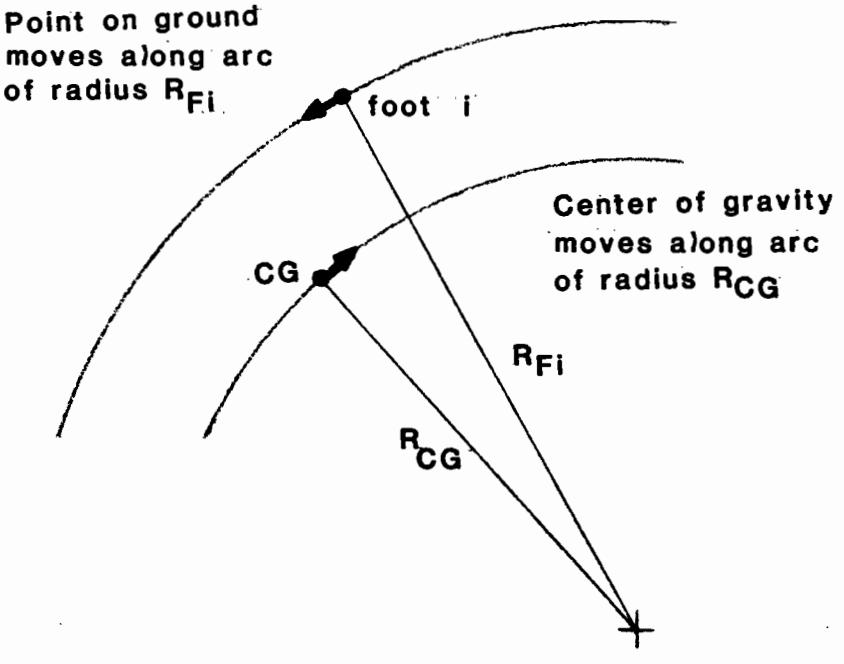


Figure 4.6 As the body center of gravity moves along a circular arc, a point on the ground appears to move along a circular arc with the same center, but different radius.

suboptimal solution in terms of maximizing the usable stride length, but it allows a suitable foothold position to be determined without excessive calculations.

The situation under consideration is depicted in Figure 4.7. The desired path of the body center of gravity is represented by the circular arc of radius R_{CG}. Point O represents the center of rotation. The relative movement of support foot i is along the circular path with center of point O and radius of R_{Fi}. This path passes through the equilibrium point of foot i. The intersection of this path with the circle of reachable area for foot i results in the two points P_1 and P_2. It is desired to calculate the location of points P_1 and P_2 for use as future foothold positions.

The location of the equilibrium position of foot i, with respect to the center of rotation, can be found from Figure 4.8. Radius R_{CG} is the distance from the body center of gravity to the center of rotation, and as shown previously:

$$R_{CG} = \frac{v_T}{|r|} = \frac{(u^2 + v^2)^{1/2}}{|r|} \tag{4.6}$$

In the body-fixed coordinate system, let (E_{xi},E_{yi}) represent the x and y coordinates of the equilibrium position for foot i, and let (X_{REi},Y_{REi}) represent the location of that equilibrium position with respect to the center of rotation. Then from Figure 4.8

$$X_{REi} = R_{CG} \sin (90° + \theta) + E_{xi}$$
$$X_{REi} = R_{CG} \cos \theta + E_{xi} \tag{4.43}$$

$$Y_{REi} = R_{CG} \cos (90° + \theta) + E_{yi}$$
$$Y_{REi} = -R_{CG} \sin \theta + E_{yi} \quad . \tag{4.44}$$

But
$$\cos \theta = \frac{v}{(u^2 + v^2)^{1/2}} \tag{4.13}$$

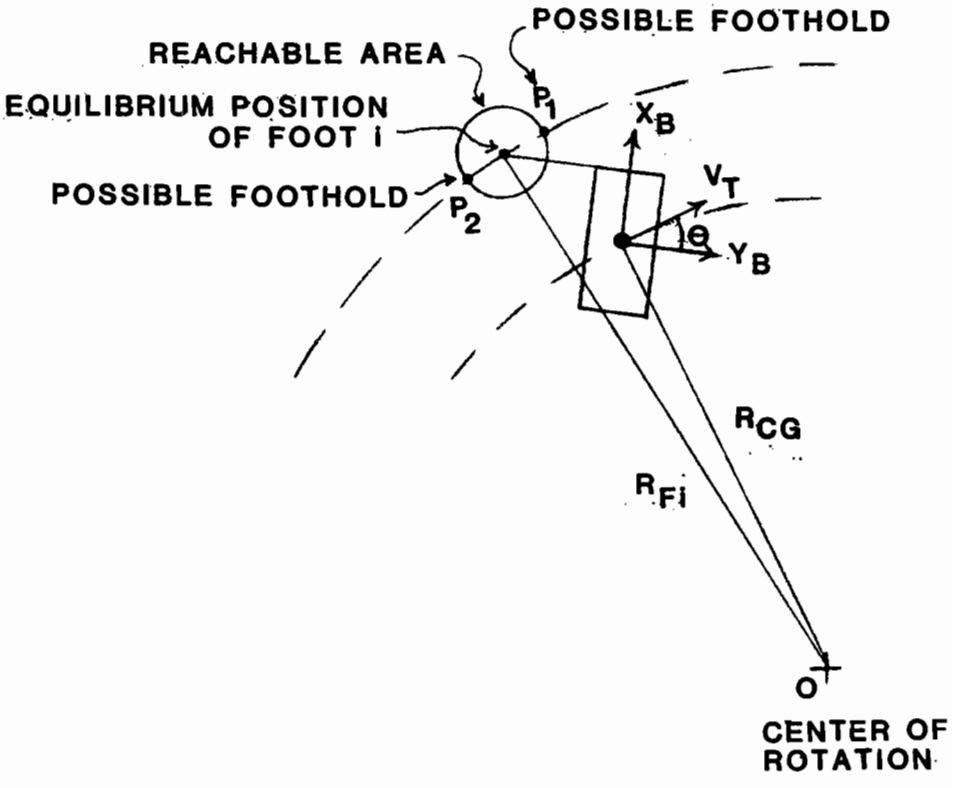


Figure 4.7 Consideration of possible footholds.

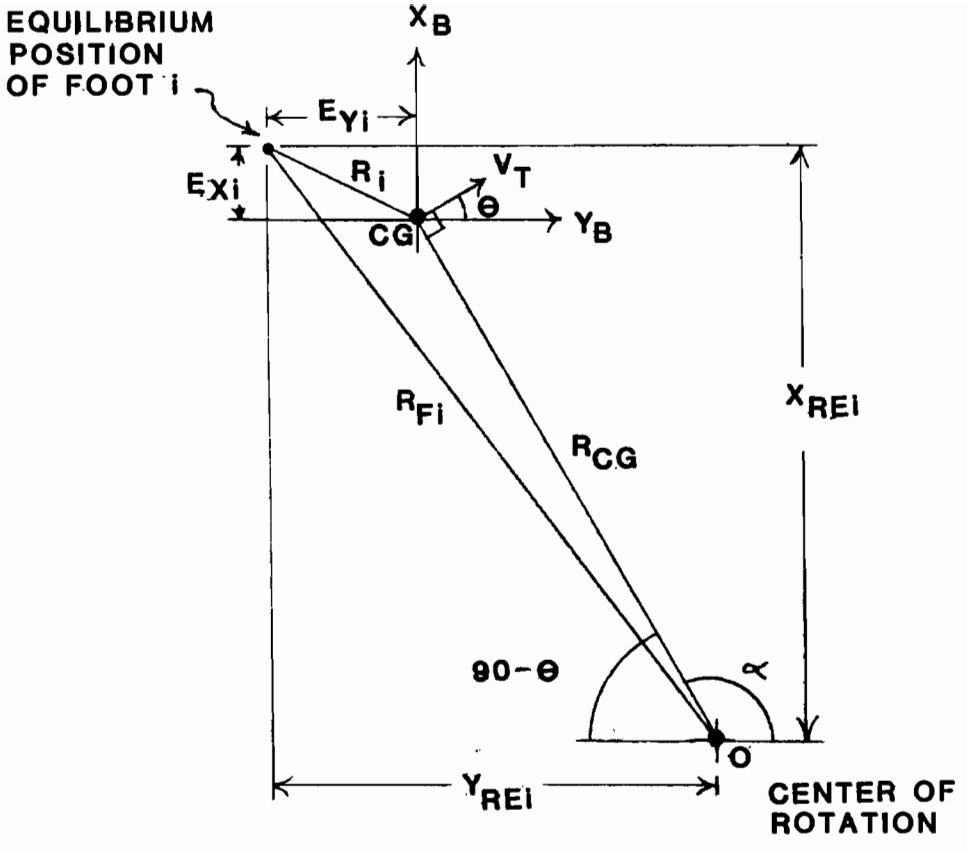


Figure 4.8 Location of the equilibrium position of foot i, with respect to the center of rotation.

and $$\sin\theta = \frac{u}{(u^2 + v^2)^{1/2}} \quad . \tag{4.14}$$

Substituting for R_{CG}, cos θ, and sin θ:

$$X_{REi} = \left(\frac{(u^2 + v^2)^{1/2}}{|r|} \right) \left(\frac{v}{(u^2 + v^2)^{1/2}} \right) + E_{xi}$$

$$X_{REi} = \frac{v}{|r|} + E_{xi} \tag{4.45}$$

and $$Y_{REi} = -\left(\frac{(u^2 + v^2)^{1/2}}{|r|} \right) \left(\frac{u}{(u^2 + v^2)^{1/2}} \right) + E_{yi}$$

$$Y_{REi} = -\frac{u}{|r|} + E_{yi} \quad . \tag{4.46}$$

Also, the radius from the center of rotation to the equilibrium position of foot i can be expressed as

$$R_{Fi} = (X_{REi}^2 + Y_{REi}^2)^{1/2} \tag{4.47}$$

and $$\alpha = \arctan \frac{X_{REi}}{Y_{REi}} \tag{4.48}$$

For each foot i in the transfer phase, the position where that foot is to be placed on the ground is determined by the intersection of the circle of reachable area with the circular arc passing through the equilibrium position. Consider the geometrical construction depicted in Figure 4.9. The circle of reachable area has radius R_A and has its center located at the equilibrium position for foot i, (X_{REi}, Y_{REi}). The circle formed by the circular arc passing through the equilibrium position has radius R_{Fi}, and the center of this circle is determined by the angle $(90° + \theta)$. Notice that the center of this larger circle is the center of rotation. This is the center of the circular path, with radius R_{CG}, representing the trajectory of the body center of gravity.

From the geometrical construction of the two intersecting circles shown in Figure 4.9, the intersection points P_1 and P_2 are the tangent

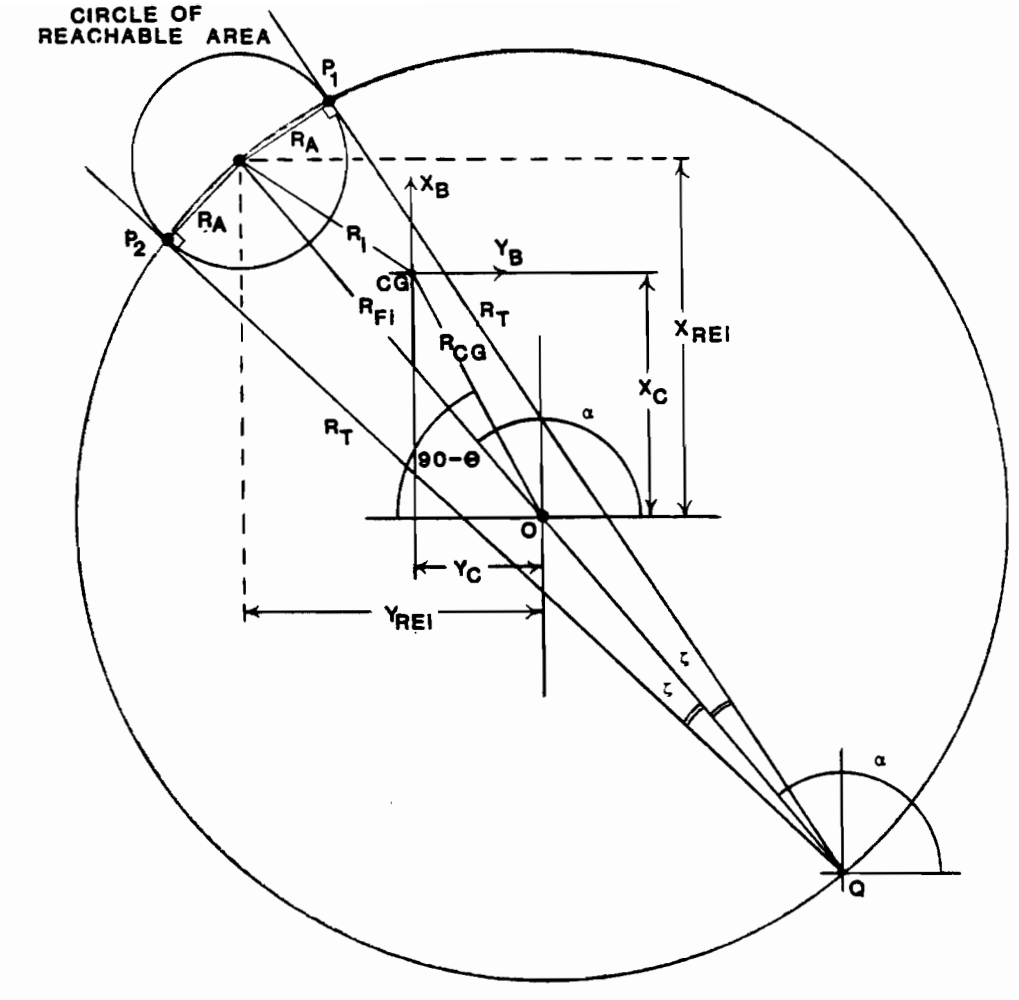


Figure 4.9 Geometrical construction of intersecting circles, to find the foothold points P_1 and P_2.

points of the lines drawn from the point Q. The point Q is located on that diagonal of the large circle which passes through the centers of the two circles. From Figure 4.9:

$$\zeta = \arcsin \frac{RA}{2R_{Fi}} \tag{4.45}$$

$$R_T = ((2R_{Fi})^2 - (R_A)^2)^{1/2} \quad . \tag{4.50}$$

The xy coordinates of points P_1 and P_2, with respect to point Q, are

for P_1:

$$X_{P1} = R_T \sin (\alpha-\zeta) \tag{4.51}$$

$$Y_{P1} = R_T \cos (\alpha-\zeta) \tag{4.52}$$

for P_2:

$$X_{P2} = R_T \sin (\alpha+\zeta) \tag{4.53}$$

$$Y_{P2} = R_T \cos (\alpha+\zeta) \quad . \tag{4.54}$$

However, as illustrated in Figure 4.10(a):

$$\sin \alpha = \frac{X_{REi}}{R_{Fi}} \quad \text{and} \quad \cos \alpha = \frac{Y_{REi}}{R_{Fi}} \quad . \tag{4.55}$$

Similarly, from Figure 4.10(b):

$$\sin \zeta = \frac{R_A}{2R_{Fi}} \quad \text{and} \quad \cos \zeta = \frac{R_T}{2R_{Fi}} \quad . \tag{4.56}$$

Using the trigonometric identity

$$\sin (\alpha - \zeta) = \sin \alpha \cos \zeta - \cos \alpha \sin \zeta \tag{4.57}$$

$$\sin (\alpha - \zeta) = \left(\frac{X_{REi}}{R_{Fi}}\right) \left(\frac{R_T}{2R_{Fi}}\right) - \left(\frac{Y_{REi}}{R_{Fi}}\right) \left(\frac{R_A}{2R_{Fi}}\right) \tag{4.58}$$

$$\sin (\alpha - \zeta) = \frac{1}{2R_{Fi}^2} [X_{REi} R_T - Y_{REi} R_A] \tag{4.59}$$

and,

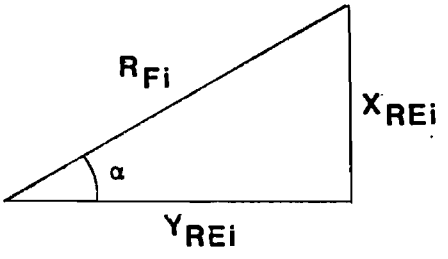
$$\sin (\alpha + \zeta) = \frac{1}{2R_{Fi}^2} [X_{REi} R_T + Y_{REi} R_A] \quad . \tag{4.60}$$

Likewise,

$$\cos (\alpha - \zeta) = \cos \alpha \cos \zeta + \sin \alpha \sin \zeta \tag{4.61}$$

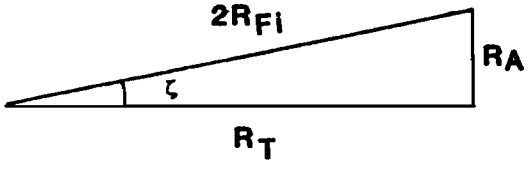
$$\cos (\alpha - \zeta) = \left(\frac{Y_{REi}}{R_{Fi}}\right) \left(\frac{R_T}{2R_{Fi}}\right) + \left(\frac{X_{REi}}{R_{Fi}}\right) \left(\frac{R_A}{2R_{Fi}}\right) \tag{4.62}$$

$$\cos (\alpha - \zeta) = \frac{1}{2R_{Fi}^2} [Y_{REi} R_T + X_{REi} R_A] \tag{4.63}$$



(a) $$\alpha = \arctan \frac{X_{REi}}{Y_{REi}}$$

$$\sin \alpha = \frac{X_{REi}}{R_{Fi}} \qquad \cos \alpha = \frac{Y_{REi}}{R_{Fi}}$$



(b) $$\zeta = \arcsin \frac{R_A}{2R_{Fi}}$$

$$\sin \zeta = \frac{R_A}{2R_{Fi}} \qquad \cos \zeta = \frac{R_T}{2R_{Fi}}$$

Figure 4.10 Trigonmetric relationships for the angles α and ζ.

and, $$\cos(\alpha + \zeta) = \frac{1}{2R_{Fi}^2}[Y_{REi} R_T + X_{REi} R_A] \quad (4.64)$$

As a result, the x and y coordinates of points P_1 and P_2, with respect to point Q, are

for P_1: $$X_{P1} = \frac{R_T}{2R_{Fi}^2}[X_{REi} R_T - Y_{REi} R_A] \quad (4.65)$$

$$Y_{P1} = \frac{R_T}{2R_{Fi}^2}[Y_{REi} R_T + X_{REi} R_A] \quad (4.66)$$

for P_2: $$X_{P2} = \frac{RT}{2R_{Fi}^2}[X_{REi} R_T + Y_{REi} R_A] \quad (4.67)$$

$$Y_{P2} = \frac{R_T}{2R_{Fi}^2}[Y_{REi} R_T - X_{REi} R_A] \quad . \quad (4.68)$$

Given that (X_P, Y_P) is the location of point P_1 or P_2 with respect to the point Q, and letting (X_P', Y_P') represent their location with respect to the center of the circular arc, then:

$$X_P' = X_P - X_{REi} \quad (4.69)$$

$$Y_P' = Y_P - Y_{REi} \quad . \quad (4.70)$$

Letting (X_P'', Y_P'') represent the location of point P_1 or P_2 with respect to the body center of gravity, then from Figure 4.9:

$$X_P'' = X_P' - X_C$$

$$X_P'' = X_P - X_{REi} - X_C \quad (4.71)$$

and $$Y_P'' = Y_P' - Y_C$$

$$Y_P'' = Y_P - Y_{REi} - Y_C \quad (4.72)$$

where $$X_C = R_{CG} \sin(90° + \theta)$$

$$X_C = R_{CG} \cos\theta \quad (4.73)$$

and $$Y_C = R_{CG} \cos (90° + \theta)$$

$$Y_C = -R_{CG} \sin \theta \quad . \tag{4.74}$$

Substituting equations (4.73) and (4.74) into equations (4.57) and (4.72):

$$X''_P = X_P - X_{REi} - R_{CG} \cos \theta \tag{4.75}$$

$$Y''_P = Y_P - Y_{REi} + R_{CG} \sin \theta \quad . \tag{4.76}$$

Using equations (4.6), (4.13), (4.14), (4.45), and (4.46):

$$X''_P = X_P - \left| \frac{v}{|r|} + E_{xi} \right| - \left| \frac{(u^2 + v^2)^{1/2}}{|r|} \cdot \frac{v}{(u^2 + v^2)^{1/2}} \right| \tag{4.77}$$

$$X''_P = X_P - \frac{v}{|r|} - E_{xi} - \frac{v}{|r|} \tag{4.78}$$

$$X''_P = X_P - \frac{2v}{|r|} - E_{xi} \tag{4.79}$$

and $$Y''_P = Y_P - \left| \frac{-u}{|r|} + E_{yi} \right| + \left| \frac{(u^2 + v^2)^{1/2}}{|r|} \cdot \frac{u}{(u^2 + v^2)^{1/2}} \right| \tag{4.80}$$

$$Y''_P = Y_P + \frac{u}{|r|} - E_{yi} + \frac{u}{|r|} \tag{4.81}$$

$$Y''_P = Y_P + \frac{2u}{|r|} - E_{yi} \quad . \tag{4.82}$$

Therefore, the intersection of the circle of reachable area with the circular arc results in the two points, P_1 and P_2, shown in Figure 4.9. Using equations (4.51) through (4.68) with equations (4.79) and (4.82), the x and y coordinates of points P_1 and P_2, with respect to the body center of gravity are

for P_1: $$X''_{P1} = \frac{R_T}{2R_{Fi}^2} [X_{REi} R_T - Y_{REi} R_A] - \frac{2v}{|r|} - E_{xi} \tag{4.83}$$

$$Y''_{P1} = \frac{R_T}{2R_{Fi}^2} [Y_{REi} R_T + X_{REi} R_A] + \frac{2u}{|r|} - E_{yi} \tag{4.84}$$

for P_2: $$X''_{P2} = \frac{R_T}{2R_{Fi}^2}\left[X_{REi}\, R_T + Y_{REi}\, R_A\right] - \frac{2v}{|r|} - E_{xi} \qquad (4.85)$$

$$Y''_{P2} = \frac{R_T}{2R_{Fi}^2}\left[Y_{REi}\, R_T - X_{REi}\, R_A\right] + \frac{2u}{|r|} - E_{yi} \qquad (4.86)$$

where $$R_T = \left[(2R_{Fi})^2 - R_A^2\right]^{1/2} \qquad (4.50)$$

$$R_{Fi} = \left(X_{REi}^2 + Y_{REi}^2\right)^{1/2} \qquad (4.47)$$

$$X_{REi} = \frac{v}{|r|} + E_{xi} \qquad (4.45)$$

$$Y_{REi} = \frac{-u}{|r|} + E_{yi} \quad . \qquad (4.46)$$

A decision as to whether to use point P_1 or P_2 can be determined by checking the direction of the rotational velocity r. If the velocity r is positive, then the body is to rotate clockwise, and point P_1 is to be used. If the velocity r is negative, then the body is to rotate counterclockwise, and point P_2 is to be used.

For the special case when there is no rotational motion, i.e. r=0, the position of the foothold where the foot is to be placed on the ground is determined by the intersection of the velocity vector V_T with the circle of reachable area. Consider the geometrical construction shown in Figure 4.11. The x and y coordinates of the point P_1, with respect to the body center of gravity, are

$$X_{P1_o} = E_{xi} + R_A \sin\theta \qquad (4.87)$$

$$Y_{P1_o} = E_{yi} + R_A \cos\theta \quad . \qquad (4.88)$$

But $\cos\theta = \frac{v}{(u^2 + v^2)1/2}$ and, $\sin\theta = \frac{u}{(u^2 + v^2)1/2}$.

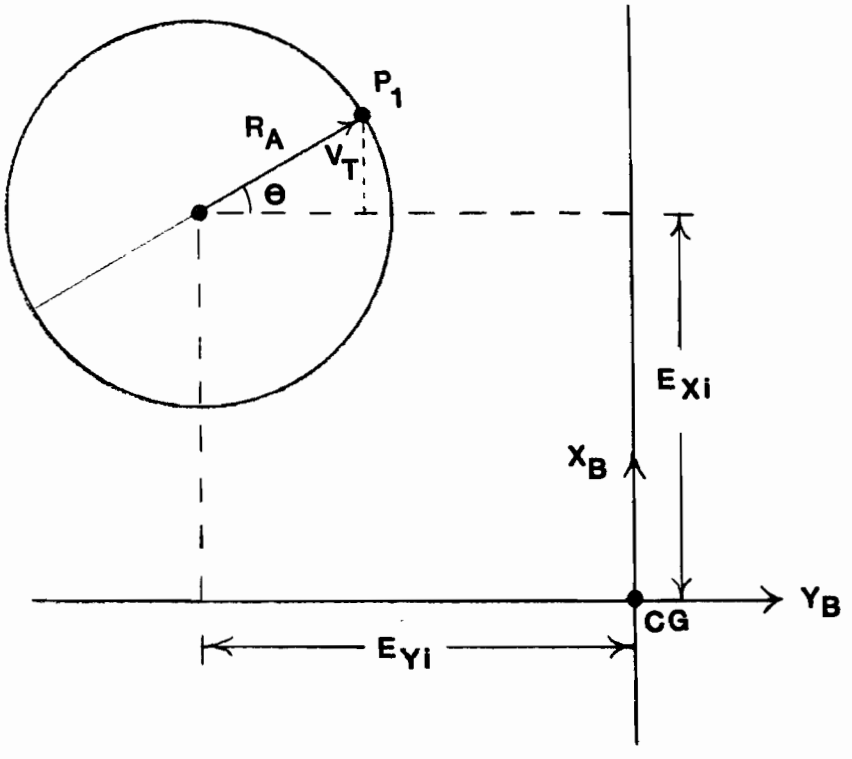


Figure 4.11 Geometrical construction to find the foothold point P_1 for the special case when there is no rotational velocity.

So $$X_{P1_o} = E_{xi} + \frac{u\ R_A}{(u^2 + v^2)1/2} \quad (4.89)$$

$$Y_{P1_o} = E_{yi} + \frac{v\ R_A}{(u^2 + v^2)1/2} \quad (4.90)$$

For this special case, only the one point P_1 is needed since the angle θ determines the proper direction.

4.6 Trajectory of a Foot in the Transfer Phase

A foot is classified as being in the Support Phase for that period of time during which the foot is in contact with the ground. For the remainder of the locomotion cycle, the foot is in the air and is classified as being in the Transfer Phase. When any of the Support Phase feet approach the boundary of their reachable limits, the new tripod should be placed into the Support Phase. After this new Support Tripod is in place, the previous Support Tripod can be lifted and moved, in anticipation, to the next desired foothold. The movement of a foot in the Transfer Phase should be performed in such a way as to avoid possible collisions with surrounding obstacles.

Figure 4.12 and 4.13 illustrates the desired trajectory, in the z-direction, for the tip of a foot in the Transfer Phase. As indicated in Figure 4.12, the trajectory of the foot tip, with respect to the earth-fixed coordinate system, should be lifted vertically upward to some predetermined height h. At this point, the foot may be moved to the location directly above the next desired foothold. The Transfer Phase foot remains at this height h above the desired foothold until the time when it is necessary for the new tripod to descend into the Support Phase. When the Transfer Phase foot is lowered to the ground, it should again follow a vertical path. Raising and lowering the Transfer Phase

Figure 4.12 Desired trajectory of a transfer phase foot, in the z-direction, with respect to the earth-fixed coordinate system.

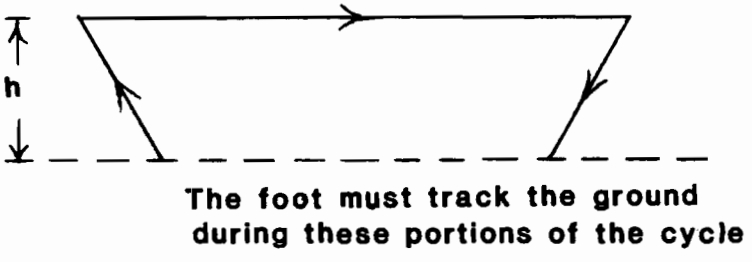


Figure 4.13 Desired trajectory of a transfer phase foot, in the z-direction, with respect to the body-fixed coordinate system.

foot along a vertical path, with respect to the earth-fixed coordinate system, protects the foot from colliding with any nearby objects. Keeping the foot at some preset height, h, insures that the transfer phase legs will not collide with any objects below this height. Further collision protection could be obtained by incorporation of proximity sensors [43], but this extra feature was not included in the algorithms developed in this research.

This same Transfer Phase trajectory, when viewed with respect to the body-fixed coordinate system, appears as shown in Figure 4.13. Notice that during the portion of the cycle where the foot is being lifted off the ground or being lowered to the ground, the foot must continue moving in the XY direction exactly as if it had remained on the ground. In this manner, the tip of the foot continues to track the point on the ground. As a result, the trajectory appears as shown in Figure 4.12 when viewed from the earth-fixed coordinate system.

Once the foot has been lifted to the predetermined height h, the foot can be moved to the XY coordinate location of the next desired foothold. The desired foothold is predicted based upon the current joystick commanded velocity inputs. Since these joystick commands can change at any time, the desired foothold location is continuously updated. As long as the Transfer Phase foot is at the height h, the XY position of this foot can be continuously modified in such a way that it always remains above the proper desired foothold. Whenever the new tripod is commanded to begin its descent to the ground, the foothold location can no longer be modified. The descending feet must track the ground in order to obtain the desired trajectory for obstacle avoidance.

4.7 Implementation and Results

The Dual Tripod Algorithm for leg placement was incorporated as a subroutine procedure which is invoked by the main program shown in Figure 3.12. The DUAL TRIPOD subroutine performs all the necessary calculations, limit checks, and logic decisions in order to determine the desired positions and velocities of the vehicle's feet. This information is passed back to the main program where it is used by the Servo Routine to control the actual leg movements.

Prior to invoking the DUAL TRIPOD routine, the vehicle configuration must be initialized. Typically, the operator can make use of two special-purpose subroutines, NORMALIZE BODY and TRIPOD INITIALIZE, to automatically place the vehicle into the proper configuration. The NORMALIZE BODY subroutine [28] moves the vehicle body and legs so that the vehicle is in a configuration referred to as the normalized position. In the normalized position, the vehicle legs are positioned such that all six feet are on the ground, each at the center of its reachable area. In addition, the body rotational angles are adjusted such that the body pitch and roll are zero. The TRIPOD INITIALIZE subroutine positions the legs to the initial configuration for a tripod gate. Legs 1, 4, and 5 remain on the ground in the support phase, and each of these feet is located at the center of its reachable area. Legs 2, 3, and 6 are lifted to the top of the transfer phase trajectory, and each of these feet is located at the center of its reachable area.

4.7.1 Flow Chart and Explanation of the Dual Tripod Algorithm

A flow chart of the DUAL TRIPOD subroutine is shown in Figure 4.14. This routine, in turn, may invoke any of three lower-level subroutines,

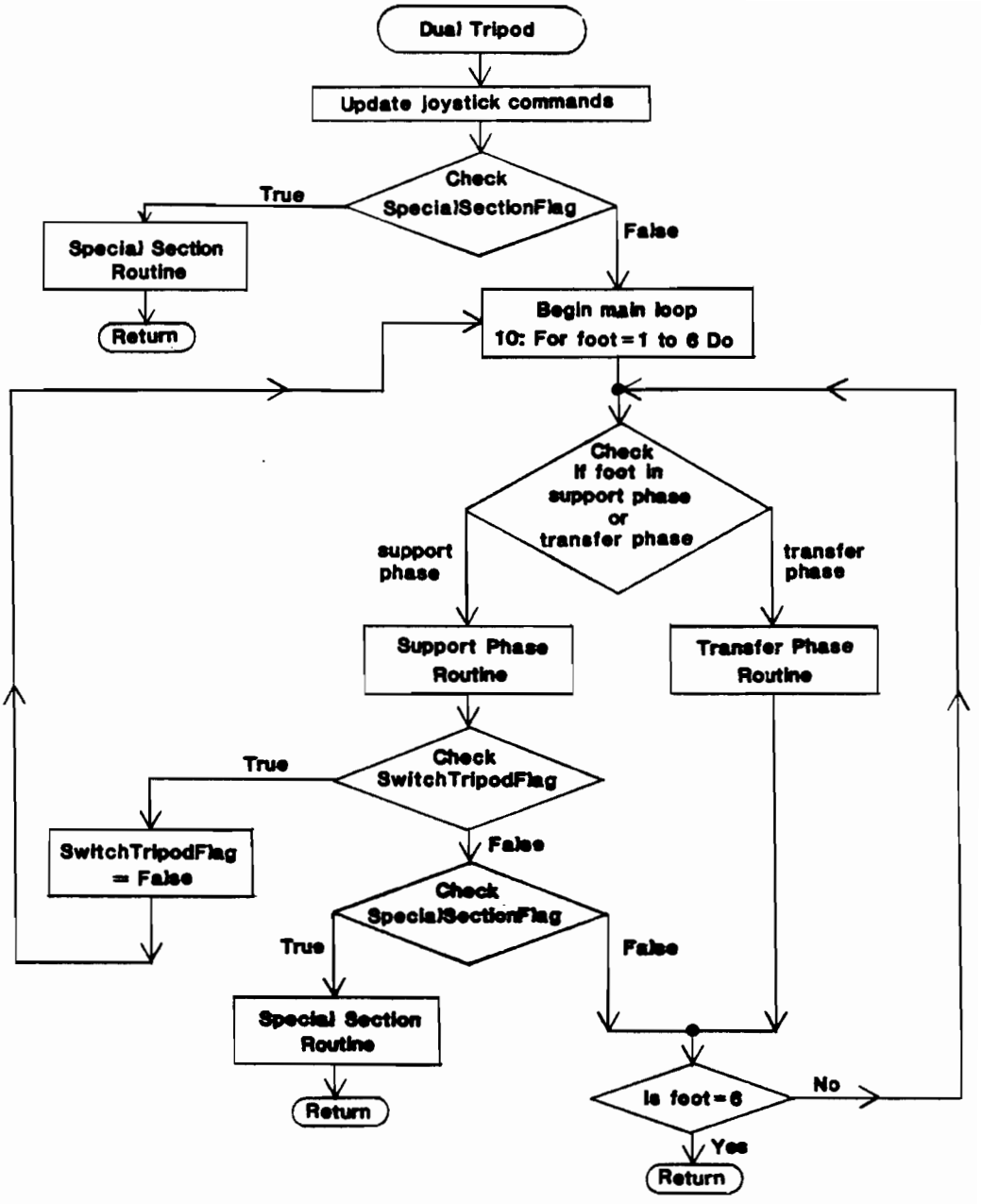


Figure 4.14 Flow chart of the DUAL TRIPOD subroutine.

SUPPORT PHASE, TRANSFER PHASE, and SPECIAL SECTION. As the names imply, the SUPPORT PHASE and TRANSFER PHASE subroutines compute the desired position and velocity for a foot which is in the Support phase or Transfer phase respectively. The SPECIAL SECTION subroutine calculates the necessary leg movements during the "special" situations when a support foot is determined to be too close to the edge of its reachable limits and the new support tripod has not yet been placed on the ground. This situation requires special action to insure the proper operation of the vehicle.

Figure 4.15 shows a flow chart for the SUPPORT PHASE subroutine. This subroutine concentrates on determining the desired movements of a given support phase leg. The subroutine can be partitioned into two sections. The first section consists of a series of logical decisions in order to determine the state of the leg under consideration. Determination of this state is based upon the location of the given leg within its reachable area, and the condition of the other vehicle legs. The second section of this subroutine calculates the desired position and velocity for the given leg, based upon the particular state of the leg at that time. When necessary, appropriate flags are set to affect the movement of other vehicle legs and to achieve proper coordination of all the legs.

A flow chart for the TRANSFER PHASE subroutine is shown in Figure 4.16. This subroutine controls the movement of the transfer phase legs in order to achieve the desired transfer phase trajectory discussed in Section 4.6. The given transfer phase leg under consideration is classified as being in one of four possible states: (1) it may be on the ground but ready to be lifted; (2) it may be in the process of being

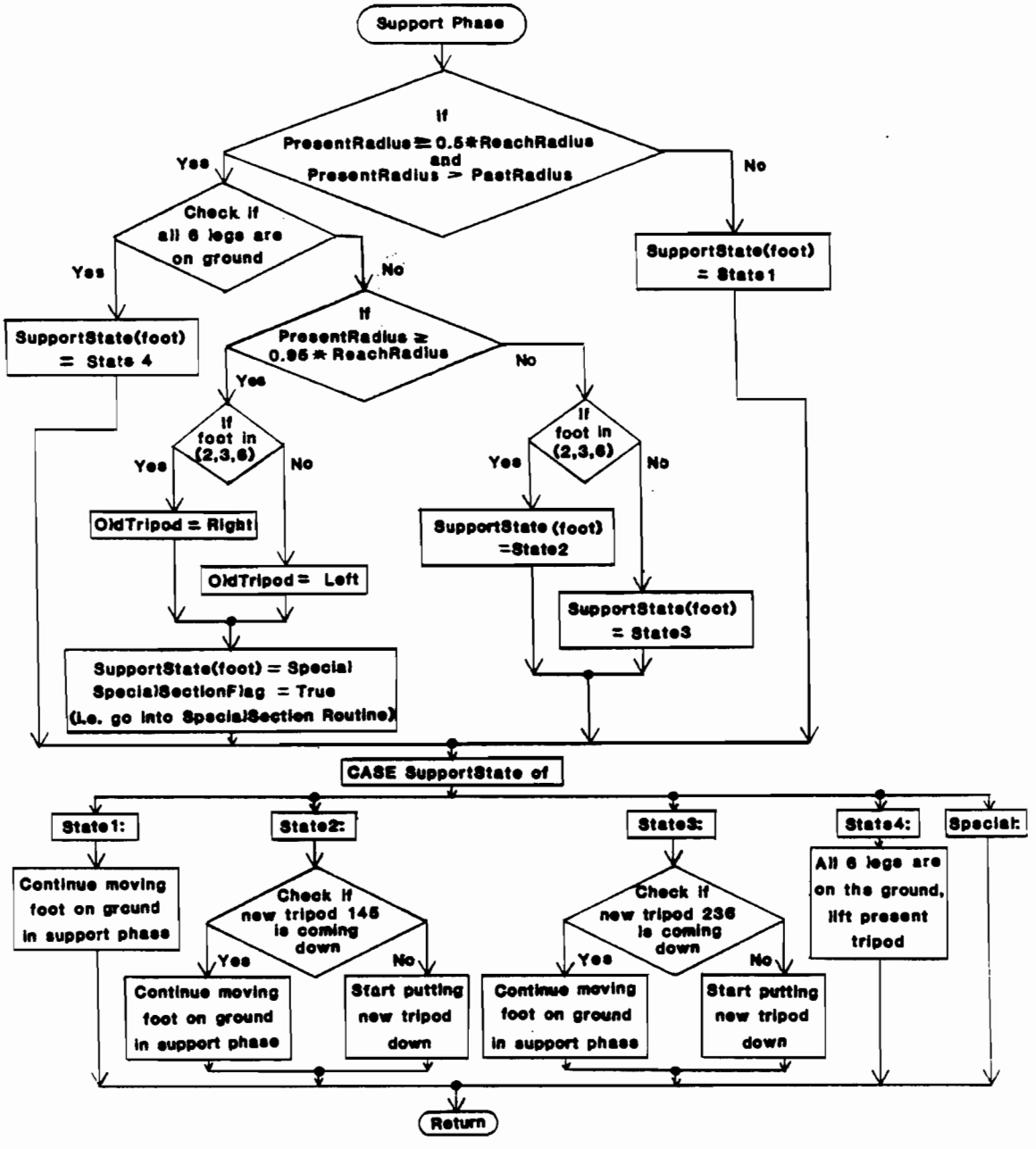


Figure 4.15 Flow chart of the SUPPORT PHASE subroutine.

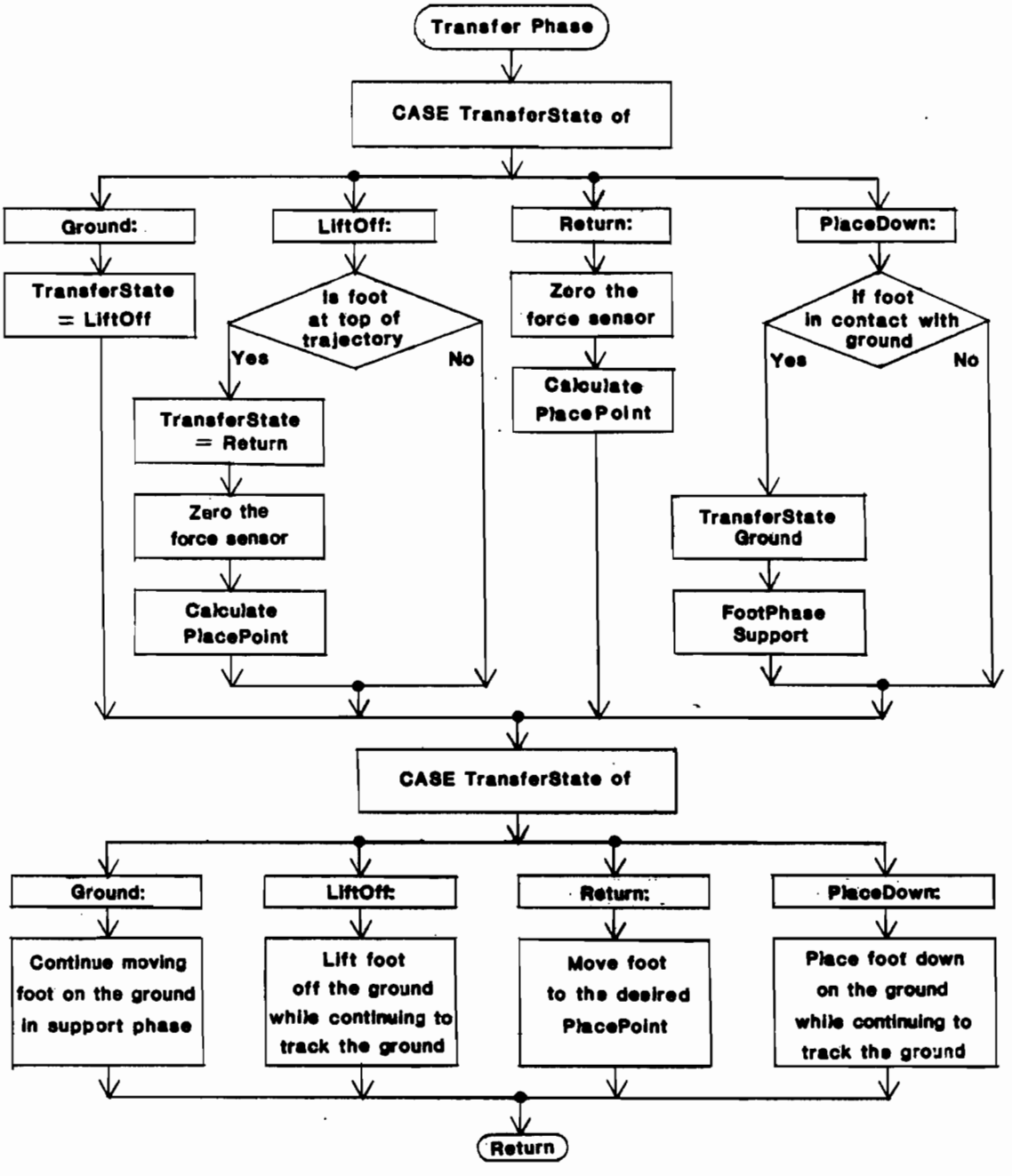


Figure 4.16 Flow chart of the TRANSFER PHASE subroutine.

lifted; (3) it may be at the top of its trajectory while being moved towards the next desired place point; or (4) it may be in the process of being lowered to the ground. The desired position and velocity of the leg are determined according to which of these four states the leg is in, at that time.

The SPECIAL SECTION subroutine is used in the situation where a support leg is near the end of its reachable limit and the new support tripod has not been placed down. A flow chart for this subroutine is shown in Figure 4.17. The SPECIAL SECTION subroutine controls the movements of all six legs, rather than concentrating on only one given leg at a time as was done in the SUPPORT PHASE and TRANSFER PHASE subroutines.

In order to insure that the support legs are not stretched beyond their reachable limits, the SPECIAL SECTION subroutine inhibits any further movement of the vehicle body. The transfer phase legs are lowered to the ground and placed into the support phase. The previous support tripod is lifted off the ground and raised to the top of the transfer phase trajectory. This new transfer phase tripod is moved to the desired place point, based upon the present joystick commanded velocities, in anticipation of the next desired foothold. At this point, the SPECIAL SECTION routine is completed and, as a result, program control is returned to the normal mode.

4.7.2 Results

The Dual Tripod Algorithm for leg placement was implemented on a computer graphics simulation of the Adaptive Suspension Vehicle (ASV) as well as on the OSU Hexapod vehicle. Figure 4.18 is a typical view of

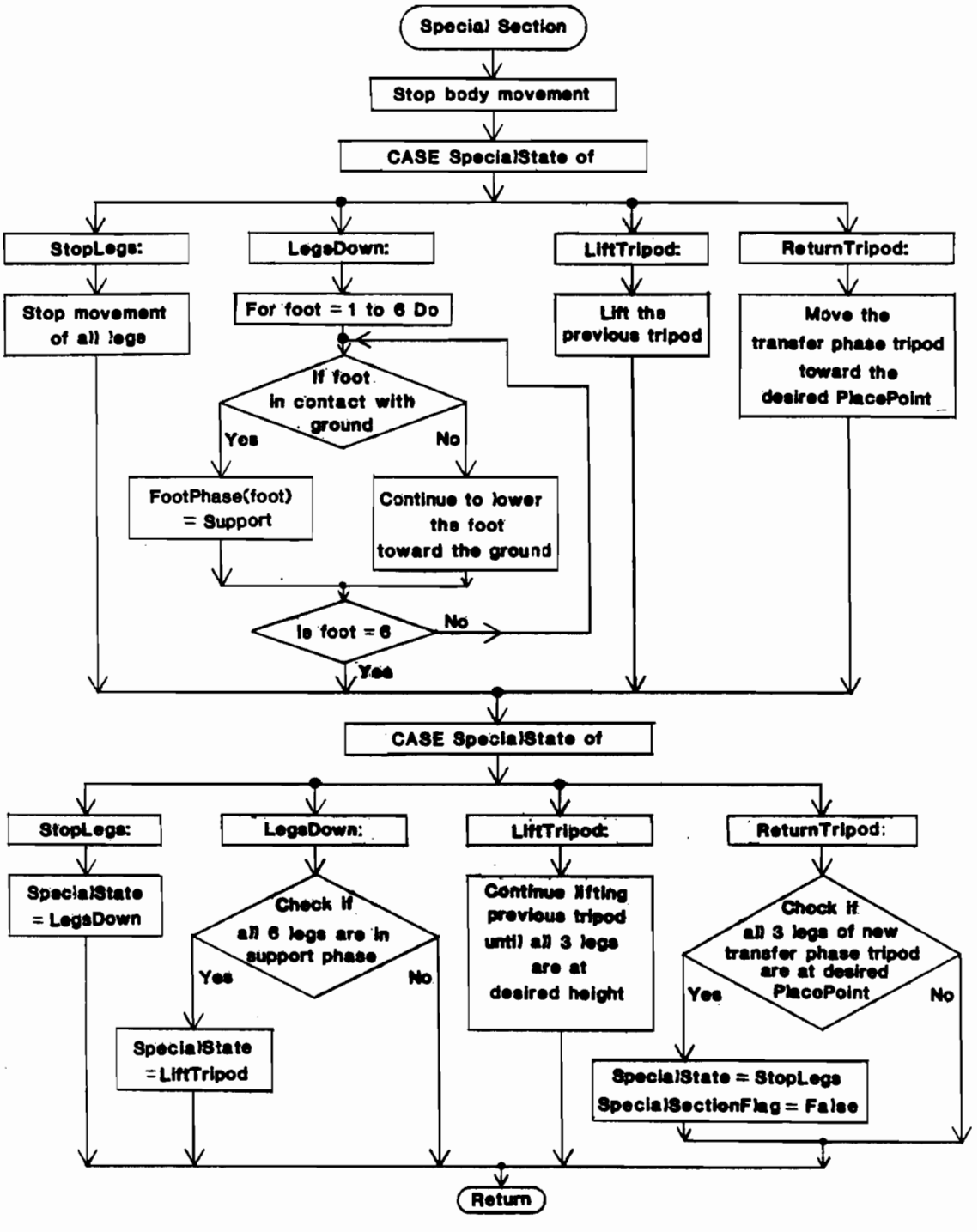


Figure 4.17 Flow chart of the SPECIAL SECTION subroutine.

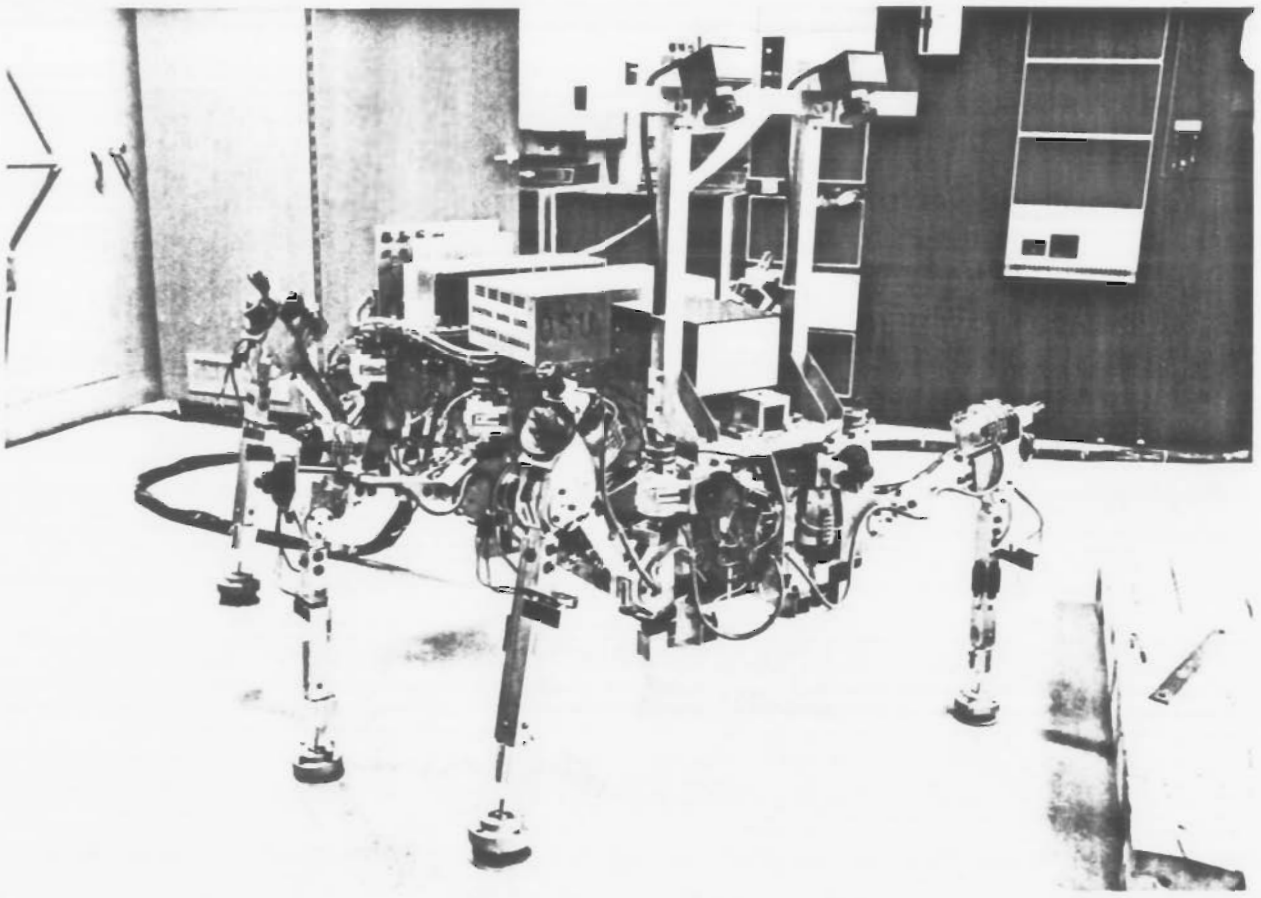


Figure 4.18 The OSU Hexapod vehicle walking in the Dual Tripod mode.

the OSU Hexapod vehicle walking in the Dual Tripod mode. In this view, the tripod formed by legs 1, 4, and 5 is in the support phase, while the tripod formed by legs 2, 3, and 6 is in the transfer phase.

