NPS52-87-023

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A COMPUTER SIMULATION STUDY OF TRIPOD FOLLOW-THE-LEADER GAIT COORDINATION FOR A HEXAPOD WALKING MACHINE

Relle Lewis Lyman, Jr.

June 1987

Thesis Advisor: R. B. McGhee

Approved for public release; distribution is unlimited.

Prepared for:
Ohio State University Research Foundation
Columbus, OH 43212

This thesis prepared in conjunction with research sponsored in part by Ohio State Univ. Research Foundation under RF Project No. 716520.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-87-023	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable) 52	7a NAME OF MONITORING ORGANIZATION Prof. Kenneth J. Waldron, Dept. of Mech. Eng., Ohio State Univ.
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) 2075 Robinson Laboratory, 206 W. 18th Ave., Columbus, Ohio 43210
8a NAME OF FUNDING/SPONSORING ORGANIZATION Ohio State Univ. Research Foundation; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER RF Project No. 716520, RF Purchase Order No. 496549
8c ADDRESS (City, State, and ZIP Code) 1314 Kinnear Road, Columbus, OH 43212	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
A COMPUTER SIMULATION STUDY OF TRIPOD FOLLOW-THE-LEADER GAIT COORDINATION FOR A HEXAPOD WALKING MACHINE

12 PERSONAL AUTHOR(S)
Lyman, Relle L.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
MS and EE Thesis	FROM ______ TO ______	1987 June	177

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Robotics, Walking Machines, Adaptive Suspension Vehicle

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A new type of gait and steering algorithm for use by a six-legged walking machine is developed and presented in this study. The spatially oriented tripod follow-the-leader gait is an extension of previous studies of temporal follow-the-leader gaits, and should prove useful for all-terrain walking vehicles, such as the Adaptive Suspension Vehicle. Tractor-trailer style steering is introduced as an effort to tailor steering control for this type of gait. Both gait and steering algorithms are implemented on a color graphics computer simulation for study and comparison with other walking algorithms.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Robert B. McGhee	22b TELEPHONE (Include Area Code) 408-646-2095; 22c OFFICE SYMBOL 52Mz

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

A Computer Simulation Study of Tripod Follow–the–Leader Gait Coordination for a Hexapod Walking Machine

by

Relle Lewis Lyman, Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 1980

Submitted in partial fulfillment of the
requirements for the degree(s) of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
ELECTRICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL

June 1987

ABSTRACT

A new type of gait and steering algorithm for use by a six-legged walking machine is developed and presented in this study. The spatially oriented tripod follow-the-leader gait is an extension of previous studies of temporal follow-the-leader gaits. and should prove useful for all-terrain walking vehicles, such as the Adaptive Suspension Vehicle. Tractor-trailer style steering is introduced as an effort to tailor steering control for this type of gait. Both gait and steering algorithms are implemented on a color graphics computer simulation for study and comparison with other walking algorithms.

Thesis
L9535
c.1

TABLE OF CONTENTS

I. INTRODUCTION

It is estimated that nearly half of the Earth's land surface is inaccessible to wheeled and tracked vehicles [Ref. 1]. Yet almost all of this same area can be successfully traversed by animals and man. This great difference in mobility has motivated research into the creation of a practical legged vehicle or *walking machine*.

The advantages of legged locomotion can largely be attributed to the flexibility offered in leg placement and support. Wheeled vehicles, and to a lesser extent tracked vehicles, are confined to a more or less continuous, relatively flat and obstruction free paths along the ground. The leg's flexibility allows the utilization of discontinuous support regions on the ground and the adaptation to terrain slope. A legged vehicle may potentially use obstructions for support as it climbs over those obstacles which it decides to not simply ignore.

A second advantage of legs involves the means of obtaining traction in soft soil. A wheel or track creates a depression or rut from which it must continually work to climb out. Slippage causes the wheel or track spin, possibly digging a deeper hole. A leg, however, may be lifted vertically out of its depression, minimizing the work required. In addition, any back slip caused by the vehicle stepping pushes up soil behind the foot and improves traction. [Ref. 2]

The combination of flexible coordination and increased traction provides a potential for greater speed and less power consumption while operating over rough and otherwise unsuitable terrain. Other advantages of legged locomotion include possible improved comfort in ride due to the adaptive nature of legged support on uneven terrain, the ability to test soil conditions prior to placement of weight on the legs, and the relatively small footprint left in the soil. The latter may prove especially important for agricultural work, where the disturbance of crops is to be minimized, or for military vehicles navigating areas suspected of containing landmines.