The operator maintains control over the vehicle motion by the use of a three-axis joystick. The joystick commanded velocities are processed immediately, allowing the vehicle to quickly respond to these commands. Previous algorithms had required a tradeoff of speed versus agility. This was necessary because the velocity commands required a long time-constant filter in order to avoid sudden changes in command. Since the Dual Tripod algorithm is based upon the concept that leg motion is limited only by geometrical considerations there are no time sequencing constraints in this design. The result is an extremely agile walking algorithm which is well suited for the Close Maneuvering mode.

One particular refinement to the Dual Tripod Algorithm, which greatly improved the maneuverability of the vehicle, involved the SPECIAL SECTION subroutine. Initial implementation of the subroutine was designed to lower the new tripod into the support phase, lift the previous support tripod into the transfer phase, and then move this new tripod to the equilibrium position before returning control to the normal Dual Tripod mode. This original implementation resulted in satisfactory operation of the vehicle. However, it was noted that under certain conditions the algorithm would revert to the SPECIAL SECTION subroutine for several consecutive cycles before returning to normal operation. This occurred in situations where the new transfer phase tripod was forced into the support phase before reaching the actual desired foothold which corresponds to the joystick commanded velocities. Under these circumstances, the full stride length of the legs is not utilizied. In order to avoid this situation, the SPECIAL SECTION

subroutine was revised such that when the previous tripod is lifted into the transfer phase it is moved to the desired foothold, rather than the equilibrium position. The desired foothold is determined by the present joystick commanded velocities. Therefore, the new transfer phase tripod is positioned above the next desired foothold location before the algorithm returns to normal operation. If the new tripod is forced into the support phase, the full stride length can be utilized. This modification resulted in improved performance of the SPECIAL SECTION subroutine and enhanced the overall performance of the Dual Tripod Algorithm.

Through computer graphics simulation of the ASV, the potential of the Dual Tripod Algorithm to provide exceptional maneuverability was exhibited. A key factor is the ability of the algorithm to quickly respond to any combination of joystick inputs.

Using the Dual Tripod Algorithm, the OSU Hexapod vehicle has demonstrated the capability for maneuvers such as walking forward and backward, sidestepping, and turning in place. More importantly, the vehicle can respond to any combination of the joystick commanded velocities: longitudinal, lateral, and rotational. Furthermore, because of the trajectory chosen for the transfer phase legs, and because the algorithm uses the force sensors to determine when a foot has reached the supporting surface, the Dual Tripod Algorithm enables the vehicle to travel over rough terrain while maintaining its high degree of maneuverability.

4.8 Summary

The derivation of the Dual Tripod Algorithm was presented in this chapter. This algorithm treats the six legs of the vehicle as two

independent sets of tripods. At least one of these two sets of tripods supports the vehicle at all times. The tripod support pattern provides a stable stance, especially on rough terrain.

The operator can control the vehicle movement through the use of a three-axis joystick. The deflection of the joystick corresponds to commanded velocities with respect to the vehicle's body coordinate system. The operator can simultaneously control the vehicle's longitudinal velocity, lateral velocity, and rotational velocity.

The derivation of the equations of motion in this leg placement algorithm is based upon the observation that, in general, the center of gravity of the body can be considered as moving along a circular arc. The radius and center of this circular arc are determined by the commanded velocities u, v, and r. Using this approach, equations were derived for expressing the motion of the body center of gravity in response to the commanded velocities. Equations were also developed to determine the necessary movements for the support feet in order to propel the body along the desired trajectory.

The concept of defining a reachable volume for each foot provides a unique criteria for determining when to lift a foot off the ground and when to place a foot down. Using this criteria, the leg motion is limited only by geometrical considerations. The Dual Tripod Algorithm has no time-sequencing constraints as in the typical wave-gait formulations.

Lee [31] included the concept of a reachable volume in his computer simulations. However, that algorithm was based on previous wave gait concepts. Both the duty factor β and the realtive phase ϕ_i were held fixed; these quantities were then used to determine the sequencing of

the legs in the wave gait. Because of these wave gait restrictions, this algorithm was not as agile and maneuverable as the Dual Tripod Algorithm.

The concept of reachable volume, together with the concept of travelling along a circular path, also leads to a unique method for determining the location of the desired footholds for those vehicle legs which are in the transfer phase. The equations were derived for determining the desired footholds by considering the intersection of the circle of reachable area with the circular arc followed by a projected support foot.

By allowing the transfer phase feet to track the ground while they are being lifted off the ground or placed on the ground, the trajectory of the transfer phase legs insures that they will not collide with surrounding obstacles below some predetermined height. The use of force sensors on each foot allows the control algorithm to determine when a transfer phase leg has been placed down upon a supporting surface. This sensory ability enables the vehicle to walk over rough terrain.

A predominant problem in the Close Maneuvering mode is the gait transition of the vehicle legs as the direction of body motion is changed. Previous algorithms required a tradeoff of speed versus agility - that is, the velocity input commands required a filter with a long time-constant. This filtering action, and its inherent drawback of reduced response time, was required in order to avoid sudden changes in operator commands. On the other hand, by the nature of its design, the Dual Tripod Algorithm does not have a gait transition problem; therefore filtering of the input commands is no longer necessary. As a result, the Dual Tripod Algorithm is capable of responding to sudden changes in

the operator's joystick commands. In addition, the danger that a vehicle leg may collide with adjacent legs has been eliminated.

This incorporation of all of these features into the Dual Tripod Algorithm results in an extremely agile walking algorithm, which is especially well-suited for the Close Maneuvering operational mode.

CHAPTER 5

DEVELOPMENT OF PRECISION FOOTING LEG PLACEMENT ALGORITHMS

5.1 Introduction

The Precision Footing mode is, by definition, extremely operator intensive. The operator is given the capability to control individual leg movements and, in addition, can control body motion. The Precision Footing operational mode can provide maximum maneuverability, particularly for complex tasks such as climbing over large obstacles or crossing ditches. However, coordination of the leg movements in such a way as to achieve the desired locomotion of the vehicle is an extremely complex task, especially on rough terrain. Previous studies have shown that if the leg coordination is left entirely to the human operator, even a relatively simple walking machine presents such a highly complex task that the operator becomes exhausted after only a short period of operation [5]. In order to make this control mode more useful, it is essential that the Precision Footing computer control algorithm include features to relieve the operator of as much of this complex task as possible. However, this must be accomplished without greatly restricting the freedom of movement of the vehicle, which is inherent to this mode.

This chapter presents the development of three algorithms which have been implemented to achieve vehicle operation in the Precision

Footing mode. Each of these algorithms provides the operator with a different level of vehicle control and computer interaction.

A description of the Manual Tripod Leg Placement Algorithm is given in Section 5.2. The algorithm partitions the six legs of the vehicle into two independent sets of tripods, as in the Dual Tripod Algorithm. However, in the Manual Tripod Algorithm, the vehicle operator directly controls the motion of each set of tripods.

In Section 5.3, a description is given of the Individual Foot Control Algorithm. This algorithm allows the operator to directly control the motion of each leg individually. In addition, the operator can also control the motion of the vehicle body.

The development of different Automatic Body Regulation schemes is presented in Section 5.4. These regulation schemes allow automatic movement of the vehicle body in order to aid the operator in maneuvering the vehicle. A major development of this research is the derivation of the Energy Stability Margin which provides a more accurate measure of stability than previous approaches. The improvement in stability measurement due to the Energy Stability Margin is even more pronounced when the vehicle is on rough terrain.

Application of the Energy Stability Margin, together with a consideration of constraints on the kinematic limits of individual legs, leads to the development of schemes for Automatic Body Regulation. The Automatic Body Regulation schemes are incorporated into the Individual Foot Control Algorithm to provide a high degree of vehicle maneuverability while reducing the operator's burden.

All of these Precision Footing Algorithms were implemented on both the OSU Hexapod vehicle and a computer graphic simulation of the Adaptive Suspension Vehicle (ASV).

5.2 The Manual Tripod Leg Placement Algorithm

The Manual Tripod Leg Placement Algorithm treats the six legs of the vehicle as two independent sets of tripods, using the same leg partitioning as the Dual Tripod Algorithm. However, in the Manual Tripod Algorithm, the vehicle operator directly controls the movement of the tripod sets. The operator may lift a tripod off the ground, position this tripod so that all three feet are above solid footholds, then lower this tripod to the ground to support the vehicle. The operator may also control the motion of the vehicle body with respect to the feet which have remained on the ground.

5.2.1 Description of Operator Control

For implementation of the Manual Tripod Leg Placement Algorithm on the OSU Hexapod, the vehicle operator can issue commands to the algorithm by using a three-axis joystick and a selected set of keys on the computer terminal keyboard. When construction of the Adaptive Suspension Vehicle (ASV) is completed, the operator control mechanism will consist of a custom-designed arm controller and a set of function-select switches [39]. Regardless of the hardware interface, the algorithm functions remain the same.

Using a function-select switch, the vehicle operator may choose to control the movement of the transfer phase tripod with respect to the vehicle body. The operator may then use the three-axis joystick to command the desired velocities of the transfer phase tripod feet in the longitudinal, lateral, and vertical directions. The three legs forming the transfer phase tripod are moved in unison, responding to the operator's commands via the joystick. A safety feature is included in the control algorithm to insure that none of the vehicle's legs are

extended beyond the kinematic limits. In addition, the position of each foot is monitored to see that no foot is extended outside of its reachable volume. This automatic safety feature allows the vehicle operator to concentrate on positioning the transfer phase tripod, without being concerned about the possibility of overextending any of the vehicle's legs.

When the operator has positioned the transfer phase tripod in such a way that each of the three feet are above solid footholds, he can then activate a function select switch which will cause the support phase tripod and the transfer phase tripod to be interchanged. The transfer phase tripod is lowered to the supporting surface by restricting the legs to be extended only in the z-direction of the body coordinate system. The position of these feet remain fixed with respect to the x and y coordinate locations at which they were located when the operator initiated the tripod interchange. Force sensors located in each leg provide the necessary feedback to determine when each of these feet has made firm contact with the supporting surface. The downward motion of a leg is halted when that leg contacts the surface. After the three legs of the transfer phase tripod are placed on the supporting surface, the previous support phase tripod is lifted off the ground and raised to a preset height. This completes the interchange of the transfer phase tripod and the support phase tripod.

Upon completion of the tripod interchange, the operator may control the movement of the vehicle body with respect to those feet which have remained on the ground. The three-axis joystick is used to command the desired body velocities in the longitudinal, lateral, and rotational directions. Again, the control algorithm includes a safety feature to insure that none of the vehicle's support legs are extended beyond the

kinematic limits. If any support foot is extended to the edge of its reachable volume, the body motion is halted and a warning message is issued to the operator. As a result, this safety feature protects both the operator and the vehicle, while enabling the operator to maneuver the vehicle without being overly concerned about the kinematic limits of the vehicle.

After the operator has moved the vehicle body to the desired position, he may again activate the appropriate function select switch which will enable the joystick controller to be utilized for commanding the movement of the new transfer phase tripod.

A state diagram illustrating the function of the operator controls for the Manual Tripod Leg Placement Algorithm is shown in Figure 5.1. The two input signals controlled by the function-select switches are represented by the symbols S and T. When input switch T is asserted, the control algorithm operates in the Transfer State. In the Transfer State, the operator may use the joystick to control the motion of the transfer phase tripod. If the input switch T is asserted while the control algorithm is in the Transfer State, the algorithm remains in the Transfer State. On the other hand, asserting the input switch S while the control algorithm is in the Transfer State causes the tripods to be interchanged, and places the control algorithm into the Support State. The tripod interchange is accomplished automatically by the control algorithm. The present transfer phase tripod is first lowered to the supporting surface. When all six vehicle legs are on the ground, the previous support tripod is lifted off the ground and switched to the transfer phase. After this tripod interchange is completed, the operator may use the joystick to control the movement of the vehicle body, with respect to the support phase tripod. If the input switch S

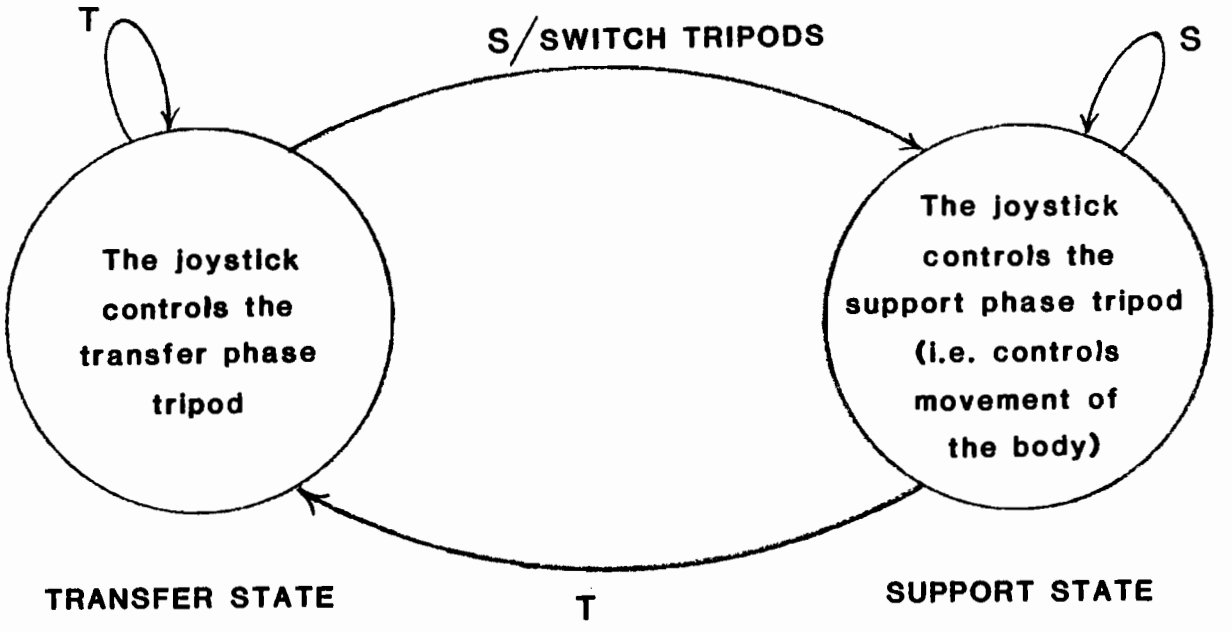


Figure 5.1 State diagram for the Manual Tripod Algorithm.

is asserted while the control algorithm is in the Support State, the algorithm remains in the Support State. Note that the only time that the two tripods are interchanged is when the control algorithm changes from the Transfer State to the Support State.

5.2.2 Implementation and Results

The Manual Tripod Algorithm for leg placement was incorporated as a subroutine procedure, which is invoked by the main program shown in Figure 3.12. The MANUAL TRIPOD subroutine determines the desired positions and velocities of the vehicle's feet. It performs all the necessary calculations, limit checks, and logic decisions to maintain coordination of the feet. The leg position and velocity information is transferred back to the main program, where it will then be used by the Servo Routine to control the actual leg movements.

Prior to invoking the MANUAL TRIPOD control algorithm, the vehicle configuration must be initialized. The operator can make use of two special-purpose subroutines, NORMALIZE BODY and TRIPOD INITIALIZE, to automatically place the vehicle into the proper configuration. The NORMALIZE BODY subroutine [28] moves the vehicle body and legs such that the vehicle is in a configuration referred to as the normalized position - i.e. all six feet are placed on the ground and the body pitch and roll are zero. The TRIPOD INITIALIZE subroutine positions the legs so as to create an initial tripod gait configuration. Legs 1, 4, and 5 remain on the ground in the support phase, with each of these feet located at the center of its reachable area. Legs 2, 3, and 6 are lifted to the top of the transfer phase trajectory, and each of these feet is located at the center of its reachable area.

5.2.2.1 Flow Chart

A flow chart of the MANUAL TRIPOD subroutine is shown in Figure 5.2. The subroutine is partitioned into two sections, which correspond to the two states indicated in the state diagram of Figure 5.1. The two "SUBCOMMAND" symbols, S and T, represent the input signals controlled by the function select switches.

During the time when the "SUBCOMMAND" is equal to T, the MANUAL TRIPOD subroutine operates in the Transfer Phase state. Movement of the vehicle body is halted. The operator may now control the motion of the transfer phase tripod, using the joystick to input the desired velocities of the tripod feet. The subroutine also includes safety checks to insure that a vehicle leg is not extended beyond its kinematic limits.

On the other hand, during the time when the "SUBCOMMAND" is equal to S, the MANUAL TRIPOD subroutine operates in the Support Phase state. When the subroutine state changes from the Transfer state to the Support state, the function of the two tripod sets are interchanged. The present transfer phase tripod is lowered to the ground, so that all six legs are now in the support phase. The previous support tripod is then lifted off the ground and switched to the transfer phase. After the tripod sets are interchanged, the operator may now control the movement of the vehicle body, using the joystick to input the desired velocity commands.

5.2.2.2 Results

The Manual Tripod Algorithm for leg placement was implemented on a computer graphics simulation of the Adaptive Suspension Vehicle (ASV), as well as on the OSU Hexapod vehicle. As discussed previously for the

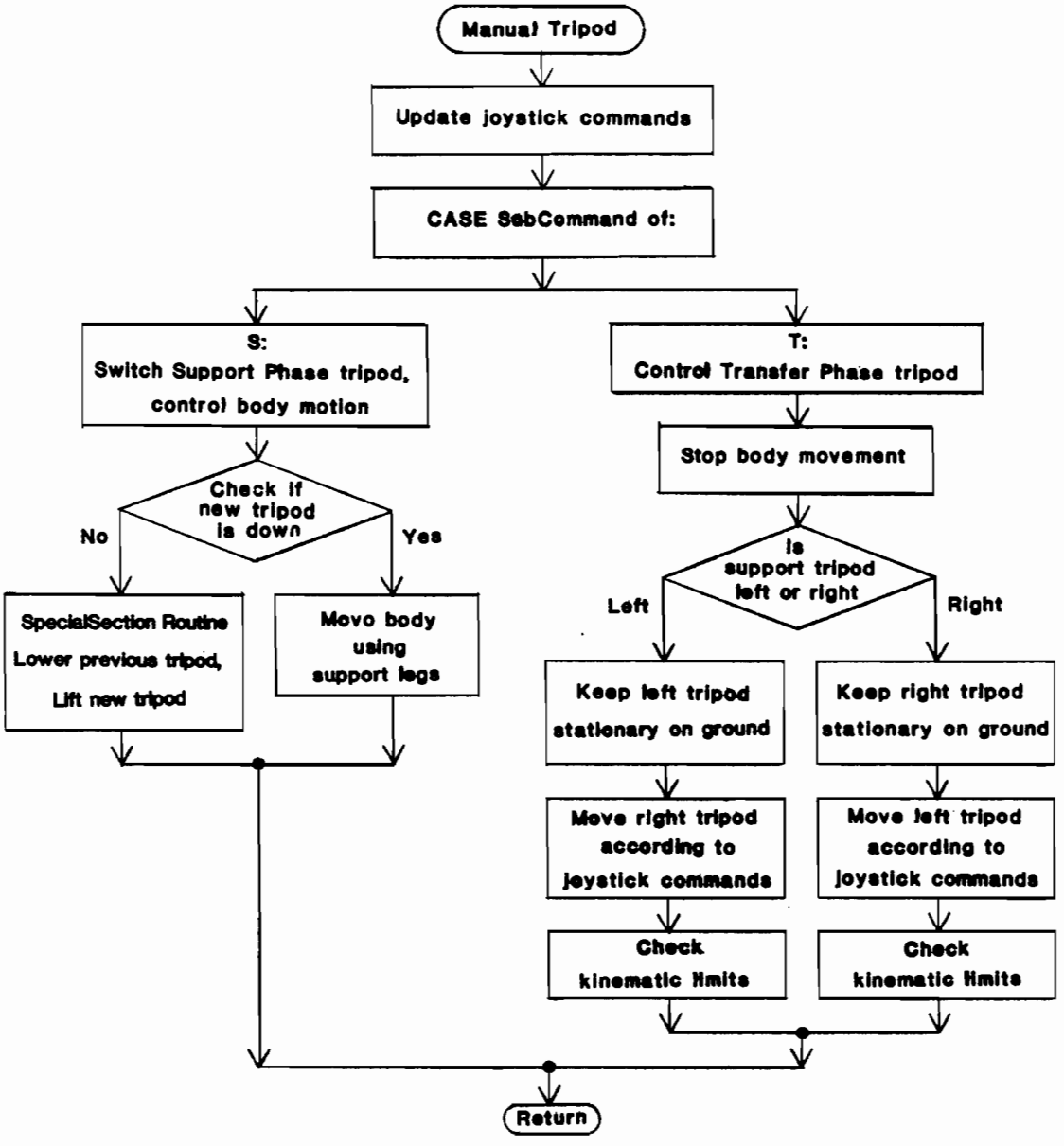


Figure 5.2 Flow chart of the MANUAL TRIPOD subroutine.

Dual Tripod Algorithm, the six legs of the vehicle are partitioned into two tripod sets. Legs 1, 4, and 5 are referred to as the left tripod set, while legs 2, 3, and 6 are referred to as the right tripod set. The operator's controls consist of a three-axis joystick and two function-select switches. Through the use of these function-select switches, the operator may choose to control the movement of either the transfer phase tripod or the support phase tripod. Whichever of these two modes is selected, the three-axis joystick provides the means of commanding the direction and velocity of the desired movement. The algorithm does not require any filtering of the joystick velocity commands, therefore the tripod movement responds quickly to the operator's commands. Furthermore, the algorithm includes a safety check to insure that no leg is extended beyond its kinematic limit.

The Manual Tripod Algorithm enables the operator to maneuver the vehicle body along trajectories similar to those which may be obtained in the Dual Tripod Algorithm. However, the Manual Tripod Algorithm permits the operator to control the precise position of each tripod. This capability is extremely useful, and in some instances absolutely essential when traversing terrain which contains regions not suitable for footholds. Using the Manual Tripod mode, the operator can position the transfer phase tripod so that those feet will be placed down on acceptable foothold locations when put into the support phase. The Manual Tripod Algorithm allows the operator to enter further into the control loop, since the vehicle operator makes the determination of terrain locations that provide acceptable foothold supports.

Initial implementation of the Manual Tripod Algorithm enabled the operator to control the motion of the transfer phase Tripod during operation in the Transfer State only. In this state, the operator may

use the three-axis joystick to command the desired velocities of the transfer phase tripod feet in the longitudinal, lateral, and vertical directions. This form of control allows the operator to position the transfer phase tripod anywhere within the boundaries of the reachable areas. However, it was found that certain vehicle maneuvers, such as turning in place, could be slightly awkward since the operator needed to devote more attention to the exact positioning of the transfer phase tripod. To reduce the awkwardness of such maneuvers, an optional feature was incorporated into the Manual Tripod Algorithm. During the transition from the Transfer State to the Support State, the present transfer phase tripod is lowered to the ground, at which time the previous support tripod is lifted off the ground and switched to the transfer phase. Normally, the previous support tripod is lifted off the ground along the z-direction of the body coordinate system, and remains at that position while the operator controls the body motion. Inclusion of the optional feature permits the operator to control the motion of the new transfer phase tripod prior to controlling the body motion. In this optional control mode, the joystick commands are interpreted as longitudinal, lateral, and rotational velocities. If the operator provides these desired velocity commands during the transition from the Transfer State to the Support State, the new transfer phase tripod will be moved to the desired foothold corresponding to the desired velocity command. Once the transfer phase legs are positioned at the desired foothold, the algorithm continues in a normal manner allowing the operator to control the body motion. This optional feature is particularly useful for positioning the transfer phase legs when performing vehicle maneuvers involving rotation of the body. This

feature positions the transfer legs to obtain maximum stride length with a minimum of effort from the operator.

Enabling the operator to control individually the movement of each of the two tripods results in a high degree of vehicle maneuverability. At the same time, the operator burden is kept to a minimum since the vehicle's legs are controlled as tripod sets rather than treating each leg independently. The Manual Tripod Algorithm is particularly useful, and in certain situations essential, for maneuverability over moderately rough terrain, where the full capabilities of the Precision Footing mode may not be required.

5.3 The Individual Foot Control Algorithm

The Individual Foot Control Algorithm enables the vehicle operator to directly control the movement of each leg individually. The operator is also able to control the movement of the vehicle body with respect to the supporting feet. The ability to control the motion of each leg individually can significantly increase the vehicle maneuverability. However, the control algorithm should provide the operator as much help as possible in order to alleviate some of the burden of manipulating the vehicle body and limbs. To aid the vehicle operator, the control algorithm is designed to provide a feedback of information to the operator via a computer graphics display terminal. In addition, the Individual Foot Control Algorithm includes various automatic monitoring features to assure the safety of the operator and vehicle.

5.3.1 Description of Operator Controls

The hardware interface between the vehicle operator and the control computer is similar to that for the Manual Tripod Algorithm. For the

OSU Hexapod, the vehicle operator can issue commands to the algorithm by using a three-axis joystick and a selected set of keys on the computer terminal keyboard. Plans for the Adaptive Suspension Vehicle (ASV), currently under construction, include an operator control mechanism consisting of a custom-designed arm controller and a set of function-select switches [39]. In either vehicle configuration, the operation of the Individual Foot Control Algorithm, which will be described in this section, functions in a similar manner, regardless of the hardware interface.

The operator may select the desired foot to be moved by choosing one of a set of six switches. The three-axis joystick may then be used to command the desired velocities of the foot in the longitudinal, lateral, and vertical directions. Movement of each individual vehicle foot is simplified for the operator since the algorithm uses Resolved Motion Rate Control [44]. As described by Whitney, this principle allows the operator to specify rectilinear velocities of the foot, rather than specifying actuator motion velocities.

The force sensors located in each foot are used to detect when that particular foot comes in contact with a supporting surface. While a foot is in the air, the operator can move that foot by making use of three-axis joystick to command any combination of longitudinal, lateral, and vertical velocities. If the foot is positioned so that it makes contact with a supporting surface, the control algorithm is designed such that it will automatically restrict the foot movement whenever the contact force is greater than a specified threshold value. This feature alleviates the operator of any concern as to whether or not a foot has been lowered the proper amount to make contact with the supporting surface. The operator needs only to command the foot to be lowered at

some velocity, and when that foot makes contact with the surface, the motion will terminate. In addition, while a foot is in contact with a supporting surface, the algorithm restricts the longitudinal and lateral movement of that foot. This protects the vehicle leg from unnecessary stress which would occur if the operator were to inadvertently command some longitudinal or lateral movement of a foot that is in the support state.

Using a function-select switch, the operator may choose to control the movement of the vehicle body, with respect to the supporting feet. The joystick may be used to command the three linear body velocities - longitudinal body motion, lateral body motion, and body altitude.

By activating a different function-select switch, the operator may choose to control the body attitude. The joystick may be used to command the three angular body velocities - the pitch, roll, and yaw rates.

As the operator controls the motion of the vehicle body, the algorithm is designed to maintain any transfer phase legs stationary, with respect to the body. However, to protect these transfer phase legs from colliding with obstacles, the force sensors are used to detect when a foot comes into contact with such an obstacle. If the contact force is greater than a specified threshold value, the control algorithm automatically treats that foot as a support phase foot. All support phase feet are kept stationary with respect to the earth-fixed coordinate system. This feature protects the transfer phase legs from collisions, without requiring special action from the vehicle operator. Therefore, the operator can concentrate on commanding the desired motion of the vehicle, while the control algorithm automatically provides certain protection features.

5.3.2 Graphics Display for Information Feedback

The Precision Footing operational mode is inherently operator intensive. The Individual Foot Control Algorithm provides the operator with the ability to control the movements of individual legs, as well as controlling the motion of the vehicle body. The vehicle operator must make decisions such as: which legs should be moved, where should a particular leg be placed, how should the vehicle body be moved, what can be done to increase the stability of the vehicle, etc. To assist the operator in making these decisions, the control algorithm is designed to provide a feedback of information through the use of a computer graphics display terminal. Figure 5.3 is a photograph showing the information presented on the graphic display terminal during operation of the OSU Hexapod. Information contained on the display consists of:

(1) The location of each supporting foot, denoted by an X.

(2) The present support polygon, denoted by the darker polygonal lines. (The support polygon is a convex polygon in the horizontal plane; the vertices of the polygon consists of the vertical projection of the support feet.)

(3) The predicted support polygon, denoted by the lighter polygonal lines. (The predicted support polygon appears only when one of the feet are being controlled individually. This polygon indicates the shape that the actual support polygon would have if the leg presently being controlled were to be placed in the support phase.)

(4) The vertical projection of the vehicle center of gravity, denoted by the small square near the center of the polygon. (The location of the center of gravity within the support

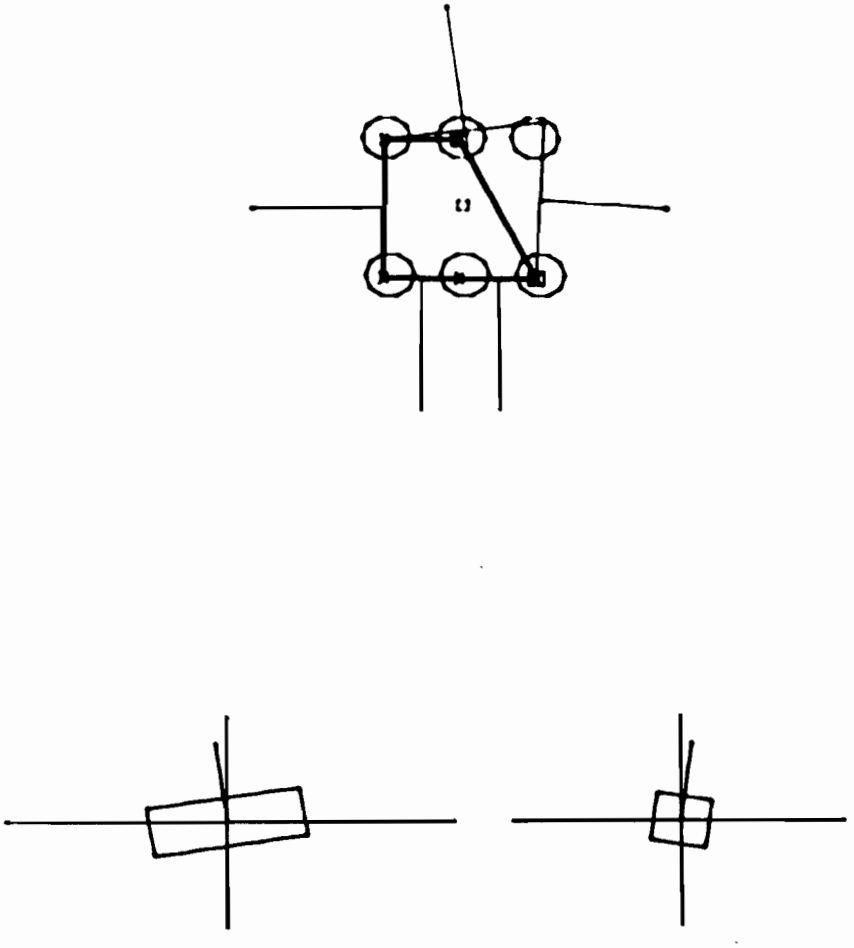


Figure 5.3 The information presented on the graphics display terminal during operation of the OSU Hexapod vehicle. The two views at the bottom portion of the screen indicate the pitch and roll of the vehicle body.

polygon provides a useful indication of the vehicle's stability.)

(5) The reachable area of each foot, denoted as circles centered at the equilibrium position of each foot. (The reachable areas provide an indication of the position of a foot with respect to its kinematic limits.)

(6) The critical feet, denoted by a square which encloses that foot. (A critical foot is defined as a foot which, if lifted, would cause the body to be statically unstable.)

(7) The Energy Stability Levels, denoted as line segments perpendicular to the lines forming the support polygons. (The Energy Stability Levels represent a new measure of vehicle stability that will be discussed in Section 5.4.2.)

(8) The pitch and roll of the vehicle body, denoted by the vehicle outlines at the bottom of the screen.

The graphics display provides essential information to simplify the operator's control task and, because of the graphical format, the operator can quickly assimilate the needed information. The operator can readily discern the location of the support feet, observe which feet can or cannot be moved, and examine the area within which each foot may be moved. Furthermore, the operator receives continuous feedback on the stability of the vehicle. In this way the effects of body and limb movements can be quickly evaluated, and with this prior knowledge the operator can avoid placing the vehicle in an unstable condition.

In addition, the Individual Foot Control Algorithm includes various automatic monitoring features to assure the safety of both the operator and the vehicle. Furthermore, these automatic monitoring features help to reduce the operator's burden by allowing the operator to concentrate

more on the overall control task, rather than being concerned with all the details of manual control.

One of these automatic features is the monitoring of the position of all legs in order to insure that the kinematic limits are not exceeded. A reachable volume is defined for each of the vehicle's legs. The reachable volume of a foot is the locus of all points to which that foot may be moved. For implementation in this algorithm, the reachable volume of each foot is considered to be a cylinder with a radius R_A and height H. The center of the cylinder is located at the equilibrium position of each foot. The algorithm monitors the position of each leg, and restricts any motion which would cause a foot to extend beyond its reachable volume. Therefore, the vehicle operator can freely manipulate the movement of any of the vehicle legs or the vehicle body without worrying about causing a leg to exceed its kinematic limits.

Another automatic safeguard feature is the monitoring of the position of the body center of gravity to insure that the vehicle body is not moved to a statically unstable position. The vehicle is statically stable as long as the vertical projection of the body center of gravity lies within the support pattern. The control algorithm monitors the distance from the vertical projection of the body center of gravity to each side of the support polygon. Whenever the shortest distance is less than some preset minimum value, the vehicle operator is alerted by the word "UNSTABLE" appearing on the graphics display. In addition, the control algorithm inhibits any vehicle movement which would further reduce the minimum distance from the center of gravity to the critical edge of the support polygon. This feature insures that the vehicle operator does not inadvertantly move the vehicle body to a statically unstable position. The algorithm contains an additional

safety margin since it is possible to adjust the minimum distance from the center of gravity to the edge of the support polygon, which establishes the threshold for determining stability. However, the operator retains the capability to move the body in other directions which would improve vehicle stability.

A third automatic safeguard feature inhibits the operator from lifting a critical foot, since this action would also cause the vehicle to be statically unstable. The graphics display continuously indicates all critical feet by enclosing each of those feet with a square. However, if the operator inadvertantly selects to move one of these critical feet, the graphics display alerts the operator by displaying the word "CRITICAL". In addition, the control algorithm inhibits any movement of the critical feet. Thus, the operator is permitted to move only those feet which are not classified as critical.

These automatic monitoring features provide safeguards in case the operator does not heed the information provided via the graphics display, or if the operator decides to continue a certain leg or body movement until a limit is reached. Furthermore, the combination of these automatic monitoring features together with the feedback information provided via the graphics display terminal greatly simplifies the operator's task when operating the vehicle in the Precision Footing mode. All of these features were designed to relieve the burden of manipulating the vehicle body and legs.

These features also make it possible for less experienced operators to maneuver the vehicle. The graphics display provides feedback information in a straightforward graphical format. In addition, the automatic monitoring features protect the operator and vehicle in case

the operator attempts a maneuver which would otherwise cause the vehicle to become unstable.

5.3.3 Implementation and Results

The Individual Foot Control Algorithm is incorporated as a family of three subroutine procedures which are invoked by the main program shown in Figure 3.12. The LEG MOTION subroutine controls the movement of the individual leg selected by the operator. The LINEAR MOTION subroutine interprets the three-axis joystick commands as linear body velocities along the three-coordinate axis of the vehicle body. This subroutine controls the movement of the support legs in order to achieve the desired linear body motion. Likewise, the ANGULAR MOTION subroutine interprets the three-axis joystick commands as angular body velocities rotating about the three-coordinate axis of the vehicle body. This subroutine controls the movement of the support legs in order to achieve the desired angular body motion. Each of these three subroutines contains safeguards to insure that no leg is extended beyond its reachable volume.

5.3.3.1 Flow Chart

The flow charts for each of the three subroutines used in the Individual Foot Control Algorithm are shown in Figures 5.4 - 5.6.

The LEG MOTION subroutine shown in Figure 5.4 controls the movement of the individual leg selected by the operator. The remaining legs of the vehicle, as well as the vehicle body, are kept stationary. The LEG MOTION subroutine makes use of the KINEMATIC LIMITS subroutine to insure that the leg under consideration is not extended beyond its

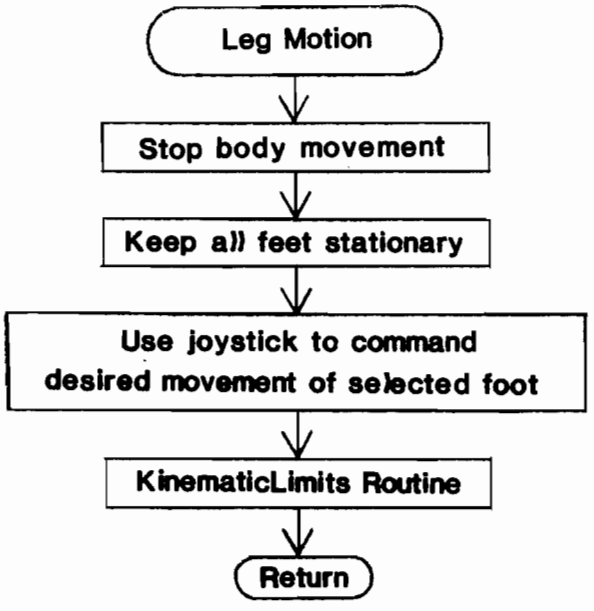


Figure 5.4 Flow chart of the LEG MOTION subroutine.

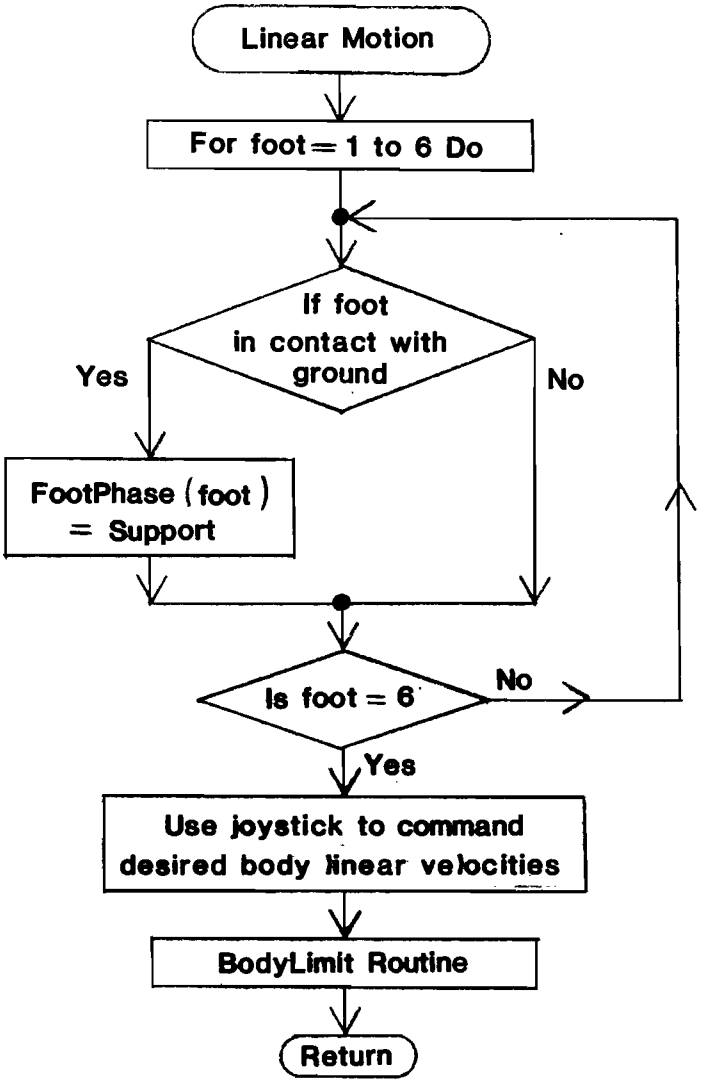


Figure 5.5 Flow chart of the LINEAR MOTION subroutine.

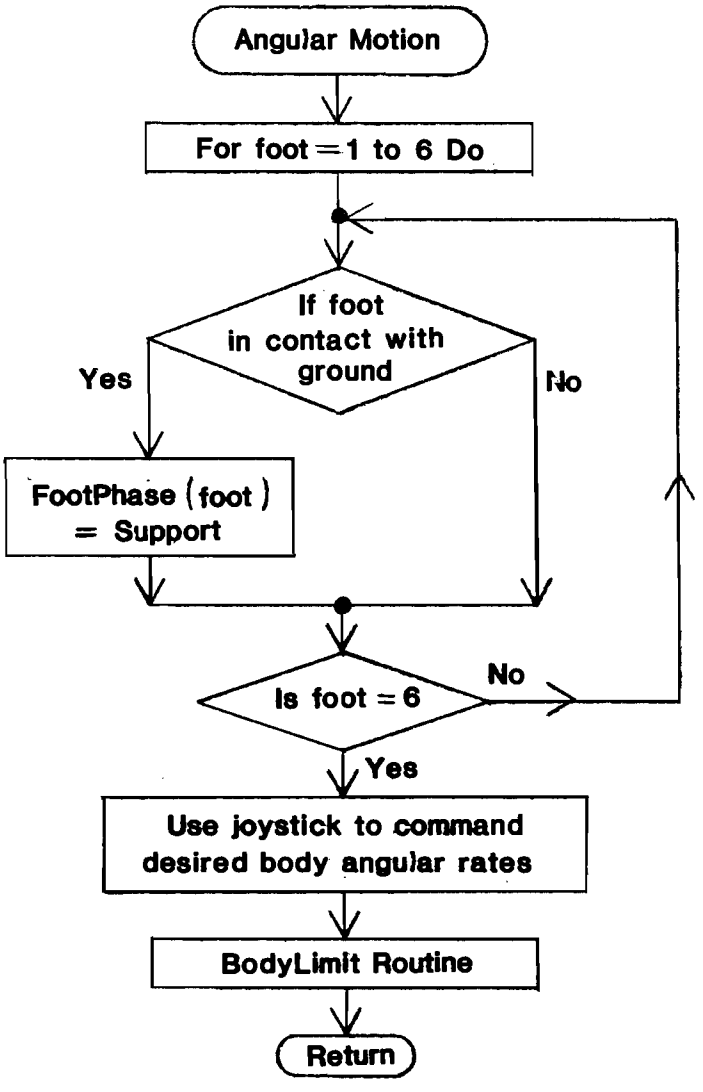


Figure 5.6 Flow chart of the ANGULAR MOTION subroutine.

reachable volume. The flow chart of the KINEMATIC LIMITS subroutine is shown in Figure 5.7. This subroutine utilizes the force sensors attached to each leg to determine if a foot is in contact with the ground. When the foot which has been selected for individual control is in contact with the ground, the KINEMATIC LIMITS subroutine inhibits the operator from moving that foot in any direction other than lifting it off the ground. While the individually controlled foot is not in contact with the ground, the KINEMATIC LIMITS subroutine checks that (1) the foot is not lifted above the maximum allowable height; (2) the foot is not lowered further than the maximum allowable distance; and (3) the foot is not moved outside of the reachable area. If the individually controlled foot approaches any of these limits, the motion of that foot is automatically inhibited.

The LINEAR MOTION subroutine shown in Figure 5.5 controls the movement of the support legs in order to move the vehicle body along the three translational body vectors. The three-axis joystick commands are interpreted as linear body velocities in the longitudinal, lateral, and vertical directions. The LINEAR MOTION subroutine makes use of the BODY LIMIT subroutine which checks that all of the support legs remain within their reachable volume. The flow chart of the BODY LIMIT subroutine is shown in Figure 5.8. If any support leg approaches the boundary of its reachable volume, further motion of the vehicle body in that direction is inhibited.

The ANGULAR MOTION subroutine shown in Figure 5.6 controls the movement of the support legs which would be necessary in order to rotate the vehicle body about the three rotational body vectors. The three-axis joystick commands are interpreted as angular body velocities, specifying the roll, pitch, and yaw rates of the vehicle body. The

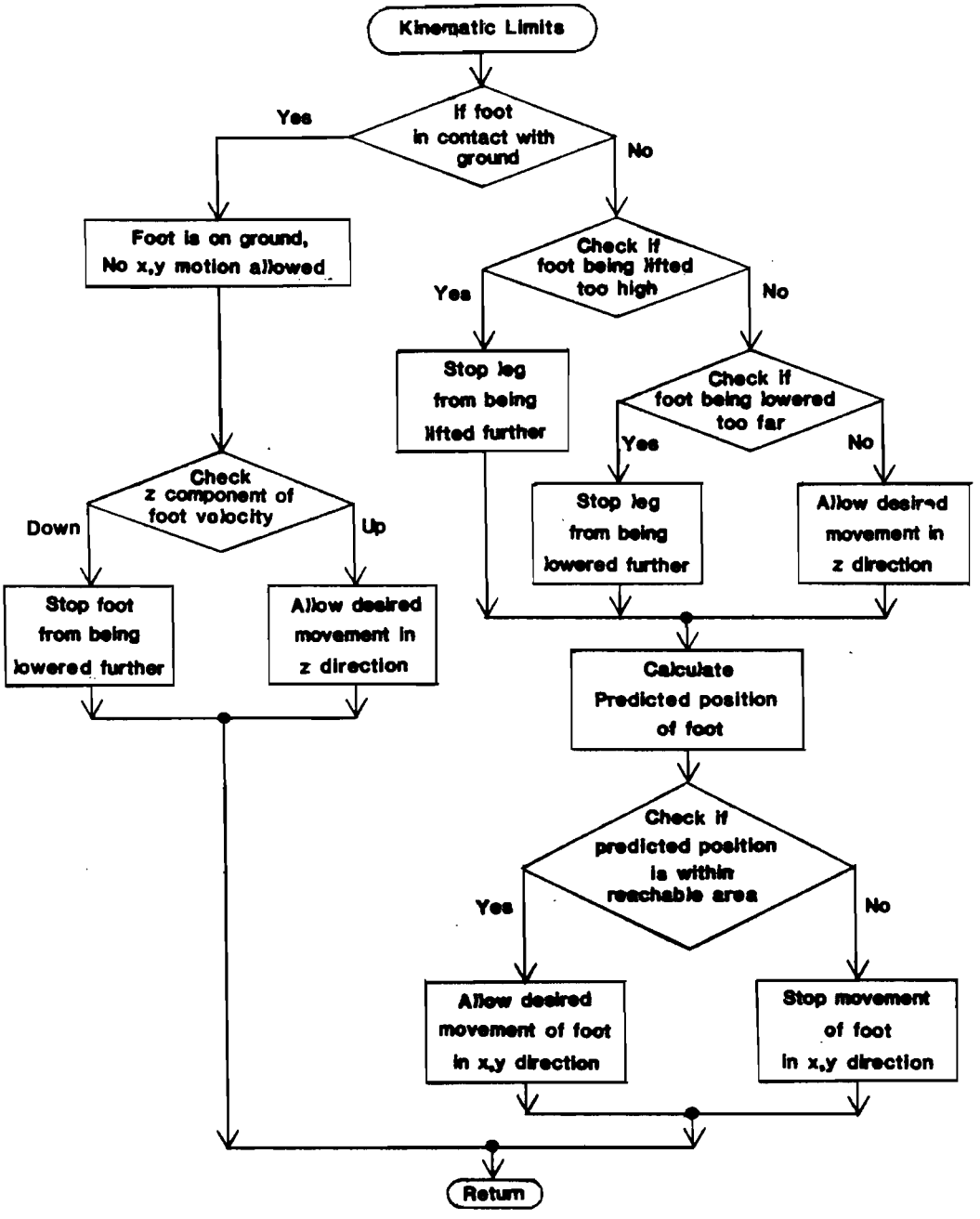


Figure 5.7 Flow chart of the KINEMATIC LIMIT subroutine.

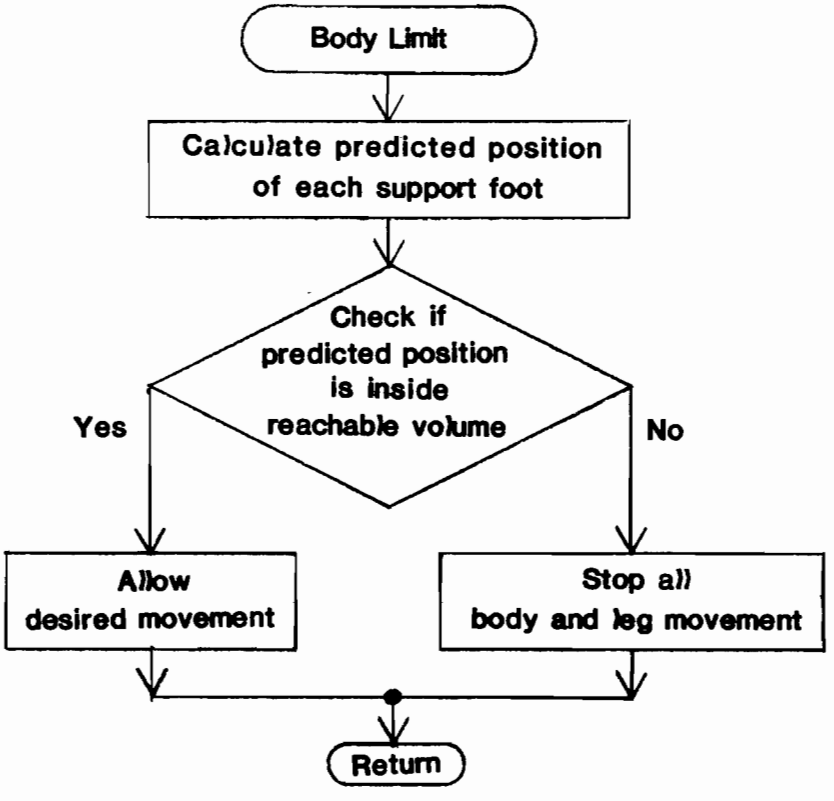


Figure 5.8 Flow chart of the BODY LIMIT subroutine.

ANGULAR MOTION subroutine also makes use of the BODY LIMIT subroutine which checks that all of the support legs remain within their reachable volume.

5.3.3.2 Results

The Individual Foot Control Algorithm was implemented on the OSU Hexapod vehicle, as well as on a computer graphics simulation of the Adaptive Suspension Vehicle (ASV). The operator's controls consist of a three-axis joystick, three function-select switches, and six leg-select switches. Through the use of the function-select switches the operator may choose to control either the linear body motion of the vehicle, the angular body motion of the vehicle, or the motion of an individual vehicle leg. The particular leg to be controlled is indicated by actuating the appropriate leg-select switch. In all cases, the three-axis joystick provides the means of inputting the direction and velocity of the desired movement.

The vehicle operator is provided with a feedback of information via the graphics display terminal. Figure 5.9 is a photograph of the information presented on the graphics display terminal during simulation of the Adaptive Suspension Vehicle (ASV). The feedback information includes the location of the support feet, the present and predicted support polygons, demarcation of the critical feet, and indications of the vehicle stability. Operation of the vehicle in the Individual Foot Control mode is greatly enhanced by the graphical feedback feature. The graphical format enables the operator to quickly interpret the information. With this feedback information the operator can easily evaluate the effects of body and limb movements. In addition, the operator is provided with continuous feedback on the stability of the

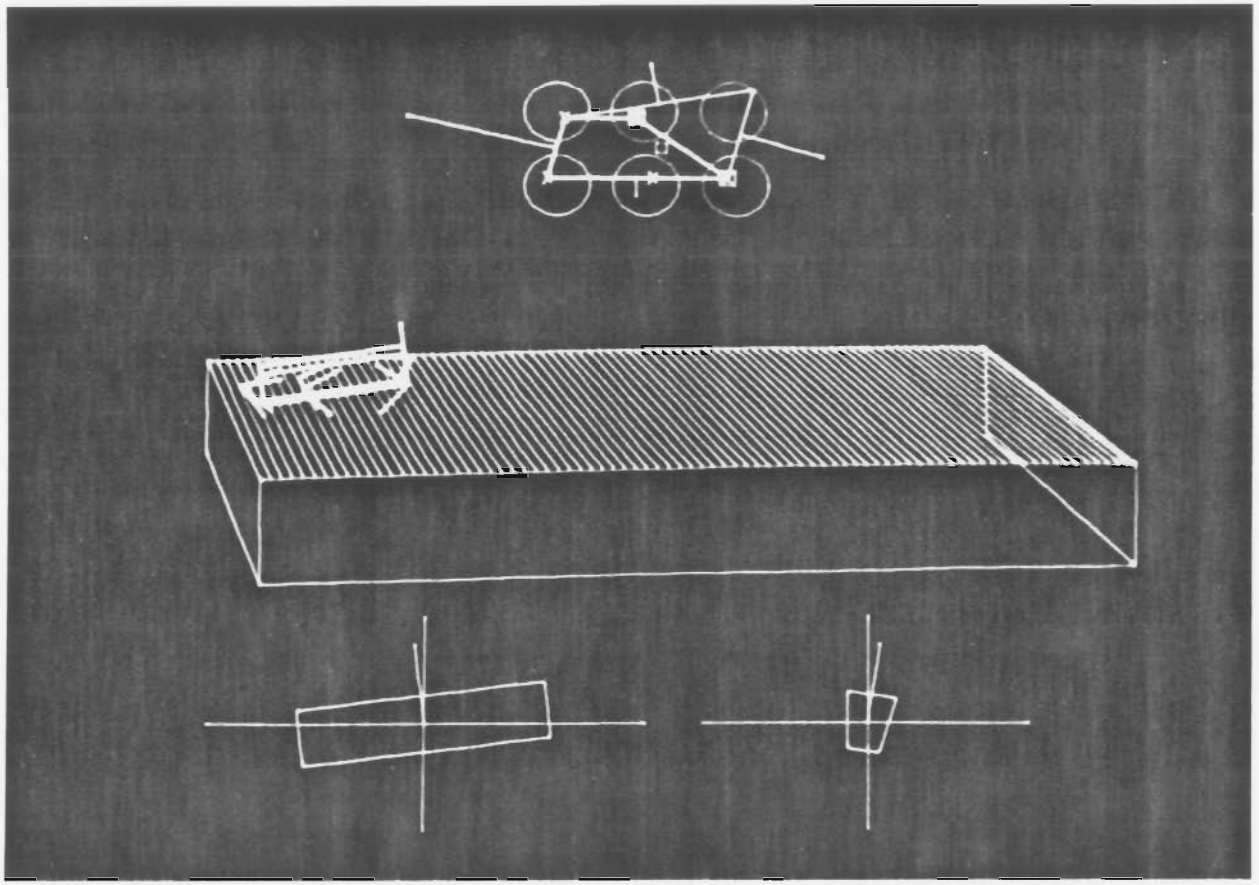


Figure 5.9 Computer graphics simulation of the Adaptive Suspension Vehicle (ASV), complete with graphical feedback information provided for the vehicle operator.

vehicle. This feedback information is extremely useful in helping the operator to determine the proper manner in which to maneuver the vehicle body and legs.

The implementation of this graphical feedback display involved careful consideration regarding the type of information to be displayed and the manner in which it should be depicted. At various stages during the development of this feature, additional complexity was incorporated into the computer program in order to provide accurate information without overwhelming the operator with an abundance of data. A particular area of tradeoff was the increased complexity required to display an accurate projected view of the vehicle feet and the reachable areas. This additional complexity was incorporated into the computer program in order to provide the operator with a better indication of the location of the vehicle feet and, simultaneously, with those positions to which the feet may be moved.

The automatic monitoring features, which are included in the Individual Foot Control Algorithm, assure the safety of both the operator and the vehicle. These automatic safeguards allow the vehicle operator to concentrate on the overall control task, rather than being concerned with the many details of manual control of the vehicle.

The ability to control the motion of each individual leg significantly increases the vehicle maneuverability. However, without the special features provided by the graphical feedback information and the automatic monitoring features, this type of control mode can represent an extremely complex task for the vehicle operator. The Individual Foot Control Algorithm relieves the vehicle operator of much of this burden of manipulating the vehicle body and legs. As a result, this algorithm enables the practical operation of a walking vehicle in

the Individual Foot Control mode without overburdening the operator. The Individual Foot Control algorithm is particularly useful for maneuvering over extremely rough terrain and for climbing over large obstacles.

5.4 Automatic Body Regulation

Maneuvering a vehicle over rough terrain is an extremely complex task. The features in the Precision Footing control algorithms which have been discussed to this point greatly simplify the operator's control task, provide feedback information, and assure safety. To enhance the capabilities of the Precision Footing operational mode, the concept of Automatic Body Regulation was developed whereby the operator can allow the computer control algorithm to automatically adjust the position of the vehicle body in accordance with some predefined criteria.

5.4.1 Description of Operation

Two Automatic Body Regulation schemes were developed and have been incorporated into the Individual Foot Control Algorithm. The two schemes are referred to as Body Accommodation and Body Stabilization [45].

5.4.1.1 Body Accommodation

As explained in Section 5.3, the Individual Foot Control Algorithm enables the vehicle operator to select and control the motion of individual legs of the vehicle. The algorithm also allows the operator to directly control the body motion in its six degrees of freedom. These features make the vehicle highly maneuverable for extremely rough terrain situations. However, each vehicle leg has a limited reach, due

to the kinematic limits of the leg. This limited reach may sometimes require the operator to perform increased maneuvering in order to place a foot at a desired foothold. In order to alleviate the operator of some of this maneuvering task, a "Body Accommodation" feature was incorporated into the control algorithm.

In the Individual Foot Control Algorithm, whenever the operator selects a vehicle leg to be manipulated, the position of that leg is monitored to insure it is never extended beyond the kinematic limits. With the Body Accommodation scheme, if this individually controlled leg reaches the kinematic limits, then the vehicle body is automatically commanded to move in such a direction as to accommodate the operator's desired motion of that individual leg. This accommodation increases the ability of the vehicle to reach a desired foothold. Of course, the position of all of the support legs must be monitored during this accommodation movement to insure that the body movement does not extend a support leg beyond the kinematic limits. Also, vehicle stability must be monitored to insure that the body is not shifted to a position where the vehicle is unstable. This concept of Body Accommodation is analogous to the situation where a human being must lean the trunk of his body in such a manner as to enable him to extend his reach to a desired position which would otherwise have been unobtainable.

5.4.1.2. Body Stabilization

While traversing a region of extremely rough terrain, the vehicle operator may find that, due to the terrain conditions, the vehicle body and legs have become oriented in a rather complex configuration. The feedback information provided via the cockpit's computer graphics display indicates such things as: the critical support feet, which

cannot be lifted; the support polygon; and the location of the vertical projection of the body center of gravity. Furthermore, the control algorithm includes safeguards to keep all legs within kinematic limits and to maintain static stability. If the display indicates that the present vehicle situation is not highly stable, the operator may desire to shift the vehicle body to a position of greater stability before proceeding with leg maneuvers. This repositioning task is simplified by incorporating a "Body Stabilization" feature into the control algorithm.

The Body Stabilization feature is activated when the operator chooses the appropriate function-select switch. Based upon the current body orientation and leg positions, the Body Stabilization scheme determines the location to which the body center of gravity should be moved in order to optimize the vehicle stability. The vehicle is then automatically shifted in such a way that the body center of gravity is located at the desired position. After Body Stabilization is completed, the operator can proceed with whatever maneuvers are desired.

5.4.2 Energy Stability Margin

An important consideration in the development of any control algorithm for a legged vehicle is the ability to maintain stability. If the vehicle becomes unstable at any point during the locomotion, there is the possibility that the vehicle will overturn, unless the vehicle can dynamically compensate in such a way as to remain upright [11]. Since the Precision Footing operational mode is primarily intended for use on irregular terrain, the vehicle operator is particularly concerned about the vehicle overturning. Furthermore, the Automatic Body Regulation concept referred to as Body Stabilization requires the capability of determining the location to which the body center of gravity should

be moved in order to optimize the vehicle stability. In order to satisfy these requirements, a new measure of stability was derived. This new stability measure, referred to as the Energy Stability Margin, provides a more accurate measure of stability than previous measures. The improvement in stability measurement is even more notable when the vehicle is on rough terrain.

5.4.2.1 Previous Measure of Stability

Previous formalizations of the criteria for determining the stability of a legged vehicle have been based upon the assumption of constant speed, straight line locomotion over flat terrain. Based upon these assumptions, McGhee and Frank [18] developed a series of definitions and theorems concerning the static stability of a legged machine. These criteria were later generalized to the situation for rough terrain [19]. The following definitions provide the basis for determining if a vehicle is statically stable:

DEFINITION 1: The support pattern associated with a given support state is the convex polygon, in a horizontal plane, which contains the vertical projection of all of the supporting feet [19].

DEFINITION 2: The magnitude of the Static Stability Margin for an arbitrary support pattern is equal to the shortest distance from an edge of the support pattern to the vertical projection of the vehicle center of gravity onto the horizontal plane [18].

Until recently, the majority of research activity has dealt with locomotion over relatively level terrain, and the previous definition of stability has been extremely useful. However, the Static Stability Margin is independent of vehicle height, since it is based solely upon the vertical projections onto a horizontal plane. Because the Precision

Footing operational mode is specifically intended for use on extremely rough terrain, it is necessary to define a new measure of stability which takes into account the effects of uneven terrain.

5.4.2.2 Motivation for the Energy Stability Margin

In order to better understand the reasons for defining a new measure of vehicle stability, consider as an example, the situation depicted in Figure 5.10. The vehicle has four supporting legs and is standing on an inclined plane, with the body horizontal. According to the previous definitions, the support pattern in this case is a rectangle formed by the vertical projection of the four supporting feet onto the horizontal plane. Assuming that the center of gravity of the vehicle is located at the center of the vehicle body, then the position shown represents the maximum Static Stability Margin for this type of situation. This position represents the maximum because the vertical projection of the center of gravity will be at the center of the rectangular support pattern. However, intuition seems to indicate that the vehicle is more likely to tip "downhill" rather than "uphill." This observation suggests that the maximum static stability would be achieved for the given situation if the body were shifted some distance in the "uphill" direction, to the point were there would be an equal likelihood of a "downward" tip or an "upward" tip.

This observation leads to the realization that the Static Stability Margin does not provide a satisfactory measure for the amount of stability when the terrain is not a horizontal plane, although it does provide a limit which indicates whether the body is stable or unstable. In order to take into account the effects of uneven terrain, a new measure of stability, called the Energy Stability Margin, has been

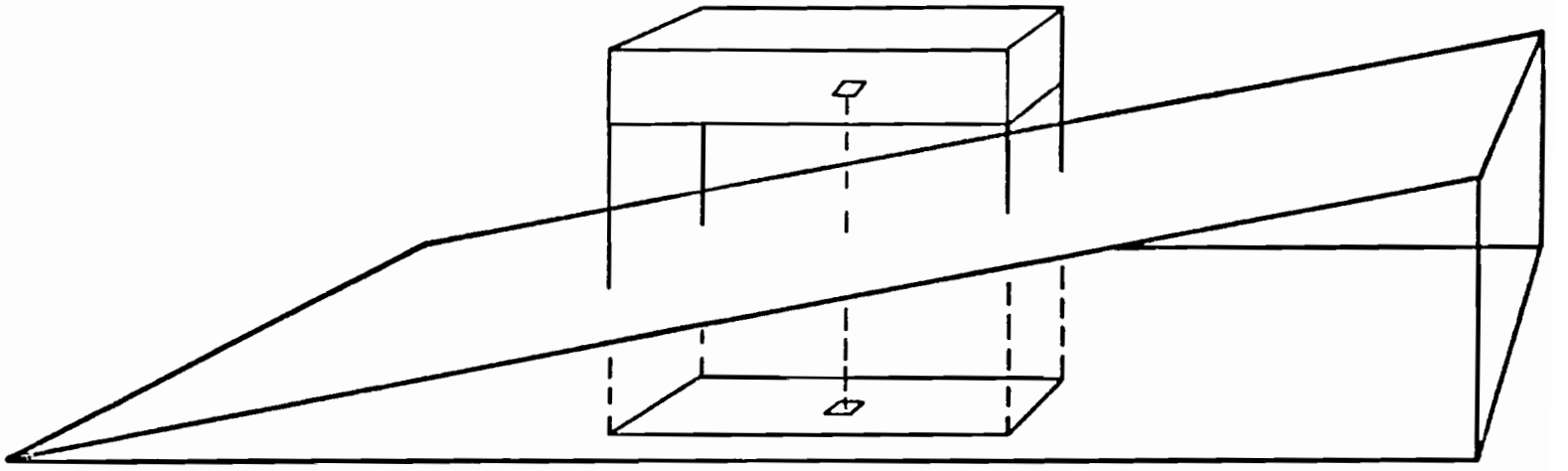


Figure 5.10 An example of the support pattern when the vehicle is standing on an inclined plane, with the body horizontal. The dotted lines show the vertical projection of the support feet. Also shown is the vertical projection of the body center of gravity.

developed. The following definitions form a basis for determining the Energy Stability Margin:

DEFINITION 3: The support boundary associated with a given support state consists of the line segments which connect the tips of the support feet that form the support pattern.

Notice that since the support pattern is a convex polygon, its vertices may not necessarily consist of all of the support feet. Likewise, the vertices of the support boundary may not necessarily consist of all of the support feet. In addition, notice that the support boundary is a three-dimensional curve, as opposed to the two-dimensional support pattern.

DEFINITION 4: The Energy Stability Level associated with a particular edge of a support boundary is equal to the potential energy required to rotate the body center of gravity about that edge, to the position where the vertical projection of the body center of gravity lies along that edge of the support boundary.

DEFINITION 5: The Energy Stability Margin for an arbitrary support boundary is equal to the minimum of the Energy Stability Levels associated with each edge of that support boundary.

The Energy Stability Margin gives a quantitative measure of the impact energy which can be sustained by the vehicle without overturning. A similar concept was discussed for biped locomotion [46], however that concept was limited to locomotion on a level horizontal surface. The present definitions apply on any type of terrain.

For extremely irregular terrain there is a geometric possibility that even when the supporting feet form a convex polygon, the body may not be able to rotate outward about a line between adjacent pairs of feet. This may occur because another foot braces the vehicle against

turning over. This unlikely possibility does not enter into the energy stability measure and therefore this measure provides a conservative estimate of instability danger.

The application of these definitions is demonstrated in Figures 5.11 and 5.12. Again, the vehicle has four supporting legs and is standing on an inclined plane, with the body horizontal. Figure 5.11 shows the support boundary, which in this case lies in the plane of the incline. Figure 5.12 shows a geometrical comparison of the Energy Stability Levels for the front edge and rear edge of the support boundary. The line segment from point F_1 (rear edge of support boundary) to the point CG (body center of gravity) represents the radius R_1 of an arc which the body center of gravity would trace if the body were rotated about the rear edge of the support boundary. If the body were rotated to the position where the body center of gravity is vertically above the rear edge of the support boundary, then the vehicle would be on the verge of instability. This would correspond to zero Static Stability Margin according to Definition 2. At this position, the body center of gravity would be at a distance of R_1 directly above the rear edge of the support boundary. The change in vertical height through which the body center of gravity is moved from its original position to this position of zero Static Stability Margin is given by the distance h_1. Therefore, the amount of potential energy required to rotate the body center of gravity about the rear edge of the support boundary, from its original position to the point of zero static stability is mgh_1, where m represents the mass of the vehicle body and g represents the acceleration due to gravity. Likewise, the amount of potential energy required to rotate the body center of gravity about the front edge of the support boundary, to the point of zero static

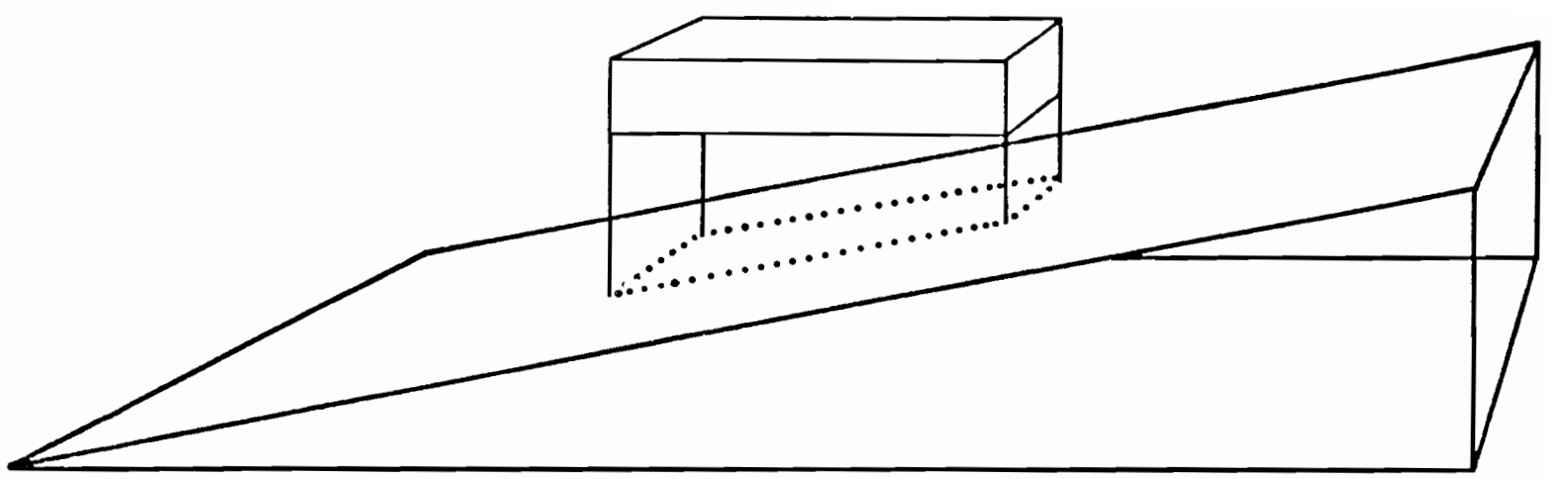


Figure 5.11 The dotted line represents the support boundary for this configuration where the vehicle is standing on an inclined plane, with the body horizontal. The support boundary is found by interconnecting the tips of the support feet that form the support pattern.

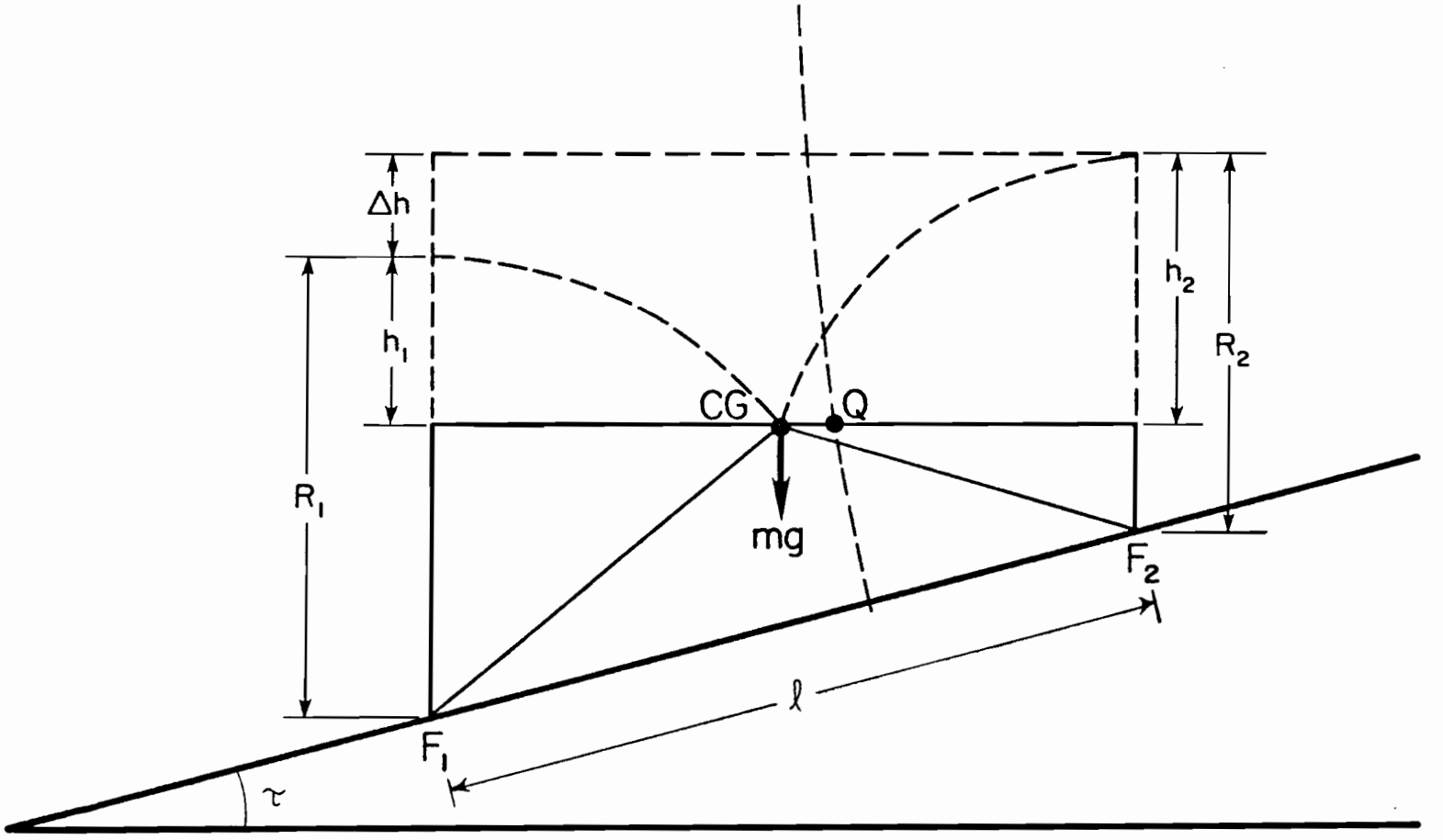


Figure 5.12 Side view of the configuration in Figure 5.11, showing a geometrical comparison of the Energy Stability Level for the front and rear edges of the support boundary.

stability is mgh_2. Since h_2 equals $h_1 + \Delta h$, the situation depicted in Figure 5.12 would require less energy to overturn the vehicle about the rear support legs as opposed to the front support legs. Therefore, if it were desired to shift the body to a position of greater overall stability, the body should be shifted such that h_1 equals h_2, at which point the Energy Stability Levels for the front and rear edge would be equal.

The configuration shown in Figure 5.12 represents a relatively simple case in which the location of the body center of gravity, such that h_1 equals h_2 can be solved geometrically. In general, the locus of all points whose distances from two fixed points differs by a constant describes a hyperbola. For the present configuration, h_1 will be equal to h_2 for any point which satisfies the equation

$$R_1 - R_2 = \ell \sin\tau \quad . \tag{5.1}$$

The locus of points which satisfy this equation is described by the hyperbola which has focal points at F_1 and F_2. This hyperbola is symmetric about the line F_1F_2; the relevant portion for this situation is indicated by the dashed curve in Figure 5.12. The location where this hyperbola intersects the line F_1F_2 is at a distance $\frac{1}{2}\ell(1 + \sin\tau)$ with respect to F_1.

Although h_1 equals h_2 when the body center of gravity is located anywhere along this hyperbola, notice that the magnitude of the Energy Stability Level increases as the vehicle body height is decreased. Since body height directly affects the obstacle clearance capability of the vehicle, it is generally not desirable to move the body closer to the ground and sacrifice obstacle clearance simply for the sake of attaining a greater Energy Stability Margin. Therefore, when choosing the location to which the body center of gravity should be shifted so

that h_1 equals h_2, the vehicle body is restricted to move only within the present plane of the body. With this restriction, the point O in Figure 5.12 represents the desired position of the body center of gravity so that h_1 equals h_2.

It should be noted that the previous discussion was concerned only with the front and rear edges of the support boundary. However, the definition of the Energy Stability Margin requires the consideration of all edges of the support boundary. Also, in more general situations, the position of the body and legs, especially in the case of very rough terrain, may not permit a simple geometric solution. Therefore, it is desirable to derive a general equation which yields a measure of the Energy Stability Level about any given edge of the support boundary.

5.4.2.2.1 Derivation of the Equation for the Energy Stability Level

The Energy Stability Level associated with a particular edge of the support boundary is a measure of the potential energy required to rotate the body about that edge. The potential energy is given by

$$PE = mgh \tag{5.2}$$

where the mass m and acceleration of gravity g are constant. Then for the purpose of finding the Energy Stability Level, it is necessary to find the vertical height h through which the body center of gravity would move if the body were rotated about the given edge of the support boundary to the point of zero static stability margin.

Consider the general situation depicted in Figure 5.13, where points F_1 and F_2 represent the footholds of two support feet and the line segment connecting F_1 and F_2 represents one edge of the support boundary. Plane 1 is a vertical plane containing the line segment F_1F_2. The point CG represents the location of the body center of gravity. The

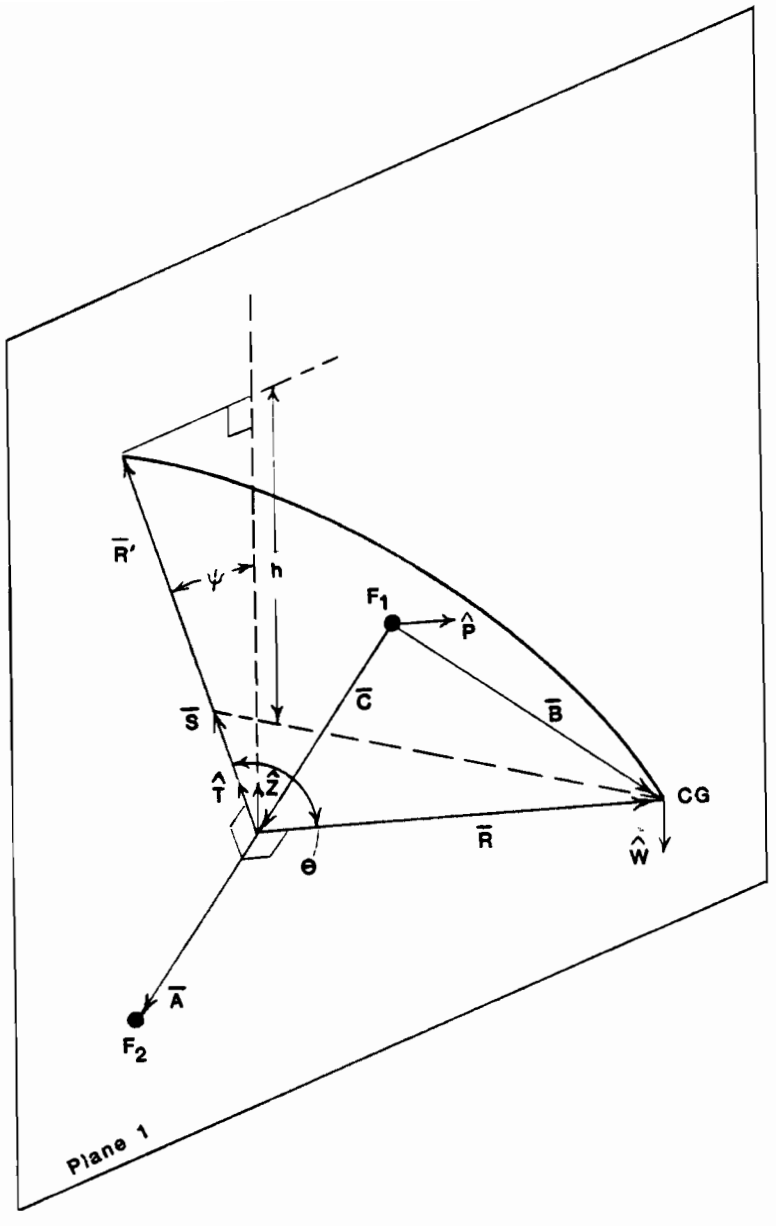


Figure 5.13 A general configuration, used for the derivation of the Energy Stability Level equation. The line F_1F_2 represents one edge of a support boundary. The point CG represents the body center of gravity. The vertical distance h gives a measure of the Energy Stability Level.

vector $\bar{W}$ is the weight vector of the vehicle; it is directed vertically through the point CG.

Since the points F_1 and F_2 represent the locations of two of the vehicle's support feet, the coordinates of these points are known in the body coordinate system, with the origin located at the point CG. In order to simplify the calculations, the coordinates of these points are transformed to an earth-referenced coordinate system. This coordinate system has the origin at the point CG, with the z coordinate axis in the upward vertical direction and the x and y coordinate axes in the horizontal plane.

Let $\bar{A}$ denote the vector from point F_1 to point F_2, and let $\bar{B}$ denote the vector from point F_1 to point CG. Let $\bar{C}$ denote the vector which represents the orthogonal projection of $\bar{B}$ onto the direction of vector $\bar{A}$. Using the property of the dot product

$$\bar{C} = (\hat{A} \cdot \bar{B})\hat{A} \tag{5.3}$$

where $\hat{A}$ is the unit vector in the direction of vector $\bar{A}$,

$$\hat{A} = \frac{\bar{A}}{|\bar{A}|} \quad . \tag{5.4}$$

Vector $\bar{R}$ is a vector from line F_1F_2 to the point CG, and is orthogonal to line F_1F_2. As can be seen in Figure 5.13,

$$\bar{R} = \bar{B} - \bar{C} \quad . \tag{5.5}$$

Let $\bar{P}$ represent a vector that is orthogonal to plane 1, and whose direction points toward the inside region of the support polygon. Using the property of the cross product:

$$\bar{P} = \hat{A} \times \hat{W} \quad . \tag{5.6}$$

By definition,

$$\hat{W} = -\hat{Z} \tag{5.7}$$

then, $$\bar{P} = -\hat{A} \times \hat{Z} \quad . \tag{5.8}$$

The direction of the unit vector $\hat{T}$, shown in Figure 5.13, is obtained by rotating vector $\bar{R}$, about the line F_1F_2, until it lies in plane 1. Then, $\hat{T}$ is a vector that is orthogonal to the line F_1F_2, lying in plane 1, and pointing in the upward direction. Consequently,

$$\hat{T} = \hat{A} \times \hat{P} \tag{5.9}$$

where $\hat{P}$ is the unit vector in the direction of vector $\bar{P}$,

$$\hat{P} = \frac{\bar{P}}{|\bar{P}|} \quad . \tag{5.10}$$

Let $\bar{S}$ denote the orthogonal projection of vector $\bar{R}$ onto the direction of vector $\hat{T}$, then

$$\bar{S} = (\hat{T} \cdot \bar{R})\hat{T} \quad . \tag{5.11}$$

If the vector $\bar{T}$ represents the vector $\bar{R}$ when it has been rotated about the line F_1F_2 until it lies in plane 1, then

$$\bar{T} = |\bar{R}|\hat{T} \tag{5.12}$$

The difference between these two vectors is:

$$\bar{T} - \bar{S} = |\bar{R}|\hat{T} - (\hat{T}\cdot\bar{R})\hat{T} \tag{5.13}$$

$$\bar{T} - \bar{S} = [|\bar{R}| - (\hat{T}\cdot\bar{R})]\hat{T} \tag{5.14}$$

Then, the vertical height h, through which the point CG moves when the vector $\bar{R}$ is rotated to the vertical plane, is equal to the orthogonal projection of the vector $(\bar{T} - \bar{S})$ onto the z-axis:

$$h = (\bar{T}\cdot\bar{S})\cdot\hat{Z} \tag{5.15}$$

$$h = \{[|\bar{R}| - \hat{T}\cdot\bar{R}]\hat{T}\}\cdot\hat{Z} \tag{5.16}$$

$$h = [|\bar{R}| - \hat{T}\cdot\bar{R}](\hat{T}\cdot\hat{Z}) \tag{5.17}$$

or,

$$h = [|\bar{R}| - |\bar{R}|\ \hat{R}\cdot\hat{T}](\hat{T}\cdot\hat{Z}) \tag{5.18}$$

$$h = |\bar{R}|\ [1 - \hat{R}\cdot\hat{T}](\hat{T}\cdot\hat{Z}) \tag{5.19}$$

This equation can also be written in the form:

$$h = |\bar{R}| \ (1 - \cos\theta) \cos\psi \qquad (5.20)$$

where, as shown in Figure 5.13,

θ is the angle between $\bar{R}$ and $\hat{T}$,

and ψ is the angle between $\hat{T}$ and $\hat{Z}$.

Therefore, the general equation for the Energy Stability Level associated with a particular edge of the support boundary is given by:

$$E = m\ g\ |\bar{R}|\ [1 - \hat{R}\cdot\hat{T}](\hat{T}\cdot\hat{Z}) \quad . \qquad (5.21)$$

Notice that all of the vectors required in this equation can be determined solely as a function of the known locations of the two footholds F_1 and F_2. During operation of the walking vehicle, the location of all the feet can be found, with respect to the body center of gravity, and the vector formulation provides a simple, efficient method of calculating the Energy Stability Margin for any position of the body or legs. Most importantly, this equation is a general formulation which is applicable for any type of terrain conditions.

5.4.2.3 Level Energy Curves

Having developed a general equation which allows the calculation of the Energy Stability Margin, it is possible to analyze the Energy Stability Margin for various configurations of body and leg positions. Consider the Energy Stability Margin as a function of the position of the center of gravity, projected into the present plane of the vehicle body. Then one can draw Level Energy Curves which are the locus of all points in the present plane of the body to which the body center of gravity could be moved while maintaining a constant value of Energy Stability Margin. For any configuration of the vehicle, a family of Level Energy Curves can be drawn where each curve represents a different

level of Energy Stability Margin. These Level Energy Curves provide significant information concerning the characteristics of the Energy Stability Margin. Analysis of these characteristics leads to a method for determining an Optimally Stable Position, as will be discussed in Section 5.4.2.4.

One method for drawing the Level Energy Curves for a given configuration is to utilize the gradient of the Energy Stability Margin.

In general, a two-dimensional coordinate system (a,b) can be defined in order to describe the location of the body center of gravity within a given plane. At any point in time, the location of the body center of gravity in the ab plane can be expressed as

$$\overline{CG} = \overline{CG}_0 + a\hat{a} + b\hat{b} \tag{5.22}$$

where $\overline{CG}_0$ is the initial location of the body center of gravity. Using this expression, the equation for the Energy Stability Level can be expressed as a function of the variables a and b. As the body center of gravity is moved within the ab plane, the variation of the Energy Stability Level is described by the gradient

$$\bar{\nabla}E = \begin{bmatrix} \frac{\partial E}{\partial a} \\ \frac{\partial E}{\partial b} \end{bmatrix} . \tag{5.23}$$

As discussed in Section 5.4.2.2, if the vehicle center of gravity is to be shifted to a position of greater Energy Stability Margin, the vehicle body should be restricted to move only within the present plane of the vehicle body. This restriction guards against sacrificing obstacle clearance in order to increase the Energy Stability Margin. For this reason, the Level Energy Curves are defined to lie in the

present plane of the vehicle body. Therefore, the two-dimensional coordinate system (a,b) is defined to coincide with the (X_B, Y_B) coordinates of the body coordinate system, having its origin fixed at the center of the vehicle body.

5.4.2.3.1 Determining the Gradient

In order to graph the Level Energy Curves, it would be desirable to express the Energy Stability Level as a function of the variables a and b. Then for any given configuration of the vehicle body and legs, the gradient function can be used to determine the variation of the Energy Stability Level as the body is shifted in the ab plane.

Recall from the derivation of the Energy Stability Level equation that the coordinates of all points must be transformed to an earth-referenced coordinate system. This coordinate system, with the z coordinate axis in the upward vertical direction and the x and y coordinate axes in the horizontal plane, has the origin at the point CG_0, which is the initial location of the body center of gravity. Any point in the ab coordinate system can be transformed to the earth-referenced coordinate system by using the following rotation transformation [47]:

$$\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\theta \sin\phi & \cos\theta \sin\phi \\ 0 & \cos\theta & -\sin\theta \\ -\sin\phi & \sin\theta \cos\phi & \cos\theta \cos\phi \end{bmatrix} \begin{bmatrix} a \\ b \\ 0 \end{bmatrix} \qquad (5.24)$$

where ϕ = the body pitch angle

and θ = the body roll angle.

Equation (5.22) describes the location of the body center of gravity as it is shifted within the ab plane:

$$\overline{CG} = \overline{CG_0} + a\hat{a} + b\hat{b} \quad . \qquad (5.22)$$

This equation can then be expressed in terms of the earth-referenced coordinate system as:

$$\overline{CG} = \overline{CG_0} + e_x \hat{X} + e_y \hat{Y} + e_z \hat{Z} \qquad (5.25)$$

where $e_x = a\cos\phi + b \sin\theta \sin\phi$

$e_y = b\cos\theta$

$e_z = -a \sin\phi + b \sin\theta \cos\phi$.

The points F_1 and F_2 in Figure 5.13 represent the locations of two of the vehicle's support feet. The coordinates of these points are known in the body coordinate system. These coordinates can be transformed to the earth-referenced coordinate system by using the same rotation transformation matrix given in equation (5.24).

Referring to Figure 5.13, the vector $\bar{A}$ from point F_1 to point F_2 may be written as

$$\bar{A} = \bar{F}_2 - \bar{F}_1 \qquad (5.26)$$

where $\bar{F}_1$ and $\bar{F}_2$ represent vectors in the earth-referenced coordinate system, from the origin to the points F_1 and F_2, respectively. If the origin of the earth-referenced coordinate system is fixed at the initial location of the body center of gravity, CG_0, then $\bar{F}_1$ and $\bar{F}_2$ are independent of variations of the body center of gravity in the ab plane. Then the vector $\bar{A}$ is independent of the variables a and b.

The vector $\bar{B}$ from point F_1 to the point CG may be written as:

$$\bar{B} = \overline{CG} - \bar{F}_1 \quad . \qquad (5.27)$$

Substituting equation (5.25) for $\overline{CG}$:

$$\bar{B} = (\overline{CG_0} + e_x \hat{X} + e_y \hat{Y} + e_z \hat{Z}) - \bar{F}_1 \quad . \qquad (5.28)$$

The vector $\bar{C}$, which represents the orthogonal projection of $\bar{B}$ onto the direction of vector $\bar{A}$ is given by equation (5.3) and (5.4) as:

$$\bar{C} = (\hat{A} \cdot \bar{B})\hat{A} \tag{5.3}$$

where $$\hat{A} = \frac{\bar{A}}{|\bar{A}|} \quad . \tag{5.4}$$

Substituting equation (5.28) for $\bar{B}$:

$$\bar{C} = [\hat{A}\cdot(\overline{CG_0} + e_x \hat{X} + e_y \hat{Y} + e_z \hat{Z} - \bar{F}_1)]\hat{A} \tag{5.29}$$

or $$\bar{C} = [\hat{A}\cdot(\overline{CG_0} - \bar{F}_1) + e_x\hat{A}\cdot\hat{X} + e_y\hat{A}\cdot\hat{Y} + e_z\hat{A}\cdot\hat{Z}]\hat{A} \quad . \tag{5.30}$$

The vector $\bar{R}$, given by equation (5.5), was expressed as

$$\bar{R} = \bar{B} - \bar{C} \quad . \tag{5.5}$$

Substituting equations (5.27) and (5.30) for $\bar{B}$ and $\bar{C}$, respectively:

$$\bar{R} = [(\overline{CG_0} - \bar{F}_1) + e_x\hat{X} + e_y\hat{Y} + e_z\hat{Z}] -$$
$$[\hat{A}\cdot(\overline{CG_0} - F_1) + e_x\hat{A}\cdot\hat{X} + e_y\hat{A}\cdot\hat{Y} + e_z\hat{A}\cdot\hat{Z}]\hat{A} \quad . \tag{5.31}$$

Since $\overline{CG_0}$, $\bar{F}_1$, and $\hat{A}$ are constants, independent of variations in the coordinates a and b,

let $$\bar{D} = \overline{CG_0} - \bar{F}_1 \tag{5.32}$$

$$k = \hat{A}\cdot(\overline{CG_0} - \bar{F}_1) \tag{5.33}$$

and $$A_x = \hat{A}\cdot\hat{X}$$

$$A_y = \hat{A}\cdot\hat{Y}$$

$$A_z = \hat{A}\cdot\hat{Z} \quad . \tag{5.34}$$

Notice that equations (5.34) represents the components of $\hat{A}$ in the x, y, and z directions of the earth-referenced coordinate system. Then equation (5.31) may be rewritten as

$$\bar{R} = \bar{D} + e_X\hat{X} + e_Y\hat{Y} + e_Z\hat{Z} -$$

$$[k + e_X A_X + e_Y A_Y + e_Z A_Z]\hat{A} \quad . \tag{5.35}$$

The vector $\bar{P}$ is given by equation (5.8) as

$$\bar{P} = -\hat{A} \times \hat{Z} \quad . \tag{5.8}$$

Since both $\hat{A}$ and $\hat{Z}$ are independent of variations in (a,b), the vector $\bar{P}$ is a constant.

The unit vector $\hat{T}$ is given by equation (5.9) as

$$\hat{T} = \hat{A} \times \hat{P} \quad . \tag{5.9}$$

Again, $\hat{A}$ and $\hat{P}$ are constant, so $\hat{T}$ is constant.

The Energy Stability Level can be written, using equations (5.17) and (5.21) as

$$E = m\ g\ [|\bar{R}| - \hat{T}\cdot\bar{R}](\hat{T}\cdot\hat{Z}) \quad . \tag{5.36}$$

Since the vectors $\hat{T}$ and $\hat{Z}$ are independent of variations in the coordinates a and b,

let $$T_Z = \hat{T}\cdot\hat{Z} \quad . \tag{5.37}$$

This represents the component of $\hat{T}$ in the z-direction.