A. GOALS

The purpose of this study is to explore a new type of gait and steering algorithm for the use of legged walking machines. The gait is a particular type of tripod gait, which can be considered as an extension of the temporal follow-the-leader gait [Ref. 3], into the spatial domain. The steering algorithm to be investigated along with this style of gait borrows from the concept of driving a wheeled tractor-trailer vehicle. It is believed that this steering algorithm may be particularly well suited for the fixed foothold position requirements of follow-the-leader gaits.

The machine chosen as a physical reference for the study is the Adaptive Suspension Vehicle (ASV). This is a self-contained, six-legged vehicle currently being evaluated at the Ohio State University for rough-terrain locomotion. The ASV is a Defense Advanced Research Projects Agency (DARPA) proof of concept project.

A secondary goal is to develop a simulation model with which to study walking gaits and control algorithms in general for the ASV. This model is developed along the general lines of the simulation previously presented by Lee [Ref. 4], incorporating several of his model's features, including omni-directional control, foot movement, and body attitude and altitude regulation algorithms. In addition, this simulation is to have the features of operation in either the new follow-the-leader tripod gait mode or in Lee's "forward wave" tripod gait mode, an enhancement of realism with a detailed color graphics display, and a menu system controlled with a single mouse button.

B. ORGANIZATION

Chapter II provides a brief overview of the previous work relating to this study. It includes a discussion of state of the art legged vehicles, tripod follow-the-leader gaits, tripod gaits, stability and simulation displays.

A detailed discussion of the ASV simulation problem is presented in Chapter III. This chapter covers the configuration of the vehicle, the gait and steering algorithms, the simplifications assumed in the construction of the model, and the kinematics involved in making the ASV model walk. The final section in this chapter describes the IRIS-2400 simulation hardware and software on which the model was developed.

The simulation program's operation and functions are presented in Chapter IV. This includes a complete description of the operation of the program controls and display features. This is followed by a discussion of the means by which graphics are programmed on the IRIS, and by a description of the organization and flow of the program and its modules.

Chapter V is a review of the performance of the simulation. It includes a brief subjective view on the feel of driving in the two modes.

The final chapter summarizes the contributions of this study. It also contains comments on possible directions for future research. The program code is listed in the appendix.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

The last quarter of a century has witnessed intense efforts to build machines that walk. Difficulties facing researchers include the problems of controlling the many degrees of freedom necessary in a maneuverable leg, maintaining vehicle stability, creating energy efficient motion, and adapting the walking motions to unstructured terrain. With the advent of compact computer technology and computer-aided simulation and design, serious progress is now being made in overcoming these problems. [Ref. 5]

Several promising working designs have emerged in the last ten years. Some of the most prominent include the Perambulating Vehicle II (PVII) at the Tokyo Institute of Technology, the Carnegie-Mellon University hexapod, the Odetics Inc. ODEX I, and the Adaptive Suspension Vehicle (ASV) developed at the Ohio State University.

The PVII is a light-weight laboratory model quadruped, developed in 1980. It features one of the first pantograph leg constructions designed specifically to provide simplified leg coordination and energy efficient walking. Using tactile foot sensors and a microcomputer mounted near the vehicle, the PVII is able to probe for footholds and maneuver over obstacles. [Ref. 6]

The hexapod developed at the Carnegie-Mellon University in 1982 is a self-contained walking machine large enough to carry its operator. It uses a gasoline engine to provide power to the legs via a set of hydraulic actuators. The movements of the individual legs are controlled by a series of passive hydraulic circuits. A built-in microprocessor interprets the driver's commands and specifies the correct series of leg movement patterns to be used. This arrangement frees the single microprocessor from the need to compute each foot trajectory. [Ref. 7]

The ODEX I is a commercial design introduced in 1983 [Ref. 8]. An improved version, sometimes referred to as ODEX II, is being developed for near-term use in nuclear power plants [Ref. 9]. The ODEX series makes use of a unique circular arrangement of six planar pantograph legs which allow the vehicles to adjust their profile for negotiation of narrow passages. The ODEX walking machines are directed through a radio or fiber-optic link from the operator to an on-board supervisory-level microprocessor. Each leg is controlled by a dedicated lower-level microprocessor which receives instructions from the supervisory level microprocessor. The new ODEX hexapod is also being equipped with a center-mounted arm for remote manipulation of objects, such as valves, in hazardous environments. [Ref. 8]