Then, $$E = m\ g\ T_Z\ [|\bar{R}| - \hat{T}\cdot\bar{R}] \quad . \tag{5.38}$$

In equation (5.38), the vector $\bar{R}$ is the only component which is a function of the variables a and b. Therefore, as the body center of gravity is shifted within the ab plane, the variation of the Energy Stability Level is described by the gradient as:

$$\bar{\nabla}E = m\ g\ T_Z\ [\bar{\nabla}|\bar{R}| - \bar{\nabla}(\hat{T}\cdot\bar{R})] \quad . \tag{5.39}$$

This may also be written as:

$$\frac{\partial E}{\partial a} = m\ g\ T_Z\ [\frac{\partial}{\partial a}\ |\bar{R}|\ -\ \frac{\partial}{\partial a}(\hat{T}\cdot\bar{R})] \tag{5.40}$$

$$\frac{\partial E}{\partial b} = m\ g\ T_Z\ [\frac{\partial}{\partial b}\ |\bar{R}|\ -\ \frac{\partial}{\partial b}(\hat{T}\cdot\bar{R})] \quad . \tag{5.41}$$

Using equation (5.35),

$$(\hat{T}\cdot\bar{R}) = \hat{T}\cdot\bar{D} + \hat{T}\cdot(e_X\hat{X} + e_Y\hat{Y} + e_Z\hat{Z}) -$$

$$[k + e_XA_X + e_YA_Y + e_ZA_Z]\hat{T}\cdot\hat{A} \quad . \tag{5.42}$$

Since $\hat{T}$, $\bar{D}$, and $\hat{A}$ are constants, independent of variations in the coordinates a and b,

let $$\ell = \hat{T}\cdot\bar{D} - \hat{T}\cdot(\overline{CG}_0-\bar{F}_1) \tag{5.43}$$

$$m = \hat{T}\cdot\hat{A} \tag{5.44}$$

and let $$T_X = \hat{T}\cdot\hat{X}$$

$$T_Y = \hat{T}\cdot\hat{Y}$$

$$T_Z = \hat{T}\cdot\hat{Z} \quad . \tag{5.45}$$

Then equation (5.42) may be rewritten as

$$(\hat{T}\cdot\bar{R}) = \ell + e_XT_X + e_YT_Y + e_ZT_Z -$$

$$[k + e_XA_X + e_YA_Y + e_ZA_Z]m \quad . \tag{5.46}$$

Or, $$(\hat{T}\cdot\bar{R}) = (\ell-mk) + e_X(T_X-mA_X) +$$

$$e_Y(T_Y-mA_Y) + e_Z(T_Z-mA_Z) \quad . \tag{5.47}$$

Substituting for e_X, e_Y, and e_Z in terms of the variables a and b:

$$\hat{T}\cdot\bar{R} = (\ell-mk) + (a\cos\phi + b\sin\theta\sin\phi)(T_X-mA_X)$$

$$+ (\qquad b\cos\theta \qquad)(T_Y-mA_Y)$$

$$+ (-a\sin\phi + b\sin\theta\cos\phi)(T_Z-mA_Z) \tag{5.48}$$

Then $\bar{\nabla}(\hat{T}.\bar{R})$ is given by:

$$\frac{\partial}{\partial a}(\hat{T}\cdot\bar{R}) = \cos\phi(T_X - mA_X) - \sin\phi(T_Z - mA_Z) \tag{5.49}$$

$$\frac{\partial}{\partial b}(\hat{T}\cdot\bar{R}) = \sin\theta\,\sin\phi(T_X - mA_X) + \cos\theta(T_Y - mA_Y) + \sin\theta\,\cos\phi(T_Z - mA_Z) \quad . \tag{5.50}$$

In order to determine $|\bar{R}|$, use equation (5.35):

$$\bar{R} = \bar{D} + e_X\hat{X} + e_Y\hat{Y} + e_Z\hat{Z} - [k + e_XA_X + e_YA_Y + e_ZA_Z]\hat{A} \quad . \tag{5.35}$$

Let $$\bar{D} = D_X\hat{X} + D_Y\hat{Y} + D_Z\hat{Z} \tag{5.51}$$

and $$\hat{A} = A_X\hat{X} + A_Y\hat{Y} + A_Z\hat{Z} \quad . \tag{5.52}$$

Then equation (5.35) may be written:

$$\bar{R} = (D_X\hat{X} + D_Y\hat{Y} + D_Z\hat{Z}) + e_X\hat{X} + e_Y\hat{Y} + e_Z\hat{Z} - [k + e_XA_X + e_YA_Y + e_ZA_Z](A_X\hat{X} + A_Y\hat{Y} + A_Z\hat{Z}) \quad . \tag{5.53}$$

Or,
$$\bar{R} = (D_X + e_X - A_X[k + e_XA_X + e_YA_Y + e_ZA_Z])\hat{X} + (D_Y + e_Y - A_Y[k + e_XA_X + e_YA_Y + e_ZA_Z])\hat{Y} + (D_Z + e_Z - A_Z[k + e_XA_X + e_YA_Y + e_ZA_Z])\hat{Z} \quad . \tag{5.54}$$

Notice that equation (5.54) is of the form:

$$\bar{R} = R_X\hat{X} + R_Y\hat{Y} + R_Z\hat{Z} \tag{5.55}$$

where R_X, R_Y, and R_Z are components of $\bar{R}$ in the x, y, and z directions. The magnitude of the vector $\bar{R}$ may be written as:

$$|\bar{R}| = (R_X^2 + R_Y^2 + R_Z^2)^{1/2} \quad . \tag{5.56}$$

Then $\bar{\nabla}|\bar{R}|$ is given by:

$$\frac{\partial}{\partial a}|\bar{R}| = \frac{R_x \frac{\partial R_x}{\partial a} + R_y \frac{\partial R_y}{\partial a} + R_z \frac{\partial R_z}{\partial a}}{|\bar{R}|} \tag{5.57}$$

$$\frac{\partial}{\partial b}|\bar{R}| = \frac{R_x \frac{\partial R_x}{\partial b} + R_y \frac{\partial R_y}{\partial b} + R_z \frac{\partial R_z}{\partial b}}{|\bar{R}|} \quad . \tag{5.58}$$

Each of the partial derivative terms in equation (5.57) and (5.58) can be found using equation (5.54) and using the equations for e_x, e_y, and e_z in terms of the variables a and b as follows:

(1) $$\frac{\partial R_x}{\partial a} = \frac{\partial}{\partial a}\{D_x + e_x - A_x [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_x}{\partial a} = 0 + \frac{\partial e_x}{\partial a} - A_x \left[\underline{0 + A_x \frac{\partial e_x}{\partial a} + A_y \frac{\partial e_y}{\partial a} + A_z \frac{\partial e_z}{\partial a}} \right]$$

$$\frac{\partial R_x}{\partial a} = \cos\phi - A_x [A_x \cos\phi + A_y (0) + A_z (-\sin\phi)]$$

$$\frac{\partial R_x}{\partial a} = \cos\phi - A_x^2 \cos\phi + A_x A_z \sin\phi \tag{5.59}$$

(2) $$\frac{\partial R_x}{\partial b} = \frac{\partial}{\partial b}\{D_x + e_x - A_x [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_x}{\partial b} = 0 + \frac{\partial e_x}{\partial b} - A_x \left[\underline{0 + A_x \frac{\partial e_x}{\partial b} + A_y \frac{\partial e_y}{\partial b} + A_z \frac{\partial e_z}{\partial b}} \right]$$

$$\frac{\partial R_x}{\partial b} = \sin\theta \sin\phi - A_x [A_x \sin\theta \sin\phi + A_y \cos\theta + A_z \sin\theta \cos\phi]$$

$$\frac{\partial R_x}{\partial b} = \sin\theta \sin\phi - A_x^2 \sin\theta \sin\phi - A_x A_y \cos\theta - A_x A_z \sin\theta \cos\phi \tag{5.60}$$

(3) $$\frac{\partial R_y}{\partial a} = \frac{\partial}{\partial a}\{D_y + e_y - A_y [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_y}{\partial a} = 0 + \frac{\partial e_y}{\partial a} - A_y \left| \underline{0} + A_x \frac{\partial e_x}{\partial a} + A_y \frac{\partial e_y}{\partial a} + A_z \frac{\partial e_z}{\partial a} \right|$$

$$\frac{\partial R_y}{\partial a} = 0 - A_y[A_x \cos\phi + A_y (0) + A_z (-\sin\phi)]$$

$$\frac{\partial R_y}{\partial a} = -A_x A_y \cos\phi + A_y A_z \sin\phi \qquad (5.61)$$

(4) $$\frac{\partial R_y}{\partial b} = \frac{\partial}{\partial b}\{D_y + e_y - A_y [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_y}{\partial b} = 0 + \frac{\partial e_y}{\partial b} - A_y \left| \underline{0} + A_x \frac{\partial e_x}{\partial b} + A_y \frac{\partial e_y}{\partial b} + A_z \frac{\partial e_z}{\partial b} \right|$$

$$\frac{\partial R_y}{\partial b} = \cos\theta - A_y[A_x \sin\theta \sin\phi + A_y \cos\theta + A_z \sin\theta \cos\phi]$$

$$\frac{\partial R_y}{\partial b} = \cos\theta - A_x A_y \sin\theta \sin\phi - A_y^2 \cos\theta - A_y A_z \sin\theta \cos\phi \qquad (5.62)$$

(5) $$\frac{\partial R_z}{\partial a} = \frac{\partial}{\partial a}\{D_z + e_z - A_z [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_z}{\partial a} = 0 + \frac{\partial e_z}{\partial a} - A_z \left| \underline{0} + A_x \frac{\partial e_x}{\partial a} + A_y \frac{\partial e_y}{\partial a} + A_z \frac{\partial e_z}{\partial a} \right|$$

$$\frac{\partial R_z}{\partial a} = -\sin\phi - A_z[A_x \cos\phi + A_y (0) + A_z (-\sin\phi)]$$

$$\frac{\partial R_z}{\partial a} = -\sin\phi - A_x A_z \cos\phi + A_z^2 \sin\phi \qquad (5.63)$$

(6) $$\frac{\partial R_z}{\partial b} = \frac{\partial}{\partial b}\{D_z + e_z - A_z [k + e_x A_x + e_y A_y + e_z A_z]\}$$

$$\frac{\partial R_z}{\partial b} = 0 + \frac{\partial e_z}{\partial b} - A_z\left[\overline{0 + A_x \frac{\partial e_x}{\partial b} + A_y \frac{\partial e_y}{\partial b} + A_z \frac{\partial e_z}{\partial b}}\right]$$

$$\frac{\partial R_z}{\partial b} = \sin\theta \cos\phi - A_z[A_x \sin\theta \sin\phi + A_y \cos\theta + A_z \sin\theta \cos\phi]$$

$$\frac{\partial R_z}{\partial b} = \sin\theta \cos\phi - A_x A_z \sin\theta \sin\phi - A_y A_z \cos\theta - A_z^2 \sin\theta \cos\phi \quad (5.64)$$

Therefore, the gradient of the Energy Stability Level is found by substituting equations (5.49) and (5.50) together with (5.57) and (5.58) into equations (5.40) and (5.41) and using equations (5.59) through (5.64). The result is

$$\frac{\partial E}{\partial a} = m\, g\, T_z \left[\overline{\frac{R_x \frac{\partial R_x}{\partial a} + R_y \frac{\partial R_y}{\partial a} + R_z \frac{\partial R_z}{\partial a}}{|\bar{R}|} - \{\cos\phi(Tx - mA_x) - \sin\phi(Tz - mA_z)\}}\right] \quad (5.65)$$

$$\frac{\partial E}{\partial b} = m\, g\, T_z \left[\overline{\frac{R_x \frac{\partial R_x}{\partial b} + R_y \frac{\partial R_y}{\partial b} + R_z \frac{\partial R_z}{\partial b}}{|\bar{R}|} - \{\sin\theta \sin\phi\,(Tx - mA_x) + \cos\theta\,(Ty - mA_y) + \sin\theta \cos\phi\,(Tz - mA_z)\}}\right] \quad (5.66)$$

where $R_x = D_x + e_x - A_x [k + e_xA_x + e_yA_y + e_zA_z]$

$R_y = D_y + e_y - A_y [k + e_xA_x + e_yA_y + e_zA_z]$

$Rz = D_z + e_z - A_z [k + e_xA_x + e_yA_y + e_zA_z]$

and $$\frac{\partial R_x}{\partial a} = \cos\phi - A_x^2 \cos\phi + A_xA_z \sin\phi$$

$$\frac{\partial R_y}{\partial a} = -A_xA_y \cos\phi + A_yA_z \sin\phi$$

$$\frac{\partial R_z}{\partial a} = -\sin\phi - A_xA_z \cos\phi + A_z^2 \sin\phi$$

$$\frac{\partial R_x}{\partial b} = \sin\theta \sin\phi - A_x^2 \sin\theta \sin\phi - A_xA_y \cos\theta - A_xA_z \sin\theta \cos\phi$$

$$\frac{\partial R_y}{\partial b} = \cos\theta - A_xA_y \sin\theta \sin\phi - A_y^2 \cos\theta - A_yA_z \sin\theta \cos\phi$$

$$\frac{\partial R_z}{\partial b} = \sin\theta \cos\phi - A_xA_z \sin\theta \sin\phi - A_yA_z \cos\theta - A_z^2 \sin\theta \cos\phi \quad .$$

5.4.2.3.2 Plots of Level Energy Curves

As a means of analyzing the characteristics of the Energy Stability Margin, it is desirable to plot a family of Level Energy Curves for any given configuration of the vehicle body and legs. The gradient of the Energy Stability Level provides significant information regarding the variation of the Energy Stability Level as the body center of gravity is moved within the ab plane. By utilizing this information provided by the gradient, it is possible to plot the Level Energy Curves. The following iterative equation is used to calculate the location of those points which lie on the same Level Energy Curve:

$$P = P_o + \begin{vmatrix} \bar{0} & -\bar{\alpha} \\ \bar{\alpha} & \bar{0} \end{vmatrix} \frac{\bar{\nabla}E(P_o)}{|E(P_o)|} + \frac{A - E(P_o)}{|\bar{\nabla}E(P_o)|^2} \bar{\nabla}E(P_o) \qquad (5.67)$$

where P = the (a,b) coordinates of a point on the Level Energy Curve,

P_0 = the (a,b) coordinates of the current point on the Level Energy Curve,

A = the constant value of the Energy Stability Level on this particular Level Energy Curve,

$E(P_0)$ = the value of the Energy Stability Level at the point P_0,

$\bar{\nabla}E(P_0)$ = the gradient of the Energy Stability Level, evaluated at the point P_0,

α = a scalar constant representing the step-length.

Equation 5.58 is an iterative equation which, for each iteration, makes use of a current point P_0 to calculate a new point P. These points approximate the curve with Energy Stability Margin equal to A.

The second term in equation (5.67) provides a discrete step, of length α, in the direction orthogonal to the gradient of the Level Energy Curve at the point P_0. Recall that the gradient is always normal to the curve. Therefore, moving in a direction orthogonal to the gradient gives an approximation of a neighboring point on the same curve, provided that the step-length, α, is sufficiently small.

The third term in equation (5.67) provides error-correction to allow more accurate tracking of the Level Energy Curve. This term is derived by considering the Taylor series expansion of a function of several variables [48]:

$$f(x) = f(x') + \bar{\nabla}f(x')^T \Delta x + \frac{1}{2}\Delta x^T \bar{\nabla}^2 f(x')\Delta x + \ldots \qquad (5.68)$$

where x' = the current, or expansion, point,

$\Delta x = x - x'$ = the change in the value of x,

$\bar{\nabla}f(x')$ = the gradient of f, evaluated at x',

$\bar{\nabla}^2 f(x')$ = the Hessian matrix of f, evaluated at x'.

If Δx is relatively small, then the higher order terms can be considered neglible, consequently,

$$f(x) = f(x') + \bar{\nabla} f(x')^T \Delta x \quad . \tag{5.69}$$

Consider that the next value of x is to be determined from the value of x' by an equation of the form:

$$x = x' + \delta\, s(x') \tag{5.70}$$

where δ = a scalar step-length

$s(x')$ = the search direction.

Then $$\Delta x = x - x' = \delta\, s(x') \quad . \tag{5.71}$$

In equation (5.69), the second term determines the change in f(x), since f(x') is constant. The greatest change in Δx is determined by the direction s(x') in equation (5.71), and occurs when

$$s(x') = \bar{\nabla} f(x') \quad . \tag{5.72}$$

Then $$\Delta x = \delta f(x') \quad . \tag{5.73}$$

Substituting into equation (5.69):

$$f(x) = f(x') + \bar{\nabla} f(x')^T [\delta \bar{\nabla} f(x')] \tag{5.74}$$

$$f(x) = f(x') + \delta\, |\bar{\nabla} f(x')|^2 \quad . \tag{5.75}$$

Solving for δ:

$$\delta = \frac{f(x) - f(x')}{|\bar{\nabla} f(x')|^2} \quad . \tag{5.76}$$

Substituting into equation (5.73):

$$\Delta x = \frac{f(x) - f(x')}{|\bar{\nabla} f(x')|^2} \bar{\nabla} f(x') \quad . \tag{5.67}$$

For the purpose of tracking the Level Energy Curve, f(x) represents the constant value of the Energy Stability Level associated with the particular Level Energy Curve under consideration; therefore f(x) = A.

Replacing $f(x')$ with $E(P_o)$, it is seen that equation (5.77) is the third term in equation (5.67). This term assists in maintaining the point P on the desired Level Energy Curve.

Based upon equation (5.67), a computer program was written which is capable of plotting Level Energy Curves for any given configuration of the vehicle body and legs. Figures 5.14, 5.15, and 5.16 illustrate several families of Level Energy Curves, for various configurations, plotted with this computer program. In these drawings, the X's represent the support feet which form the support boundary. The positions of these X's represent the vertical projection of the support feet onto the plane of the body. The polygon formed by connecting these X's represents the curve of zero Energy Stability Margin. If the body center of gravity is shifted outside of this boundary, the vehicle will be statically unstable. On the other hand, as the body center of gravity is moved inward, each Level Energy Curve represents a higher Energy Stability Margin.

The Level Energy Curves may be thought of as a contour map of the Energy Stability Margin. The coordinates (a,b) give the location of the body center of gravity as it is shifted in the plane of the body. Consider these coordinates (a,b) to be in the plane of the paper. Consider the value of the Energy Stability Margin as being the third coordinate, whose axis comes out of the page; then the Energy Stability Margin may be thought of as a three-dimensional surface. As the body center of gravity is shifted in the plane of the body, the Energy Stability Margin is varying along this surface. The greater the value of the Energy Stability Margin, the greater the vertical distance along this energy surface.

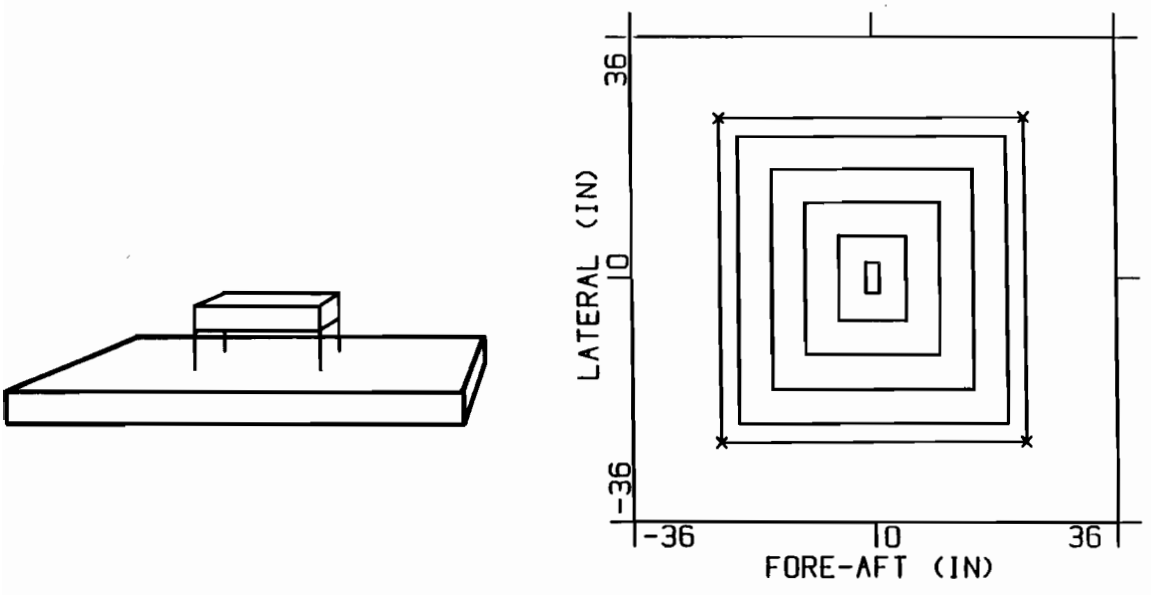


Figure 5.14 Level Energy Curves for the vehicle standing on level terrain, with the body horizontal.

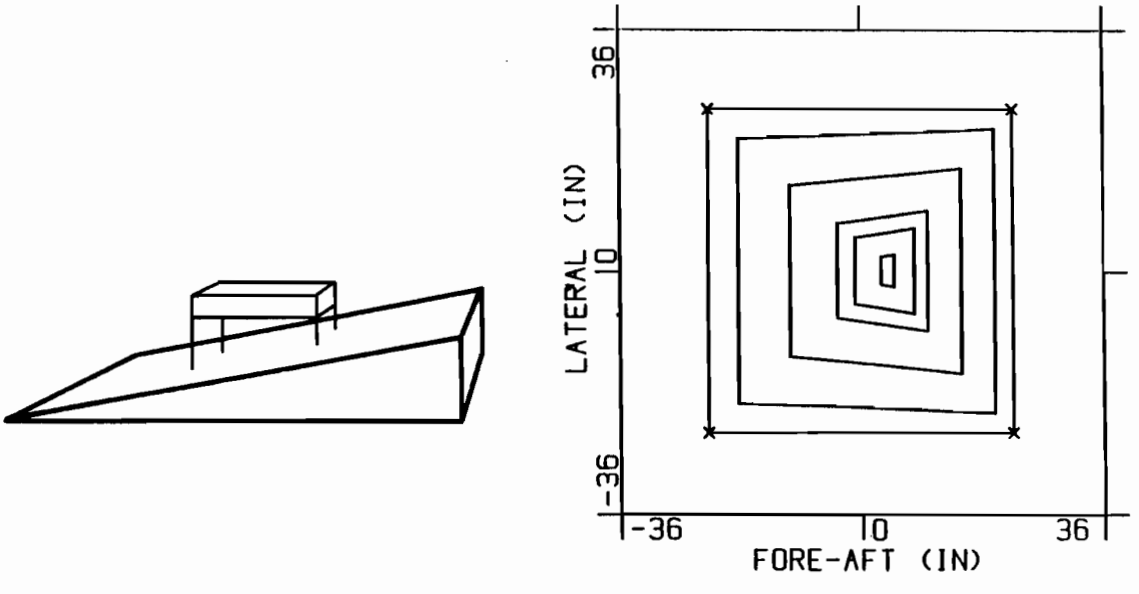


Figure 5.15 Level Energy Curves for the vehicle standing on a 20 degree inclined plane, with the body horizontal.

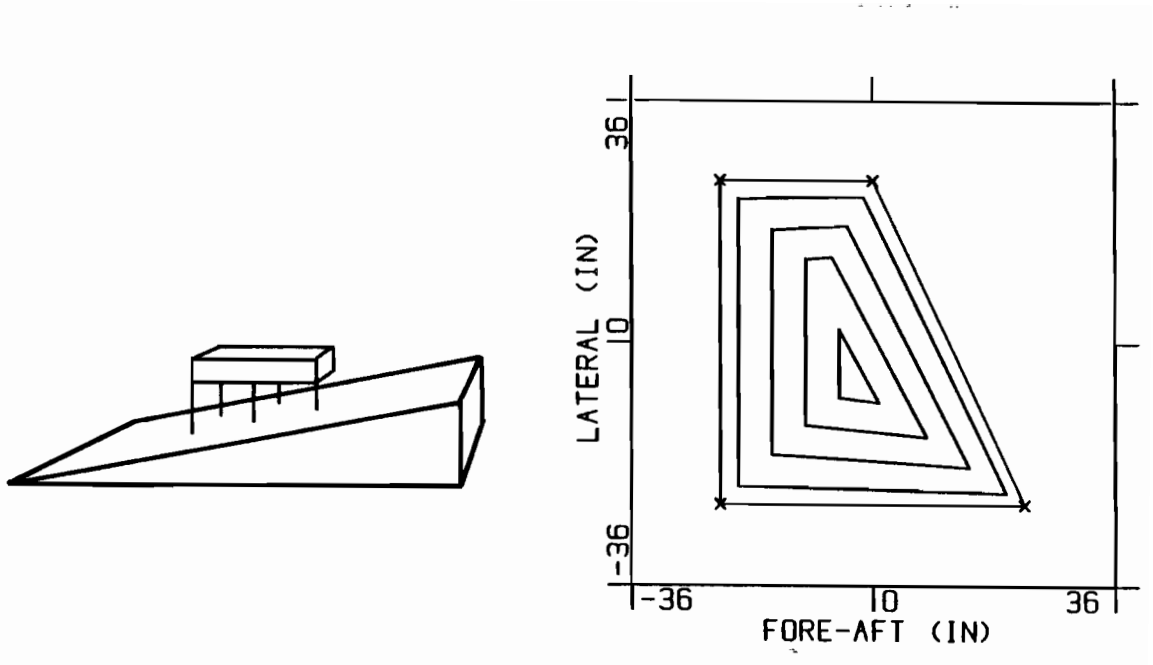


Figure 5.16 Level Energy Curves for the vehicle standing on a 20 degree inclined plane, with the body horizontal and the left front leg off the ground.

5.4.2.4 Optimally Stable Position

To fully apply the concept of Energy Stability Margin to the control of a legged vehicle, it is desirable to locate the position to which the body center of gravity could be moved in order to obtain the maximum Energy Stability Margin for a given configuration. In other words, it is desirable to move to the highest level on the Energy Stability Margin surface. It should be noted from the Level Energy Curves of Figures 5.14, 5.15, and 5.16 that the Energy Stability Margin surface does not necessarily have a unique peak point. The energy surface may contain a ridge at its highest level, which corresponds to having a line as the highest Level Energy Curve, as shown in Figures 5.14 and 5.15. This leads to the following definition.

DEFINITION 6: An Optimally Stable Position is any position in the plane of the body at which the Energy Stability Margin would be maximal if the center of gravity were moved to that position.

Again, notice that in this definition, and in the preceeding discussions, the vehicle movement is constrained to be in the plane of the body. Without this type of restriction, the Optimally Stable Position would require that the body center of gravity be at ground level; or more precisely, for the situation of rough terrain, it would require the body center of gravity to be in the vicinity of the contact points of the support feet. The closer the body center of gravity is to the vertical position of the support feet, the greater the Energy Stability Margin. At the same time, however, lowering the body height adversely affects the vehicle's obstacle clearance capability. On the other hand, constraining the Optimally Stable Position to be in the plane of the body defines a position where the Energy Stability Margin is maximal without directly affecting the body height.

A study of the Level Energy Curves discussed in Section 5.4.2.3 indicates that the three-dimensional energy surface is monotonically increasing, up to the maximal level. Because of this characteristic, it is possible to find the optimal path which the body center of gravity should follow in order to shift from its present location to the Optimally Stable Position. This optimal path can be found by utilizing the gradient of the Energy Stability Margin.

The following iterative equation, based upon the optimal gradient, or steepest descent, method [49], can be used to determine the location of the Optimally Stable Position:

$$P = P_o + k \frac{\bar{\nabla}E(P_o)}{|\bar{\nabla}E(P_o)|} \tag{5.78}$$

where P = the (a,b) coordinates of a point on the optimal path,

P_o = the (a,b) coordinates of the current point on the optimal path,

$\bar{\nabla}E(P_o)$ = the gradient of the Energy Stability Level, evaluated at the point P_o,

k = a scalar constant representing the step length.

Beginning at a given present location of the body center of gravity, equation (5.68) can be used to find the optimal path by following the direction of the Energy Stability Margin gradient. This optimal path traces the steepest slope from the given body center of gravity location to the maximal level of the Energy Stability Margin; this maximal point is an Optimally Stable Position.

Recall from Definition 5 that the Energy Stability Margin is the minimum of all the Energy Stability Levels for a given support boundary. Therefore a Level Energy Curve is the locus of all the points where the minimum Energy Stability Level is equal to some constant value. As can

be seen in Figures 5.14-5.16, the trace of a Level Energy Curve involves sharp changes in direction. These direction changes indicate that the minimum Energy Stability Level has switched from one edge of the support boundary to another edge. These direction changes on the Level Energy Curves correspond to ridges on the three-dimensional energy surface.

The optimal path leading to the Optimally Stable Position traces the maximum increase of the minimum Energy Stability Level. Since the algorithm to trace the optimal path is implemented on a digital computer, the gradient of the Energy Stability Margin is calculated at discrete intervals. Therefore as the optimal path approaches a ridge on the energy surface, there will be sharp direction changes in the optimal path. This occurs because the direction of the optimal path is normal to the Level Energy Curves. The effect is that the optimal path oscillates as it follows the ridge; an example is shown in Figure 5.17. Although this optimal path does lead to the Optimally Stable Position, the oscillation would obviously be undesirable for use in controlling a physical system such as a legged vehicle.

To reduce the occurrence of these oscillations when the optimal path follows a ridge on the energy surface, a "blending function" is introduced. The objective of the blending function is to allow the optimal path to follow the center of the ridge, without oscillating across the ridge. The blending function is only active when the optimal path enters within a narrow band on either side of a ridge. Inside this band, the optimal path traces a trajectory determined by the weighted combination of the gradients of the two smallest Energy Stability Levels. On the other hand, outside the band enveloping the ridge, the optimal path traces the gradient of the minimum Energy Stability Level, as normally occurs.

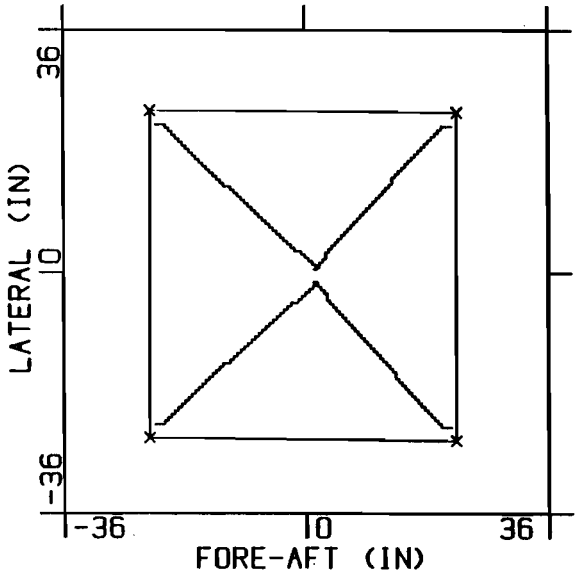


Figure 5.17 Tracing the optimal paths of the vehicle center of gravity, when blending function is not included. Note the oscillations, which indicate sharp direction changes in the optimal path, due to the discrete time calculation of the Energy Stability Margin gradient.

Figure 5.18 depicts the active region of the blending function. The solid line, labeled "E_2-E_1=0," represents a ridge on the energy surface. This ridge consists of the locus of points at which two adjacent edges of the support boundary have equal Energy Stability Levels. The two dashed lines in Figure 5.18 represent the boundaries of the active region. Inside the active region, the two edges of the support boundary have Energy Stability levels such that

$$|E_2 - E_1| \leq \epsilon_{max} \quad . \tag{5.79}$$

Inside the active region, the points along the optimal path are determined by a weighted combination of the gradients of the two Energy Stability Levels according to the equation:

$$P = P_o + [1-\beta]\, k \frac{\bar{\nabla}E_1(Po)}{|\bar{\nabla}E_1(P_o)|} + [\beta]\, k \frac{\bar{\nabla}E_2(Po)}{|\bar{\nabla}E_2(P_o)|} \tag{5.80}$$

where P = the (a,b) coordinates of a point on the optimal path,

P_o = the (a,b) coordinates of the current point on the optimal path,

$\bar{\nabla}E_1(P_o)$ = the gradient of the minimum Energy Stability Level, evaluated at the point P_o.

$\bar{\nabla}E_2(P_o)$ = the gradient of the next smallest Energy Stability Level, evaluated at the point P_o,

k = a scalar constant representing the step length,

β = a scalar constant representing the relative weight of the terms in the blending function.

The weight, β, of the blending function can be determined in various ways. At the center of the active region, where E_2-E_1=0, it is desired to have equal weighting of the two Energy Stability Levels; therefore β=1/2 at this point. For use in the present situation, a linear variation of the weight β was chosen, as shown in Figure 5.19.

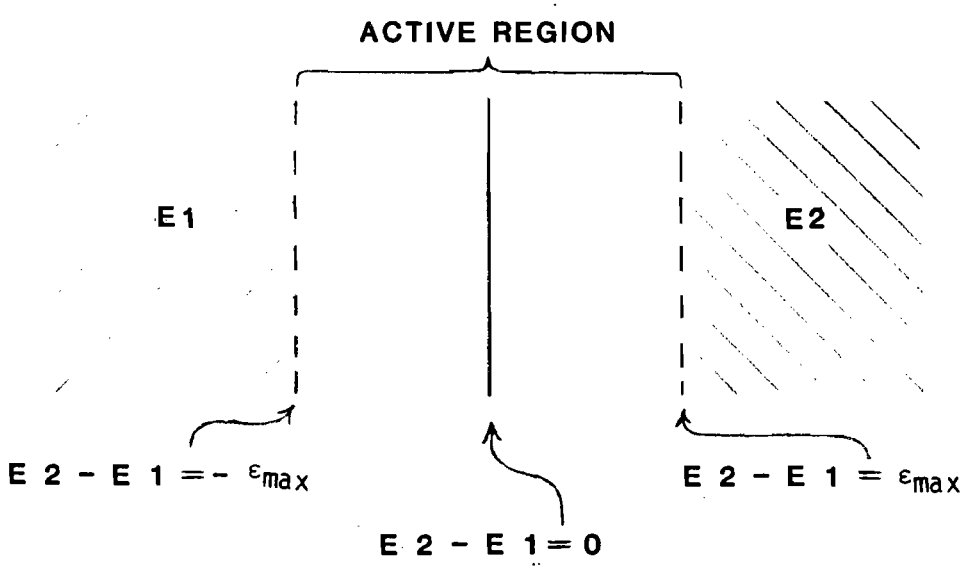


Figure 5.18 The active region of the blending function.

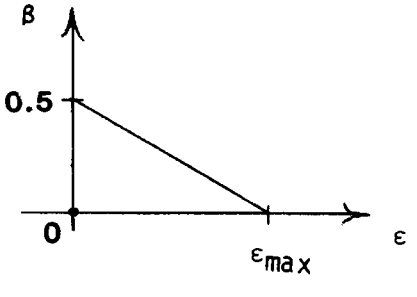


Figure 5.19 Variation of the weight β as a function of ε, where ε = E 2 - E 1.

The weight β is determined by the equation:

$$\beta = \frac{1}{2}\left(1 - \frac{\varepsilon}{\varepsilon_{max}}\right) \tag{5.81}$$

and

$$\varepsilon = E_2 - E_1 \tag{5.82}$$

where E_1 = the minimum Energy Stability Level,

E_2 = the next smallest Energy Stability Level,

ε_{max} = the value of ε at the boundary of the active region.

Figures 5.20-5.25 show the traces of various optimal paths for several vehicle configurations, when the blending function is included. In addition, the Level Energy Curves are shown as dotted lines on these graphs. These curves depict some of the optimal paths for the body center of gravity to move from various initial positions to an Optimally Stable Position. Notice, as mentioned previously, that there is not always a unique Optimally Stable Position.

5.4.2.5 Comparison of Stability Criteria

The concepts of Energy Stability Margin and Optimally Stable Position provide a more quantitative measure of vehicle stability than the concept of Static Stability Margin. The Energy Stability Margin takes into account such factors as the vertical height of the body center of gravity, the pitch and roll of the vehicle body, and the location of the support feet in three-dimensional space. On the other hand, the Static Stability Margin deals only with the vertical projection of the support feet and body center of gravity onto a horizontal plane.

The Level Energy Curves of Figures 5.14 and 5.15, and the corresponding traces of optimal paths in Figures 5.20 and 5.21 provide

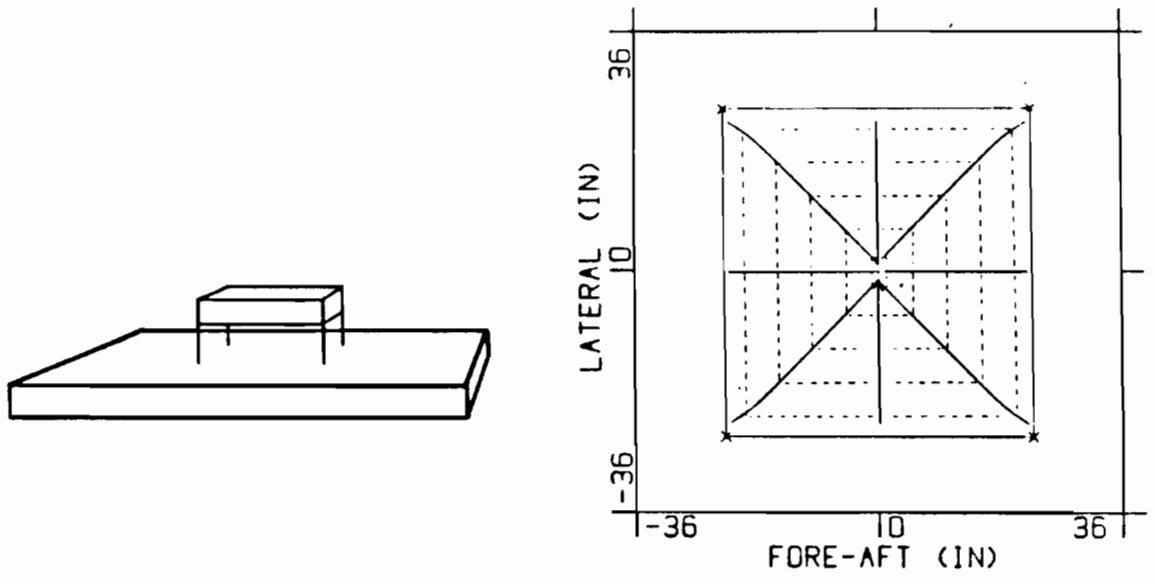


Figure 5.20 Optimal Paths of the vehicle center of gravity for the vehicle standing on level terrain, with the body horizontal. The dotted lines represent the Level Energy Curves.

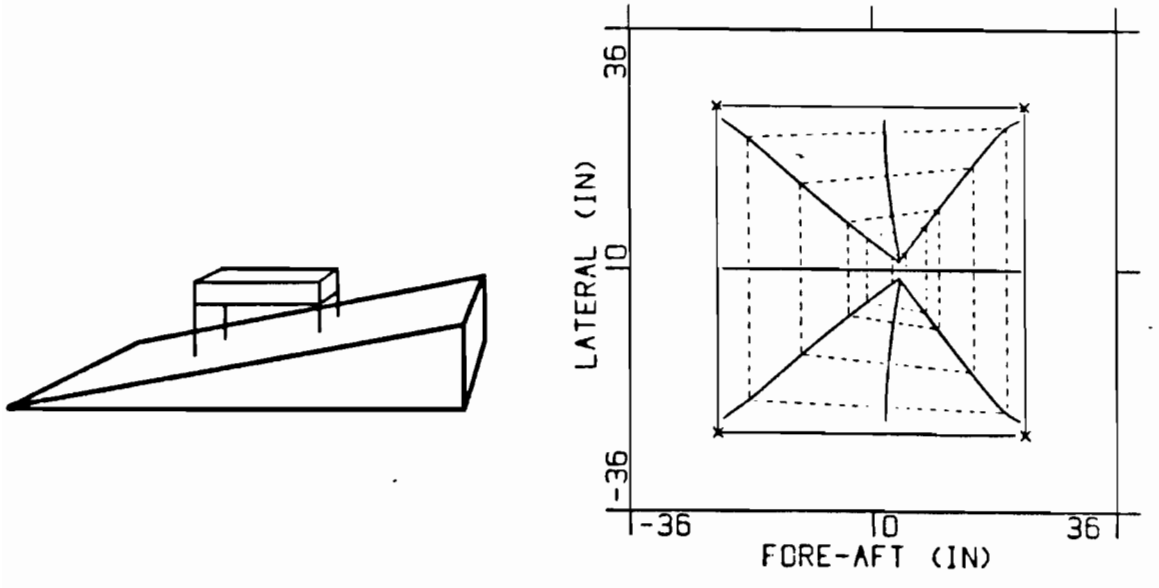


Figure 5.21 Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal. The dotted lines represent Level Energy Curves.

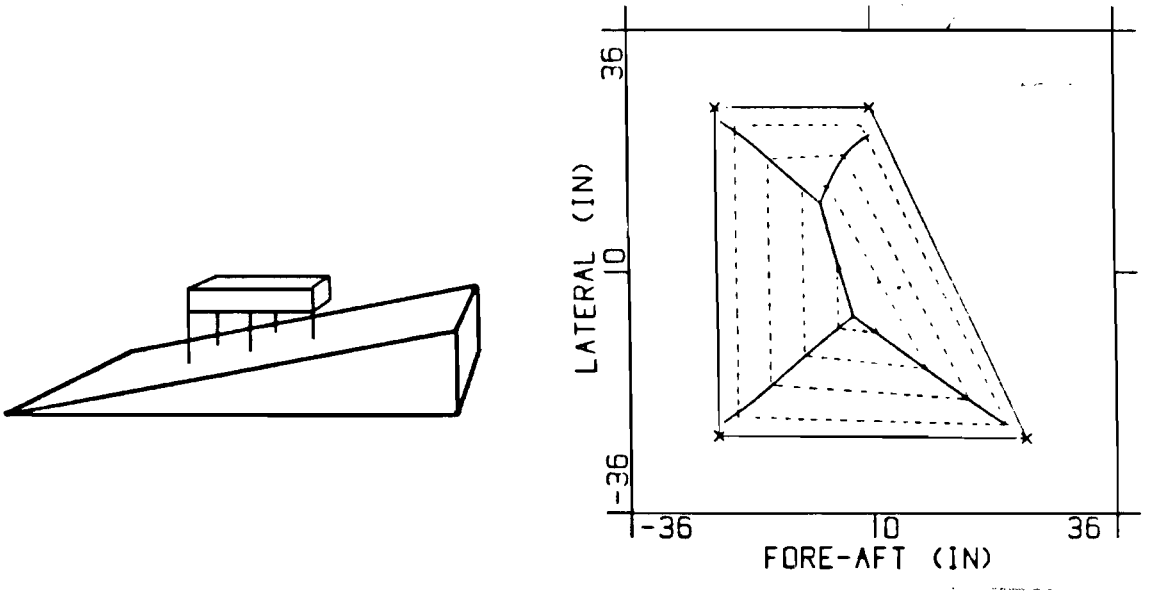


Figure 5.22 Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal and the left front leg off the ground. The dotted lines represent Level Energy Curves.

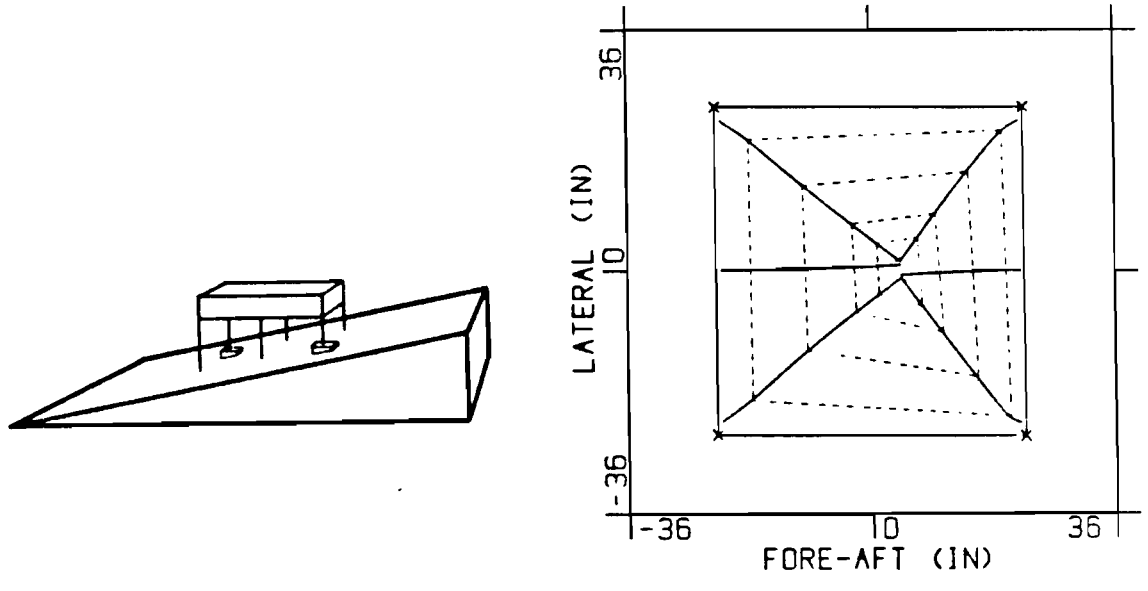


Figure 5.23 Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body horizontal and the right front leg and left rear leg on rocks. The dotted lines represent Level Energy Curves.

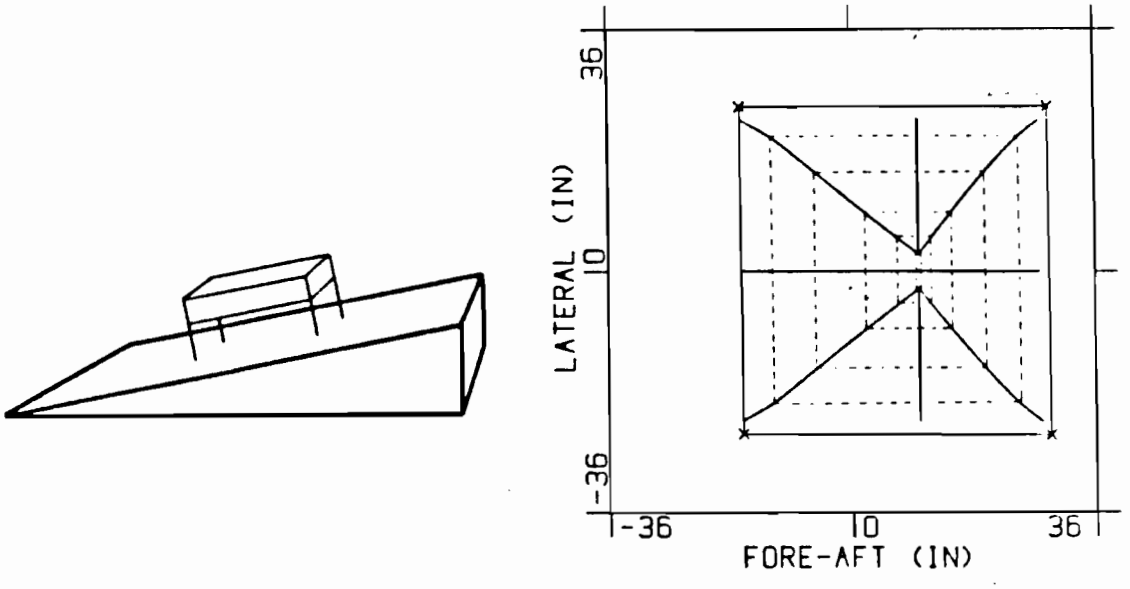


Figure 5.24 Optimal Paths of the vehicle center of gravity for the vehicle standing on a 20 degree inclined plane, with the body pitched at 20 degrees. The dotted lines represent Level Energy Curves.

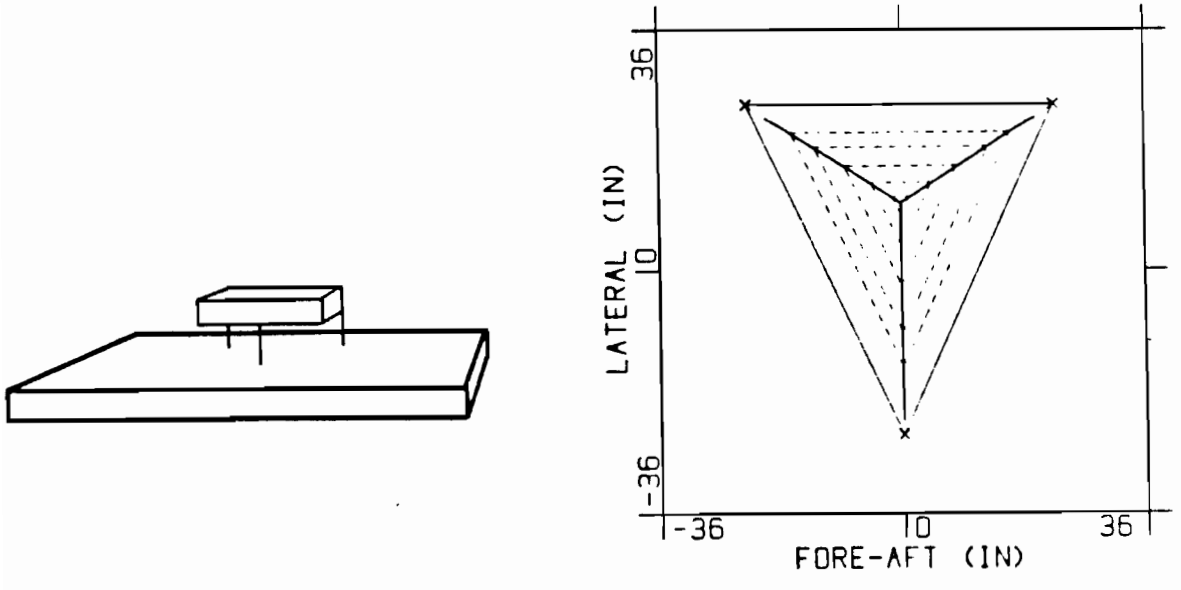


Figure 5.25 Optimal Paths of the vehicle center of gravity for the vehicle standing on level terrain, with a tripod support pattern. The dotted lines represent Level Energy Curves.

a means of comparison of the two stability criteria - Energy Stability Margin versus Static Stability Margin. In both of these figures, the vehicle body is horizontal, and, as a result, the curve of zero Energy Stability Margin is the same as the support pattern of Definition 1.

In Figure 5.14, where the vehicle is on level terrain, the Level Energy Curves indicate that the distance from the center of gravity to a boundary of the support polygon does indeed give a relative measure of the vehicle stability. For the same situation, Figure 5.20 indicates that the greatest measure of stability would be achieved when the center of gravity is at the center of the support polygon. These results agree with the criteria of Static Stability Margin. Therefore, for this level terrain situation, the two stability criteria yield similar conclusions.

In Figure 5.15, the vehicle body remains horizontal, although the vehicle is standing on an inclined plane. Notice that the curve of zero Energy Stability Margin still corresponds to the support pattern of Definition 1. In addition, notice that this zero energy curve is exactly the same as in Figure 5.14. Therefore, the concept of Static Stability Margin leads to identical conclusions as in the previous example. However, the concept of Energy Stability Margin leads to different conclusions. The Level Energy Curves of Figure 5.15 indicate that the distance from the center of gravity to a boundary of the support polygon does not represent a relative measure of vehicle stability. Instead, it demonstrates that the Level Energy Curves are skewed toward the front of the vehicle and that these curves are no longer rectangular, as they were in Figure 5.20. Therefore, the relative stability of the vehicle is not necessarily dependent solely upon the distance from the center of gravity to the support polygon.

Furthermore, Figure 5.21 shows that the Optimally Stable Position is no longer at the center of the support polygon, but instead has shifted toward the front of the vehicle.

The concept of Energy Stability Margin provides a more accurate, quantitative measure of stability than the concept of Static Stability Margin, although both of these concepts provide a qualitative measure to determine whether or not a vehicle is statically stable.

5.4.3 Implementation and Results

The two Automatic Body Regulation schemes, Body Accommodation and Body Stabilization, were incorporated into the computer control algorithm. The Body Accommodation scheme was incorporated as part of the Precision Footing operational mode by modifying the KINEMATIC LIMIT subroutine. The Body Stabilization scheme was incorporated as a subroutine procedure which is invoked by the main program, shown in Figure 3.12.

5.4.3.1 Flow Chart and Explanation of Body Accommodation

The Body Accommodation scheme for Automatic Body Regulation was incorporated as part of the Precision Footing operational mode by modifying the KINEMATIC LIMIT subroutine. The flow chart of the revised KINEMATIC LIMIT subroutine is shown in Figure 5.26. The primary modifications affect those legs of the vehicle which are not in contact with the ground. Two lower-level subroutines are incorporated: UPDOWN LIMIT and XY LIMIT.

The UPDOWN LIMIT subroutine enables the vehicle body to accommodate movements of individual legs in the z-axis direction of the body coordinate system. The flow chart of the UPDOWN LIMIT subroutine is

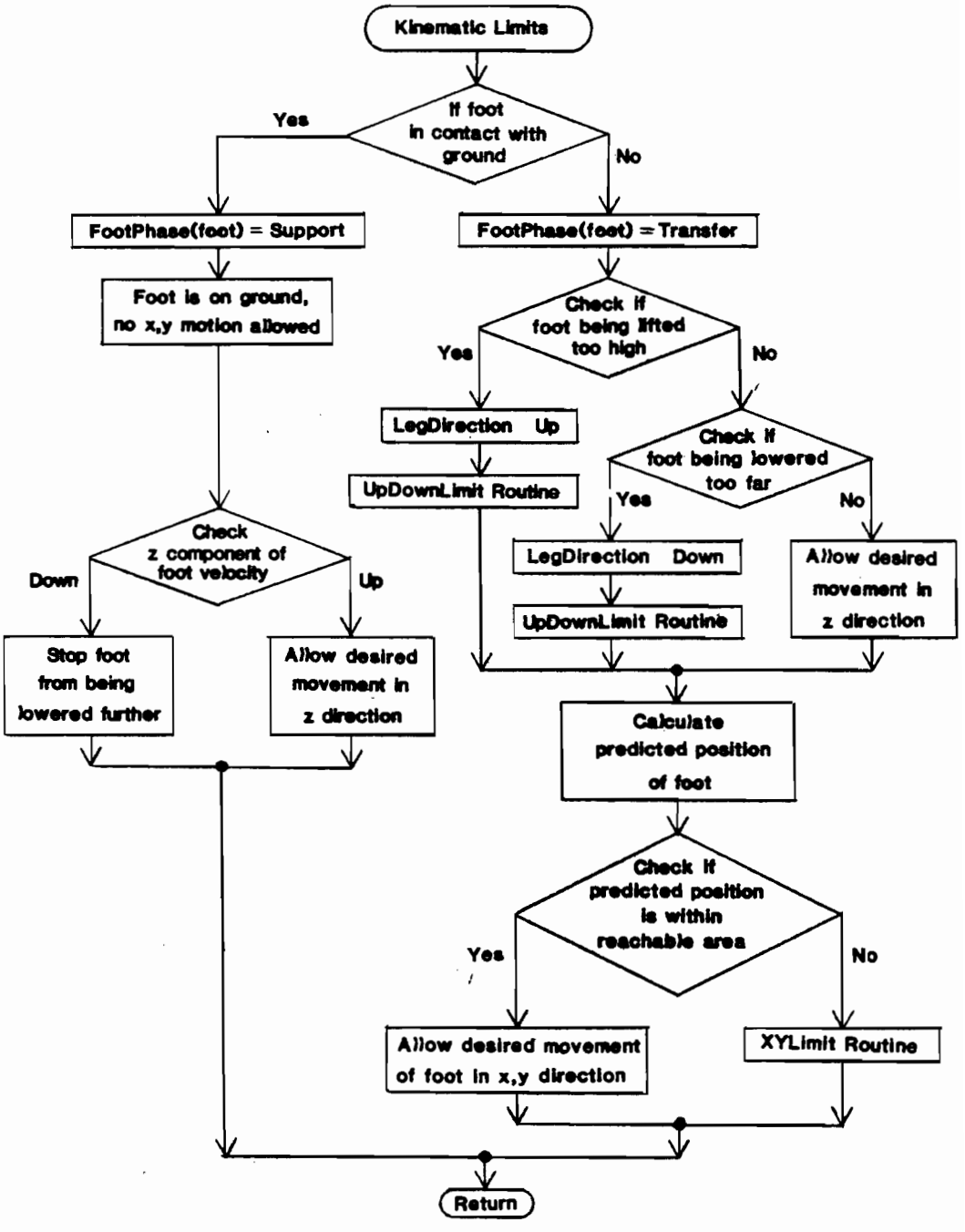


Figure 5.26 Flow chart of the KINEMATIC LIMITS subroutine, with modification for Body Accommodation.

shown in Figure 5.27. This surbroutine includes a check of all support legs to insure that these legs can accommodate the desired z-axis motion without extending beyond the kinematic limits. If any support leg approaches the kinematic limits, the entire vehicle body and its leg motion are halted immediately.

The XY LIMIT subroutine enables the vehicle body to accommodate movements of individual legs in the x and y directions of the body coordinate system. The flow chart of the XY LIMIT subroutine is shown in Figure 5.28. This subroutine is invoked when an individually controlled leg is extended to the edge of its reachable limits. The subroutine checks all of the remaining vehicle legs to insure that the vehicle body can be moved to accommodate the desired leg motion without extending any leg beyond the kinematic limits. If any leg approaches the kinematic limits, both the vehicle and leg motion are halted.

5.4.3.2 Flow Chart and Explanation of Body Stabilization

The Body Stabilization scheme for Automatic Body Regulation was incorporated as an independent subroutine procedure which is invoked by the main program. The flow chart of the OPTIMIZE STABILITY subroutine is shown in Figure 5.29. The subroutine is activated when the operator chooses the appropriate function-select switch. Based upon the present location of all of the support feet, with respect to the body center of gravity, the subroutine determines the optimal path of the body center of gravity and locates the Optimally Stable Position. The subroutine then automatically controls the motion of the vehicle body in order to move the vehicle center of gravity to an Optimally Stable Position. This vehicle movement to the Optimally Stable Position is restricted to remain within the present plane of the vehicle body. After body

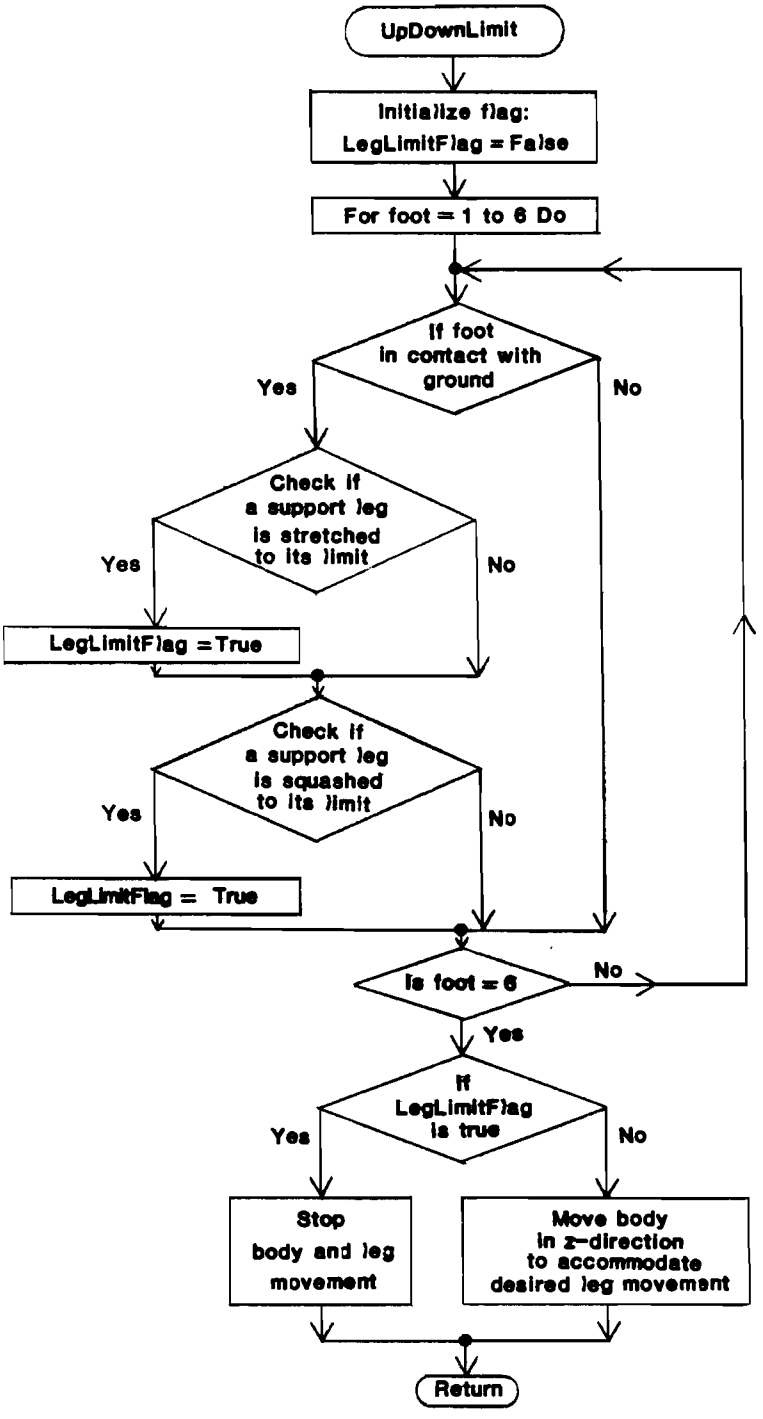


Figure 5.27 Flow chart of the UPDOWN LIMIT subroutine.

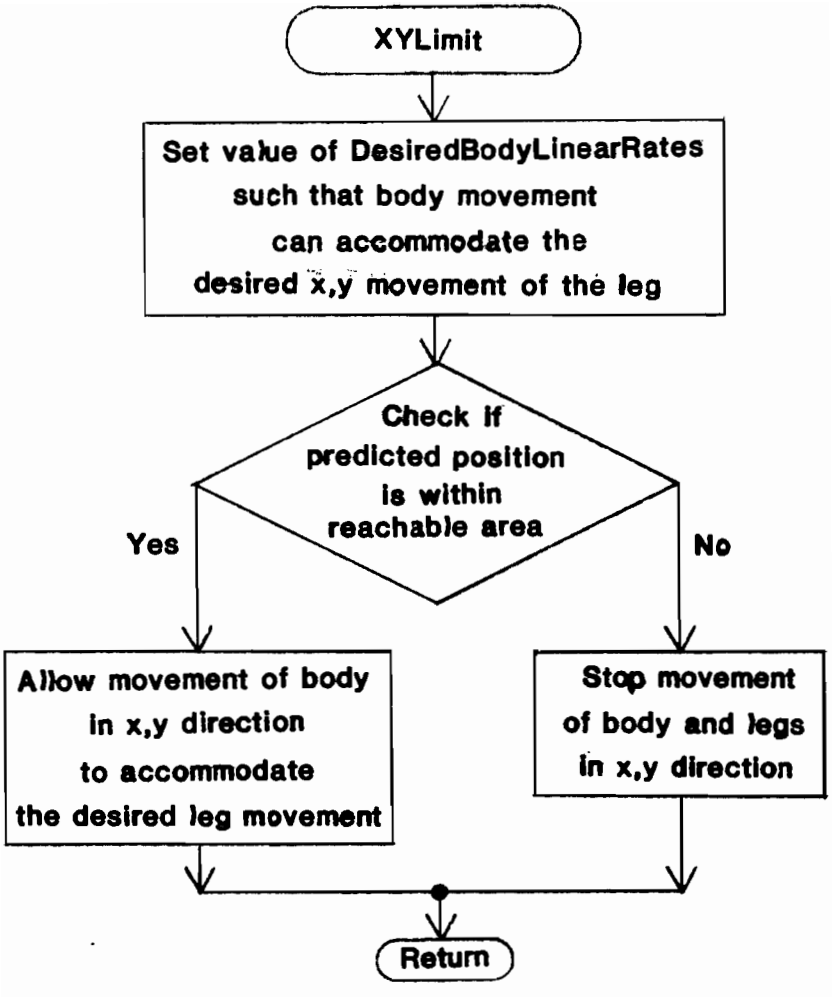


Figure 5.28 Flow chart of the XY LIMIT subroutine.

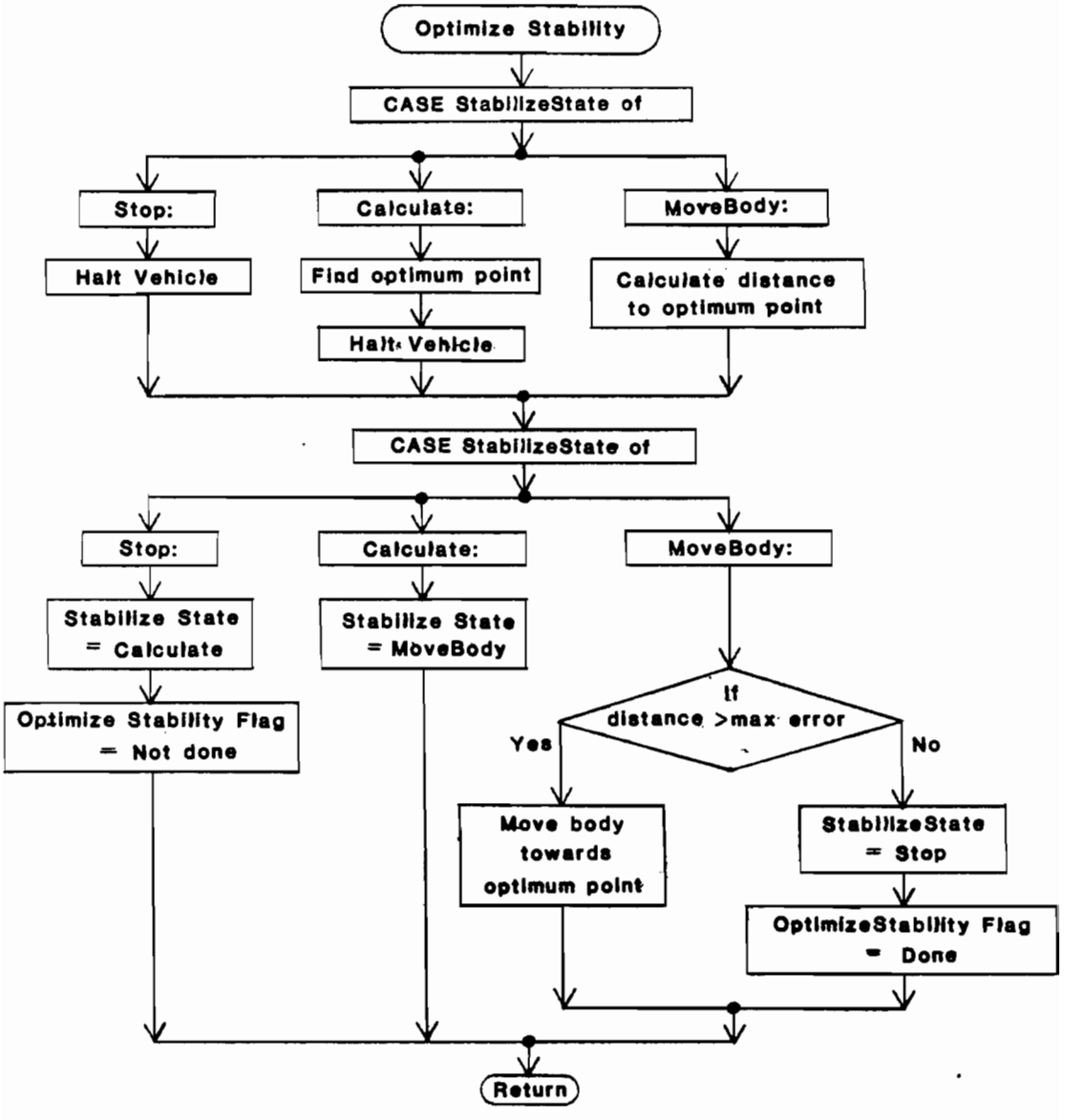


Figure 5.29 Flow chart of the OPTIMIZE STABILITY subroutine.

stabilization is completed, the opertor can proceed with whatever maneuvers are desired.

It should be noted that it may not always be possible to move the vehicle body in such a way that the body center of gravity coincides with an Optimally Stable Position. This movement may be limited when the body orientation and leg positions are such that the leg kinematic limits prohibit the necessary body movement. In these instances the Body Stabilization scheme would move the body toward the Optimally Stable Position until further movement is prohibited by these kinematic limits. The result would be that the body is positioned at the point of maximum stability allowable with the present body orientation, leg positions, and kinematic limits.

5.4.3.3 Results

The Automatic Body Regulation schemes were implemented on the OSU Hexapod vehicle, as well as on a computer graphics simulation of the Adaptive Suspension Vehicle (ASV).

The Body Accommodation scheme for Automatic Body Regulation was incorporated as part of the Individual Foot Control Algorithm. The operator's controls consist of a three-axis joystick, two function-select switches, and six leg-select switches. Through the use of the function-select switches the operator may choose to control either the linear body motion of the vehicle or the angular body motion of the vehicle. In addition, the operator may choose to control the motion of an individual vehicle leg. The particular leg to be controlled is indicated by activating the appropriate leg-select switch. In all cases, the three-axis joystick provides the means of inputting the direction and velocity of the desired movement. As before, the

vehicle operator is provided with a feedback of information via the graphics display terminal.

Figure 5.30 depicts the OSU Hexapod climbing onto an elevated wooden platform. During this maneuver, the vehicle is operating in the Individual Foot Control mode, with the Body Accommodation feature. If the Body Accommodation feature had not been used, this type of maneuver would require a much greater degree of operator input. Furthermore, without the various automatic monitoring features of the Individual Foot Control mode this maneuver would require an extensive amount of concentration by the operator.

The Body Accommodation scheme greatly enhances the maneuverability of the vehicle during the Precision Footing operational mode. Since the movement of the vehicle body is automatically commanded to accommodate the operator's desired movement of an individual leg, this feature reduces the complexity of the operator's control task. Body Accommodation allows the operator to manipulate a vehicle leg within a much larger volume than was possible with the typical reachable volume of each leg, as determined by the kinematic limits. Therefore, the vehicle can be maneuvered over a much larger range, while simplifying the operator's control task.

The Body Stabilization scheme for Automatic Body Regulation is activated when the vehicle operator chooses the appropriate function-select switch. The operation of the Body Stabilization routine is demonstrated in Figures 5.31 and 5.32. Figure 5.31 shows the OSU Hexapod in a rather precarious configuration -the vehicle is climbing over an obstacle and has been maneuvered into an awkward position. The graphics display information corresponding to this situation is also shown in Figure 5.31. The small square inside the support polygon

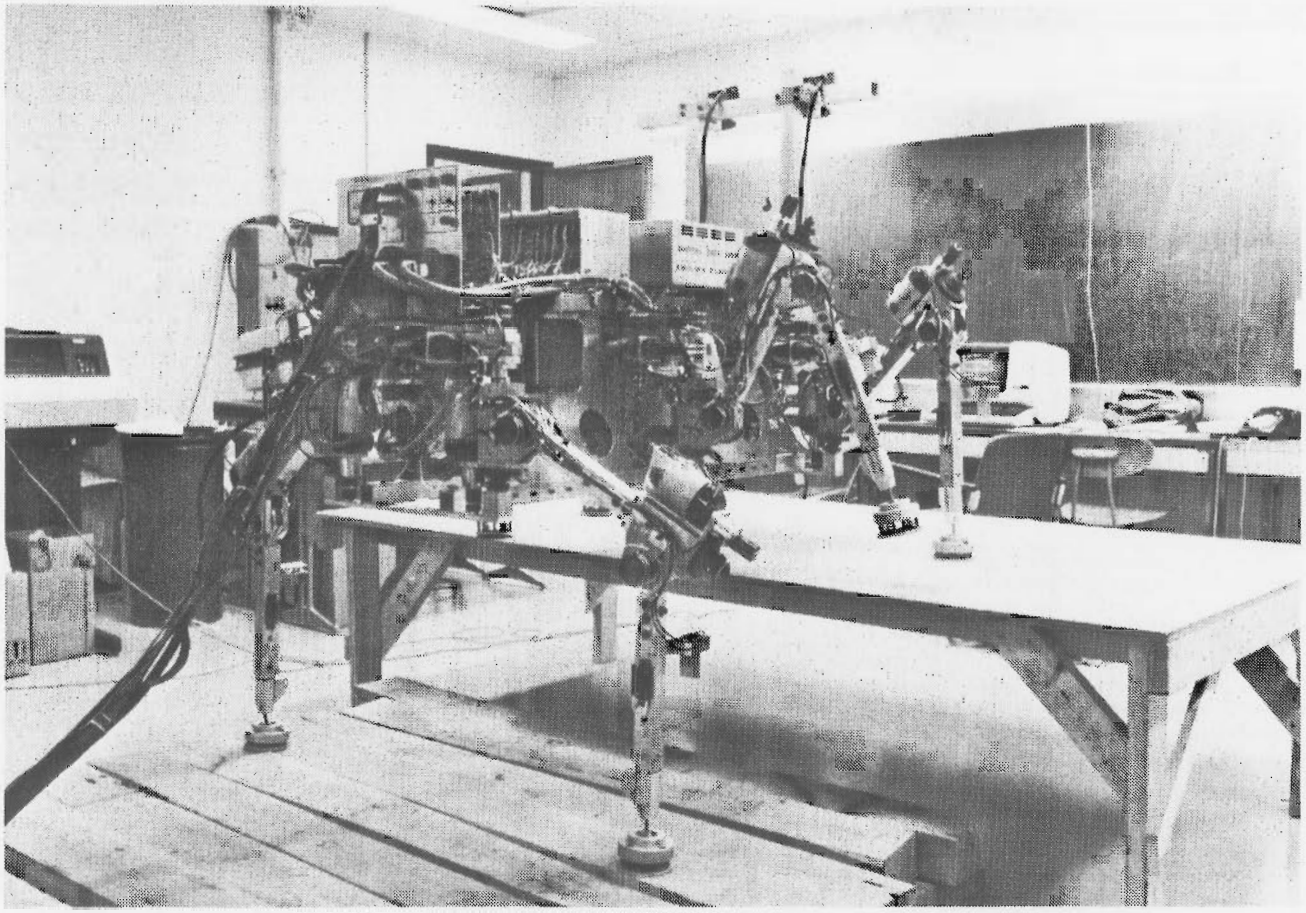


Figure 5.30 Photograph of the OSU Hexapod climbing onto a wooden platform. The vehicle is operating in the Individual Foot Control mode, with the Body Accommodation feature.

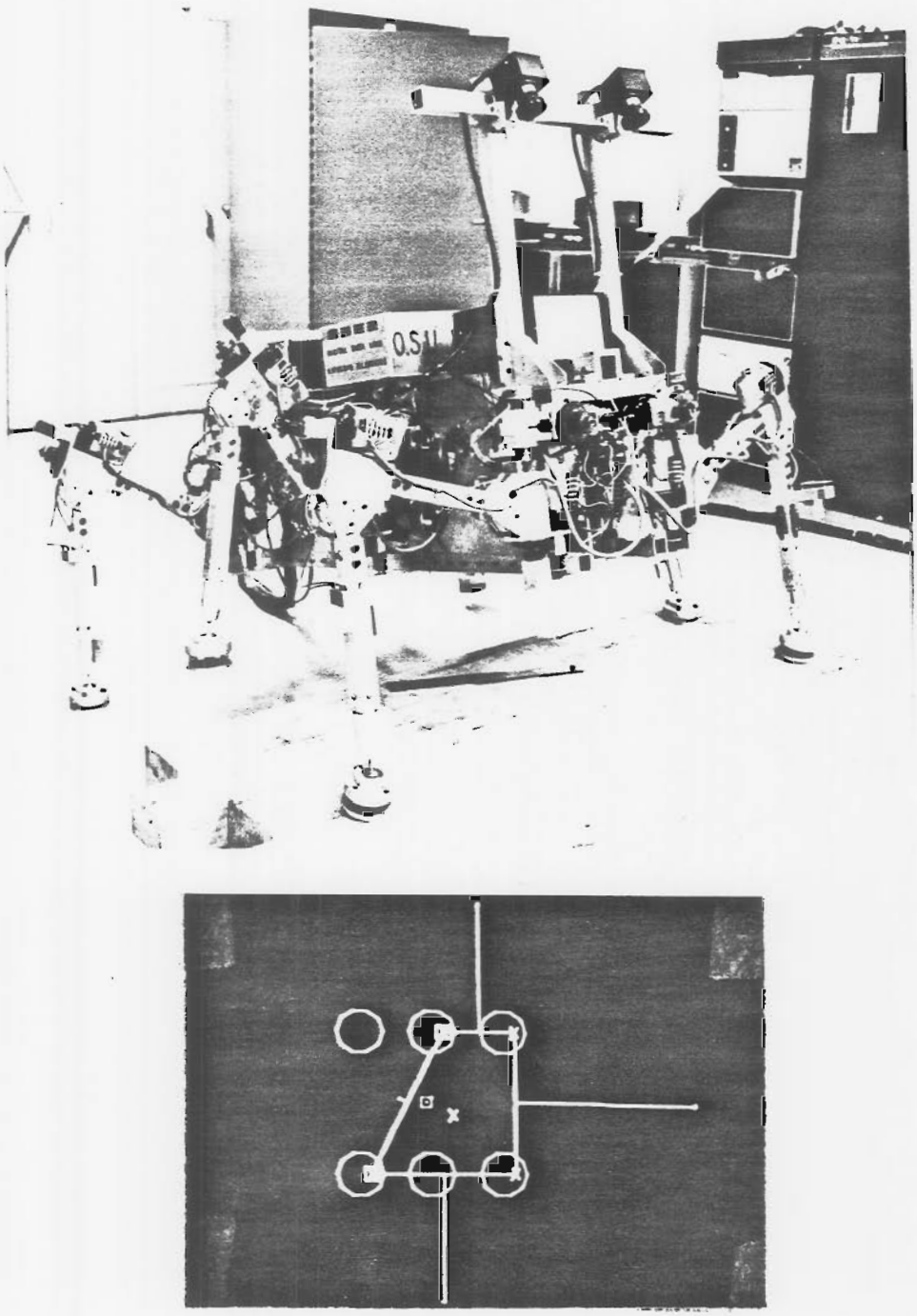


Figure 5.31 The OSU Hexapod, and corresponding graphics display at the start of the Body Stabilization algorithm. The X symbol, located inside the support polygon, indicates the Optimally Stable Position to which the center of gravity should move.

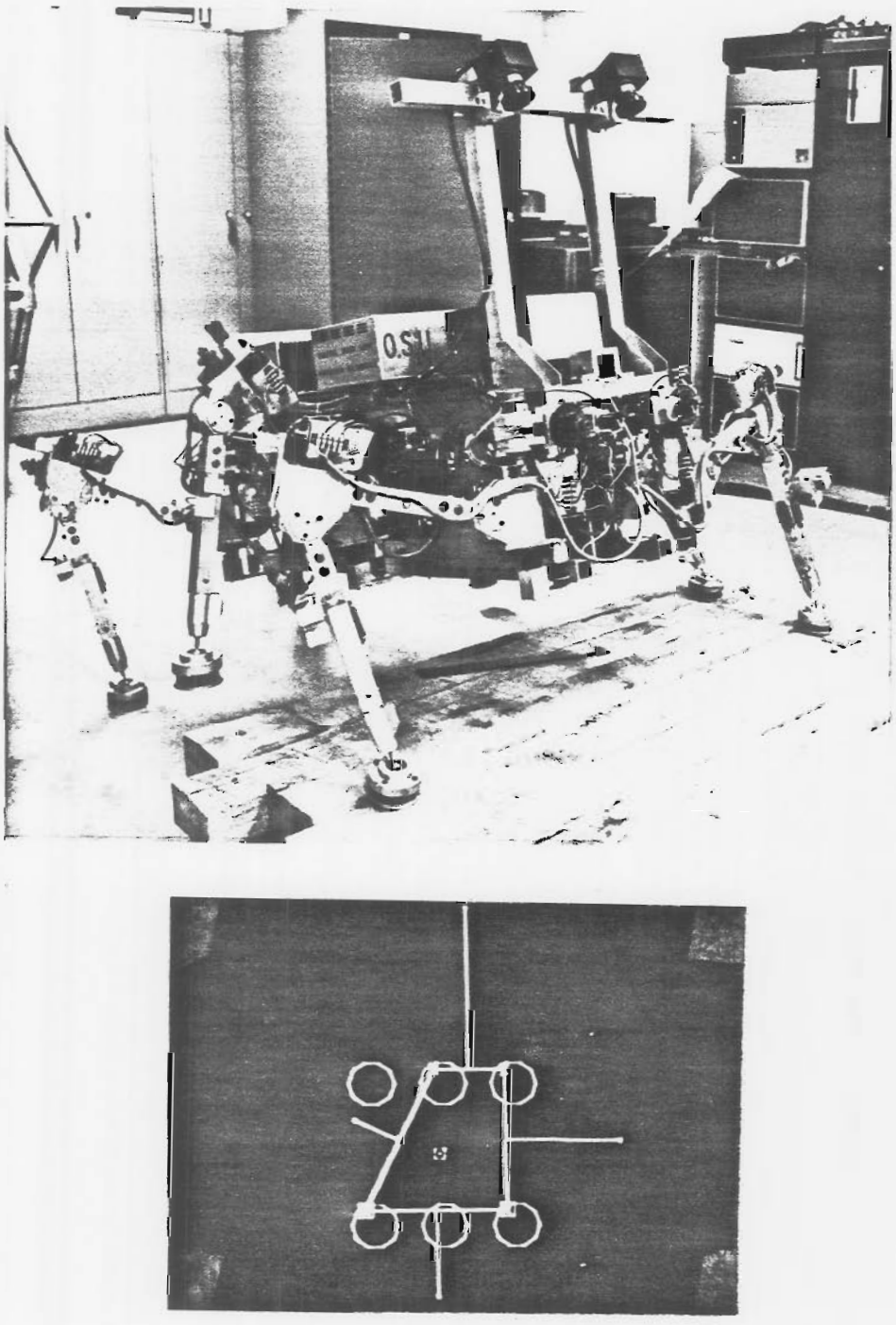


Figure 5.32 The OSU Hexapod, and corresponding graphics display, after the completion of the Body Stabilization algorithm. The body has shifted so that the center of gravity is at the Optimally Stable Position.

represents the location of the body center of gravity. The X symbol, also located inside the support polygon, indicates the Optimally Stable Position of the body center of gravity. This X symbol appears when the Body Stabilization routine is activated. Figure 5.32 shows the results after invoking the Body Stabilization routine. Notice that the vehicle body remains in the same plane as it was initially and the body has been shifted to the Optimally Stable Position. The graphics display information of Figure 5.32 indicates that the body center of gravity is now at the Optimally Stable Position as evidenced by the location of the present center of gravity, and the magnitudes of the Energy Stability Levels.

The Body Stabilization scheme provides the operator a means of automatically repositioning the vehicle body such that the Energy Stability Margin is optimal for the given leg configuration. This feature significantly reduces the operator's burden since the vehicle may now be maneuvered more freely. Whenever the operator determines that the Energy Stability Margin has decreased below a reasonable level, the Body Stabilization routine may then be activated. As a result, the vehicle body is shifted such that the body center of gravity is at the Optimally Stable Position. The Body Stablization feature may also be quite useful when the operator desires to position the vehicle at an Optimally Stable Position before attempting to maneuver across certain obstacles. Since the Body Stabilization routine is completely automatic once activated, it permits the vehicle stability to be optimized easily and efficiently, for any vehicle orientation and leg positions.

5.5 Summary

The development of various leg placement algorithms, designed for the Precision Footing operational mode, has been presented in this chapter. Each of these algorithms provides the operator with a different level of vehicle control and computer interaction.

The Manual Tripod Leg Placement Algorithm treats the six legs of the vehicle as two independent sets of tripods. At least one of these two sets of tripods is supporting the vehicle at all times. The tripod support pattern provides a stable stance, especially on rough terrain. In the Manual Tripod Algorithm, the vehicle operator directly controls the movement of the tripod sets. The operator's controls consist of a three-axis joystick and two function-select switches. Through the use of the function-select switches, the operator may choose to control the movements of either the transfer phase tripod or the support phase tripod. Whichever of these two modes is selected, the three-axis joystick provides the means of commanding the direction and velocity of the desired movement. The operator may lift a tripod off the ground, position this tripod such that all three feet are above solid footholds, then lower this tripod to the ground in order to support the vehicle. The operator may also control the motion of the vehicle body with respect to those feet which have remained in contact with the ground. The algorithm does not require any filtering of the joystick velocity commands and, as a result, the tripod movement responds quickly to the operator's commands. Furthermore, the algorithm includes a safety check to insure that no leg is extended beyond its kinematic limits.

The Individual Foot Control Algorithm enables the operator to directly control the movement of each vehicle leg individually. In addition, the operator is able to control the movement of the vehicle

body with respect to the supporting feet. The vehicle operator can issue commands to the algorithm through the use of a three-axis joystick and a set of function-select switches. The desired foot to be moved is selected by activating the appropriate select switch which corresponds to the proper foot. The three-axis joystick may then be used to command the desired velocities of the foot in the longitudinal, lateral, and vertical directions. Through activation of the proper function-select switch, the operator may also choose to control the movement of the vehicle body, with respect to the support feet. The joystick may now be used to command the three linear body velocities: longitudinal body motion, lateral body motion, and body altitude. Another function-select switch is activated if the operator chooses to control the body attitude. The joystick may then be used to command the three angular body velocities: pitch, roll, and yaw rates. To aid the vehicle operator, the Individual Foot Control Algorithm is designed to provide feedback of information to the operator via a computer graphics display terminal. In addition, the control algorithm includes various automatic monitoring features to assure the safety of both the operator and the vehicle.

One of the major developments presented in this chapter is the concept of Energy Stability Margin. A general equation was derived which allows the calculation of the Energy Stability Margin for any given position of the body and legs. By utilizing the gradient of this function, an optimal path can be found leading from a given initial location of the body center of gravity to an Optimally Stable Position. In comparison to the concept of an Optimally Stable Point, as introduced by Tsai [29], the concept of an Optimally Stable Position as defined in the present work represents a more general approach which is applicable

for any type of terrain condition. All previous work was only accurate on flat terrain. Furthermore, the heuristic algorithm suggested by Tsai required an excessive amount of calculation and could not be implemented. On the other hand, the Energy Stability Margin can be calculated efficiently for any position of the vehicle body and legs. As a result, the stability of the vehicle can be automatically optimized during real-time control.

Application of the Energy Stability Margin, together with a consideration of constraints on the kinematic limits of individual legs, resulted in the development of two Automatic Body Regulation schemes. These Automatic Body Regulation schemes allow greater vehicle maneuverability, particularly during rough-terrain locomotion. With the Automatic Body regulation schemes, the operator may choose to allow the computer control algorithm to automatically adjust the position of the vehicle body in accordance with certain predefined criteria. The two Automatic Body Regulation schemes are referred to as Body Accommodation and Body Stabilization.

The Body Accommodation scheme is incorporated as part of the Individual Foot Control Algorithm. With Body Accommodation, the vehicle body is automatically commanded to accommodate the operator's desired movement of an individual leg. This feature allows the vehicle to be maneuvered over a much larger range, while simplifying the operator's control task.

The Body Stabilization scheme provides the operator a means of automatically repositioning the vehicle body in such a way that the Energy Stability Margin is optimal for the given leg configuration. This Body Stabilization feature is activated when the operator chooses the appropriate function-select switch. Based upon the present vehicle

body orientation and leg positions, the Body Stabilization scheme determines the location to which the body center of gravity should be moved in order to optimize the vehicle stability. The vehicle is then automatically shifted in such a way that the body center of gravity is located at the Optimally Stable Position. An analogy can be drawn between the concept of an Optimally Stable Position and the concept of a point in a potential well in physics [46].

All of these algorithms were implemented on both the OSU Hexapod vehicle and a computer graphic simulation of the Adaptive Suspension Vehicle (ASV). These algorithms enable the operator to control the vehicle in the Precision Footing operational mode. The various algorithms allow the operator to select different levels of vehicle control and computer interaction. Therefore, the operator may choose the desired level of control, in accordance with the terrain conditions and the desired degree of vehicle maneuverability. Furthermore, the algorithms enable the vehicle to perform highly complex maneuvers without overburdening the vehicle operator. The features incorporated into these algorithms are designed to alleviate the burden of manipulating the vehicle body and limbs, but without greatly restricting the freedom of movement which is inherent to the Precision Footing operational mode.

CHAPTER 6

SUMMARY AND CONCLUSIONS

6.1 Research Contributions

The research efforts documented in this dissertation have culminated in the implementation of a family of computer control algorithms which enable the locomotion of a legged vehicle over rough terrain with a high degree of maneuverability. All of these control algorithms were implemented on both the OSU Hexapod vehicle and a computer graphic simulation of the Adaptive Suspension Vehicle (ASV). The implementation of these algorithms demonstrates the feasability of utilizing the potential maneuverability features of a legged vehicle. At the same time, the control algorithms were designed to relieve the vehicle operator of much of the complex task of maneuvering a legged vehicle.

One of the major contributions of this research has been the development of the Dual Tripod Leg Placement Algorithm. This is a highly maneuverable walking algorithm which is particularly well suited for the Close Maneuvering operational mode.

The Dual Tripod Leg Placement Algorithm represents the first hardware implementation of supervisory control for an omnidirectional walking algorithm capable of rough terrain locomotion. In this

algorithm, leg motion is limited only by geometrical considerations. As a result, there are no time-sequencing constaints as in the typical wave-gait formulations. The derivation of the equations of motion is based upon the observation that, in general, the center of gravity of the vehicle can be considered as moving along a circular arc. This observation provides a concise description of the motion of the vehicle body and completely describes the required motion of the vehicle's support legs in order to propel the body along the desired trajectory. Using this approach, equations were derived for expressing the motion of the body center of gravity in response to the commanded velocities. The concept of defining a reachable volume for each foot, based upon the kinematic limits, provides a unique criteria for determining when to lift a foot off the ground and when to place a foot down. The concept of reachable volume, together with the concept of travelling along a circular path, also leads to a unique method for determining the location of the desired footholds for those vehicle legs which are in the transfer phase. Furthermore, by allowing the transfer phase feet to track the ground while they are being lifted off the ground or placed on the ground, the trajectory of the transfer phase legs insures that they will not collide with surrounding obstacles whose elevation are less than some predetermined height.

A predominant problem in the Close Maneuvering mode is the gait transition of the vehicle legs as the direction of body motion is changed. Previous algorithms required a tradeoff of speed versus agility - that is, the velocity input commands required a filter with a long time-constant. This filtering action was required in order to

avoid sudden changes in operator commands. On the other hand, by the nature of its design, the Dual Tripod Algorithm does not have a gait transition problem; therefore filtering of the input commands is no longer necessary. As a result, the Dual Tripod Algorithm is capable of responding to sudden changes in the operator's joystick commands. The result is an extremely agile walking algorithm which is especially well-suited for the Close Maneuvering operational mode.

In this research work various leg placement algorithms for the Precision Footing operational mode have been developed and implemented. Each of these algorithms provides the operator with a different level of vehicle control and computer interaction.

The Manual Tripod Leg Placement Algorithm enables the vehicle operator to directly control the movement of the two tripod sets of legs. This level of Precision Footing control results in a high degree of vehicle maneuverability. At the same time, the operator burden is reduced to a minimum since the vehicle's legs are controlled as tripod sets rather than treating each leg independently. The Manual Tripod Algorithm provides a level of vehicle control which was previously unavailable. It is particularly useful over moderately rough terrain, where the full capabilities of the Precision Footing mode may not necessarily be required.

The Individual Foot Control Algorithm is another leg placement algorithm developed for the Precision Footing operational mode in this research. This algorithm enables the vehicle operator to control the movement of each leg individually. The algorithm includes various automatic monitoring features to assure the safety of both the operator

and the vehicle. The incorporation of the Body Accommodation scheme of Automatic Body Regulation greatly enhances the maneuverability of the vehicle in this mode. It is this Body Accommodation scheme which allows automatic movement of the vehicle body in order to accommodate the operator's desired movement of an individual leg.

One of the important contributions of this research has been the use of a computer graphics display to provide a feedback of information to the vehicle operator. The graphics display provides essential information to simplify the operator's control task. Furthermore, the graphical format enables the operator to interpret the information easily and efficiently.

Another major contribution of this research has been the development of the concept of Energy Stability Margin. A general equation was derived which allows the calculation of the Energy Stability Margin for any given position of the body and legs. This equation makes it possible to plot families of Level Energy Curves, which serve as contour maps of the Energy Stability Margin. Similarly, by utilizing the gradient of this function, an optimal path can be determined, which leads from a given initial location of the body center of gravity to an Optimally Stable Position.

The Energy Stability Margin provides a quick and accurate quantitative measure of vehicle stability, particularly for rough-terrain conditions. Previous stability criteria provided a qualitative measure of stability, but did not account for these rough-terrain conditions. In this respect, they did not provide a useful quantitative stability measure. It is this quantitative measure, provided by the Energy

Stability Margin, which allowed the development of a computer algorithm for determining an Optimally Stable Position.

Furthermore, this capability for determining an Optimally Stable Position led to the development of the Body Stabilization scheme of Automatic Body Regulation. The Body Stabilization scheme provides the operator a means of automatically repositioning the vehicle body so that the Energy Stability Margin is optimal for the given leg configuration.

All of these developments have been incorporated into a complete computer control program which fully demonstrates the maneuverability of a legged vehicle for rough terrain locomotion. This computer program enables the vehicle operator to control the vehicle in either the Close Maneuvering or the Precision Footing operational modes. The operator can select the level of control in the Precision Footing mode by choosing between the Manual Tripod or the Individual Foot Control Algorithms. At all times, the operator is provided a feedback of information via the graphics display terminal. Furthermore, the operator is capable of switching between these various control modes as is warranted by the terrain conditions. This increased level of flexibility exemplifies the high degree of maneuverability capable of being achieved by such a legged vehicle.

6.2 Research Extensions

The developments resulting from this research demonstrate a measure of the potential maneuverability of a legged vehicle. These results lead to the realization that further refinements and new developments

may utilize even more of the potential of a legged vehicle. Several suggestions for future work are presented in the following paragraphs.

Vehicle operation could be enhanced and the computer control algorithms could be simplified by the incorporation of additional sensor systems. The use of contact sensors on each foot could provide an on/off indication of whether or not a foot is in contact with an object. This information is presently obtained through interpretation of the force sensor readings. However, the force sensors are prone to drift in value and as a result may require occassional resetting.

Similarly, the incorporation of proximity sensor information may enhance the rough terrain mobility of the vehicle. The proximity sensors provide a measure of the distance from the tip of the vehicle foot to surrounding objects. The proximity sensor information could be used to modify the foot placement velocity so that the vehicle feet make a softer contact with the supporting surface [43]. In addition, the proximity sensor information could better enable the vehicle feet to automatically avoid collisions with obstacles.

A terrain scanning system could be included as part of the vehicle's sensory system. Sensory information from this system could be incorporated into a hybrid of the Dual Tripod and Manual Tripod Algorithms. This hybrid algorithm would reduce the operator burden by automatically reacting to the sensory information of the terrain scanner. The operator could then concentrate on the higher level control inputs and be less concerned with the precise movement of the vehicle legs.

In addition, the terrain scanner could provide information which would be used for automatic control of the attitude and altitude of the vehicle body. The incorporation of this automatic attitude and altitude control would improve the vehicle's mobility over certain types of overall terrain conditions.

This research work introduced the concept of using a computer graphics display to provide a feedback of information to the vehicle operator. This innovative concept could be extended to provide the operator a full spectrum of information. However, care should be taken to avoid inundating the vehicle operator with an overwhelming amount of information. Particularly in critical maneuvering situations, the operator may desire to have only a selected set of essential parameters displayed on the graphics display screen. It may be desirable to provide the operator a means of selecting, from a menu, the parameters and other information to be displayed on the screen.

The general formulation of the equation for calculating the Energy Stability Margin allows an interesting extension. Since the OSU Hexapod vehicle is equipped with force sensors on each foot, this force information could be used to actively compute the location of the vehicle center of gravity [50]. This active center of gravity would take into account any effects of vehicle cargo loading, changes in fuel level, etc. The Energy Stability Margin could be calculated using the active center of gravity. The result would be improved vehicle stability since any variation in the vehicle center of gravity could now be compensated.

The present work defined the Optimally Stable Position to lie in the plane of the vehicle body. As an alternative, the Optimally Stable Position could be defined to lie in any other plane, such as a plane which is parallel in the supporting surface. The derivations presented are sufficiently general to allow such extensions.

The analysis of vehicle stability considered only the effect of gravitational acceleration, i.e. the situation of static stability. As vehicle speed and size is increased, it may become necessary to consider the dynamic stability of the vehicle. The concept of the Energy Stability Margin could be generalized to include the effects of the vehicle's kinetic energy as well as the effects of the acceleration of the vehicle with respect to the earth-fixed coordinate system.

One of the particularly interesting future applications for the concepts developed in this research would be in the development of free-gait algorithms [19]. The Dual Tripod Leg Placement Algorithm has demonstrated that the OSU Hexapod vehicle is capable of highly maneuverable locomotion. This algorithm has none of the time-sequencing constraints which are a major limitation in wave gait formulations. The concepts used in the Dual Tripod Algorithm could be extended to create a free-gait algorithm. Furthermore, the Automatic Body Regulation schemes presented in this research could be incorporated into such a free-gait algorithm. The Body Accommodation feature would allow the vehicle body to move in such a manner as to provide a wider selection of allowable footholds. The Body Stabilization feature could be used to maintain maximal stability of the vehicle. With the additional sensory systems

mentioned previously, a free-gait algorithm could provide an extremely agile walking machine.

The concepts of Energy Stability Margin and an Optimally Stable Position can be used in a wide variety of situations, and should lead to the development of more sophisticated control algorithms for legged vehicles. The new algorithms can further the realization of a legged vehicle's potential for maneuverability over rough terrain.