The Adaptive Suspension Vehicle (Figure 2.1), currently being tested at Ohio State University, is the first computer-coordinated legged vehicle designed and built for operation on natural terrain [Ref. 10]. This hexapod walking machine is completely self-contained, and is capable of carrying the driver, a 500 lb. internal

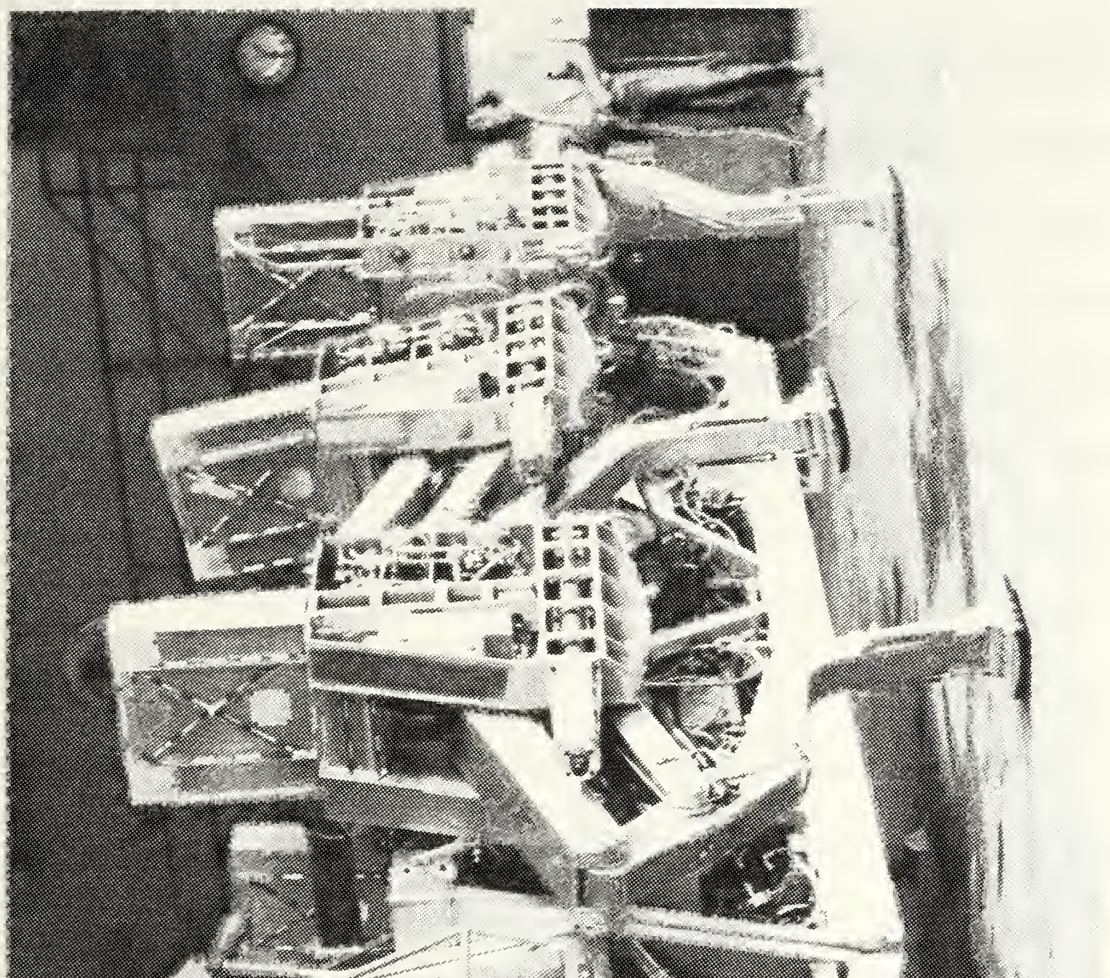


Figure 2.1 Adaptive Suspension Vehicle

payload, computer and control circuitry, and power system, in an outdoor environment. The ASV is the vehicle modeled in this study. A more detailed description of the ASV follows in Section 3.B.

The remaining sections of this chapter concern gaits used by walking machines, vehicle steering, the walking machine stability problem, and graphical representation of the vehicle's motion. The gait and stability sections are oriented towards six-legged vehicles such as the ASV.

B. GAIT SELECTION

1. Definitions

A *gait* is a mode of locomotion for a vehicle or animal distinguished by a specific pattern of lifting and placing of the feet. Gaits in general may be described using the *event sequence* notation introduced by McGhee and Jain [Ref. 11]. The integer i in such a sequence corresponds to the event of placing foot i on the ground. The lifting of the same foot is represented by the integer $i + n$, where n equals the number of legs. For the ASV, legs are numbered on the left side (1, 3, 5) from the front to the rear, and on the right side (2, 4, 6) in the same order.