APPENDIX

COMPUTER PROGRAMS OF THE WALKING ALGORITHMS

```
TYPE    Vector  = RECORD
                    X,Y,Z : Real;
                  END;

        Matrix  = Array[ 1..3 ] of Vector;

        Homogeneous
                = RECORD
                    Rotation    : Matrix;
                    Translation : Vector;
                  END;

        SixVectors = Array[ 1..6 ] of Vector;
        SixMatrices = Array[ 1..6 ] of Matrix;

        Array3 =  Array [0..2] of Real;
        Array6  = Array [1..6] of Real;

        SixBooleans = Array[ 1..6 ] of Boolean;

        LegSet = SET OF 1..6;

        NormalizeMode = ( NotReady, Ready, Erection, Move1, Move2, Elevation );

        FootState = ( Support, Transfer );
        SixStates = Array[ 1..6 ] of FootState;

        FootSubstate = ( Ground, LiftOff, Return, PlaceDown, Error );
        SixSubstates = Array[ 1..6 ] of FootSubstate;

        RateType = ( Absolute, Relative );
        SixRateTypes = Array[ 1..6 ] OF RateType;

        Completion = ( NotDone, Done );


        Coordinate = ( X, Y, Z );
        CoordinateSet = SET of Coordinate;
        SixCoordSets = Array[ 1..6 ] of CoordinateSet;

{ New TYPE declarations }

        Vector2 = RECORD
                    X,Y,R : Real;
                  END;

        FiveSubstates = ( State1, State2, State3, State4, Special );
        FiveStates = Array[1..6] of FiveSubstates;
        TripodGroup = ( Left, Right );
        SpecialSubstates = ( StopLegs, LegsDown, LiftTripod, ReturnTripod );
        Direction = ( Up, Down );
        SevenVectors = Array[1..7] of Vector;
        Array2 = Array[1..2] of Real;
        StabilizeMode = ( Stop, Calculate, MoveBody );
```

```
CONST   Zero = Vector( 0.0, 0.0, 0.0 );
        Identity = Matrix( ( 1.0, 0.0, 0.0 ),
                           ( 0.0, 1.0, 0.0 ),
                           ( 0.0, 0.0, 1.0 ) );
        Pi = 3.14159;

{ new CONST declarations }

        MaxFootHeight = -10.0;   { inches }
        MinFootHeight = -19.436; { inches }
        ReachRadius   =  7.25;   { inches }

        EquilibriumPosition = SixVectors( (  22.75,  24.00, 0.0 ),
                                          (  22.75, -24.00, 0.0 ),
                                          (  0.0,    24.00, 0.0 ),
                                          (  0.0,   -24.00, 0.0 ),
                                          ( -22.75,  24.00, 0.0 ),
                                          ( -22.75, -24.00, 0.0 )  );

        PlacePoint = SixVectors( (  22.75,  24.00, -10.0 ),
                                 (  22.75, -24.00, -10.0 ),
                                 (  0.0,    24.00, -10.0 ),
                                 (  0.0,   -24.00, -10.0 ),
                                 ( -22.75,  24.00, -10.0 ),
                                 ( -22.75, -24.00, -10.0 )  );


        LiftRate = 2.0;          { inches per second }
        ReturnRate = 4.0;        { inches per second }
        PlaceRate = 3.0;         { inches per second }
        MaxError = 0.4;

        FastLiftRate = 3.7;      { inches per second }
        FastReturnRate = 4.0;    { inches per second }
        FastPlaceRate = 4.0;     { inches per second }

        Xhip = Array6(22.75, 22.75, 0.0, 0.0, -22.75, -22.75);
        Yhip = 24.00;
```

```
{$own}

{********************* FILE:  OSUTEST13.PAS ****************************}


%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declaration }
%INCLUDE DMEXTERN9;     { External procedure declarations }
%INCLUDE EXTPROJ3;      { External procedures and types for Projection }
%INCLUDE EXTPROJHP;     { HP NonPascal procedure declarations }

VAR     ActualBodyAngularRates          : Vector;
        DesiredBodyLinearRates          : Vector;
        DesiredBodyAngularRates         : Vector;
        BodyServoAngularRates           : Vector;
        BodyServoAngularRatesEarth      : Vector;


        DesiredBodyTransform            : Homogeneous;
        ActualBodyTransform             : Homogeneous;
        PresentBodyTransform            : Homogeneous;

        ActualFootPosition              : SixVectors;
        DesiredFootPosition             : SixVectors;

        PlannedFootRate                 : SixVectors;
        FootRateType                    : SixRateTypes;
        BodyServoFootRate               : SixVectors;
        DesiredFootRate                 : SixVectors;
        LegServoFootRate                : SixVectors;

        ActualForce                     : SixVectors;
        DesiredForce                    : SixVectors;

        FootContact                     : SixBooleans;

        InverseJacobian                 : SixMatrices;
        Tactilian                       : SixMatrices;
        ForceSet                        : SixCoordSets;

        ZeroForce                       : SixVectors;
        ActualFootForce                 : SixVectors;
        ActualJointAngles               : SixVectors;
        ActualJointRates                : SixVectors;
        CommandJointRate                : SixVectors;
        OutVoltage                      : SixVectors;

        Pitch                           : Real;
        Roll                            : Real;
        Yaw                             : Real;

        foot                            : Integer;

        JoyStickDeflection              : Array3;

        LegNumber                       : Integer;

        CurrentVehicleMode              : Char;
        CommandedVehicleMode            : Char;

        CurrentTime                     : Real;
        LastTime                        : Real;
        dt                              : Real;
```

```
        WaitUnitType                    : Integer;
        NumberWaitUnits                 : Integer;
        WaitStatus                      : Integer;

        WriteFlag                       : Boolean;
        NormalizeBodyFlag               : Completion;
        InitializeTripodFlag            : Completion;
        OptimizeStabilityFlag           : Completion;

        Contact                         : LegSet;
        FootHeight                      : Array6;
        Dummy                           : Array6;

        PresentRadius                   : Array6;
        FootPhase                       : SixStates;
        TransferState                   : SixSubstates;

        SubCommand                      : Char;
        NewTripodDown                   : Boolean;
        SupportTripod                   : TripodGroup;

        FirstTimeP                      : Boolean;      { used in Projection }
        CriticalFeet                    : LegSet;
        StabilityFlag                   : Boolean;
        ForceThreshold                  : Real;

BEGIN  { Main }

{ initialize graphics display }
Initz;

IntLimitText;
IntCriticalText;
IntUnstableText;

FirstTimeP := True;     { used in Projection routine }

CalculateCircle( ReachRadius );

Initialize(     ActualBodyTransform,
                DesiredBodyTransform,
                ActualFootPosition,
                DesiredFootPosition,
                ActualForce,
                DesiredForce,
                Tactilian,
                ForceSet,
                FootContact );

SupportTripod := Left;          { really belongs in TripodInitialize }

FetchInitialize;
NormInitialize;
LegInitialize( ActualForce );


WaitUnitType := 0;      { Select clock ticks for wait units }
NumberWaitUnits := 1;   { Wait for one clock tick }

OnMotors;

LastTime := Time;
CurrentVehicleMode := 'H';
CommandedVehicleMode := 'H';
WriteFlag := FALSE;
BodyServoAngularRates := Zero;
Yaw := 0.0;
```

```
REPEAT { UNTIL CommandedVehicleMode = 'X'}

        CurrentTime := Time;
        dt := ( CurrentTime - LastTime ) * 3600.0;
        LastTime := CurrentTime;

        FetchCommand(   CommandedVehicleMode,
                        LegNumber,
                        JoystickDeflection,
                        WriteFlag,
                        SubCommand,
                        NewTripodDown,
                        SupportTripod,
                        CriticalFeet );


        IF WriteFlag
           THEN Writeln('dt=', dt );


        {**************************************************************************}
        {*              OPERATIONAL MODE SEQUENCER SECTION                        *}
        {**************************************************************************}

        CASE CurrentVehicleMode OF

           'L': CurrentVehicleMode := CommandedVehicleMode;

           'A': CurrentVehicleMode := CommandedVehicleMode;

           'F': CurrentVehicleMode := CommandedVehicleMode;

           'N': IF NormalizeBodyFlag = Done
                    THEN
                        BEGIN
                        CurrentVehicleMode := 'H';
                        CommandedVehicleMode := 'H';
                        END;

           'I': IF InitializeTripodFlag = Done
                    THEN
                        BEGIN
                        CurrentVehicleMode := 'H';
                        CommandedVehicleMode := 'H';
                        END;


           'D': CurrentVehicleMode := CommandedVehicleMode;

           'M': CurrentVehicleMode := CommandedVehicleMode;

           'O': IF OptimizeStabilityFlag = Done
                    THEN
                        BEGIN
                        CurrentVehicleMode := 'H';
                        CommandedVehicleMode := 'H';
                        END;

           'H': CurrentVehicleMode := CommandedVehicleMode;

            END; { CASE CurrentVehicleMode }
```

```
{*****************************************************************}
{*              REFERENCE GENERATION SECTION                     *}
{*****************************************************************}

PresentBodyTransform := DesiredBodyTransform;  {for CheckBodyStability}

CASE CurrentVehicleMode OF

   'L': LinearMotion(   JoyStickDeflection,
                        DesiredFootPosition,
                        FootContact,
                        dt,
                        FootPhase,
                        PlannedFootRate,
                        FootRateType,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        DesiredBodyTransform );

   'A': AngularMotion(  JoyStickDeflection,
                        DesiredFootPosition,
                        FootContact,
                        dt,
                        FootPhase,
                        PlannedFootRate,
                        FootRateType,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        DesiredBodyTransform );

   'F': LegMotion(      LegNumber,
                        JoyStickDeflection,
                        dt,
                        ActualFootPosition,
                        DesiredFootPosition,
                        FootContact[ LegNumber ],
                        FootPhase,
                        ZeroForce[ LegNumber ],
                        PlannedFootRate,
                        FootRateType,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        DesiredBodyTransform );

   'N': NormalizeBody(  NormalizeBodyFlag,
                        FootContact,
                        DesiredFootPosition,
                        dt,
                        ZeroForce,
                        PlannedFootRate,
                        FootRateType,
                        FootPhase,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        DesiredBodyTransform );
```

```
'I': TripodInitialize( InitializeTripodFlag,
                       NormalizeBodyFlag,
                       DesiredFootPosition,
                       dt,
                       FootPhase,
                       TransferState,
                       PresentRadius,
                       ZeroForce,
                       PlannedFootRate,
                       FootRateType,
                       DesiredBodyLinearRates,
                       DesiredBodyAngularRates,
                       DesiredBodyTransform );

'D': DualTripod(       JoystickDeflection,
                       FootContact,
                       DesiredFootPosition,
                       dt,
                       FootPhase,
                       TransferState,
                       PresentRadius,
                       ZeroForce,
                       PlannedFootRate,
                       FootRateType,
                       DesiredBodyLinearRates,
                       DesiredBodyAngularRates,
                       DesiredBodyTransform );

'M': ManualTripod(     JoystickDeflection,
                       SubCommand,
                       NewTripodDown,
                       SupportTripod,
                       dt,
                       ActualFootPosition,
                       DesiredFootPosition,
                       FootPhase,
                       TransferState,
                       FootContact,
                       PresentRadius,
                       ZeroForce,
                       PlannedFootRate,
                       FootRateType,
                       DesiredBodyLinearRates,
                       DesiredBodyAngularRates,
                       DesiredBodyTransform  );

'O': OptimizeStability( OptimizeStabilityFlag,
                       FootPhase,
                       DesiredFootPosition,
                       Pitch,
                       Roll,
                       dt,
                       LastTime,
                       PlannedFootRate,
                       FootRateType,
                       DesiredBodyLinearRates,
                       DesiredBodyAngularRates,
                       DesiredBodyTransform );

'H': HaltVehicle(      dt,
                       PlannedFootRate,
                       FootRateType,
                       DesiredBodyLinearRates,
                       DesiredBodyAngularRates,
                       DesiredBodyTransform );
```

```
    'X': ;

    END; { CASE CurrentVehicleMode }


CheckBodyStability(     FootPhase,
                        Pitch,
                        Roll,
                        DesiredFootPosition,
                        PresentBodyTransform,
                        dt,
                        PlannedFootRate,
                        FootRateType,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        DesiredBodyTransform );


OrthoNorm(              DesiredBodyTransform.Rotation,
                        DesiredBodyTransform.Rotation );


{**************************************************************************}
{*                      SERVO ROUTINE SECTION                             *}
{**************************************************************************}


GyroFeedBack(   Pitch,
                Roll  );


{ Integrate commanded yaw rate to simulate yaw gyro feedback }
MatrixMult(     BodyServoAngularRatesEarth,
                ActualBodyTransform.Rotation,
                BodyServoAngularRates );
Yaw := Yaw + BodyServoAngularRatesEarth.Z * dt;


GetOrientation( Yaw,
                Pitch,
                Roll,
                ActualBodyTransform.Rotation );


IF WriteFlag THEN
        BEGIN
        WriteLn( 'Yaw = ', Yaw:10:4 );
        WriteLn( 'Pitch = ', Pitch:10:4 );
        WriteLn( 'Roll = ', Roll:10:4 );

        {* WriteMatrix( ActualBodyTransform.Rotation, 'ActRotat  ' );*}
        WriteLn;
        END;

BodyServo(      DesiredBodyTransform.Rotation,
                DesiredBodyTransform.Rotation, { for ActualTransform }
                DesiredBodyAngularRates,
                DesiredBodyAngularRates, { for ActualBodyAngularRates }
                BodyServoAngularRates );
```

```
FOR foot := 1 TO 6 DO   { Convert body rates to foot rates }
        BEGIN
        CrossProduct(   BodyServoFootRate[ foot ],
                        DesiredFootPosition[ foot ],
                        BodyServoAngularRates );

        VectSub(        BodyServoFootRate[ foot ],
                        BodyServoFootRate[ foot ],
                        DesiredBodyLinearRates );
        END;

MixFootRates(   PlannedFootRate,
                BodyServoFootRate,
                FootRateType,
                dt,
                DesiredFootRate,
                DesiredFootPosition );


PositionFeedback( ActualJointAngles );

ForceFeedback(  ActualFootForce );

FOR foot := 1 to 6 DO { subtract force offsets from actual force }
    VectSub(    ActualFootForce[ foot ],
                ActualFootForce[ foot ],
                ZeroForce[ foot ] );


Cartesian(      ActualJointAngles,
                ActualFootForce,
                ActualFootPosition,
                ActualForce,
                InverseJacobian );

IF WriteFlag THEN Write( 'Actual Forces : ' );
IF ( CurrentVehicleMode = 'D' ) OR ( CurrentVehicleMode = 'M' )
    THEN ForceThreshold := 25.0  { pounds }
    ELSE ForceThreshold := 10.0; { pounds }
FOR foot := 1 TO 6 DO
    BEGIN
    IF -ActualForce[ foot ].Z >= ForceThreshold
        THEN
           FootContact[ foot ] := TRUE
        ELSE
           FootContact[ foot ] := FALSE;

    IF WriteFlag THEN Write( -ActualForce[ foot ].Z:10:2 );
    END;

IF WriteFlag THEN Writeln;


LegServo(       DesiredFootPosition,
                DesiredForce,
                DesiredFootRate,
                ActualFootPosition,
                ActualForce,
                Tactilian,
                ForceSet,
                dt,
                LegServoFootRate );
```

```
        FOR foot := 1 TO 6 DO  { convert foot rates to joint rates }
                MatrixMult(     CommandJointRate[ foot ],
                                InverseJacobian[ foot ],
                                LegServoFootRate[ foot ] );

        RateFeedback(   ActualJointRates );

        JointServo(     CommandJointRate,
                        ActualJointRates,
                        OutVoltage );

        OutputVolt(     OutVoltage );

        Projection(     FootPhase,
                        DesiredFootPosition,
                        FirstTimeP,
                        CurrentVehicleMode,
                        LegNumber,
                        Pitch,
                        Roll,
                        CriticalFeet,
                        StabilityFlag );


        {*************************************************************************}
        {*                    END OF SERVO ROUTINES                              *}
        {*************************************************************************}


        WriteFlag := FALSE;

        Wait(           NumberWaitUnits,
                        WaitUnitType,
                        WaitStatus );

        IF WaitStatus <> 1 THEN Writeln( 'WaitStatus =', WaitStatus );

UNTIL CommandedVehicleMode = 'X';


FOR foot := 1 TO 6 DO
    OutVoltage[ foot ] := Zero;

OutputVolt( OutVoltage );

OffMotors;
OffElectronics;

END.
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;       { Type declarations }
%INCLUDE DMEXTERN9;      { Extrernal procedure declarations }
%INCLUDE EXTPROJ3;       { External procedures and types for projection }
%INCLUDE EXTPROJHP;      { HP NonPascal procedure declarations }

{*****************************************************************}
{               File Name: DMHUMAN1T.PAS                          }
{                                                                 }
{ Function: Human interface routine. Operation mode is            }
{           determined by keyboard input. Requested vehicle       }
{           velocities are read from the joystick.                }
{                                                                 }
{*****************************************************************}


{***   STATIC VARIABLE DECLARATION     ***}

VAR     FirstTime       : Boolean;
        CriticalText    : ARRAY[1..8] of Char;
        DisplayFlag     : Boolean;


PROCEDURE IntCriticalText;

VAR     Cfile, Csize, Cchar     : Integer;
        Xtext, Ytext            : Integer;
        I                       : Integer;

BEGIN { IntCriticalFlag }
Cfile := 13;            { file number }
Csize := 1;             { size of character, 0 is smallest }
Cchar := 8;             { number of characters }
Xtext := 432;
Ytext := 252;

CriticalText[1]  := 'C';
CriticalText[2]  := 'R';
CriticalText[3]  := 'I';
CriticalText[4]  := 'T';
CriticalText[5]  := 'I';
CriticalText[6]  := 'C';
CriticalText[7]  := 'A';
CriticalText[8]  := 'L';

TEXT( Xtext, Ytext, Csize, Cfile, Cchar, CriticalText[1] );

BLANK( Cfile );
DisplayFlag := False;

END; { IntCriticalText }



PROCEDURE FetchInitialize;
BEGIN
  { set FirstTime flag }
  FirstTime := TRUE;
END;
```

```
PROCEDURE FetchCommand{( VAR CommandedVehicleMode       : Char;
                         VAR LegNumber                  : Integer;
                         VAR JoystickDeflection         : Array3;
                         VAR WriteFlag                  : Boolean;
                         VAR SubCommand                 : Char;
                         VAR NewTripodDown              : Boolean;
                         VAR SuportTripod               : TripodGroup;
                             CriticalFeet               : LegSet )};

VAR      Letter          : Char;
         foot            : Integer;
         Cfile           : Integer;

BEGIN { FetchCommand }

Cfile := 13;             { file number }

JoyStick( JoystickDeflection, 1.0 );


(*********************************}
{***  DETERMINE OPERATING MODE  ***}
(*********************************}

KeyCk( Letter );  { Test for keyboard input }

IF ORD( Letter ) <> 0
    THEN  { an input has occurred }
      BEGIN
        IF DisplayFlag
          THEN
              BEGIN
              BLANK( Cfile );
              DisplayFlag := False;
              END;

        CASE Letter OF

            'L', 'l': CommandedVehicleMode := 'L';  { Linear rate input mode }

            'A', 'a': CommandedVehicleMode := 'A';  { Angular rate input mode }

            'N', 'n': CommandedVehicleMode := 'N';  { Normalization mode }

            'I', 'i': CommandedVehicleMode := 'I';  { Initialize tripod mode }

            'D', 'd': CommandedVehicleMode := 'D';  { DualTripod mode }

            'T', 't': BEGIN
                      CommandedVehicleMode := 'M';  { ManualTripod mode }
                      SubCommand := 'T';            { TransferPhase tripod }
                      { set FirstTime flag }
                      FirstTime := TRUE;
                      END;

            'S', 's': BEGIN
                      CommandedVehicleMode := 'M';  { ManualTripod mode }
                      SubCommand := 'S';          {Switch SupportPhase tripod}
                      IF FirstTime
                        THEN
                          BEGIN
                          NewTripodDown := FALSE;
                          { switch support tripod }
```

```
                    IF SupportTripod = Left
                      THEN SupportTripod := Right
                      ELSE SupportTripod := Left;
                    { reset FirstTime flag }
                    FirstTime := FALSE;
                    END;
                  END;

          '0', 'o': CommandedVehicleMode := '0';  { OptimizeStability mode }

          'H', 'h': CommandedVehicleMode := 'H';  { Halt mode }

          'X', 'x': CommandedVehicleMode := 'X';  { Exit program execution }

          'W', 'w': WriteFlag := NOT WriteFlag;   { Toggle write flag }

          '1'     : BEGIN
                    foot := 1;
                    IF foot IN CriticalFeet
                      THEN { inhibit operator from moving this foot }
                        BEGIN
                        CommandedVehicleMode := 'H';  { Halt mode }
                      {*Writeln(' Leg 1 is a Critical leg ');*}
                        UBLANK( Cfile );
                        DisplayFlag := True;
                        END
                      ELSE
                        BEGIN
                        CommandedVehicleMode := 'F';  { Foot control mode }
                        LegNumber := 1;
                        END;
                    END;

          '2'     : BEGIN
                    CommandedVehicleMode := 'F';  { Foot control mode }
                    LegNumber := 2;
                    END;

          '3'     : BEGIN
                    CommandedVehicleMode := 'F';  { Foot control mode }
                    LegNumber := 3;
                    END;

          '4'     : BEGIN
                    CommandedVehicleMode := 'F';  { Foot control mode }
                    LegNumber := 4;
                    END;

          '5'     : BEGIN
                    CommandedVehicleMode := 'F';  { Foot control mode }
                    LegNumber := 5;
                    END;

          '6'     : BEGIN
                    CommandedVehicleMode := 'F';  { Foot control mode }
                    LegNumber := 6;
                    END;

          ' ':        ;              { Disregard blank characters }

        ELSE  WRITELN( 'ILLEGAL COMMAND' );

        END; { CASE Letter }

    END; { IF ORD }

END; { FetchCommand }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST41;     { Constant declaration }
%INCLUDE DMEXTERN9;     { External procedure declarations }

{*****************************************************************}
{                File Name: DMGUIDEKL.PAS                         }
{*****************************************************************}


PROCEDURE LinearMotion;

CONST   MaxBodyLinearRate = 2.0;  { inches per second }

VAR     foot            : Integer;
        BodyLimitFlag   : Boolean;

BEGIN  { LinearMotion }

FOR foot := 1 To 6 DO
  BEGIN
  PlannedFootRate[ foot ] := Zero;

  IF FootContact[ foot ]
    THEN { foot is on ground }
        FootPhase[ foot ] := Support;

  IF FootPhase[ foot ] = Support
    THEN FootRateType[ foot ] := Relative
    ELSE FootRateType[ foot ] := Absolute;

  END;   { DO }

DesiredBodyLinearRates.X :=  JoystickDeflection[1] * MaxBodyLinearRate;
DesiredBodyLinearRates.Y := -JoystickDeflection[0] * MaxBodyLinearRate;
DesiredBodyLinearRates.Z :=  JoystickDeflection[2] * MaxBodyLinearRate;

DesiredBodyAngularRates :=  Zero;

BodyLimit(  FootPhase,
            DesiredFootPosition,
            dt,
            PlannedFootRate,
            FootRateType,
            DesiredBodyLinearRates,
            DesiredBodyAngularRates,
            DesiredBodyTransform,
            BodyLimitFlag );

END; { LinearMotion }


{*****************************************************************************}
{*****************************************************************************}


PROCEDURE AngularMotion;

CONST   MaxBodyAngularRate = 0.1;  { radians per second }

VAR     foot            : Integer;
        BodyLimitFlag   : Boolean;
```

```
BEGIN  { AngularMotion }

FOR foot := 1 To 6 DO
  BEGIN
  PlannedFootRate[ foot ] := Zero;

  IF FootContact[ foot ]
    THEN { foot is on ground }
        FootPhase[ foot ] := Support;

  IF FootPhase[ foot ] = Support
    THEN FootRateType[ foot ] := Relative
    ELSE FootRateType[ foot ] := Absolute;

  END;   { DO }

DesiredBodyAngularRates.X := JoystickDeflection[0] * MaxBodyAngularRate;
DesiredBodyAngularRates.Y := JoystickDeflection[1] * MaxBodyAngularRate;
DesiredBodyAngularRates.Z := JoystickDeflection[2] * MaxBodyAngularRate;

DesiredBodyLinearRates :=  Zero;

BodyLimit(  FootPhase,
            DesiredFootPosition,
            dt,
            PlannedFootRate,
            FootRateType,
            DesiredBodyLinearRates,
            DesiredBodyAngularRates,
            DesiredBodyTransform,
            BodyLimitFlag );

END; { AngularMotion }



{*****************************************************************************}
{*****************************************************************************}


PROCEDURE LegMotion;

CONST   MaxFootRate = 6.0;  { inches per second }

VAR     foot                    : Integer;
        CurrentVehicleMode      : Char;

BEGIN  { LegMotion }

DesiredBodyLinearRates := Zero;
DesiredBodyAngularRates := Zero;

{ DesiredBodyTransform := DesiredBodyTransform }

FOR foot := 1 TO 6 DO
  IF FootPhase[ foot ] = Support
    THEN FootRateType[ foot ] := Relative
    ELSE FootRateType[ foot ] := Absolute;

FOR foot := 1 To 6 DO
    IF foot = LegNumber
        THEN
            BEGIN
            PlannedFootRate[ foot ].X :=  JoystickDeflection[1] * MaxFootRate;
            PlannedFootRate[ foot ].Y := -JoystickDeflection[0] * MaxFootRate;
            PlannedFootRate[ foot ].Z :=  JoystickDeflection[2] * MaxFootRate;
```

```
            CurrentVehicleMode := 'F';
            KinematicLimits(    LegNumber,
                                dt,
                                CurrentVehicleMode,
                                ActualFootPosition,
                                DesiredFootPosition,
                                FootContact,
                                PlannedFootRate,
                                FootRateType,
                                DesiredBodyLinearRates,
                                DesiredBodyAngularRates,
                                DesiredBodyTransform,
                                FootPhase,
                                ZeroForce );

            END
        ELSE
            PlannedFootRate[ foot ] := Zero;
{ END of DO }

END; { LegMotion }


{*************************************************************************}
{*************************************************************************}


PROCEDURE HaltVehicle;

VAR     foot    : Integer;

BEGIN  { HaltVehicle }

FOR foot := 1 To 6 DO
    BEGIN
    PlannedFootRate[ foot ] := Zero;
    FootRateType[ foot ] := Relative;
    END;

DesiredBodyLinearRates := Zero;
DesiredBodyAngularRates := Zero;

{ DesiredBodyTransform := DesiredBodyTransform }

END; { HaltVehicle }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declarations }
%INCLUDE DMEXTERN9;     { External Procedure declarations }
%INCLUDE EXTPROJ3;      { External declarations for projection }
%INCLUDE EXTPROJHP;     { HP NonPascalprocedure declarations }

{*************************************************************}
{               File Name: BODLIMOT.PAS                       }
{ Function: Limit body motion to insure that no legs          }
{               are extended beyond reachable limits.         }
{*************************************************************}


{***     STATIC VARIABLE DECLARATION     ***}

VAR     LimitText       : ARRAY[1..10] of Char;
        DisplayFlag     : Boolean;

PROCEDURE IntLimitText;

VAR     Lfile, Lsize, Lchar     : Integer;
        Xtext, Ytext            : Integer;
        I                       : Integer;

BEGIN { IntLimitFlag }
Lfile := 11;            { file number }
Lsize := 1;             { size of character, 0 is smallest }
Lchar := 10;            { number of characters }
Xtext := 392;
Ytext := 300;

LimitText[1]  := ' ';
LimitText[2]  := ' ';
LimitText[3]  := 'A';
LimitText[4]  := 'T';
LimitText[5]  := ' ';
LimitText[6]  := 'L';
LimitText[7]  := 'I';
LimitText[8]  := 'M';
LimitText[9]  := 'I';
LimitText[10] := 'T';

TEXT( Xtext, Ytext, Lsize, Lfile, Lchar, LimitText[1] );

BLANK( Lfile );
DisplayFlag := False;

END; { IntLimitText }


PROCEDURE BodyLimit(( FootPhase                 : SixStates;
                      DesiredFootPosition       : SixVectors;
                      dt                        : Real;
                  VAR PlannedFootRate           : SixVectors;
                  VAR FootRateType              : SixRateTypes;
                  VAR DesiredBodyLinearRates    : Vector;
                  VAR DesiredBodyAngularRates   : Vector;
                  VAR DesiredBodyTransform      : Homogeneous;
                  VAR BodyLimitFlag             : Boolean )};
```

```
LABEL   30;

CONST   AbsoluteMinFootHeight = -27.0;  { inches }
        AbsoluteMaxFootHeight = -7.0;   { inches }

VAR     foot                        : Integer;
        PlannedBodyTransform        : Homogeneous;
        DesiredFootRate             : SixVectors;
        XB, YB                      : Real;
        PresentRadius               : Real;
        Lfile, Lsize, Ldec          : Integer;
        Xtext, Ytext                : Integer;
        Xfoot                       : Real;

BEGIN   { BodyLimit }

Lfile := 11;            { file number }
Lsize := 1;             { size of character, 0 is smallest }
Ldec  := 0;             { number of places after decimal point }
Xtext := 432;
Ytext := 300;

BodyLimitFlag := False;

{ save present DesiredBodyTransform }
PlannedBodyTransform := DesiredBodyTransform;

{ predict updated BodyTransform }
TransIntegrate( PlannedBodyTransform,
                DesiredBodyLinearRates,
                DesiredBodyAngularRates,
                dt    );

OrthoNorm( PlannedBodyTransform.Rotation,
           PlannedBodyTransform.Rotation );
{ calculate the PredictedPosition of each foot }
{ NOTE: update DesiredFootPosition value, but not returned to calling routine }
PredictedPosition( PlannedFootRate,
                   FootRateType,
                   DesiredBodyLinearRates,
                   DesiredBodyAngularRates,
                   PlannedBodyTransform,
                   dt,
                   DesiredFootRate,
                   DesiredFootPosition );

FOR foot := 1 TO 6 DO
   BEGIN
   IF FootPhase[ foot ] = Support
     THEN
        BEGIN
        XB := DesiredFootPosition[ foot ].X - EquilibriumPosition[ foot ].X;
        YB := DesiredFootPosition[ foot ].Y - EquilibriumPosition[ foot ].Y;
        PresentRadius := SQRT( XB*XB + YB*YB );

        IF PresentRadius > ReachRadius
           THEN { foot is outside of reachable area }
                BEGIN
                HaltVehicle( dt,
                             PlannedFootRate,
                             FootRateType,
                             DesiredBodyLinearRates,
                             DesiredBodyAngularRates,
                             DesiredBodyTransform );
                BodyLimitFlag := True;
                Xfoot := foot;
                GoTo 30;
                END;
```

```
        IF ( DesiredFootPosition[ foot ].Z <= AbsoluteMinFootHeight )
          AND ( DesiredFootRate[ foot ].Z < 0.0 )
            THEN { a support leg is stretched to its limit }
                 BEGIN
                 HaltVehicle( dt,
                              PlannedFootRate,
                              FootRateType,
                              DesiredBodyLinearRates,
                              DesiredBodyAngularRates,
                              DesiredBodyTransform );
                 BodyLimitFlag := True;
                 Xfoot := foot;
                 GoTo 30;
                 END;

        IF ( DesiredFootPosition[ foot ].Z >= AbsoluteMaxFootHeight )
          AND ( DesiredFootRate[ foot ].Z > 0.0 )
            THEN { a support leg is squashed to its limit }
                 BEGIN
                 HaltVehicle( dt,
                              PlannedFootRate,
                              FootRateType,
                              DesiredBodyLinearRates,
                              DesiredBodyAngularRates,
                              DesiredBodyTransform );
                 Xfoot := foot;
                 BodyLimitFlag := True;
                 GoTo 30;
                 END;
        END;
   END; { DO }

DesiredBodyTransform := PlannedBodyTransform;

30: IF BodyLimitFlag
      THEN
        BEGIN
{       NUMBER( Xtext, Ytext, Lsize, Lfile, Ldec, Xfoot );      }
        UBLANK( Lfile );
        DisplayFlag := True;
        END
      ELSE
        IF DisplayFlag
          THEN
            BEGIN
            BLANK( Lfile );
            DisplayFlag := False;
            END;

END;  { BodyLimit }



PROCEDURE PredictedPosition[( PlannedFootRate           : SixVectors;
                              FootRateType              : SixRateTypes;
                              DesiredBodyLinearRates    : Vector;
                              DesiredBodyAngularRates   : Vector;
                              DesiredBodyTransform      : Homogeneous;
                              dt                        : Real;
                          VAR PredictedFootRate         : SixVectors;
                          VAR PredictedFootPosition     : SixVectors )];
```

```
{* Function: Calculate the predicted position and predicted rate        *}
{*              of each foot, based on present value of                 *}
{*              PredictedFootPosition and the planned commands.         *}

VAR     foot                    : Integer;
        BodyServoAngularRates   : Vector;
        BodyServoFootRate       : SixVectors;

BEGIN { PredictedPosition }

BodyServo( DesiredBodyTransform.Rotation,
           DesiredBodyTransform.Rotation,       { for ActualTransform }
           DesiredBodyAngularRates,
           DesiredBodyAngularRates,     { for ActualBodyAngularRates }
           BodyServoAngularRates );

FOR foot := 1 TO 6 DO           { convert body rates to foot rates }
        BEGIN
        CrossProduct( BodyServoFootRate[ foot ],
                      PredictedFootPosition[ foot ],
                      BodyServoAngularRates );

        VectSub( BodyServoFootRate[ foot ],
                 BodyServoFootRate[ foot ],
                 DesiredBodyLinearRates );
        END;

MixFootRates( PlannedFootRate,
              BodyServoFootRate,
              FootRateType,
              dt,
              PredictedFootRate,
              PredictedFootPosition );

END;  { PredictedPosition }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMEXTERN9;     { Extrernal procedure declarations }
%INCLUDE EXTPROJ3;      { External procedures and types for projection }
%INCLUDE EXTPROJHP;     { HP NonPascal procedure declarations }

{*************************************************************}
{*              FileName: BODSTABLT.PAS                      *}
{* Function: Stops body before it moves to                   *}
{*              an unstable position.                        *}
{*************************************************************}


{***    STATIC VARIABLE DECLARATION     ***}

VAR     UnstableText    : ARRAY[1..8] of Char;
        DisplayFlag     : Boolean;

PROCEDURE IntUnstableText;

VAR     Ufile, Usize, Uchar     : Integer;
        Xtext, Ytext            : Integer;
        I                       : Integer;

BEGIN { IntUnstableFlag }
Ufile := 12;            { file number }
Usize := 1;             { size of character, 0 is smallest }
Uchar := 10;            { number of characters }
Xtext := 432;
Ytext := 276;

UnstableText[1]  := 'U';
UnstableText[2]  := 'N';
UnstableText[3]  := 'S';
UnstableText[4]  := 'T';
UnstableText[5]  := 'A';
UnstableText[6]  := 'B';
UnstableText[7]  := 'L';
UnstableText[8]  := 'E';

TEXT( Xtext, Ytext, Usize, Ufile, Uchar, UnstableText[1] );

BLANK( Ufile );
DisplayFlag := False;

END; { IntUnstableText }



PROCEDURE CheckBodyStability(( FootPhase                : SixStates;
                               Pitch                    : Real;
                               Roll                     : Real;
                               DesiredFootPosition      : SixVectors;
                               PresentBodyTransform     : Homogeneous;
                               dt                       : Real;
                           VAR PlannedFootRate          : SixVectors;
                           VAR FootRateType             : SixRateTypes;
                           VAR DesiredBodyLinearRates   : Vector;
                           VAR DesiredBodyAngularRates  : Vector;
                           VAR DesiredBodyTransform     : Homogeneous ));
```

```
VAR     DesiredFootRate                 : SixVectors;
        ZeroYaw                         : Real;
        VerticalRotationTransform       : Matrix;
        SupportFeet, PolygonFeet        : LegSet;
        foot                            : Integer;
        ProjectedFootPosition           : SixVectors;
        ProjectedCG                     : Vector;
        StabilityFlag                   : Boolean;
        Ufile                           : Integer;

BEGIN   { CheckBodyStability }

Ufile := 12;            { file number }

{NOTE: Update DesiredFootPosition value, but not returned to calling routine}
PredictedPosition( PlannedFootRate,
                   FootRateType,
                   DesiredBodyLinearRates,
                   DesiredBodyAngularRates,
                   DesiredBodyTransform,
                   2.0*dt,
                   DesiredFootRate,
                   DesiredFootPosition );
ZeroYaw := 0.0;
GetOrientation( ZeroYaw,
                Pitch,
                Roll,
                VerticalRotationTransform );

{ Transform location of each foot to "Earth Based" coordinates }
SupportFeet := [];      { empty set }
FOR foot := 1 TO 6 DO
  BEGIN
  MatrixMult( ProjectedFootPosition[ foot ],
              VerticalRotationTransform,
              DesiredFootPosition[ foot ] );
  IF ( FootPhase[ foot ] = Support )
    THEN  SupportFeet := SupportFeet + [ foot ];
  END; { DO }

{ Assuming initial location of Center of Gravity is at (0,0,0) then }
{  Projected Center of Gravity in "Earth Based" coord is:           }
ProjectedCG.X := 0.0;
ProjectedCG.Y := 0.0;
ProjectedCG.Z := 0.0;

CheckRedundantFeet( ProjectedFootPosition,
                    SupportFeet,
                    PolygonFeet );
```

```
CheckPolygonStability( ProjectedFootPosition,
                       PolygonFeet,
                       ProjectedCG,
                       StabilityFlag );
IF NOT StabilityFlag
  THEN
    BEGIN
    DesiredBodyTransform := PresentBodyTransform;
    HaltVehicle( dt,
                 PlannedFootRate,
                 FootRateType,
                 DesiredBodyLinearRates,
                 DesiredBodyAngularRates,
                 DesiredBodyTransform );
    UBLANK( Ufile );
    DisplayFlag := True;
    END
  ELSE
    IF DisplayFlag
      THEN
        BEGIN
        BLANK( Ufile );
        DisplayFlag := False;
        END;

END;  { CheckBodyStability }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declarations }
%INCLUDE DMEXTERN9;     { External procedure declarations }

{*****************************************************************}
{              File Name: OSUKINLI5.PAS                           }
{                                                                 }
{ Function: To check if a leg is within its kinematic limits,    }
{           and to modify the planned foot rates if necessary.    }
{ Revised: 12-Dec-83 Add automatic body regulation.               }
{*****************************************************************}



PROCEDURE UpDownLimit(  LegDirection            : Direction;
                        ActualFootPosition      : SixVectors;
                        dt                      : Real;
                    VAR PlannedFootRate         : Vector;
                    VAR FootRateType            : SixRateTypes;
                        FootPhase               : SixStates;
                    VAR DesiredBodyLinearRates  : Vector;
                        DesiredBodyAngularRates : Vector;
                    VAR DesiredBodyTransform    : Homogeneous );

CONST   AbsoluteMinFootHeight = -27.0;  { inches }
        AbsoluteMaxFootHeight = -7.0;   { inches }

VAR     footI           : Integer;
        LegLimitFlag    : Boolean;

BEGIN   { UpDownLimit }

LegLimitFlag := False;          { initialize flag }

FOR footI := 1 TO 6 DO
  IF FootPhase[ footI ] = Support
    THEN  { foot is on ground }
        BEGIN
        { leave FootRateType[ footI ] = Relative }

        IF ( LegDirection = Up ) AND
                ( ActualFootPosition[ footI ].Z <= AbsoluteMinFootHeight )
          THEN  { a support leg is stretched to its limit }
                LegLimitFlag := True;

        IF ( LegDirection = Down ) AND
                ( ActualFootPosition[ footI ].Z >= AbsoluteMaxFootHeight )
          THEN  { a support leg is squashed to its limit }
                LegLimitFlag := True;

        END
    ELSE  { foot is not on ground }
        FootRateType[ footI ] := Absolute;

IF LegLimitFlag
  THEN  { stop body and leg movement }
    PlannedFootRate.Z := 0.0
    { leave DesiredBodyLinearRates := Zero }
  ELSE
    BEGIN
```

```
    { move body in Z-direction to accomodate desired leg movement }
    DesiredBodyLinearRates.Z := (PlannedFootRate.Z) / 3.0;
    { stop direct movement of the selected leg }
    PlannedFootRate.Z := 0.0;
    TransIntegrate( DesiredBodyTransform,
                    DesiredBodyLinearRates,
                    DesiredBodyAngularRates,
                    dt );
    END;

END;    { UpDownLimit }



PROCEDURE XYLimit(  foot                          : Integer;
                    dt                            : Real;
                    DesiredFootPosition           : SixVectors;
                VAR PlannedFootRate               : SixVectors;
                VAR FootRateType                  : SixRateTypes;
                VAR DesiredBodyLinearRates        : Vector;
                    DesiredBodyAngularRates       : Vector;
                VAR DesiredBodyTransform          : Homogeneous );

LABEL   20;

VAR     PresentRadius           : Real;
        XB, YB                  : Real;
        BodyServoAngularRates   : Vector;
        BodyServoFootRate       : SixVectors;
        footI                   : Integer;
        DesiredFootRate         : SixVectors;
        OriginalBodyTransform   : Homogeneous;

BEGIN   { XYLimit }

{ move body in XY direction to accomodate desired leg movement }
DesiredBodyLinearRates.X := (PlannedFootRate[ foot ].X) / 3.0;
DesiredBodyLinearRates.Y := (PlannedFootRate[ foot ].Y) / 3.0;

{ stop direct movement of the selected leg }
PlannedFootRate[ foot ].X := 0.0;
PlannedFootRate[ foot ].Y := 0.0;

FootRateType[ foot ] := Absolute;
OriginalBodyTransform := DesiredBodyTransform;
TransIntegrate( DesiredBodyTransform,
                DesiredBodyLinearRates,
                DesiredBodyAngularRates,
                dt );

{ check if any foot will be moved outside of reachable area }
{ calculate desired location of feet }
BodyServo( DesiredBodyTransform.Rotation,
           DesiredBodyTransform.Rotation,       { for ActualTransform }
           DesiredBodyAngularRates,
           DesiredBodyAngularRates,     { for ActualBodyAngularRates  }
           BodyServoAngularRates );

FOR footI := 1 TO 6 DO    { convert body rates to foot rates }
        BEGIN
        CrossProduct( BodyServoFootRate[ footI ],
                      DesiredFootPosition[ footI ],
                      BodyServoAngularRates     );
```

```
        VectSub( BodyServoFootRate[ footI ],
                 BodyServoFootRate[ footI ],
                 DesiredBodyLinearRates );
        END;

MixFootRates( PlannedFootRate,
              BodyServoFootRate,
              FootRateType,
              dt,
              DesiredFootRate,
              DesiredFootPosition );
{NOTE: update DesiredFootPosition value, but not returned to calling routine }

FOR footI := 1 TO 6 DO
  IF footI <> foot
    THEN
        BEGIN
        { find future location of footI within reachable area }
        XB := DesiredFootPosition[ footI ].X - EquilibriumPosition[ footI ].X;
        YB := DesiredFootPosition[ footI ].Y - EquilibriumPosition[ footI ].Y;
        PresentRadius := SQRT( XB*XB + YB*YB );

        { check if foot is inside reachable area }
        IF PresentRadius > ReachRadius
          THEN
            BEGIN
            { leg outside reachable area; stop movement in X,Y directions }
            DesiredBodyLinearRates := Zero;
            { PlannedFootRate already zero }
            { restore original value of DesiredBodyTransform }
            DesiredBodyTransform := OriginalBodyTransform;
            GoTo 20;
            END;
        END;    { and end of DO }

20:END; { XYLimit }
```

```
PROCEDURE KinematicLimits{( foot                          : Integer;
                            dt                            : Real;
                            CurrentVehicleMode            : Char;
                            ActualFootPosition            : SixVectors;
                            DesiredFootPosition           : SixVectors;
                            FootContact                   : Boolean;
                       VAR  PlannedFootRate               : SixVectors;
                       VAR  FootRateType                  : SixRateTypes;
                       VAR  DesiredBodyLinearRates        : Vector;
                            DesiredBodyAngularRates       : Vector;
                       VAR  DesiredBodyTransform          : Homogeneous;
                       VAR  FootPhase                     : SixStates;
                       VAR  ZeroForce                     : Vector )};

CONST   AbsoluteMinFootHeight = -27.0;  { inches }
        AbsoluteMaxFootHeight = -7.0;   { inches }

VAR     PresentRadius           : Real;
        XB, YB                  : Real;
        BodyServoAngularRates   : Vector;
        BodyServoFootRate       : SixVectors;
        footI                   : Integer;
        TemporaryForce          : SixVectors;
        DesiredFootRate         : SixVectors;
        LegDirection            : Direction;

BEGIN   { KinematicLimits }

IF FootContact
    THEN
        BEGIN { foot is on ground }

        FootPhase[ foot ] := Support;

        { foot is on ground; no x,y motion allowed }
        PlannedFootRate[ foot ].X := 0.0;
        PlannedFootRate[ foot ].Y := 0.0;

        { check z-component of foot rate }
        IF PlannedFootRate[ foot ].Z < 0.0
            THEN
                { stop foot from being lowered further }
                PlannedFootRate[ foot ].Z := 0.0
            ELSE
                BEGIN
                { allow desired movement in z-direction }
                { leave PlannedFootRate[ foot ].Z as is }

                { NOTE: The following IF statement is necessary only to    }
                {       compensate for the drift in the force sensors.     }
                {       The drift may cause a leg which is not on the      }
                {       ground to still appear as if it is on the ground.  }
                {       The following IF will allow the operator to        }
                {       re-calibrate the force sensor if this drift occurs.}

                IF ( ActualFootPosition[ foot ].Z >= AbsoluteMaxFootHeight )
                  THEN { leg being lifted too high }
                    BEGIN
                    ForceFeedback( TemporaryForce );
                    ZeroForce := TemporaryForce[ foot ];

                    { stop foot from being lifted further }
                    PlannedFootRate[ foot ].Z := 0.0;
                    END;
                END;
        END
```

```
ELSE
    BEGIN { foot is not on ground }

    FootPhase[ foot ] := Transfer;

    IF ( PlannedFootRate[ foot ].Z > 0.0 )  AND
            ( ActualFootPosition[ foot ].Z >= AbsoluteMaxFootHeight )
       THEN { leg being lifted too high }
            BEGIN

            { ForceFeedback( TemporaryForce );
            ZeroForce := TemporaryForce[ foot ]; }

            ZeroForce := Zero;

            { stop foot from being lifted further }
            LegDirection := Up;
            IF CurrentVehicleMode = 'M'
              THEN
                PlannedFootRate[ foot ].Z := 0.0
              ELSE
                UpDownLimit( LegDirection,
                             ActualFootPosition,
                             dt,
                             PlannedFootRate[ foot ],
                             FootRateType,
                             FootPhase,
                             DesiredBodyLinearRates,
                             DesiredBodyAngularRates,
                             DesiredBodyTransform );
            END
       ELSE
          IF (PlannedFootRate[ foot ].Z < 0.0 ) AND
                ( ActualFootPosition[ foot ].Z <= AbsoluteMinFootHeight )
           THEN { leg being lowered too far }
             BEGIN
             { stop foot from being lowered further }
             LegDirection := Down;
             IF CurrentVehicleMode = 'M'
               THEN
                  PlannedFootRate[ foot ].Z := 0.0
               ELSE
                  UpDownLimit( LegDirection,
                               ActualFootPosition,
                               dt,
                               PlannedFootRate[ foot ],
                               FootRateType,
                               FootPhase,
                               DesiredBodyLinearRates,
                               DesiredBodyAngularRates,
                               DesiredBodyTransform );
             END;
           { ELSE allow desired movement in z-direction }
           {      leave PlannedFootRate[ foot ].Z as is.}
```

```
        { calculate desired location of feet }
        BodyServo( DesiredBodyTransform.Rotation,
                   DesiredBodyTransform.Rotation,   { for ActualTransform }
                   DesiredBodyAngularRates,
                   DesiredBodyAngularRates,   { for ActualBodyAngularRates }
                   BodyServoAngularRates   );

        FOR footI := 1 TO 6 DO   { convert body rates to foot rates }
                BEGIN
                CrossProduct( BodyServoFootRate[ footI ],
                              DesiredFootPosition[ footI ],
                              BodyServoAngularRates   );

                VectSub( BodyServoFootRate[ footI ],
                         BodyServoFootRate[ footI ],
                         DesiredBodyLinearRates     );

                END;

        MixFootRates(   PlannedFootRate,
                        BodyServoFootRate,
                        FootRateType,
                        dt,
                        DesiredFootRate,
                        DesiredFootPosition  );
{NOTE: update DesiredFootPosition value, but not returned to calling routine }

        { find future location of foot within reachable area }
        XB := DesiredFootPosition[ foot ].X - EquilibriumPosition[ foot ].X;
        YB := DesiredFootPosition[ foot ].Y - EquilibriumPosition[ foot ].Y;
        PresentRadius := SQRT( XB*XB + YB*YB );

        { check if foot is inside reachable area }
        IF (PresentRadius > ReachRadius) AND (CurrentVehicleMode = 'M')
          THEN { leg outside reachable area; stop movement in x,y direction }
            BEGIN
            PlannedFootRate[ foot ].X := 0.0;
            PlannedFootRate[ foot ].Y := 0.0;
            END;

        IF (PresentRadius > ReachRadius) AND (CurrentVehicleMode <> 'M')
          THEN { leg outside reachable area; use automatic body regulation }
            XYLimit( foot,
                     dt,
                     DesiredFootPosition,
                     PlannedFootRate,
                     FootRateType,
                     DesiredBodyLinearRates,
                     DesiredBodyAngularRates,
                     DesiredBodyTransform );
            { ELSE leg inside, allow desired movement          }
            {      leave PlannedFootRate[ foot ].X and .Y as is }

        END;

END;    { KinematicLimits }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declaration }
%INCLUDE DMEXTERN2;     { External procedure declarations }

{*************************************************************************}
{*                       FILE: OSUNORMAL.PAS                             *}
{* Function: Move body and legs to normalized position.                  *}
{*           All legs normalized to center of reachable area.            *}
{*************************************************************************}


{*** STATIC VARIABLE DECLARATION ***}

VAR     NormalState             : NormalizeMode;                { norm }
        ErectBodyFlag           : Completion;                   { norm }
        Move1Flag               : Completion;                   { norm }
        Move2Flag               : Completion;                   { norm }
        MoveFootFlag            : Array[ 1..6 ] OF Completion;  { norm }
        ElevateBodyFlag         : Completion;                   { norm }
        MoveFootState           : SixSubStates;                 { move }


PROCEDURE NormInitialize;

VAR     foot    : Integer;

BEGIN

NormalState := Ready;

For foot := 1 TO 6 DO
    MoveFootState[ foot ] := Ground;

END; { NormInitialize }



PROCEDURE Erectbody(
        DesiredRotation         : Matrix;
    VAR DesiredBodyAngularRates : Vector;
    VAR ErectBodyFlag           : Completion
              );

CONST   TwoDegree = 0.0349;             { Radians }
        CosineTwoDegree = 0.99939;
        OneDegree = 0.01745;            { Radians }
        CosineOneDegree = 0.99985;
        ErectionRate = 0.05;            { Radians per second }


VAR     TanTheta        : Real;
        Theta           : Real;
        TwoSinTheta     : Real;
        Axis            : Vector;
        VerticalAxis    : Boolean;
        SmallAngle      : Boolean;
```

```
BEGIN  { ErectBody }

{*** COMPUTE BODY ROTATION ANGLE ***}

TanTheta := SQRT(   SQR( DesiredRotation[2].Z - DesiredRotation[3].Y )
                  + SQR( DesiredRotation[3].X - DesiredRotation[1].Z )
                  + SQR( DesiredRotation[1].Y - DesiredRotation[2].X ) )
            /  (   DesiredRotation[1].X
                  + DesiredRotation[2].Y
                  + DesiredRotation[3].Z
                  - 1.0                    );

Theta := ArcTan( TanTheta );

SmallAngle := ABS( Theta ) <= TwoDegree;

IF SmallAngle
    THEN
        BEGIN
        Axis.X := 0.0;
        Axis.Y := 0.0;
        Axis.Z := 1.0;
        END
    ELSE
        BEGIN
        IF Theta < 0.0 THEN Theta := Theta + Pi;

        {*** COMPUTE BODY ROTATION AXIS ***}

        TwoSinTheta := 2.0 * Sin( Theta );

        Axis.X := ( DesiredRotation[2].Z - DesiredRotation[3].Y )
                        / TwoSinTheta;
        Axis.Y := ( DesiredRotation[3].X - DesiredRotation[1].Z )
                        / TwoSinTheta;
        Axis.Z := ( DesiredRotation[1].Y - DesiredRotation[2].X )
                        / TwoSinTheta;

        VectDiv( Axis, Axis, Magnitude(Axis));{ Renormalize for small Theta's }

        END; { ELSE }

    { END IF SmallAngle }


{*** EVALUATE BOUNDARY VALUE CONDITION ***}

VerticalAxis := ABS( Axis.Z ) >= CosineTwoDegree;

IF VerticalAxis OR SmallAngle
    THEN
        ErectBodyFlag := Done

    ELSE
        BEGIN
        ErectBodyFlag := NotDone;

        Axis.Z := 0.0;  { Do not yaw the body during erection }
        VectDiv( Axis, Axis, Magnitude( Axis ) ); { Renormalize }

        { Transform the rotation axis into body coordinates }
        MatrixMult( Axis, DesiredRotation, Axis );
```

```
        {*** COMPUTE DESIRED BODY RATES ***}

        VectMult( DesiredBodyAngularRates, -ErectionRate, Axis );

        END; { Else }

END; { ErectBody }



PROCEDURE MoveFoot(
        foot                    : Integer;
        FootContact             : Boolean;
        DesiredFootPosition     : Vector;
        dt                      : Real;
    VAR ZeroForce               : Vector;
    VAR PlannedFootRate         : Vector;
    VAR FootRateType            : RateType;
    VAR FootPhase               : FootState;    { for one foot }
    VAR MoveFootFlag            : Completion
                );

VAR     DeltaPosition   : Vector;       { Intermediate position vector }
        Distance        : Real;         { Distance to PlacePoint }
        MotionDirection : Vector;       { Unit vector toward PlacePoint }
        TemporaryForce  : SixVectors;

BEGIN { MoveFoot }

{*** EVALUATE BOUNDARY VALUE CONDITIONS ***}

CASE MoveFootState[ foot ] OF

    Ground:     BEGIN
                MoveFootState[ foot ] := LiftOff;
                FootPhase := Transfer;
                MoveFootFlag := NotDone;
                END;

    LiftOff:    IF DesiredFootPosition.Z >= PlacePoint[ foot ].Z
                    THEN { foot is at top of trajectory }
                        BEGIN
                        MoveFootState[ foot ] := Return;

                        ForceFeedback( TemporaryForce );
                        ZeroForce := TemporaryForce[ foot ];

                        VectSub( DeltaPosition,
                                        PlacePoint[foot], DesiredFootPosition );
                        Distance := Magnitude( DeltaPosition );
                        END;

    Return:     BEGIN
                VectSub( DeltaPosition,
                                PlacePoint[foot], DesiredFootPosition );
                Distance := Magnitude( DeltaPosition );

                IF Distance <= MaxError
                    THEN { foot ready for placing }
                        MoveFootState[ foot ] := PlaceDown;
                END;
```

```
    PlaceDown:  IF FootContact
                    THEN
                        BEGIN
                        MoveFootState[ foot ] := Ground;
                        FootPhase := Support;
                        MoveFootFlag := Done;
                        END;

    END; { CASE MoveFootState }


{*** COMPUTE FOOT VELOCITY COMMAND ***}

CASE MoveFootState[ foot ] OF

    Ground:     BEGIN
                PlannedFootRate := Zero;

                FootRateType := Relative;
                END;

    LiftOff:    BEGIN
                PlannedFootRate.X := 0.0;
                PlannedFootRate.Y := 0.0;
                PlannedFootRate.Z := LiftRate;

                FootRateType := Relative;
                END; { LiftOff }

    Return:     BEGIN
                VectDiv( MotionDirection,
                                DeltaPosition, Distance );

                VectMult( PlannedFootRate,
                                ReturnRate, MotionDirection );

                FootRateType := Absolute;

                END; { Return }

    PlaceDown:  BEGIN
                PlannedFootRate.X := 0.0;
                PlannedFootRate.Y := 0.0;
                PlannedFootRate.Z := -PlaceRate;

                FootRateType := Relative;
                END; { PlaceDown }

    END; { CASE MoveFootState }


END; { MoveFoot }
```

```
PROCEDURE ElevateBody(
        DesiredFootPosition      : Sixvectors;
        DesiredBodyHeight        : Real;
    VAR DesiredBodyLinearRates   : Vector;
    VAR ElevateBodyFlag          : Completion
                );

CONST   MaxHeightError = 0.25;   { inches }
        ElevationRate = 2.0;     { inches per second }

VAR     AverageHeight            : Real;
        BodyHeightError          : Real;
        foot                     : Integer;


BEGIN

{*** DETERMINE AVERAGE LEG EXTENSION ***}

AverageHeight := 0.0;
FOR foot := 1 TO 6 DO
    AverageHeight := AverageHeight - DesiredFootPosition[ foot ].Z;
AverageHeight := AverageHeight / 6.0;

BodyHeightError := DesiredBodyHeight - AverageHeight;


{*** EVALUATE BOUNDARY VALUE CONDITION ***}

IF ABS( BodyHeightError ) <= MaxHeightError
    THEN
        ElevateBodyFlag := Done

    ELSE
        BEGIN
        ElevateBodyFlag := NotDone;

        {*** CALCULATE DESIRED LINEAR BODY RATES ***}

        DesiredBodyLinearRates.X := 0.0;
        DesiredBodyLinearRates.Y := 0.0;

        IF BodyHeightError > 0.0
            THEN
                DesiredBodyLinearRates.Z := ElevationRate
            ELSE
                DesiredBodyLinearRates.Z := -ElevationRate;

        END; { Else }


END;  { BodyElevate }
```

```
PROCEDURE NormalizeBody;

CONST   DesiredBodyHeight = 18.0;       { inches }

VAR     foot                    : Integer;

BEGIN  { NormalizeBody }

{*** CALL THE APPROPRIATE MOTION SUBROUTINE ***}

CASE NormalState OF

    Ready:

        BEGIN
        DesiredBodyLinearRates := Zero;
        DesiredBodyAngularRates := Zero;

        FOR foot := 1 to 6 DO
            BEGIN
            PlannedFootRate[ foot ] := Zero;
            FootRateType[ foot ] := Relative;
            END;

        END;


    Erection:

        BEGIN
        ErectBody(      DesiredBodyTransform.Rotation,
                        DesiredBodyAngularRates,
                        ErectBodyFlag );

        DesiredBodyLinearRates := Zero;

        FOR foot := 1 to 6 DO
            BEGIN
            PlannedFootRate[ foot ] := Zero;
            FootRateType[ foot ] := Relative;
            END;

        END;


    Move1:

        BEGIN
        DesiredBodyLinearRates := Zero;
        DesiredBodyAngularRates := Zero;

        FOR foot := 1 TO 6 DO
            IF ( foot IN [ 1, 4, 5 ] ) AND ( MoveFootFlag[ foot ] = NotDone )
                THEN
                    MoveFoot(   foot,
                                FootContact[ foot ],
                                DesiredFootPosition[ foot ],
                                dt,
                                ZeroForce[ foot ],
                                PlannedFootRate[ foot ],
                                FootRateType[ foot ],
                                FootPhase[ foot ],
                                MoveFootFlag[ foot ] )
```

```
                ELSE
                    BEGIN
                    PlannedFootRate[ foot ] := Zero;
                    FootRateType[ foot ] := Relative;
                    END;
        END;


    Move2:

        BEGIN
        DesiredBodyLinearRates := Zero;
        DesiredBodyAngularRates := Zero;

        FOR foot := 1 TO 6 DO
            IF ( foot IN [ 2, 3, 6 ] ) AND ( MoveFootFlag[ foot ] = NotDone )
                THEN
                    MoveFoot(   foot,
                                FootContact[ foot ],
                                DesiredFootPosition[ foot ],
                                dt,
                                ZeroForce[ foot ],
                                PlannedFootRate[ foot ],
                                FootRateType[ foot ],
                                FootPhase[ foot ],
                                MoveFootFlag[ foot ] )
                ELSE
                    BEGIN
                    PlannedFootRate[ foot ] := Zero;
                    FootRateType[ foot ] := Relative;
                    END;
        END;


    Elevation:

        BEGIN
        ElevateBody(    DesiredFootPosition,
                        DesiredBodyHeight,
                        DesiredBodyLinearRates,
                        ElevateBodyFlag );

        DesiredBodyAngularRates := Zero;

        FOR foot := 1 to 6 DO
            BEGIN
            PlannedFootRate[ foot ] := Zero;
            FootRateType[ foot ] := Relative;
            END;

        END;

    END; { CASE NormalState }


{*** COMPUTE NEW BODY TRANSFORM ***}

TransIntegrate( DesiredBodyTransform,
                DesiredBodyLinearRates,
                DesiredBodyAngularRates,
                dt );
```

```
{*** DETERMINE NEW STATE BASED ON MOTION BOUNDARY VALUES ***}

CASE NormalState OF

    Ready:

        BEGIN
        NormalState := Erection;
        NormalizeBodyFlag := NotDone;
        END;


    Erection:

        IF ErectBodyFlag = Done
            THEN
                BEGIN
                NormalState := Move1;
                For foot := 1 To 6 DO
                    MoveFootFlag[ foot ] := NotDone;
                END;

    Move1:

        BEGIN
        Move1Flag := Done;
        FOR foot := 1 to 6 DO
            IF ( foot IN [ 1, 4, 5 ] ) AND ( MoveFootFlag[ foot ] = NotDone )
                THEN
                    Move1Flag := NotDone;

        IF Move1Flag = Done
            THEN
                NormalState := Move2;

        END; { Move1 }


    Move2:

        BEGIN
        Move2Flag := Done;
        FOR foot := 1 to 6 DO
            IF ( foot IN [ 2, 3, 6 ] ) AND ( MoveFootFlag[ foot ] = NotDone )
                THEN
                    Move2Flag := NotDone;

        IF Move2Flag = Done
            THEN
                NormalState := Elevation;

        END; { Move2 }


    Elevation:

        IF ElevateBodyFlag = Done
            THEN
                BEGIN
                NormalState := Ready;
                NormalizeBodyFlag := Done;
                END;


    END; { Case NormalState }

END;  { NormalizeBody }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declarations }
%INCLUDE DMEXTERN2;     { External procedure declarations }

{*****************************************************************}
{               File Name: 0SUINTRI.PAS                           }
{                                                                 }
{ Function: Move legs to initial position for tripod gate.        }
{           Keep legs 1,4,5 on the ground at the center of        }
{           their reachable area. Lift leg 2,3,6 straight above   }
{           their center of reachable area.                       }
{*****************************************************************}

PROCEDURE MoveFootInit (
                        foot                    : Integer;
                        DesiredFootPosition     : Vector;
                        dt                      : Real;
                    VAR ZeroForce               : Vector;
                    VAR PlannedFootRate         : Vector;
                    VAR FootRateType            : RateType;
                    VAR InitFootFlag            : Completion
                        );

VAR     TemporaryForce  : SixVectors;

BEGIN { MoveFootInit }

IF DesiredFootPosition.Z >= MaxFootHeight
    THEN  { foot is at top of trajectory }
        BEGIN
        InitFootFlag := Done;

        ForceFeedback( TemporaryForce );
        ZeroForce := TemporaryForce[ foot ];

        PlannedFootRate := Zero;
        END
    ELSE
        BEGIN
        PlannedFootRate.X := 0.0;
        PlannedFootRate.Y := 0.0;
        PlannedFootRate.Z := LiftRate;
        END;

FootRateType := Relative;

END;  { MoveFootInit }



PROCEDURE TripodInitialize;

VAR     foot            : Integer;
        InitFootFlag    : Array[1..6] OF Completion;

BEGIN  { TripodInitialize }

{ Initialize flags for DualTripod }
DTInitialize;
```

```
{ Check that body has been normalized. }
IF NormalizeBodyFlag = NotDone
     THEN
        BEGIN
        WriteLn( ' Body must first be normalized.' );
        Writeln( ' Enter  N  to normalize body. '  );
        {* CurrentVehicleMode := 'H'; *}
        {* CommandedVehicleMode := 'H'; *}
        END
     ELSE
        BEGIN
        DesiredBodyLinearRates := Zero;
        DesiredBodyAngularRates := Zero;

        FOR foot := 1 TO 6 DO
             InitFootFlag[ foot ] := NotDone;

        FOR foot := 1 TO 6 DO
             IF ( foot IN [2,3,6] ) AND (InitFootFlag[foot] = NotDone)
                THEN
                   MoveFootInit(   foot,
                                   DesiredFootPosition[ foot ],
                                   dt,
                                   ZeroForce[ foot],
                                   PlannedFootRate[ foot ],
                                   FootRateType[ foot ],
                                   InitFootFlag[ foot ] )
                ELSE
                   BEGIN
                   PlannedFootRate[ foot ] := Zero;
                   FootRateType[ foot ] := Relative;
                   END;


        {*** COMPUTE NEW BODY TRANSFORM ***}
        TransIntegrate( DesiredBodyTransform,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        dt    );

        {* set FootPhase *}
        FootPhase[1] := Support;
        FootPhase[4] := Support;
        FootPhase[5] := Support;

        FootPhase[2] := Transfer;
        FootPhase[3] := Transfer;
        FootPhase[6] := Transfer;

        {* Check if initial tripod is completely lifted *}
        InitializeTripodFlag := Done;
        FOR foot := 1 TO 6 DO
                IF ( foot IN [2,3,6] ) AND (InitFootFlag[foot] = NotDone)
                   THEN
                        InitializeTripodFlag := NotDone;
```

```
        IF InitializeTripodFlag = Done
            THEN
                BEGIN
                TransferState[1] := Ground;
                TransferState[4] := Ground;
                TransferState[5] := Ground;

                TransferState[2] := LiftOff;
                TransferState[3] := LiftOff;
                TransferState[6] := LiftOff;

                FOR foot := 1 TO 6 DO
                    PresentRadius[ foot ] := 0.0;

                END;  { IF InitializeTripodFlag }

        END; { IF NormalizeBodyFlag }


END;  { TripodInitialize }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;       { Type declarations }
%INCLUDE DMCONST0;       { Constant declarations }
%INCLUDE DMEXTERN2;      { External procedure declarations }

{*****************************************************************}
{               File Name: OSUUPDATE.PAS                          }
{ Function: Update CommandedVelocity and                          }
{               PastRadius and PresentRadius.                     }
{*****************************************************************}


PROCEDURE UpdateVelocity{(
                        JoystickDeflection      : Array3;
                        DesiredFootPosition     : SixVectors;
                    VAR CommandedVelocity       : Vector2;
                    VAR PresentRadius           : Array6;
                    VAR PastRadius              : Array6
                                )};

CONST   MaxLinearRate = 1.1;    { inches per second }
        MaxAngularRate = 0.075; { radians per second }

VAR     foot    : Integer;
        XB,YB   : Real;

BEGIN   { UpdateVelocity }

{ Update joystick commands }
CommandedVelocity.X :=  JoystickDeflection[1]*MaxLinearRate;
CommandedVelocity.Y := -JoystickDeflection[0]*MaxLinearRate;
CommandedVelocity.R :=  JoystickDeflection[2]*MaxAngularRate;

{ Calculate location of each foot }
{ within circle of reachable area }
FOR foot := 1 TO 6 DO
   BEGIN
   PastRadius[ foot ] := PresentRadius[ foot ];
   XB := DesiredFootPosition[ foot ].X - EquilibriumPosition[ foot ].X;
   YB := DesiredFootPosition[ foot ].Y - EquilibriumPosition[ foot ].Y;
   PresentRadius[ foot ] := SQRT(XB*XB + YB*YB);
   END;

END;    { UpdateVelocity }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declarations }
%INCLUDE DMEXTERN2;     { External procedure declarations }

{*********************************************************************}
{              File Name: OSUDUTRI1.PAS                               }
{ Function: Move body and legs based on                               }
{              Dual Tripod leg placement algorithm.                   }
{*********************************************************************}

{***  STATIC VARIABLE DECLARATION ***}

VAR     CommandedVelocity       : Vector2;
        PastRadius              : Array6;
        SwitchTripodFlag        : Boolean;
        SpecialSectionFlag      : Boolean;
        Tripod145Down           : Boolean;
        Tripod236Down           : Boolean;
        OldTripod               : TripodGroup;
        SupportState            : FiveStates;
        SpecialState            : SpecialSubstates;


PROCEDURE DTInitialize;

BEGIN { DTInitialize }

{ Initialize flags used in DualTripod }

SpecialSectionFlag := False;

SpecialState := StopLegs;

END; { DTInitialize }



PROCEDURE PlacePointCalculated(
                                foot                    : Integer;
                                CommandedVelocity       : Vector2;
                                DesiredFootPosition     : Vector;
                            VAR DesiredPlacePoint       : Vector
                                                        );

VAR     XBNF,YBNF               : Real;
        XRE,YRE                 : Real;
        RFSQ,RT,RTRF            : Real;
        VelocityMagnitude       : Real;

BEGIN   { PlacePointCalculated }

IF ABS( CommandedVelocity.R ) > 0.00001
    THEN
        BEGIN
        { Find Equilibrium Position of given foot, WRT Center of Circular Arc }
        XRE := CommandedVelocity.Y / CommandedVelocity.R
                + EquilibriumPosition[ foot ].X;
```

```
        YRE := -CommandedVelocity.X / CommandedVelocity.R
                + EquilibriumPosition[ foot ].Y;

        RFSQ := XRE*XRE + YRE*YRE;
        IF RFSQ > ( ReachRadius*ReachRadius / 4.0 )
            THEN
                BEGIN
                RT := SQRT( 4.0*RFSQ - ReachRadius*ReachRadius );
                RTRF := RT / ( 2.0*RFSQ );
                END;
END;

{ Calculate desired foot position WRT body, in body coordinates }
IF ABS( CommandedVelocity.R ) < 0.00001
   THEN
      BEGIN { no rotational velocity }
        VelocityMagnitude :=
              SQRT( SQR( CommandedVelocity.X ) + SQR( CommandedVelocity.Y ) );
        IF VelocityMagnitude > 0.00001
          THEN
            BEGIN
              XBNF := (ReachRadius*CommandedVelocity.X) / VelocityMagnitude
                                + EquilibriumPosition[ foot ].X;

              YBNF := (ReachRadius*CommandedVelocity.Y) / VelocityMagnitude
                                + EquilibriumPosition[ foot ].Y;
            END
          ELSE
            BEGIN { no x or y velocity }
              XBNF := DesiredFootPosition.X;
              YBNF := DesiredFootPosition.Y;
            END;

      END
   ELSE
      BEGIN
      IF RFSQ <= ( ReachRadius*ReachRadius / 4.0 )
        THEN
          BEGIN
          { center of circular arc is inside one-half of reachable area }
          XBNF := XRE + EquilibriumPosition[ foot ].X;
          YBNF := YRE + EquilibriumPosition[ foot ].Y;
          END
        ELSE
          BEGIN
          IF CommandedVelocity.R > 0.00001
             THEN
               BEGIN { clockwise rotational velocity }
               XBNF := RTRF*( XRE*RT - YRE*ReachRadius )
                       - 2.0*CommandedVelocity.Y / CommandedVelocity.R
                       - EquilibriumPosition[ foot ].X;
               YBNF := RTRF*( YRE*RT + XRE*ReachRadius )
                       + 2.0*CommandedVelocity.X / CommandedVelocity.R
                       - EquilibriumPosition[ foot ].Y;
               END { clockwise }
             ELSE
               BEGIN { counter-clockwise rotational velocity }
               XBNF := RTRF*( XRE*RT + YRE*ReachRadius )
                       - 2.0*CommandedVelocity.Y / CommandedVelocity.R
                       - EquilibriumPosition[ foot ].X;
               YBNF := RTRF*( YRE*RT - XRE*ReachRadius )
                       + 2.0*CommandedVelocity.X / CommandedVelocity.R
                       - EquilibriumPosition[ foot ].Y;
               END; { counter-clockwise }

          END;
      END;
```

```
{ calculate desired foot position WRT hip }
DesiredPlacePoint.X := XBNF;
DesiredPlacePoint.Y := YBNF;
DesiredPlacePoint.Z := MaxFootHeight;

END;    { PlacePointCalculated }



PROCEDURE SupportPhase(
                        foot                    : Integer;
                        PresentRadius           : Real;
                        PastRadius              : Real;
                    VAR PlannedFootRate         : Vector;
                    VAR FootRateType            : RateType;
                    VAR OldTripod               : TripodGroup;
                    VAR FootPhase               : SixStates;
                    VAR TransferState           : SixSubstates;
                    VAR SwitchTripodFlag        : Boolean;
                    VAR SpecialSectionFlag      : Boolean
                                );

VAR     LegsDownFlag    : Completion;
        Tfoot           : Integer;      { Temporary, for DO Loop }

BEGIN   { SupportPhase }

{***  Evaluate Boundary Value Conditions  ***}

IF ( PresentRadius >= (0.5*ReachRadius) )  AND
   ( PresentRadius > PastRadius )
      THEN
        BEGIN
        { Check if all 6 legs are on ground }
        LegsDownFlag := Done;
        FOR Tfoot := 1 TO 6 DO
            IF FootPhase[ Tfoot ] <> Support
                THEN
                   LegsDownFlag := NotDone;

        IF LegsDownFlag = Done
            THEN
                SupportState[ foot ] := State4
            ELSE
                IF PresentRadius >= (0.95*ReachRadius)
                    THEN
                        BEGIN
                        { record old tripod }
                        IF foot IN [2,3,6]
                            THEN
                                OldTripod := Right
                            ELSE
                                OldTripod := Left;
                        SupportState[ foot ] := Special;
                        SpecialSectionFlag := True;
                        END
                    ELSE
                        IF foot IN [2,3,6]
                            THEN
                                SupportState[ foot ] := State2
                            ELSE
                                SupportState[ foot ] := State3;
        END
```

```
      ELSE
        SupportState[ foot ] := State1;

{***  Compute Foot Velocity Command  ***}

CASE SupportState[ foot ] OF

   State1:      BEGIN
                PlannedFootRate := Zero;
                FootRateType := Relative;
                END;  { State1 }

   State2:      BEGIN
                { check if new tripod coming down }
                IF Tripod145Down
                    THEN { new tripod coming down }
                        BEGIN { continue moving foot on ground }
                        PlannedFootRate := Zero;
                        FootRateType := Relative;
                        END
                    ELSE { start putting new tripod down }
                        BEGIN
                        TransferState[1] := PlaceDown;
                        TransferState[4] := PlaceDown;
                        TransferState[5] := PlaceDown;

                        Tripod145Down := True;

                        SwitchTripodFlag := True;
                        END;
                END;  { State2 }


   State3:      BEGIN
                { check if new tripod coming down }
                IF Tripod236Down
                    THEN { new tripod coming down }
                        BEGIN { continue moving foot on ground }
                        PlannedFootRate := Zero;
                        FootRateType := Relative;
                        END
                    ELSE { start putting new tripod down }
                        BEGIN
                        TransferState[2] := PlaceDown;
                        TransferState[3] := PlaceDown;
                        TransferState[6] := PlaceDown;

                        Tripod236Down := True;

                        SwitchTripodFlag := True;
                        END;
                END;  { State3 }

   State4:      BEGIN
                { all 6 legs are on ground, lift present tripod }
                IF foot IN [1,4,5]
                    THEN
                        BEGIN
                        FootPhase[1] := Transfer;
                        FootPhase[4] := Transfer;
                        FootPhase[5] := Transfer;

                        Tripod145Down := False;
                        END
```

```
                    ELSE
                        BEGIN
                        FootPhase[2] := Transfer;
                        FootPhase[3] := Transfer;
                        FootPhase[6] := Transfer;

                        Tripod236Down := False;
                        END;

                SwitchTripodFlag := True;
                END;  { State4 }

   Special:     { do nothing };

   END;   { CASE SupportState }

END;  { SupportPhase }



PROCEDURE TransferPhase(
                        foot                    : Integer;
                        FootContact             : Boolean;
                        CommandedVelocity       : Vector2;
                        DesiredFootPosition     : Vector;
                    VAR ZeroForce               : Vector;
                    VAR PlannedFootRate         : Vector;
                    VAR FootRateType            : RateType;
                    VAR FootPhase               : FootState;
                    VAR TransferState           : FootSubstate
                            );

VAR     DeltaPosition   : Vector;       { Intermediate position vector }
        Distance        : Real;         { Distance to DesiredPlacePoint }
        MotionDirection : Vector;       {Unit vector toward DesiredPlacePoint}
        DesiredPlacePoint : Vector;
        TemporaryForce  : SixVectors;

BEGIN   { TransferPhase }

{***  Evaluate Boundary Value Conditions ***}

CASE TransferState OF

   Ground:      TransferState := LiftOff;

   LiftOff:     IF DesiredFootPosition.Z >= MaxFootHeight
                    THEN {foot is at top of trajectory }
                        BEGIN
                        TransferState := Return;

                        ForceFeedback( TemporaryForce );
                        ZeroForce := TemporaryForce[ foot ];

                        PlacePointCalculated( foot,
                                              CommandedVelocity,
                                              DesiredFootPosition,
                                              DesiredPlacePoint );
                        VectSub( DeltaPosition,
                                 DesiredPlacePoint, DesiredFootPosition );
                        Distance := Magnitude( DeltaPosition );
                        END;
```

```
   Return:       BEGIN

                 ForceFeedback( TemporaryForce );
                 ZeroForce := TemporaryForce[ foot ];

                 PlacePointCalculated( foot,
                                       CommandedVelocity,
                                       DesiredFootPosition,
                                       DesiredPlacePoint );
                 VectSub( DeltaPosition,
                                  DesiredPlacePoint, DesiredFootPosition );
                 Distance := Magnitude( DeltaPosition );
                 END;

   PlaceDown:    IF FootContact
                     THEN
                         BEGIN
                         TransferState := Ground;
                         FootPhase := Support;
                         END;

   END;  { CASE TransferState }


{***  Compute Foot Velocity Command  ***}

CASE TransferState OF

   Ground:       BEGIN
                 PlannedFootRate := Zero;
                 FootRateType := Relative;
                 END; { Ground }

   LiftOff:      BEGIN
                 PlannedFootRate.X := 0.0;
                 PlannedFootRate.Y := 0.0;
                 PlannedFootRate.Z := FastLiftRate;

                 FootRateType := Relative;
                 END; { LiftOff }

   Return:       BEGIN
                 IF Distance > MaxError
                   THEN
                     BEGIN
                     VectDiv( MotionDirection, DeltaPosition, Distance );
                     VectMult(PlannedFootRate, FastReturnRate, MotionDirection);
                     END
                   ELSE
                     PlannedFootRate := Zero;

                 FootRateType := Absolute;
                 END; { Return }

   PlaceDown:    BEGIN
                 PlannedFootRate.X := 0.0;
                 PlannedFootRate.Y := 0.0;
                 PlannedFootRate.Z := -FastPlaceRate;

                 FootRateType := Relative;
                 END; { PlaceDown }

   END; { CASE TransferState }

END;    { TransferPhase }
```

```
PROCEDURE SpecialSection(
                        FootContact              : SixBooleans;
                        CommandedVelocity        : Vector2;
                        DesiredFootPosition      : SixVectors;
                        OldTripod                : TripodGroup;
                    VAR PlannedFootRate          : SixVectors;
                    VAR FootRateType             : SixRateTypes;
                    VAR DesiredBodyLinearRates   : Vector;
                    VAR DesiredBodyAngularRates  : Vector;
                    VAR FootPhase                : SixStates;
                    VAR TransferState            : SixSubstates;
                    VAR ZeroForce                : SixVectors;
                    VAR SpecialSectionFlag       : Boolean
                                                 );

VAR     foot             : Integer;
        UP               : SixBooleans;
        LegsDownFlag     : Completion;
        DeltaPosition    : Vector;        { Intermediate position vector }
        Distance         : Real;          { Distance to EquilibriumPosition }
        MotionDirection  : Vector;        { Unit vector toward Eq. Pos. }
        DesiredPlacePoint : Vector;
        ReturnFlag       : SixBooleans;
        TemporaryForce   : SixVectors;

BEGIN   { SpecialSection }

{ Stop body movement }
DesiredBodyLinearRates := Zero;
DesiredBodyAngularRates := Zero;

{*** Compute Foot Velocity Command ***}

CASE SpecialState OF

   StopLegs:    BEGIN { stop leg movement }
                FOR foot := 1 TO 6 DO
                   BEGIN
                   PlannedFootRate[ foot ] := Zero;
                   FootRateType[ foot ] := Relative;
                   END;

                END; { StopLegs }

   LegsDown:    BEGIN
                FOR foot := 1 TO 6 DO
                   BEGIN
                   IF FootContact[ foot ]
                      THEN
                        BEGIN
                        FootPhase[ foot ] := Support;
                        PlannedFootRate[ foot ] := Zero;
                        END
                      ELSE
                        BEGIN
                        PlannedFootRate[ foot ].X := 0.0;
                        PlannedFootRate[ foot ].Y := 0.0;
                        PlannedFootRate[ foot ].Z := -PlaceRate;
                        END;

                   FootRateType[ foot ] := Relative;
                   END;  { DO }

                END;    { LegsDown }
```

```
LiftTripod:  BEGIN
             FOR foot := 1 TO 6 DO
                BEGIN
                PlannedFootRate[ foot ] := Zero;
                FootRateType[ foot ] := Relative;
                END;

             { lift old tripod }
             IF OldTripod = Left
                THEN
                   BEGIN
                   PlannedFootRate[1].Z := LiftRate;
                   PlannedFootRate[4].Z := LiftRate;
                   PlannedFootRate[5].Z := LiftRate;

                   FootPhase[1] := Transfer;
                   FootPhase[4] := Transfer;
                   FootPhase[5] := Transfer;
                   END
                ELSE
                   BEGIN
                   PlannedFootRate[2].Z := LiftRate;
                   PlannedFootRate[3].Z := LiftRate;
                   PlannedFootRate[6].Z := LiftRate;

                   FootPhase[2] := Transfer;
                   FootPhase[3] := Transfer;
                   FootPhase[6] := Transfer;
                   END;
             END;    { LiftTripod }

ReturnTripod:BEGIN
             IF ( OldTripod = Left )
               THEN
                BEGIN
                  FOR foot := 1 TO 6 DO
                     IF foot IN [1,4,5]
                       THEN { foot in air }
                         BEGIN
                         PlacePointCalculated( foot,
                                               CommandedVelocity,
                                               DesiredFootPosition[ foot ],
                                               DesiredPlacePoint );
                         VectSub( DeltaPosition, DesiredPlacePoint,
                                  DesiredFootPosition[ foot ] );
                         Distance := Magnitude( DeltaPosition );
                         IF Distance <= MaxError
                           THEN { foot at DesiredPlacePoint }
                             BEGIN
                             ReturnFlag[ foot ] := TRUE;
                             PlannedFootRate[ foot ] := Zero;
                             END
                           ELSE
                             BEGIN
                             ReturnFlag[ foot ] := FALSE;
                             VectDiv( MotionDirection, DeltaPosition,
                                      Distance );
                             VectMult( PlannedFootRate[foot],ReturnRate,
                                       MotionDirection );
                             END;
                         FootRateType[ foot ] := Absolute;
                         END
                       ELSE { foot on ground }
                         BEGIN
                         PlannedFootRate[ foot ] := Zero;
                         FootRateType[ foot ] := Relative;
                         END;
                END
```

```
                    ELSE { OldTripod = Right }
                     BEGIN
                       FOR foot := 1 TO 6 DO
                          IF foot IN [2,3,6]
                            THEN { foot in air }
                              BEGIN
                              PlacePointCalculated( foot,
                                                    CommandedVelocity,
                                                    DesiredFootPosition[ foot ],
                                                    DesiredPlacePoint );
                              VectSub( DeltaPosition, DesiredPlacePoint,
                                       DesiredFootPosition[ foot ] );
                              Distance := Magnitude( DeltaPosition );
                              IF Distance <= MaxError
                                THEN { foot at DesiredPlacePoint }
                                  BEGIN
                                  ReturnFlag[ foot ] := TRUE;
                                  PlannedFootRate[ foot ] := Zero;
                                  END
                                ELSE
                                  BEGIN
                                  ReturnFlag[ foot ] := FALSE;
                                  VectDiv( MotionDirection, DeltaPosition,
                                                            Distance );
                                  VectMult( PlannedFootRate[foot],ReturnRate,
                                                            MotionDirection );
                                  END;
                              FootRateType[ foot ] := Absolute;
                              END
                            ELSE { foot on ground }
                              BEGIN
                              PlannedFootRate[ foot ] := Zero;
                              FootRateType[ foot ] := Relative;
                              END;
                     END;

                END; { ReturnTripod }

    END; { CASE SpecialState }


{*** Evaluate Boundary Value Conditions ***}

CASE SpecialState OF

    StopLegs:    SpecialState := LegsDown;

    LegsDown:    BEGIN
                 { check if all 6 legs are down }
                 LegsDownFlag := Done;
                 FOR foot := 1 TO 6 DO
                    IF FootPhase[ foot ] <> Support
                       THEN
                         LegsDownFlag := NotDone;

                 IF LegsDownFlag = Done
                    THEN { all 6 legs are on ground }
                       BEGIN
                       SpecialState := LiftTripod;
                       { reset TripodDown flags, to signify tripods are down }

                       Tripod145Down := False;
                       Tripod236Down := False;
                       END;
                 END;  { LegsDown }
```

```
LiftTripod:  BEGIN
             FOR foot := 1 TO 6 DO
              BEGIN
               IF ( OldTripod = Left )
                 THEN
                     IF ( foot IN [1,4,5] )
                       AND (DesiredFootPosition[ foot ].Z >= MaxFootHeight)
                        THEN { foot is at top of trajectory }
                           BEGIN
                           UP[ foot ] := TRUE;
                           PlannedFootRate[ foot ].Z := 0.0;
                           END
                        ELSE
                           UP[ foot] := FALSE;

               IF ( OldTripod = Right )
                 THEN
                     IF ( foot IN [2,3,6] )
                       AND (DesiredFootPosition[ foot ].Z >= MaxFootHeight)
                        THEN { foot is at top of trajectory }
                           BEGIN
                           UP[ foot ] := TRUE;
                           PlannedFootRate[ foot ].Z := 0.0;
                           END
                        ELSE
                           UP[ foot ] := FALSE;

              END;  { DO }

             { check if complete tripod is lifted }
             IF UP[1] AND UP[4] AND UP[5]
                THEN { left tripod up completely }
                   BEGIN
                   TransferState[1] := Return;
                   TransferState[4] := Return;
                   TransferState[5] := Return;

                   ForceFeedback( TemporaryForce );
                   ZeroForce[1] := TemporaryForce[1];
                   ZeroForce[4] := TemporaryForce[4];
                   ZeroForce[5] := TemporaryForce[5];

                   SpecialState := ReturnTripod;
                   END;

             IF UP[2] AND UP[3] AND UP[6]
                THEN { right tripod up completely }
                   BEGIN
                   TransferState[2] := Return;
                   TransferState[3] := Return;
                   TransferState[6] := Return;

                   ForceFeedback( TemporaryForce );
                   ZeroForce[2] := TemporaryForce[2];
                   ZeroForce[3] := TemporaryForce[3];
                   ZeroForce[6] := TemporaryForce[6];

                   SpecialState := ReturnTripod;
                   END;

             END;    { LiftTripod }

ReturnTripod:BEGIN
             { check if lifted tripod has been returned to PlacePoint }
             IF ( OldTripod = Left )
               THEN
```

```
                    BEGIN
                    IF ReturnFlag[1] AND ReturnFlag[4] AND ReturnFlag[5]
                      THEN { left tripod at DesiredPlacePoint }
                            BEGIN
                            { reset SpecialState }
                            SpecialState := StopLegs;

                            { finished with SpecialSection }
                            SpecialSectionFlag := FALSE;
                            END;
                    END
                  ELSE { OldTripod = Right }
                    BEGIN
                    IF ReturnFlag[2] AND ReturnFlag[3] AND ReturnFlag[6]
                      THEN { right tripod at DesiredPlacePoint }
                            BEGIN
                            { reset SpecialState }
                            SpecialState := StopLegs;

                            { finished with SpecialSection }
                            SpecialSectionFlag := FALSE;
                            END;
                    END;

                END; { ReturnTripod }

   END; { CASE SpecialState }

END;  { SpecialSection }



PROCEDURE DualTripod;

LABEL 10, 20;

VAR     foot    : Integer;

BEGIN   { DualTripod }

UpdateVelocity( JoystickDeflection,
                DesiredFootPosition,
                CommandedVelocity,
                PresentRadius,
                PastRadius );

{ check SpecialSectionFlag }
IF SpecialSectionFlag
    THEN { call SpecialSection procedure }
        BEGIN
        SpecialSection( FootContact,
                        CommandedVelocity,
                        DesiredFootPosition,
                        OldTripod,
                        PlannedFootRate,
                        FootRateType,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        FootPhase,
                        TransferState,
                        ZeroForce,
                        SpecialSectionFlag );
        GoTo 20;
        END;
```

```
{ Begin Main DO Loop }

10:     FOR foot := 1 TO 6 DO
           { check if foot is in support phase or transfer phase }
           IF FootPhase[ foot ] = Support
              THEN { call SupportPhase routine }
                BEGIN
                SupportPhase( foot,
                              PresentRadius[ foot ],
                              PastRadius[ foot ],
                              PlannedFootRate[ foot ],
                              FootRateType[ foot ],
                              OldTripod,
                              FootPhase,
                              TransferState,
                              SwitchTripodFlag,
                              SpecialSectionFlag );

                IF SwitchTripodFlag
                   THEN
                      BEGIN
                      SwitchTripodFlag := False;
                      GoTo 10;
                      END;
                IF SpecialSectionFlag
                   THEN { call SpecialSection procedure }
                      BEGIN
                      SpecialSection( FootContact,
                                      CommandedVelocity,
                                      DesiredFootPosition,
                                      OldTripod,
                                      PlannedFootRate,
                                      FootRateType,
                                      DesiredBodyLinearRates,
                                      DesiredBodyAngularRates,
                                      FootPhase,
                                      TransferState,
                                      ZeroForce,
                                      SpecialSectionFlag );
                      GoTo 20;
                      END;
                END
              ELSE { call TransferPhase routine }
                TransferPhase( foot,
                               FootContact[ foot ],
                               CommandedVelocity,
                               DesiredFootPosition[ foot ],
                               ZeroForce[ foot ],
                               PlannedFootRate[ foot ],
                               FootRateType[ foot ],
                               FootPhase[ foot ],
                               TransferState[ foot ] );

        DesiredBodyLinearRates.X := CommandedVelocity.X;
        DesiredBodyLinearRates.Y := CommandedVelocity.Y;
        DesiredBodyLinearRates.Z := 0.0;

        DesiredBodyAngularRates.X := 0.0;
        DesiredBodyAngularRates.Y := 0.0;
        DesiredBodyAngularRates.Z := CommandedVelocity.R;

20:     TransIntegrate( DesiredBodyTransform,
                        DesiredBodyLinearRates,
                        DesiredBodyAngularRates,
                        dt  );

END;  { DualTripod }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMCONST0;      { Constant declarations }
%INCLUDE DMEXTERN9;     { External Procedure declarations }

{*********************************************************}
{               File Name: OSUMNTRI4.PAS                  }
{ Function: Move body and legs based on                   }
{               Manual Tripod algorithm.                  }
{*********************************************************}


PROCEDURE BodyMotion( FootPhase                 : SixStates;
                      CommandedVelocity         : Vector2;
                      DesiredFootPosition       : SixVectors;
                      dt                        : Real;
                  VAR PlannedFootRate           : SixVectors;
                  VAR FootRateType              : SixRateTypes;
                  VAR DesiredBodyLinearRates    : Vector;
                  VAR DesiredBodyAngularRates   : Vector;
                  VAR DesiredBodyTransform      : Homogeneous
                                              );

LABEL   30;

VAR     foot                    : Integer;
        PlannedBodyLinearRates  : Vector;
        PlannedBodyAngularRates : Vector;
        PlannedBodyTransform    : Homogeneous;
        BodyServoAngularRates   : Vector;
        BodyServoFootRate       : SixVectors;
        PredictedFootRate       : SixVectors;
        PredictedFootRateType   : SixRateTypes;
        DesiredFootRate         : SixVectors;
        XB, YB                  : Real;
        PresentRadius           : Real;

BEGIN   { BodyMotion }

{ calculate planned location of feet }
PlannedBodyLinearRates.X := CommandedVelocity.X;
PlannedBodyLinearRates.Y := CommandedVelocity.Y;
PlannedBodyLinearRates.Z := 0.0;

PlannedBodyAngularRates.X := 0.0;
PlannedBodyAngularRates.Y := 0.0;
PlannedBodyAngularRates.Z := CommandedVelocity.R;

PlannedBodyTransform := DesiredBodyTransform;

TransIntegrate( PlannedBodyTransform,
                PlannedBodyLinearRates,
                PlannedBodyAngularRates,
                dt   );

OrthoNorm( PlannedBodyTransform.Rotation,
           PlannedBodyTransform.Rotation );

BodyServo( PlannedBodyTransform.Rotation,
           PlannedBodyTransform.Rotation,       { for ActualTransform }
           PlannedBodyAngularRates,
           PlannedBodyAngularRates,     { for ActualBodyAngularRates }
           BodyServoAngularRates  );
```

```
FOR foot := 1 TO 6 DO   { convert body rates to foot rates }
        BEGIN
        CrossProduct( BodyServoFootRate[ foot ],
                      DesiredFootPosition[ foot ],
                      BodyServoAngularRates  );

        VectSub( BodyServoFootRate[ foot ],
                 BodyServoFootRate[ foot ],
                 PlannedBodyLinearRates  );
        END;

FOR foot := 1 TO 6 DO
        { predict foot rates and type }
        IF FootPhase[ foot ] = Support
          THEN
             BEGIN
             PredictedFootRate[ foot ] := Zero;
             PredictedFootRateType[ foot ] := Relative;
             End
          ELSE
             BEGIN
             PredictedFootRate[ foot ] := Zero;
             PredictedFootRateType[ foot ] := Absolute;
             END;
{ end of DO }


MixFootRates(   PredictedFootRate,
                BodyServoFootRate,
                PredictedFootRateType,
                dt,
                DesiredFootRate,
                DesiredFootPosition  );

{NOTE: update DesiredFootPosition value, but not returned to calling routine }

FOR foot := 1 TO 6 DO
   BEGIN

   IF FootPhase[ foot ] = Support
     THEN
        BEGIN

        XB := DesiredFootPosition[ foot ].X - EquilibriumPosition[ foot ].X;
        YB := DesiredFootPosition[ foot ].Y - EquilibriumPosition[ foot ].Y;
        PresentRadius := SQRT( XB*XB + YB*YB );

        { check if this foot is within circle of reachable area }
        IF PresentRadius > ReachRadius
           THEN { Stop movement of body and legs }
                BEGIN

                HaltVehicle( dt,
                             PlannedFootRate,
                             FootRateType,
                             DesiredBodyLinearRates,
                             DesiredBodyAngularRates,
                             DesiredBodyTransform );
                Writeln('Foot', foot, 'is at end of reachable limit');
                GoTo 30;

                END;
        END;
   END; { DO }
```

```
DesiredBodyLinearRates := PlannedBodyLinearRates;
DesiredBodyAngularRates := PlannedBodyAngularRates;
DesiredBodyTransform := PlannedBodyTransform;

PlannedFootRate := PredictedFootRate;
FootRateType := PredictedFootRateType;

30:  END;       { BodyMotion }



PROCEDURE ManualTripod(( JoystickDeflection      : Array3;
                         SubCommand              : Char;
                     VAR NewTripodDown           : Boolean;
                         SupportTripod           : TripodGroup;
                         dt                      : Real;
                         ActualFootPosition      : SixVectors;
                         DesiredFootPosition     : SixVectors;
                     VAR FootPhase               : SixStates;
                     VAR TransferState           : SixSubstates;
                         FootContact             : SixBooleans;
                     VAR PresentRadius           : Array6;
                     VAR ZeroForce               : SixVectors;
                     VAR PlannedFootRate         : SixVectors;
                     VAR FootRateType            : SixRateType;
                     VAR DesiredBodyLinearRates  : Vector;
                     VAR DesiredBodyAngularRates: Vector;
                     VAR DesiredBodyTransform    : Homogeneous )};

VAR     foot                    : Integer;
        OldTripod               : TripodGroup;
        SpecialSectionFlag      : Boolean;
        CommandedVelocity       : Vector2;
        PastRadius              : Array6;
        CurrentVehicleMode      : Char;

BEGIN   { ManualTripod }

UpdateVelocity( JoystickDeflection,
                DesiredFootPosition,
                CommandedVelocity,
                PresentRadius,
                PastRadius  );

CASE SubCommand OF

   'S': BEGIN   { SubCommand  S : switch Support Phase tripod, control body }
        IF NOT NewTripodDown
          THEN { call SpecialSection routine to put new tripod down }
               BEGIN

               { set value of OldTripod, tripod to be lifted }
               IF SupportTripod = Left
                 THEN OldTripod := Right
                 ELSE OldTripod := Left;

               { set SpecialSectionFlag }
               SpecialSectionFlag := True;
```

```
          SpecialSection(   FootContact,
                            CommandedVelocity,
                            DesiredFootPosition,
                            OldTripod,
                            PlannedFootRate,
                            FootRateType,
                            DesiredBodyLinearRates,
                            DesiredBodyAngularRates,
                            FootPhase,
                            TransferState,
                            ZeroForce,
                            SpecialSectionFlag  );

          TransIntegrate( DesiredBodyTransform,
                          DesiredBodyLinearRates,
                          DesiredBodyAngularRates,
                          dt        );

          { check if finished with SpecialSection routine }

          IF NOT SpecialSectionFlag
            THEN NewTripodDown := TRUE;     { new tripod is down }
          END

        ELSE { move body using support legs }

          BodyMotion( FootPhase,
                      CommandedVelocity,
                      DesiredFootPosition,
                      dt,
                      PlannedFootRate,
                      FootRateType,
                      DesiredBodyLinearRates,
                      DesiredBodyAngularRates,
                      DesiredBodyTransform  );

      END;    { SubCommand  S }


'T': BEGIN   { SubCommand T : control Transfer Phase tripod }

      { Stop body movement }
      DesiredBodyLinearRates := Zero;
      DesiredBodyAngularRates := Zero;

      IF ( SupportTripod = Left )
        THEN
          BEGIN

          FOR foot := 1 TO 6 DO
            IF foot IN [2,3,6]
              THEN  { foot in transfer phase }
                BEGIN

                PlannedFootRate[ foot ].X := CommandedVelocity.X;
                PlannedFootRate[ foot ].Y := CommandedVelocity.Y;
                PlannedFootRate[ foot ].Z := CommandedVelocity.R;
                FootRateType[ foot ] := Absolute;
                CurrentVehicleMode := 'M';
```

```
                KinematicLimits( foot,
                                 dt,
                                 CurrentVehicleMode,
                                 ActualFootPosition,
                                 DesiredFootPosition,
                                 FootContact[ foot ],
                                 PlannedFootRate,
                                 FootRateType,
                                 DesiredBodyLinearRates,
                                 DesiredBodyAngularRates,
                                 DesiredBodyTransform,
                                 FootPhase,
                                 ZeroForce[ foot ] );

                   END
                ELSE  { foot in support phase }
                   BEGIN

                   PlannedFootRate[ foot ] := Zero;
                   FootRateType[ foot ] := Absolute;

                   END;
             END

           ELSE  { SupportTripod = Right }

             BEGIN

             FOR foot := 1 TO 6 DO
               IF foot IN [1,4,5]
                 THEN  { foot in transfer phase }
                   BEGIN

                   PlannedFootRate[ foot ].X := CommandedVelocity.X;
                   PlannedFootRate[ foot ].Y := CommandedVelocity.Y;
                   PlannedFootRate[ foot ].Z := CommandedVelocity.R;
                   FootRateType[ foot ] := Absolute;
                   CurrentVehicleMode := 'M';
                   KinematicLimits( foot,
                                    dt,
                                    CurrentVehicleMode,
                                    ActualFootPosition,
                                    DesiredFootPosition,
                                    FootContact[ foot ],
                                    PlannedFootRate,
                                    FootRateType,
                                    DesiredBodyLinearRates,
                                    DesiredBodyAngularRates,
                                    DesiredBodyTransform,
                                    FootPhase,
                                    ZeroForce[ foot ] );

                   END
                ELSE  { foot in support phase }
                   BEGIN

                   PlannedFootRate[ foot ] := Zero;
                   FootRateType[ foot ] := Absolute;

                   END;
             END;

         END;  { SubCommand  T }

    END; { CASE SumCommand }

END;  { ManualTripod }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;       { Type declarations }
%INCLUDE DMCONST0;       { Constant declarations }
%INCLUDE DMEXTERN2;      { External procedure declarations }
%INCLUDE EXTPROJ8;       { External declarations for projection }
%INCLUDE EXTPROJHP;      { HP NonPascal procedure declarations }

{*****************************************************************}
{*              FileName: OSUPRJHP8.PAS                          *}
{* Function: Shows projection of feet, support polygon,         *}
{*             and center of gravity to Z = 0 plane.            *}
{*             Also shows projection of reachable areas.        *}
{*****************************************************************}


{*** STATIC VARIABLE DECLARATION ***}
VAR     Circle : M101x2;

PROCEDURE Projection(( FootPhase                : SixStates;
                       DesiredFootPosition      : SixVectors;
                   VAR FirstTimeP               : Boolean;
                       CurrentVehicleMode       : Char;
                       LegNumber                : Integer;
                       Pitch                    : Real;
                       Roll                     : Real;
                   VAR CriticalFeet             : LegSet;
                   VAR StabilityFlag            : Boolean )};


CONST   ScaleP  = 3.0;          { scale factor for projection }

TYPE    IntegerVector = RECORD
                          X,Y,Z : Integer;
                        END;

VAR     I, J, K                 : Integer;
        foot                    : Integer;
        XC, YC                  : Real;     {locate center of projected view}
        XT, YT                  : Integer; { location of X for feet }
        Nfile3                  : Integer;
        Fsize, FNchar           : Integer;
        Ftext, Btext            : Char;
        PolygonSides            : Integer;
        Polygon                 : M101x2;  { really only need 7x2 }
        CenterOfGravity         : Vector;
        ProjectedCG             : Vector;
        ProjectedPosition       : IntegerVector;
        ProjectedFootPosition   : SixVectors;
        Box                     : M101x2;  { really only need 5x2 }
        ShiftCGX, ShiftCGY      : Integer;
        VerticalRotationTransform : Matrix;

        ZeroYaw                 : Real;
        PolygonFeet, SupportFeet : LegSet;
        NullSet                 : LegSet;
        NumberOfTraces          : Integer;
        Npoints                 : Integer;
        ZeroSixVectors          : SixVectors;
```

```
BEGIN   { Projection }

XC := 512.0/ScaleP;       { location of center of projected view }
YC := 700.0/ScaleP;

NullSet := [];
Nfile3 := 3;              { use file #3 for projected view }
Fsize  := 0;              { size of character, 0 is smallest }
FNchar := 1;              { nos. of characters to be plotted }
Ftext  := 'X';            { character to be plotted for feet }
Btext  := ' ';            { blank }

{***IF FirstTimeP
  THEN DrawAxes;***}

FirstTimeP := False;

ZeroYaw   := 0.0;
GetOrientation( ZeroYaw,
                Pitch,
                Roll,
                VerticalRotationTransform );


{ Transform location of each foot to "Earth Based" coordinates }
SupportFeet := [];        { empty set }
FOR foot := 1 TO 6 DO
  BEGIN
  MatrixMult( ProjectedFootPosition[ foot ],
              VerticalRotationTransform,
              DesiredFootPosition[ foot ]  );

  IF ( FootPhase[ foot ] = Support )
     THEN SupportFeet := SupportFeet + [ foot ];

  END;  { DO }

{ Initial location of center of gravity }
CenterOfGravity.X := 0.0;
CenterOfGravity.Y := 0.0;
CenterOfGravity.Z := 0.0;
{ Transform location of center of gravity to "Earth Based" coord }
MatrixMult( ProjectedCG, VerticalRotationTransform, CenterOfGravity );

{ plot a square, 10 by 10, for center of gravity }

ShiftCGX := ROUND( ScaleP * ( XC + ProjectedCG.X ));
ShiftCGY := ROUND( ScaleP * ( YC + ProjectedCG.Y ));

Box[1,1] := ShiftCGX + 5;
Box[1,2] := ShiftCGY + 5;
Box[2,1] := ShiftCGX - 5;
Box[2,2] := ShiftCGY + 5;
Box[3,1] := ShiftCGX - 5;
Box[3,2] := ShiftCGY - 5;
Box[4,1] := ShiftCGX + 5;
Box[4,2] := ShiftCGY - 5;
Box[5,1] := Box[1,1];
Box[5,2] := Box[1,2];

{ Plot the box for CG }
Npoints := 5;
Hpplot( Box, Npoints, Nfile3 );         { use file #3 }
```

```
{ Draw projected view of circles of reachable area }
DrawReachAreas( Circle,
                VerticalRotationTransform,
                XC,
                YC,
                DesiredFootPosition  );


CheckRedundantFeet( ProjectedFootPosition,
                    SupportFeet,
                    PolygonFeet  );

{**CheckPolygonStability( ProjectedFootPosition,
                         PolygonFeet,
                         ProjectedCG,
                         StabilityFlag );

IF NOT StabilityFlag
  THEN Writeln( 'UNSTABLE' );   **}

DetermineCriticalFeet( ProjectedFootPosition,
                       ProjectedCG,
                       SupportFeet,
                       CriticalFeet );

NumberOfTraces := 3;

PlotSupportPolygon( ProjectedFootPosition,
                    SupportFeet,
                    PolygonFeet,
                    CriticalFeet,
                    NumberOfTraces );

{ If in Individual Leg Control Mode }
IF ( CurrentVehicleMode = 'F' ) AND ( FootPhase[ LegNumber ] <> Support )
  THEN
    BEGIN
    { add this foot to SupportFeet }
    SupportFeet := SupportFeet + [ LegNumber ];
    CheckRedundantFeet( ProjectedFootPosition,
                        SupportFeet,
                        PolygonFeet );

    { set SupportFeet = empty set to avoid drawing X's on feet }
    { set CriticalFeet = empty set to avoid drawing boxes on feet }
    { draw predicted polygon, one trace }
    NumberOfTraces := 1;
    PlotSupportPolygon( ProjectedFootPosition,
                        NullSet,
                        PolygonFeet,
                        NullSet,
                        NumberOfTraces );

    END
  ELSE
    BEGIN
    FOR I := 1 TO 6 DO
      ZeroSixVectors[I] := Zero;
    NumberOfTraces := 1;
    PlotSupportPolygon( ZeroSixVectors,
                        NullSet,
                        PolygonFeet,
                        NullSet,
                        NumberOfTraces  );

    END;
```

```
{ Draw display of pitch and roll of vehicle body }
{***PitchRoll( Pitch, Roll );***}


{ compute and display Energy Stability Margin }
StabilityMargin( ProjectedFootPosition,
                 PolygonFeet,
                 ProjectedCG );


Stover( Nfile3 );                { end of file for projected view }



END;    { Projection }




PROCEDURE CalculateCircle{( Radius       : Real )};

{ Calculate 10 points for a circle of given Radius and with center at (0,0).}
{ Points are stored in static global variable Circle(10x2).}

CONST   NumberPoints = 10;
        ScaleP = 3.0;

VAR     K        : Integer;
        Theta    : Real;

BEGIN   { CalculateCircle }

FOR K := 0 TO NumberPoints DO
  BEGIN
  Theta := K * 2.0 * PI / NumberPoints;
  Circle[ K+1, 1 ] := ROUND( ScaleP * Radius * COS( Theta ) );
  Circle[ K+1, 2 ] := ROUND( ScaleP * Radius * SIN( Theta ) );
  END;

END;    { CalculateCircle }
```

```
PROCEDURE DrawReachAreas{( ReachArea                    : M101x2;
                           VerticalRotationTransform    : Matrix;
                           ViewCenterX                  : Real;
                           ViewCenterY                  : Real;
                           DesiredFootPosition          : SixVectors )};

CONST   ScaleP = 3.0;

VAR     I, foot                 : Integer;
        ProjectedVector         : Vector;
        ProjectedArea           : M101x2;
        ProjectedCenters        : SixVectors;
        Xcoord, Ycoord          : Integer;
        Npoints, Nfile          : Integer;
        ActualCenters           : SixVectors;


BEGIN   { DrawReachAreas }

{ Project ReachArea to the horizontal plane }
FOR I := 1 TO 11 DO
  BEGIN
  ProjectedVector.X := ReachArea[I,1];
  ProjectedVector.Y := ReachArea[I,2];
  ProjectedVector.Z := 0.0;
  { project that vector to horizontal plane }
  MatrixMult( ProjectedVector,
              VerticalRotationTransform,
              ProjectedVector );
  { convert projected vector to integer, and store in array }
  ProjectedArea[I,1] := ROUND( ProjectedVector.X );
  ProjectedArea[I,2] := ROUND( ProjectedVector.Y );
  END;

{ Project the EquilibriumPosition to the horizontal plane }
FOR foot := 1 TO 6 DO
  BEGIN
  ActualCenters[foot].X := EquilibriumPosition[foot].X;
  ActualCenters[foot].Y := EquilibriumPosition[foot].Y;
  ActualCenters[foot].Z := DesiredFootPosition[foot].Z;

  MatrixMult( ProjectedCenters[ foot ],
              VerticalRotationTransform,
              ActualCenters[ foot ]  );
  END;


{ Draw the 6 projected reachable areas }
FOR foot := 1 TO 6 DO
  BEGIN
  For I := 1 To 11 DO
    BEGIN
    ReachArea[I,1] := ROUND( ScaleP * ( ViewCenterX + ProjectedCenters[ foot ].X )
                  + ProjectedArea[I,1];
    ReachArea[I,2] := ROUND( ScaleP * ( ViewCenterY + ProjectedCenters[ foot ].Y )
                  + ProjectedArea[I,2];
    END;
  { plot this reachable area }
  Npoints := 11;
  Nfile   := 3;
  Hpplot( ReachArea, Npoints, Nfile );          { use file #3 }
  END;

END;    { DrawReachAreas }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMEXTERN2;     { External procedure declarations }
%INCLUDE EXTPROJ3;      { External procedure declarations for projection }
%INCLUDE EXTPROJHP;     { HP NonPascal procedure declarations }

{*************************************************************}
{*              FileName: POLY2HP.PAS                        *}
{* Function: Check if Center-of-Gravity is inside            *}
{*              of support polygon.                          *}
{*************************************************************}

PROCEDURE CheckPolygonStability(( ProjectedFootPosition : SixVectors;
                                  PolygonFeet           : LegSet;
                                  ProjectedCG           : Vector;
                              VAR StabilityFlag         : Boolean ));

LABEL 10;

CONST   MinimumStabilityMargin  = 3.0;  { inches }

VAR     PolygonVerticies        : SevenVectors;
        NumberOfVerticies       : Integer;
        PolygonVector           : Vector;
        InsideNormal            : Vector;
        K                       : Integer;
        foot                    : Integer;
        CGVector                : Vector;
        Product                 : Real;
        UnitInsideNormal        : Vector;

BEGIN   { CheckPolygonStability }

{ create array of polygon verticies }
NumberOfVerticies := 1;
FOR K := 1 TO 6 DO
  BEGIN
  { select proper feet in order 1,3,5,6,4,2 }
  IF (K <= 3 )
    THEN foot := 2*K - 1
    ELSE foot := 14 - 2*K;

  IF foot IN PolygonFeet
    THEN
        BEGIN
        PolygonVerticies[ NumberOfVerticies ] := ProjectedFootPosition[ foot ];
        NumberOfVerticies := NumberOfVerticies + 1;
        END;
  END;

{ last polygon vertex same as first vertex }
PolygonVerticies[ NumberOfVerticies ] := PolygonVerticies[ 1 ];

{ initialize StabilityFlag }
StabilityFlag := True;
```

```
  FOR K := 1 TO ( NumberOfVerticies - 1 ) DO
    BEGIN
    { find vector for edge of polygon }
    VectSub( PolygonVector, PolygonVertices[ K+1 ], PolygonVertices[ K ] );
    { find inside normal for that vector }
    InsideNormal.X := -PolygonVector.Y;
    InsideNormal.Z :=  0.0;
    { find vector from leg 6 to leg 4 }
    VectSub( MiddleVector, ProjectedFootPosition[4],
                ProjectedFootPosition[6] );
    { find dot product }
    Product := Dot( MiddleVector, InsideNormal );
    { check sign of product }
    IF Product >= 0.0
        THEN PolygonFeet := PolygonFeet - [4];  { foot 4 is redundant }
    END;

END;     { CheckRedundantFeet }



PROCEDURE DetermineCriticalFeet{( ProjectedFootPosition : SixVectors;
                                  ProjectedCG           : Vector;
                                  SupportFeet           : LegSet;
                              VAR CriticalFeet          : LegSet )};

{* Function: Determine which of the SupportFeet are critical for support, *}
{*              i.e. necessary for stability.                              *}

VAR      TemporarySupportFeet     : LegSet;
         TemporaryPolygonFeet     : LegSet;
         StabilityFlag            : Boolean;
         foot                     : Integer;

BEGIN    { DetermineCriticalFeet }

CriticalFeet := [];              { empty set }
FOR foot := 1 TO 6 DO
  BEGIN
  IF foot IN SupportFeet
    THEN
        BEGIN
        { remove this foot temporarily }
        TemporarySupportFeet := SupportFeet - [ foot ];
        { check for redundant feet }
        CheckRedundantFeet( ProjectedFootPosition,
                            TemporarySupportFeet,
                            TemporaryPolygonFeet );
        { check this temporary polygon for stability }
        CheckPolygonStability( ProjectedFootPosition,
                               TemporaryPolygonFeet,
                               ProjectedCG,
                               StabilityFlag  );
        IF NOT StabilityFlag
          THEN CriticalFeet := CriticalFeet + [ foot ];
        END;
  END;  { DO }

END;     { DetermineCriticalFeet }
```

```
PROCEDURE PlotSupportPolygon(( ProjectedFootPosition    : SixVectors;
                               SupportFeet              : LegSet;
                               PolygonFeet              : LegSet;
                               CriticalFeet             : LegSet;
                               NumberOfTraces           : Integer  ));

{* Function: Draws a polygon interconnecting the PolygonFeet,  *}
{*              and draws X at location of each SupportFeet.   *}
{*              Also draws boxes around the critical feet.      *}

TYPE    IntegerVector = RECORD
                          X,Y,Z : Integer;
                        END;

CONST   ScaleP  = 3.0;          { scale factor for projection }
        XC = 170.666;           { center of projected view, 512.0/ScaleP }
        YC = 233.333;           { center of projected view, 700.0/ScaleP }

VAR     ProjectedPosition       : IntegerVector;
        Polygon                 : M101x2;
        PolygonSides            : Integer;
        K, I                    : Integer;
        foot                    : Integer;
        XT, YT                  : Integer;
        Box                     : M101x2;
        Fsize, FNchar           : Integer;
        Ftext, Btext            : Char;
        Npoints, Nfile3         : Integer;

BEGIN   { PlotSupportPolygon }

Fsize  := 0;            { size of character, 0 is smallest }
FNchar := 1;            { nos. of characters to be plotted }
Ftext  := 'X';          { character to be plotted for feet }
Btext  := ' ';          { blank }
Nfile3 := 3;            { use file #3 }

PolygonSides := 1;
FOR K := 1 TO 6 DO
  BEGIN
  { select proper feet in order 1,3,5,6,4,2 }
  IF ( K <= 3 )
    THEN foot := 2*K - 1
    ELSE foot := 14 - 2*K;

  { scale foot position and convert to integer form }
  ProjectedPosition.X := ROUND(ScaleP*(XC + ProjectedFootPosition[ foot ].X));
  ProjectedPosition.Y := ROUND(ScaleP*(YC + ProjectedFootPosition[ foot ].Y));
  ProjectedPosition.Z := 0;

  { include offsets to center the X-symbol on foot }
  XT := ProjectedPosition.X - 4;
  YT := ProjectedPosition.Y - 6;

  IF foot IN SupportFeet
    THEN { put X for foot on ground }
        TEXT( XT, YT, Fsize, Nfile3, FNchar, Ftext )
    ELSE { foot not on ground, put blank instead of X }
        TEXT( XT, YT, Fsize, Nfile3, FNchar, Btext );
```

```
  { plot boxes around critical feet }
  IF foot IN CriticalFeet
    THEN
        BEGIN
        Box[1,1] := ProjectedPosition.X + 7;
        Box[1,2] := ProjectedPosition.Y + 7;
        Box[2,1] := ProjectedPosition.X - 7;
        Box[2,2] := ProjectedPosition.Y + 7;
        Box[3,1] := ProjectedPosition.X - 7;
        Box[3,2] := ProjectedPosition.Y - 7;
        Box[4,1] := ProjectedPosition.X + 7;
        Box[4,2] := ProjectedPosition.Y - 7;
        Box[5,1] := Box[1,1];
        Box[5,2] := Box[1,2];

        { plot this box }
        Npoints := 5;
        Hpplot( Box, Npoints, Nfile3 );
        END
    ELSE
        BEGIN
        FOR I := 1 TO 5 Do
          BEGIN
          Box[I,1] := 0;
          Box[I,2] := 0;
          END;
        Npoints := 5;
        Hpplot( Box, Npoints, Nfile3 );
        END;


  IF foot IN PolygonFeet
    THEN { save coordinates in array to draw polygon }
        BEGIN
        Polygon[ PolygonSides, 1 ] := ProjectedPosition.X;
        Polygon[ PolygonSides, 2 ] := ProjectedPosition.Y;
        PolygonSides := PolygonSides + 1;
        END;

  END;  { DO }

{ Connect last side of polygon and                    }
{    fill in rest of array by repeating last point. }
FOR K := PolygonSides TO 7 DO
  BEGIN
  Polygon[K,1] := Polygon[1,1];
  Polygon[K,2] := Polygon[1,2];
  END;
FOR K := 1 TO NumberOfTraces DO          { plot more than once for contrast }
  BEGIN  { plot the polygon }
  Npoints := 7;
  Hpplot( Polygon, Npoints, Nfile3 );
  END;

END;    { PlotSupportPolygon }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMEXTERN9;     { External procedure declarations }
%INCLUDE EXTPROJ3;      { JUST FOR TYPE DEFINITIONS FOR EXTPROJHP}
%INCLUDE EXTPROJHP;

{*****************************************************************}
{*              FileName: OPTSTAB1.PAS                           *}
{* Function: Moves center-of-gravity of body to the              *}
{*              optimally stable point.                          *}
{*****************************************************************}

{*** STATIC VARIABLE DECLARATION ***}
VAR     StabilizeState          : StabilizeMode;
        OptimumPoint            : Vector;
        PresentCG               : Vector;

PROCEDURE OptimizeStability{( VAR OptimizeStabilityFlag : Completion;
                                  FootPhase             : SixStates;
                                  DesiredFootPosition   : SixVectors;
                                  Pitch                 : Real;
                                  Roll                  : Real;
                                  dt                    : Real;
                              VAR LastTime              : Real;
                              VAR PlannedFootRate       : SixVectors;
                              VAR FootRateType          : SixRateTypes;
                              VAR DesiredBodyLinearRates: Vector;
                              VAR DesiredBodyAngularRates: Vector;
                              VAR DesiredBodyTransform  : Homogeneous )};

CONST   MaxLinearRate = 0.6;    { inches per second }
        MaxError = 0.25;        { inches }

        XC = 170.666;
        YC = 233.333;
        ScaleP = 3.0;

VAR     foot            : Integer;
        DeltaPosition   : Vector;
        Distance        : Real;
        MotionDirection : Vector;
        CenterOfGravity : Vector;
        Offset          : Vector;
        SaveTime        : Real;
        NewTime         : Real;
        DeltaTime       : Real;
        BodyLimitFlag   : Boolean;

        FNchar,Nfile3,Fsize : Integer;
        XT, YT          : Integer;
        Ftext           : Char;

BEGIN   { OptimizeStability }

{ Initial location of CenterOfGravity }
CenterOfGravity.X := 0.0;
CenterOfGravity.Y := 0.0;
CenterOfGravity.Z := 0.0;
```

```
CASE StabilizeState OF

  Stop: HaltVehicle( dt,
                     PlannedFootRate,
                     FootRateType,
                     DesiredBodyLinearRates,
                     DesiredBodyAngularRates,
                     DesiredBodyTransform );


  Calculate: BEGIN
             SaveTime := Time;
             FindOptimumPoint( FootPhase,
                               DesiredFootPosition,
                               Pitch,
                               Roll,
                               OptimumPoint );
             NewTime := Time;
             DeltaTime := NewTime - SaveTime;
             { update value of LastTime, for use in Main program }
             LastTime := LastTime + DeltaTime;
             PresentCG := CenterOfGravity;
             HaltVehicle( dt,
                          PlannedFootRate,
                          FootRateType,
                          DesiredBodyLinearRates,
                          DesiredBodyAngularRates,
                          DesiredBodyTransform );

       Ftext := 'X';
       Fsize := 0;
       FNchar := 1;
       Nfile3 := 3;
       XT := ROUND( ScaleP*(OptimumPoint.X - PresentCG.X + XC )) - 4;
       YT := ROUND( ScaleP*(OptimumPoint.Y - PresentCG.Y + YC )) - 6;
       TEXT(XT, YT, Fsize, Nfile3, FNchar, Ftext);

            END;  { Calculate }


  MoveBody: BEGIN
            VectSub( DeltaPosition, OptimumPoint, PresentCG );
            Distance := Magnitude( DeltaPosition );
            END;  { MoveBody }

END; { CASE StabilizeState }


{*** DETERMINE NEW STATE ***}

CASE StabilizeState OF

  Stop: BEGIN
        StabilizeState := Calculate;
        OptimizeStabilityFlag := NotDone;
        END;  { Stop }


  Calculate: StabilizeState := MoveBody;
```

```
MoveBody: BEGIN
          IF Distance > MaxError
            THEN
              BEGIN
              VectDiv( MotionDirection, DeltaPosition, Distance );
              VectMult( DesiredBodyLinearRates, MaxLinearRate,
                                MotionDirection );
              { DesiredBodyAngularRates := Zero; }
              { find location of center-of-gravity WRT original position}
              TransIntegrate( DesiredBodyTransform,
                              DesiredBodyLinearRates,
                              DesiredBodyAngularRates,
                              dt );
              VectMult( Offset, dt, DesiredBodyLinearRates );
              VectAdd( PresentCG, PresentCG, Offset );
              PresentCG.Z := 0.0;

              Ftext := 'X';
              Fsize := 0;
              FNchar := 1;
              Nfile3 := 3;
              XT := ROUND( ScaleP*(OptimumPoint.X - PresentCG.X + XC )) - 4;
              YT := ROUND( ScaleP*(OptimumPoint.Y - PresentCG.Y + YC )) - 6;
              TEXT(XT, YT, Fsize, Nfile3, FNchar, Ftext);

              FOR foot := 1 TO 6 DO
                IF FootPhase[ foot ] = Support
                  THEN FootRateType[ foot ] := Relative
                  ELSE FootRateType[ foot ] := Absolute;
                { in either case, leave PlannedFootRate's = Zero }

              BodyLimit( FootPhase,
                         DesiredFootPosition,
                         dt,
                         PlannedFootRate,
                         FootRateType,
                         DesiredBodyLinearRates,
                         DesiredBodyAngularRates,
                         DesiredBodyTransform,
                         BodyLimitFlag );
              IF BodyLimitFlag
                THEN
                  BEGIN
                  StabilizeState := Stop;
                  OptimizeStabilityFlag := Done;
                  END;

              END
            ELSE
              BEGIN
              StabilizeState := Stop;
              OptimizeStabilityFlag := Done;
              END;
          END;  { MoveBody }

END;  { CASE StabilizeState }

END; { OptimizeStability }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMEXTERN2;     { External Procedure declarations }
%INCLUDE EXTPROJ3;      { External procedure declarations for projection }

{*****************************************************************}
{*              FileName: OPTPOINT.PAS                           *}
{* Function: Finds the optimally stable point,                   *}
{*              given the location of the feet.                  *}
{*****************************************************************}


{*** STATIC VARIABLE DECLARATIONS ***}
VAR     { Variables for OPTPOINT.PAS }
        PolygonVerticies                : SevenVectors;
        NumberOfVerticies               : Integer;
        ProjectedCG                     : Vector;
        VerticalRotationTransform       : Matrix;
        UpwardZHat                      : Vector;
        VectorWHat                      : Vector;
        VectorAHat                      : SixVectors;
        VectorPHat                      : SixVectors;
        VectorTHat                      : SixVectors;
        ConstantM                       : Array6;
        CosP, SinP                      : Real;
        CosTh, SinTh                    : Real;
        STHSP, STHCP                    : Real;



PROCEDURE COPInitialize( FootPhase              : SixStates;
                         DesiredFootPosition    : SixVectors;
                         Pitch                  : Real;
                         Roll                   : Real );

{* Function:    Sets values of PolygonVerticies, NumberOfVerticies, *}
{*              ProjectedCG, VerticalRotationTransform,              *}
{*              UpwardZHat, VectorWHat                               *}

VAR     ZeroYaw                 : Real;
        foot                    : Integer;
        K                       : Integer;
        CenterOfGravity         : Vector;
        SupportFeet             : LegSet;
        PolygonFeet             : LegSet;
        ProjectedFootPosition   : SixVectors;

BEGIN { COPInitialize }

{ Set value of UpwardZHat }
{ NOTE: for ASV-84, z-axis is upward }
UpwardZHat.X := 0.0;
UpwardZHat.Y := 0.0;
UpwardZHat.Z := 1.0;

VectorWHat.X := -UpwardZHat.X;
VectorWHat.Y := -UpwardZHat.Y;
VectorWHat.Z := -UpwardZHat.Z;
```

```
{ Calculate Rotation Transformation to "Earth Based" x,y,z system }
ZeroYaw := 0.0;
GetOrientation( ZeroYaw,
                Pitch,
                Roll,
                VerticalRotationTransform );

{ Transform location of each foot to "Earth Based" coordinates }
SupportFeet := [];      { empty set }
FOR foot := 1 TO 6 DO
  BEGIN
  MatrixMult( ProjectedFootPosition[ foot ],
              VerticalRotationTransform,
              DesiredFootPosition[ foot ] );

  IF ( FootPhase[ foot ] = Support )
     THEN SupportFeet := SupportFeet + [ foot ];

  END;  { DO }

CheckRedundantFeet( ProjectedFootPosition,
                    SupportFeet,
                    PolygonFeet );

{ create an array of polygon verticies }
NumberOfVerticies := 1;
FOR K := 1 TO 6 DO
  BEGIN
  { select proper feet in order 1,3,5,6,4,2 }
  IF ( K <= 3 )
    THEN foot := 2*K - 1
    ELSE foot := 14 - 2*K;

  IF foot IN PolygonFeet
    THEN
        BEGIN
        PolygonVertices[ NumberOfVerticies ] := ProjectedFootPosition[ foot ];
        NumberOfVerticies := NumberOfVerticies + 1;
        END;
END; { DO }

{ last polygon vertex same as first polygon vertex }
PolygonVertices[ NumberOfVerticies ] := PolygonVertices[1];

{ Initial location of Center of Gravity }
CenterOfGravity.X := 0.0;
CenterOfGravity.Y := 0.0;
CenterOfGravity.Z := 0.0;

{ Transform location of center of gravity to "Earth Based" x,y,z system }
MatrixMult( ProjectedCG, VerticalRotationTransform, CenterOfGravity );

END; { COPInitialize }



PROCEDURE COPConstants( Pitch   : Real;
                        Roll    : Real );

{* Function: Sets values of VectorAHat, VectorPHat, VectorTHat,   *}
{*              ConstantM, CosP, SinP, CosTh, SinTh, STHSP, STHCP *}

VAR     K               : Integer;
        VectorA         : Vector;
        VectorP         : Vector;
```

```
BEGIN   { COPConstants }
FOR K := 1 TO ( NumberOfVerticies - 1 ) DO
  BEGIN
  { find vector A for edge of polygon from point F1 to point F2 }
  VectSub( VectorA, PolygonVertices[ K+1 ], PolygonVertices[ K ] );
  { find unit vector of vectorA }
  VectNorm( VectorAHat[ K ], VectorA );
  { find vector P, inward pointing orthogonal to the vertical plane }
  CrossProduct( VectorP, VectorAHat[ K ], VectorWHat );
  { find unit normal of vectorP }
  VectNorm( VectorPHat[ K ], VectorP );
  { find unit vector THat, in the plane }
  CrossProduct( VectorTHat[ K ], VectorAHat[ K ], VectorPHat[ K ] );
  { find value of the ConstantM }
  ConstantM[ K ] := DOT( VectorTHat[ K ], VectorAHat[ K ] );
  END;

{ evaluate needed Trig functions }
CosP := COS( Pitch );
SinP := SIN( Pitch );
CosTh := COS( Roll );
SinTh := SIN( Roll );
STHSP := SinTh * SinP;
STHCP := SinTh * CosP;

END;    { COPConstants }



PROCEDURE EnergyGradient(   ABcoordinates       : Vector;
                        VAR Hmin1               : Real;
                        VAR Hmin2               : Real;
                        VAR dHdA                : Array2;
                        VAR dHdB                : Array2 );

{* Function: Given the coordinates of a point in the A,B plane, *}
{*              find the 2 smallest values of Energy Stability  *}
{*              at that point, and the gradients                *}

LABEL 20, 30, 40, 50;

VAR     K, I                    : Integer;
        TransformedABcoord      : Vector;
        NewCGLocation           : Vector;
        VectorB, VectorC        : Vector;
        VectorR, VectorRHat     : Vector;
        MagR,  NewH             : Real;
        FirstTime, SecondTime   : Boolean;
        MinRHat, MinAHat        : Array[1..2] of Vector;
        MinTHat                 : Array[1..2] of Vector;
        MinConstantM, MinMagR   : Array2;
        dTRdA, dTRdB            : Real;
        dRXdA, dRYdA, dRZdA     : Real;
        dRXdB, dRYdB, dRZdB     : Real;

BEGIN   { EnergyGradient }

FirstTime := True;
SecondTime := True;

FOR K := 1 TO ( NumberOfVerticies - 1) DO
  BEGIN
  { Transform ABcoordinates from (A,B) body coordinates to }
  { (x,y,z) "Earth Based" coordinates                      }
  MatrixMult(TransformedABcoord,VerticalRotationTransform,ABcoordinates);
```

```
  { find new location of center-of-gravity }
  VectAdd( NewCGLocation, ProjectedCG, TransformedABcoord );
  { find vector B, from point F1 to CG }
  VectSub( VectorB, NewCGLocation, PolygonVerticies[ K ] );
  { find vector C, orthogonal projection of B in direction of A }
  VectMult( VectorC, DOT( VectorAHat[K], VectorB ), VectorAHat[K] );
  { find vector R, from CG perpendicular to line F1F2 }
  VectSub( VectorR, VectorB, VectorC );
  { find unit vector of vector R }
  VectNorm( VectorRHat, VectorR );
  { find magnitude of vector R }
  MagR := Magnitude( VectorR );
  { calculate the change in vertical height }
  NewH := MagR * ( 1.0 - DOT( VectorRHat, VectorTHat[K] ) )
              * ( DOT( VectorTHat[K], UpwardZHat ) );

  { check if this is first time through loop }
  IF FirstTime
    THEN GoTo 30;
  { check if this is second time through loop }
  IF SecondTime
    THEN
        BEGIN
        IF ( NewH > Hmin1 )
          THEN GoTo 40
          ELSE GoTo 20;
        END;
  { neither first nor second time through loop }
  IF ( NewH > Hmin2 )
    THEN GoTo 50;        { NewH is bigger than both Hmin1 and Hmin2 }
  IF ( NewH > Hmin1 )
    THEN GoTo 40;        { Hmin1 < NewH <= Hmin2 }
  { Otherwise, NewH is less than both Hmin1 and Hmin2 }

20: { Shift first place to second place }
  SecondTime := False;
  Hmin2 := Hmin1;
  MinRHat[2] := MinRHat[1];
  MinMagR[2] := MinMagR[1];
  MinAHat[2] := MinAHat[1];
  MinTHat[2] := MinTHat[1];
  MinConstantM[2] := MinConstantM[1];

30: { Put new stuff in first place }
  FirstTime := False;
  Hmin1 := NewH;
  MinRHat[1] := VectorRHat;
  MinMagR[1] := MagR;
  MinAHat[1] := VectorAHat[K];
  MinTHat[1] := VectorTHat[K];
  MinConstantM[1] := ConstantM[K];
  GoTo 50;

40: { Put new stuff in second place }
  SecondTime := False;
  Hmin2 := NewH;
  MinRHat[2] := VectorRHat;
  MinMagR[2] := MagR;
  MinAHat[2] := VectorAHat[K];
  MinTHat[2] := VectorTHat[K];
  MinConstantM[2] := ConstantM[K];

50: END;   { DO K }
```

```
FOR I := 1 TO 2 DO
  BEGIN
  { Evaluate the partial derivatives of THat DOT R }
  dTRdA := CosP * ( MinTHat[I].X - MinConstantM[I]*MinAHat[I].X )
         -SinP * ( MinTHat[I].Z - MinConstantM[I]*MinAHat[I].Z );
  dTRdB := STHSP * ( MinTHat[I].X - MinConstantM[I]*MinAHat[I].X )
         +CosTH * ( MinTHat[I].Y - MinConstantM[I]*MinAHat[I].Y )
         +STHCP * ( MinTHat[I].Z - MinConstantM[I]*MinAHat[I].Z );

  { Evaluate partial derivatives of Rx, Ry, Rz WRT A and B }
  dRXdA := CosP - MinAHat[I].X*MinAHat[I].X*CosP
                + MinAHat[I].X*MinAHat[I].Z*SinP;
  dRYdA :=  - MinAHat[I].X*MinAHat[I].Y*CosP
            + MinAHat[I].Y*MinAHat[I].Z*SinP;
  dRZdA := -SinP - MinAHat[I].X*MinAHat[I].Z*CosP
                 + MinAHat[I].Z*MinAHat[I].Z*SinP;
  dRXdB := STHSP - MinAHat[I].X*MinAHat[I].X*STHSP
                 - MinAHat[I].X*MinAHat[I].Y*CosTh
                 - MinAHat[I].X*MinAHat[I].Z*STHCP;
  dRYdB := CosTh - MinAHat[I].X*MinAHat[I].Y*STHSP
                 - MinAHat[I].Y*MinAHat[I].Y*CosTh
                 - MinAHat[I].Y*MinAHat[I].Z*STHCP;
  dRZdB := STHCP - MinAHat[I].X*MinAHat[I].Z*STHSP
                 - MinAHat[I].Y*MinAHat[I].Z*CosTh
                 - MinAHat[I].Z*MinAHat[I].Z*STHCP;

  { Evaluate partial derivatives of H WRT A and B }
  dHdA[I] := MinTHat[I].Z * ( ( MinRHat[I].X*dRXdA + MinRHat[I].Y*dRYdA
                + MinRHat[I].Z * dRZdA ) / MinMagR[I]  -  dTRdA );
  dHdB[I] := MinTHat[I].Z * ( ( MinRHat[I].X*dRXdB + MinRHat[I].Y*dRYdB
                + MinRHat[I].Z * dRZdB ) / MinMagR[I]  -  dTRdB );

  END;  { DO I }

END;  { EnergyGradient }



PROCEDURE FindOptimumPoint(( FootPhase                : SixStates;
                             DesiredFootPosition      : SixVectors;
                             Pitch                    : Real;
                             Roll                     : Real;
                         VAR OptimumPoint             : Vector )};

CONST   Delta = 0.1;     { segment length, in inches }
        EpsMax = 0.5;    { width of blending boundary }
        Change = 0.001; { criterion for stopping }


VAR     Hmin1, Hmin2                  : Real;
        dHdA, dHdB                    : Array2;
        PastMin                       : Array[1..5] of Real;
        J                             : Integer;
        ABcoordinates                 : Vector;
        GradNorm1, GradNorm2          : Real;
        Del1, Del2, Del3, Del4        : Real;
        B, Epsilon                    : Real;


BEGIN   { FindOptimumPoint }

IF Pitch < 0.035                { 2 degrees }
        THEN Pitch := 0.0;
IF Roll  < 0.035
        THEN Roll  := 0.0;
```

```
COPInitialize( FootPhase,
               DesiredFootPosition,
               Pitch,
               Roll  );

FOR J := 1 TO 6 DO
Writeln(DesiredFootPosition[J].X,DesiredFootPosition[J].Y,
                          DesiredFootPosition[J].Z);
Writeln('pitch=',pitch);
Writeln('roll=',roll);
FOR J := 1 TO NumberOfVerticies DO
Writeln(PolygonVerticies[J].X,PolygonVerticies[J].Y,PolygonVerticies[J].Z);
Writeln;

COPConstants( Pitch, Roll );

{ call EnergyGradient to find initial Hmin1 }
EnergyGradient( ProjectedCG, Hmin1  Hmin2, dHdA, dHdB );
{ initialize array of past minimum energies }
PastMin[1] := Hmin1;
{ assign the other values to guarantee initial values will not stop looping}
FOR J := 1 TO 4 DO
  PastMin[ J+1 ] := PastMin[ J ] - 2.0*Change;

ABcoordinates := ProjectedCG;

REPEAT  { UNTIL ( ( PastMin[1] - PastMin[5] ) < Change ) }
  EnergyGradient( ABcoordinates, Hmin1, Hmin2, dHdA, dHdB );
  { save Hmin1, and its past 4 values }
  FOR J := 1 TO 4 DO
    PastMin[ 6-J ] := PastMin[ 5-J ];
  PastMin[1] := Hmin1;

  { calculate norms of the gradient }
  GradNorm1 := SQRT( dHdA[1]*dHdA[1] + dHdB[1]*dHdB[1] );
  GradNorm2 := SQRT( dHdA[2]*dHdA[2] + dHdB[2]*dHdB[2] );

  { calculate distance to move in A,B directions }
  DelA1 := Delta * dHdA[1] / GradNorm1;
  DelB1 := Delta * dHdB[1] / GradNorm1;
  DelA2 := Delta * dHdA[2] / GradNorm2;
  DelB2 := Delta * dHdB[2] / GradNorm2;

  { check if blending should be performed }
  Epsilon := Hmin2 - Hmin1;
  IF ( Epsilon > EpsMax )
    THEN { no blending }
        B := 0.0
    ELSE { use blending }
        B := 0.5 - 0.5 * Epsilon / EpsMax;

  { calculate new A,B coordinates, keep Z-component constant }
  ABcoordinates.X := ABcoordinates.X + ( 1.0 - B )*DelA1 + B*DelA2;
  ABcoordinates.Y := ABcoordinates.Y + ( 1.0 - B )*DelB1 + B*DelB2;
  { ABcoordinates.Z := ProjectedCG.Z }


UNTIL ( PastMin[1] - PastMin[5] ) < Change;

{ Set OptimumPoint equal to last value of ABcoordinates }
OptimumPoint := ABcoordinates;

END;    { FindOptimumPoint }
```

```
{$nomain}
{$own}

%INCLUDE DMTYPE41;      { Type declarations }
%INCLUDE DMEXTERN2;     { External procedure declarations }
%INCLUDE EXTPROJ3;      { External procedure declarations for projection }
%INCLUDE EXTPROJHP;     { HP NonPascal procedure declarations }

{*****************************************************************}
{*            FileName: STABILHP.PAS                             *}
{* Function: Compute the Energy Stability Margin for each        *}
{*           leg of the support polygon and display it           *}
{*           on screen.                                          *}
{*****************************************************************}

PROCEDURE StabilityMargin(( ProjectedFootPosition      : SixVectors;
                            PolygonFeet                : LegSet;
                            ProjectedCG                : Vector )};

CONST   ScaleP = 3.0;           { scale for projected view }
        ScaleS = 10.0;          { scale for stability margin }
        XC = 170.666;           { center of projected view, 512.0/ScaleP }
        YC = 233.333;           { center of projected view, 700.0/ScaleP }

VAR     PolygonVerticies        : SevenVectors;
        NumberOfVerticies       : Integer;
        K                       : Integer;
        foot                    : Integer;
        VectorA, VectorAHat     : Vector;
        VectorB, VectorC        : Vector;
        VectorR, VectorRHat     : Vector;
        MagR                    : Real;
        UpwardZHat, VectorWHat  : Vector;
        VectorP, VectorPHat     : Vector;
        VectorTHat              : Vector;
        Height                  : Real;
        ZeroPt                  : Integer;
        MidpointA               : Real;
        Coord                   : M101x2;       { really only need 2x2 }
        Npoints2, Nfile3        : Integer;

BEGIN   { StabilityMargin }

Npoints2 := 2;
Nfile3   := 3;   { use file #3 }

ZeroPt := 0;

{ set value of UpwardZHat }
{ NOTE: for ASV84, z-axis is upward }
UpwardZHat.X := 0.0;
UpwardZHat.Y := 0.0;
UpwardZHat.Z := 1.0;

VectorWHat.X := -UpwardZHat.X;
VectorWHat.Y := -UpwardZHat.Y;
VectorWHat.Z := -UpwardZHat.Z;

{ create array of polygon verticies }
NumberOfVerticies := 1;
FOR K := 1 TO 6 DO
  BEGIN
  { select proper feet in order 1,3,5,6,4,2 }
  IF ( K <= 3 )
```

```
    THEN foot := 2*K - 1
    ELSE foot := 14 - 2*K;

  IF foot IN PolygonFeet
    THEN
      BEGIN
      PolygonVertices[ NumberOfVertices ] := ProjectedFootPosition[ foot ];
      NumberOfVertices := NumberOfVertices + 1;
      END;

  END;

{ last polygon vertex same as first vertex }
PolygonVertices[ NumberOfVertices ] := PolygonVertices[ 1 ];


FOR K := 1 TO ( NumberOfVertices - 1 ) DO
  BEGIN
  { find vector A for edge of polygon, from point F1 to point F2 }
  VectSub( VectorA, PolygonVertices[ K+1 ], PolygonVertices[ K ] );
  { find unit vector of VectorA }
  VectNorm( VectorAHat, VectorA );
  { find vector B, from point F1 to CG }
  VectSub( VectorB, ProjectedCG, PolygonVertices[ K ] );
  { find vector C, orthogonal projection of B in direction of A }
  VectMult( VectorC, DOT( VectorAHat, VectorB ), VectorAHat );
  { find vector R, from CG perpendicular to line F1F2 }
  VectSub( VectorR, VectorB, VectorC );
  { find unit vector of vector R }
  VectNorm( VectorRHat, VectorR );
  { find magnitude of vector R }
  MagR := Magnitude( VectorR );
  { find vector P, inward pointing orthogonal to the vertical plane }
  CrossProduct( VectorP, VectorAHat, VectorWHat );
  { find unit vector of vector P }
  VectNorm( VectorPHat, VectorP );
  { find unit vector THat,  in the plane }
  CrossProduct( VectorTHat, VectorAHat, VectorPHat );
  { calculate change in vertical height }
  Height := MagR * ( 1.0 - DOT( VectorRHat, VectorTHat ) )
                 * ( DOT( VectorTHat, UpwardZHat ) );

  { draw a vector at leg K in projected view }
  MidpointA := 0.5 * Magnitude( VectorA );
  Coord[1,1] := ROUND( ScaleP * ( XC + PolygonVertices[ K ].X
                                 + MidpointA * VectorAHat.X ) );
  Coord[1,2] := ROUND( ScaleP * ( YC + PolygonVertices[ K ].Y
                                 + MidpointA * VectorAHat.Y ) );
  { draw vector in opposite direction of VectorPHat }
  Coord[2,1] := Coord[1,1] + ROUND( ScaleS * Height * ( -VectorPHat.X ) );
  Coord[2,2] := Coord[1,2] + ROUND( ScaleS * Height * ( -VectorPHat.Y ) );
  Hpplot( Coord, Npoints2, Nfile3 );

  END;  { DO }

{ zero out unused vector locations }
Coord[1,1] := 0;
Coord[1,2] := 0;
Coord[2,1] := 0;
Coord[2,2] := 0;
FOR K := NumberOfVertices TO 7 DO
  Hpplot( Coord, Npoints2, Nfile3 );

END;    { StabilityMargin }
```

```
        ;HPPLOT.MAC
        ;DOMINIC MESSURI                    2-19-81
        ;REVISED                            6-29-81
        ;THIS SUBROUTINE OUTPUTS AN ARRAY OF DATA TO
        ;THE HP GRAPHICS TRANSLATOR
        ;CALL STATEMENT; CALL HPPLOT(ARRAY,NPTS,FILE)
        ;                CALL INITZ
        ;                CALL STOVER(FILE)
        ;                CALL BLANK(FILE)
        ;                CALL UBLANK(FILE)
        ;                CALL ERASE(FILE)
        ;                CALL TEXT(XPAGE,YPAGE,SIZE,FILE,NCHAR,'TEXT')

        ;               CALL NUM(XPAGEN,YPAGEN,SIZEN,FILEN,NDPT,N)
        .LIST TTM
        GTS = 167760    ;GRAPHICS TRANSLATOR STATUS
        GTB = 167762    ;GRAPHICS TRANSLATOR BUFFER
        EOD = 010000    ;END OF DATA MARKER
        ;MACRO TO OUTPUT A FILE OF DATA TO THE HP I/O DEVICE
        .MACRO OUTPUT FILE,?LOOP,?GWAIT,?ZERO
        MOV FILE,R0     ;MOVE ADDRESS OF FILE INTO R0
LOOP:   CMP(R0),#EOD    ;CHECK IF END OF DATA

        BEQ ZERO
GWAIT:  TSTB @#GTS      ;CHECK GRAPHICS TRANSLATOR STATUS
        BPL GWAIT       ;IF BIT 7 IS 1,GT IS READY
        BIC #002,@#GTS  ;CLEAR BIT 1 (DEVCMD)
        MOV (R0)+,@#GTB ;MOVE DATA WORD INTO OUTPUT BUFFER
        BIS #002,@#GTS  ;SET BIT 1 TO SIGNAL DATA AVAILABLE

        BR LOOP
ZERO:   NOP
        .ENDM
        ;TO INITIALIZE THE GRAPHICS TRANSLATOR
INITZ:: OUTPUT  #INIT
        RTS     PC
INIT:   .WORD 146000,144000,101000,102000
        .WORD 100000,112000,110000,115000
        .WORD 010000
        ;TO MOVE POINTER TO BEGINNING OF FILE
STOVER::        TST(R5)+
        MOV @(R5)+,R1   ;R1 CONTAINS FILE NUMBER
        ADD #133000,R1  ;FIND FILE
        MOV R1,@#OVER
        OUTPUT #OVER    ;OUTPUT "FIND FILE (#)" COMMAND

        RTS PC
OVER:   .WORD 0,010000
        ;TO BLANK A FILE
BLANK::         TST(R5)+
        MOV @(R5)+,R1   ;R1 CONTAINS FILE NUMBER
        ADD #123000,R1  ;BLANK FILE
        MOV R1,@#BLNK
        OUTPUT #BLNK    ;OUTPUT "BLANK FILE (#)" COMMAND
        RTS PC
BLNK:   .WORD 0,010000
        ;TO UNBLANK A FILE
UBLANK::        TST(R5)+
        MOV @(R5)+,R1   ;R1 CONTAINS FILE NUMBER
        ADD #117000,R1  ;UNBLANK FILE
        MOV R1,@#UBLNK
        OUTPUT #UBLNK   ;OUTPUT "UNBLANK FILE (#)" COMMAND
        RTS PC
UBLNK:  .WORD 0,010000
```

```
        ;TO ERASE A FILE
ERASE:: TST(R5)+
        MOV @(R5)+,R1    ;R1 CONTAINS FILE NUMBER
        ADD #103000,R1   ;ERASE FILE
        MOV R1,@#ERAS
        OUTPUT #ERAS     ;OUTPUT "ERASE FILE (#)" COMMAND
        RTS PC
ERAS:   .WORD 0,010000
HPPLOT:: TST (R5)+
        MOV (R5)+,R1     ;ADDRESS OF ARRAY IN R1
        MOV @(R5)+,R2    ;VALUE OF NPTS IN R2
        MOV @(R5)+,R3    ;VALUE OF FILE NUMBER IN R3
        MOV #HPDATA,R4   ;ADDRESS OF DATA BLOCK IN R4
        MOV R3,(R4)      ;MOVE FILE NUMBER TO R4
        ADD #107000,(R4)+        ;"NAME FILE (#)"
        MOV #143000,(R4)+        ;PLOT USING PEN CONTROL
        MOV #141000,(R4)+        ;"PEN UP",TO MOVE TO FIRST LOCATION
        MOV (R1)+,(R4)+ ;STORE X1 IN HPDATA
        MOV (R1)+,(R4)+ ;STORE Y1 IN HPDATA
        MOV #141001,(R4)+        ;"PEN DOWN" (BEGIN TO DRAW)
        DEC R2           ;NPTS - 1
        BEQ END          ;IF ONLY ONE DATA POINT,STOP
LOOP:   MOV (R1)+,(R4)+ ;STORE XN IN HPDATA
        MOV (R1)+,(R4)+ ;STORE YN IN HPDATA
        SOB R2,LOOP
END:    MOV #112000,(R4)+        ;"STOP NAMING FILE"
        MOV #EOD,(R4)+   ;MARK END OF DATA
        OUTPUT #HPDATA   ;OUTPUT THE DATA BLOCK TO HP
        RTS PC
HPDATA: .BLKW 3000.      ;DATA BLOCK STORAGE AREA
        ;TO PRINT TEXT
TEXT::  TST(R5)+
        MOV @(R5)+,@#XPAGE       ;VALUE OF XPAGE
        MOV @(R5)+,@#YPAGE       ;VALUE OF YPAGE
        MOV @(R5)+,@#SIZE        ;VALUE OF SIZE
        MOV @(R5)+,@#FILE        ;FILE NUMBER
        MOV @(R5)+,R2            ;VALUE OF NOS. OF CHARACTERS IN R2
        MOV (R5)+,R3             ;ADDRESS OF FIRST CHAR IN R3
        MOV #TXT,R4              ;ADDRESS OF TEXT BLOCK IN R4
        MOV @#FILE,(R4)          ;MOVE FILE NUMBER TO R4
        ADD #107000,(R4)+        ;"NAME FILE (#)"
        MOV #143000,(R4)+        ;PLOT USING PEN CONTROL
        MOV #141000,(R4)+        ;"PEN UP",TO MOVE TO FIRST LOCATION
        MOV @#XPAGE,(R4)+        ;STORE XPAGE IN TXT
        MOV @#YPAGE,(R4)+        ;STORE YPAGE IN TXT
        MOV #141001,(R4)+        ;"PEN DOWN" (BEGIN TO DRAW)
        MOV @#SIZE,(R4)          ;MOVE SIZE TO R4
        ADD #145000,(R4)+        ;"CHARACTER SIZE AND ROTATION"
        MOV #140000,(R4)+        ;"TEXT"
LOOP2:  MOVB (R3)+,(R4)          ;STORE CHARACTER IN TXT
        BIC #177600,(R4)+        ;CLEAR PARITY BIT
        SOB R2,LOOP2
        MOV #146000,(R4)+        ;"END OF TEXT"
        MOV #112000,(R4)+        ;"STOP NAMING FILE"
        MOV #EOD,(R4)+           ;MARK END OF DATA
        OUTPUT #TXT              ;OUTPUT THE TEXT BLOCK TO HP
        RTS PC
XPAGE:  .WORD 0
YPAGE:  .WORD 0
SIZE:   .WORD 0
FILE:   .WORD 0
TXT:    .BLKW 1000.              ;TEXT BLOCK STORAGE AREA
```

```
        ;TO PRINT NUMBER
NUM::           TST(R5)+
        MOV @(R5)+,@#XPAGEN     ;VALUE OF XPAGE
        MOV @(R5)+,@#YPAGEN     ;VALUE OF YPAGE
        MOV @(R5)+,@#SIZEN      ;VALUE OF SIZE
        MOV @(R5)+,@#FILEN      ;FILE NUMBER
        MOV @(R5)+,R2           ;VALUE OF DPT IN R2
        MOV (R5)+,R3            ;ADDRESS OF FIRST DIGIT IN R3
        MOV #7,@#COUNT          ;INITIALIZE COUNT
        MOV #NOS,R4             ;ADDRESS OF NUMBER BLOCK IN R4
        MOV @#FILE,(R4)         ;MOVE FILE NUMBER TO R4
        ADD #107000,(R4)+       ;"NAME FILE (#)"
        MOV #143000,(R4)+       ;PLOT USING PEN CONTROL
        MOV #141000,(R4)+       ;"PEN UP" TO MOVE TO FIRST LOCATION
        MOV @#XPAGEN,(R4)+      ;STORE XPAGE IN NOS
        MOV @#YPAGEN,(R4)+      ;STORE YPAGE IN NOS
        MOV #141001,(R4)+       ;"PEN DOWN" (BEGIN TO DRAW)
        MOV @#SIZEN,(R4)        ;MOVE SIZE TO NOS
        ADD #145000,(R4)+       ;"CHARACTER SIZE AND ROTATION"
        MOV #140000,(R4)+       ;"TEXT"
LOOP3:  TST R2                  ;CHECK IF DPT IS ZERO
        BNE SKIP1
        MOV #60,(R4)+           ;IF DPT = 0, MOVE ZERO IN NOS
        MOV #56,(R4)+           ;IF DPT = 0, MOVE DECIMAL PT IN NOS
        BR SKIP2
SKIP1:  TST (R3)                ;CHECK IF LEADING DIGITS ARE ZERO
        BNE SKIP2
        TST (R3)+               ;AUTOINCREMENT, TO SKIP LEADING ZERO
        DEC COUNT
        DEC R2
        BNE SKIP1               ;IGNORE LEADING ZERO DIGITS
        MOV #60,(R4)+           ;PUT ZERO IN NOS
        MOV #56,(R4)+           ;PUT DECIMAL PT IN NOS
SKIP2:  MOV (R3)+,(R4)          ;MOV DIGIT TO NOS
        ADD #60,(R4)+           ;CONVERT TO ASCII
        DEC R2
        BNE SKIP3
        MOV #56,(R4)+           ;PUT IN DECIMAL PT
SKIP3:  DEC COUNT
        BNE SKIP2               ;CHECK IF ALL DIGITS CONVERTED

        MOV #146000,(R4)+       ;'END OF TEXT'
        MOV #112000,(R4)+       ;'STOP NAMING FILE'
        MOV #EOD,(R4)+          ;MARK END OF DATA
        OUTPUT #NOS             ;OUTPUT THE NUMBER TO THE HP
        RTS PC
XPAGEN: .WORD 0
YPAGEN: .WORD 0
SIZEN:  .WORD 0
FILEN:  .WORD 0
COUNT:  .WORD 0
NOS:    .BLKW 50.               ;NUMBER STORAGE AREA
        .END
```