A *periodic gait* is one that repeats the lifting and placing pattern, and thus is represented by one cycle of events. A periodic gait is said to be *nonsingular* if no two of its events occur simultaneously. McGhee [Ref. 12] demonstrated the existence of 39,916,800 possible nonsingular periodic hexapod gaits. The total number of possible hexapod gaits is a much larger and unknown number [Ref. 5]. This makes the selection of an optimum gait a very difficult problem. However, this thesis is concerned with a single type of gait sequence, the *tripod* sequence. These are *singular* gaits, in that more than one leg is placed at a given instant [Ref. 3].

A periodic gait is considered *symmetrical* when the stepping pattern on one side of the body is identical to that on the opposite side and separated in time by exactly one-half of the gait period [Ref. 12]. Symmetry tends to simplify the required leg coordination algorithms.

The *pitch* of a gait is the distance between footholds, measured in body lengths (defined as the distance between the front and rear leg reference positions). *Leg stroke* is the linear distance the foot travels with respect to the body when occupying a particular foothold. Leg stroke is also expressed in terms of body lengths.

2. Follow-the-Leader Gaits

A *follow−the−leader*(FTL) gait is one in which the middle and rear legs on each side of the body step in the foothold locations previously occupied by the leading legs [Ref. 13]. *Creeping* FTL gaits (in which at most one leg is in the air at any time [Ref. 14]). were first studied by Ozguner, Tsai, and McGhee [Ref. 3]. Using a *temporal* framework, they narrowed the number of possible FTL creeping gaits to 30, of which they found five to be symmetrically realizable.

Expanding to a *spatial* reference frame greatly increases the number of gaits in this category. A *tripod creeping gait* can be defined as one in which the legs are placed in alternating groups of three, with each group forming a tripod of support. In the case of the ASV, the two possible tripods are the leg sets (1 4 5) and (2 3 6).

The possible distinct tripod creeping gaits can be ennumerated using an approach similar to that in Ozguner et al. [Ref. 3]. Choosing the placement of leg 1 as a reference, evidently there are two possibilities for the relative ordering of legs 4 and 5, and three possible locations in each sequence for the insertion of the alternate group of legs. Furthermore, the placing of the alternate leg group (2 3 6) can be accomplished in six distinct ways. Table 2.1 lists the 36 possible nonsingular placing sequences.

It might first appear strange that the sequence (1 2 3 6 4 5) is included in Table 2.1. However taking two periods together, the sequence becomes (1 2 3 6 4 5 1 2 3 6 4 5), which clearly shows that the placement of the legs occur in alternating groups of three.

Comparing the entries in Table 2.1 to those in the table of Ozguner et al., one can see that none of these sequences are listed in the latter work. This is because the sequences here are not temporally follow-the-leader. Yet they all are spatial FTL gaits. This can be seen from the gait kinematics of the example shown in Figure 2.2.

TABLE 2.1. PLACING SEQUENCES FOR TRIPOD CREEPING FTL GAITS

Gait Number	Placing Sequence	Tripod 2 Insertion Position	Tripod 1 Subsequence	Tripod 2 Subsequence
1	123645	1	145	236
2	126345	1	145	263
3	132645	1	145	326
4	136245	1	145	362
5	162345	1	145	623
6	163245	1	145	632
7	123654	1	154	236
8	126354	1	154	263
9	132654	1	154	326
10	136254	1	154	362
11	162354	1	154	623
12	163254	1	154	632
13	142365	2	145	236
14	142635	2	145	263
15	143265	2	145	326
16	143625	2	145	362
17	146235	2	145	623
18	146325	2	145	632
19	152364	2	154	236
20	152634	2	154	263
21	153264	2	154	326
22	153624	2	154	362
23	156234	2	154	623
24	156324	2	154	632
25	145236	3	145	236
26	145263	3	145	263
27	145326	3	145	326
28	145362	3	145	362
39	145623	3	145	623
30	145632	3	145	632
31	154236	3	154	236
32	154263	3	154	263
33	154326	3	154	326
34	154362	3	154	362
35	154623	3	154	623
36	154632	3	154	632