REFERENCES

[1] Bekker, M.G., "Is the Wheel the Last Word in Land Locomotion?," New Scientist, No. 248, August 19, 1961.

[2] McGhee, R.B., "Vehicular Legged Locomotion," in Advances in Automation and Robotics, Ed. by G.N. Sardis, Jai Press, Inc., 1985.

[3] Pearson, K., "The Control of Walking," Scientific American, Vol. 235, No. 6, pp. 72-86, December, 1976.

[4] Schmidt, R.A., "Past and Future Issues in Motor Programming," Research Quarterly for Exercise and Sport, Vol. 51, No. 1, pp. 122-140, 1980.

[5] Mosher, R.S., "Exploring the Potential of a Quadruped," SAE Paper No. 690191, International Automotive Engineering Conference, Detroit, Mich., January, 1969.

[6] McGhee, R.B., "Robot Locomotion," in Neural Control of Locomotion, Ed. by R.M. Herman, et.al., Plenum Publishing Corp., New York, 1976, pp. 237-246.

[7] Briggs, R.L., A Real Time Digital System for Control of a Hexapod Vehicle Utilizing Force Feedback, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, June, 1979.

[8] Vukobratovic, M., Legged Locomotion Robots and Anthropomorphic Mechanisms, Research Monograph, Institute Pupin, Belgrade, Yugoslavia, 1975.

[9] Orin, D.E., Interactive Control of a Six-Legged Vehicle with Optimization of both Stability and Energy, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, March, 1976.

[10] Hirose, S., and Umetani, Y., "The Basic Motion Regulation System for a Quadruped Walking Vehicle," ASME Paper No. 80-DET-34, Design Engineering Technical Conference, Beverly Hills, Calif., September, 1980.

[11] Raibert, M.H., and Sutherland, I.E., "Machines That Walk," Scientific American, Vol. 248, No. 2, January, 1983, pp. 44-53.

[12] Russell, Jr., M., "Odex I: The First Functionoid," Robotics Age, Vol. 5, No. 5, September, 1983, pp. 12-18.

[13] Muybridge, E., Animals in Motion, Dover Publications, Inc., New York, 1957 (first published in 1899).

[14] Muybridge, E., The Human Figure in Motion, Dover Publications, Inc., New York, 1955 (first published in 1901).

[15] Hildebrand, M., "Symmetrical Gaits of Horses," Science, Vol. 150, pp. 701-708, November, 1965.

[16] McGhee, R.B., "Some Finite State Aspects of Legged Locomotion," Mathematical Biosciences, Vol. 2, No. 1/2, pp. 67-84, February, 1968.

[17] Sun, S.S., A Theoretical Study of Gaits for Legged Locomotion Systems, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, March. 1975.

[18] McGhee, R.B., and Frank, A.A., "On the Stability Properties of Quadruped Creeping Gaits," Mathematical Bioscience, Vol. 3, No. 3/4, pp. 331-351, October, 1968.

[19] McGhee, R.B., and Iswandhi, G.I., "Adaptive Locomotion for a Multilegged Robot over Rough Terrain," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-9, No. 4, pp. 176-182, April, 1979.

[20] Bessonov, A.P. and Umnov, N.V., "The Analysis of Gaits in Six-Legged Vehicles According to Their Static Stability," Proc. of the Symposium on Theory and Practice of Robots and Manipulators, International Center for Mechanical Sciences, Udine, Italy, September, 1973.

[21] Wilson, D.M., "Insect Walking," Annual Review of Entomology, Vol. 11, pp. 103-121, 1966.

[22] Kugushev, E.I., and Jaroshevskij, V.S., "Problems of Selecting A Gait for An Integrated Locomotion Robot," Proc. Fourth Int. Conf. Artificial Intelligence, Tbilisi, Georgian SSR, USSR, pp. 789-793, September, 1975.

[23] Kwak, S.H., A Simulation Study of Free-Gait Algorithms for Omni-Directional Control of Hexapod Walking Machines, M.S. thesis, The Ohio State University, Columbus, Ohio, March, 1984.

[24] Klein, C.A. and Patterson, M.R., "Computer Coordination of Limb Motion for Locomotion of a Multiple-Armed Robot for Space Assembly," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-12, No. 6, November/December, 1982.

[25] Okhotsimski, D.E. and Platonov, A.K., "Control Algorithm of the Walker Climbing over Obstacles," Proc. of the Third International Joint Conference on Artificial Intelligence, Stanford, California, August, 1973.

[26] Jaswa, V.C., An Experimental Study of a Real-Time Computer Control of a Hexapod Vehicle, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, June, 1978.

[27] Chao, C.S., Real-Time Multiprocessor Control of a Hexapod Vehicle, Ph.D. dissertation, The Ohio State University, Columbus, Ohio August, 1979.

[28] Pugh, D.R., An Autopilot for a Terrain-Adaptive Hexapod Vehicle, M.S. thesis, The Ohio State University, Columbus, Ohio, September, 1982.

[29] Tsai, S.J., An Experimental Study of a Binocular Vision System for Rough Terrain Locomotion of a Hexapod Walking Robot, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, May, 1983.

[30] Ozguner, F., Tsai, S.J., and McGhee, R.B., "An Approach to the Use of Terrain-Preview Information in Rough-Terrain Locomotion by a Hexapod Walking Machine," The International Journal of Robotics Research, Vol. 3., No. 2, Summer, 1984.

[31] Lee, W.J., A Computer Simulation Study of Omnidirectional Supervisory Control for Rough-Terrain Locomotion by a Multiplegged Robot Vehicle, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, 1984.

[32] Buckett, J.R., Design of an On-board Electronic Joint Control System for a Hexapod Vehicle, M.S. thesis, The Ohio State University, Columbus, Ohio, March, 1977.

[33] Pai, A.L., Stability and Control of Legged Locomotion Systems, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, 1971.

[34] Waldron, K.J., Song, S.M., Wang, S.L., and Vohnout, V.J., Mechanical and Geometric Design of the Adaptive Suspension Vehicle, Proc. of Fifth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators, Udine, Italy, June, 1984.

[35] McGhee, R.B., Orin, D.E., Pugh, D.R., and Patterson, M.R., "A Hierarchically-Structured System for Computer Control of a Hexapod Walking Vehicle," Proc. of ROMANSY-84 Symposium, Udine, Italy, June, 1984.

[36] Chung, T.S., Kinematic Simulation of an Adaptive Suspension Vehicle, M.S. thesis, The Ohio State University, Columbus, Ohio, December, 1982.

[37] Klein, C.A., Olson, K.W., and Pugh, D.R., "Use of Force and Attitude Sensors for Locomotion of a Legged Vehicle over Irregular Terrain," The International Journal of Robotics Research, Vol. 2, No. 2, Summer, 1983.

[38] Liegeois, A., "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-7, No. 12, 1977, pp. 868-871.

[39] Beringer, D.B., "The Design of Manually Operated Controls for a Six-degree-of-freedom Groundborne "Walking" Vehicle: Control Strategies and Stereotypes," Proc. of the Ninth Symposium on Psychology in the DOD, Colorado, April, 1984.

[40] Patterson, M.R., Reidy, J.J., and Brownstein, B.J., Guidance and Actuation Techniques for an Adaptively Controlled Vehicle, Final Technical Report, Battele Columbus Laboratories, Columbus, Ohio 43201, March, 1983.

[41] Klein, C.A., and Briggs, R.L., "Use of Active Compliance in the Control of Legged Vehicles," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-10, No. 7, July, 1980, pp. 393-400.

[42] Vertut, J., "Man-machine Aspects in the Control of the ASV," Commissaniat a l'Enegie Atomique, Institute de Protection et de Surete Nucleaire, Department de Protection technique, Annual Report on contract period July 1982-July 1983, June 27, 1983.

[43] Broerman, K.R., Development of a Proximity Sensor System for Control of Foot Altitude During Locomotion of a Hexapod Robot, M.S. thesis, The Ohio State University, Columbus, Ohio, June, 1983.

[44] Whitney, D.E., "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE Trans. on Man-Machine Systems, Vol. MMS-10, No. 2, June, 1969, pp. 47-53.

[45] Messuri, D.A., and Klein, C.A., "Automatic Body Regulation for Maintaining Stability of a Legged Vehicle During Rough-Terrain Locomotion," International Journal of Robotics and Automation, submitted February, 1985.

[46] Frank, A.A., "On the Stability of an Algorithmic Biped Locomotion Machine," Journal of Terramechanics, Vol. 8, No. 1, 1971, pp. 41-50.

[47] Paul, R.P., Robot Manipulators: Mathematics, Programming, and Control, The MIT Press, Cambridge, Massachusetts, 1981.

[48] Reklaitis, G.V., Ravimdram, A., and Ragsdell, K.M., Engineering Optimization: Methods and Applications, John Wiley and Sons, Inc., New York, 1983.

[49] Fox, R.L., Optimization Methods for Engineering Design, Addison-Wesley Publishing Company, Reading, Massachusetts, 1971.

[50] Ju, J.T., Safety Checking System with Voice Response for the OSU Hexapod, M.S. thesis, The Ohio State University, Columbus, Ohio, August, 1982.