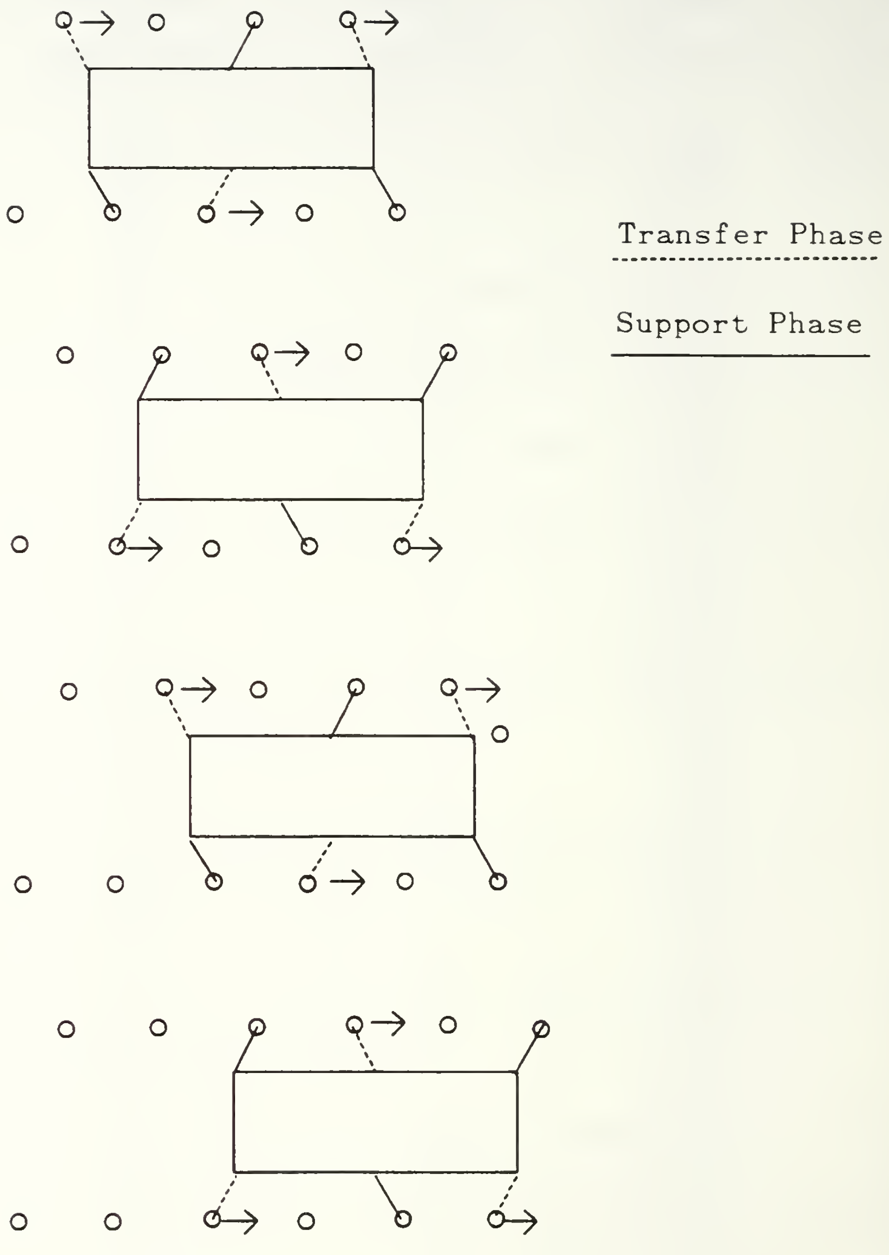


Figure 2.2 Sequence of Stepping for a Tripod FTL with Pitch of 1/3 and Continuous Body Motion

It is possible to alternate body motion with leg placement in the tripod gait (Figure 2.3). This yields a pattern of movement that is compatible with the general notion of a creeping gait [Ref. 3]. It should be noted, however, that while such a strategy may improve the *static stability* of the gait [Ref. 3], the intermittent body motion increases the leg stroke by a factor of two, which greatly increases the required working volume of the legs. For this reason, and also because intermittent body motion slows the average vehicle forward speed, only the continuous body motion alternative will be considered further in this thesis.

3. Singular Tripod Gaits

Tripod gaits have proved to provide a good compromise between stability, maneuverability, and ease of control for the Ohio State University Hexapod, the ODEX I, and the ASV. For this reason tripod gaits were chosen for this simulation study.

It can be seen that a tripod gait is actually a special limiting case of a creeping gait, where the time between the placement of individual legs within a tripod grouping approaches zero. Of the very large (unknown) number of gait sequences possible, only one can be classified as a singular tripod gait sequence. All differences among varieties of tripod gaits are therefore a function of kinematics only.

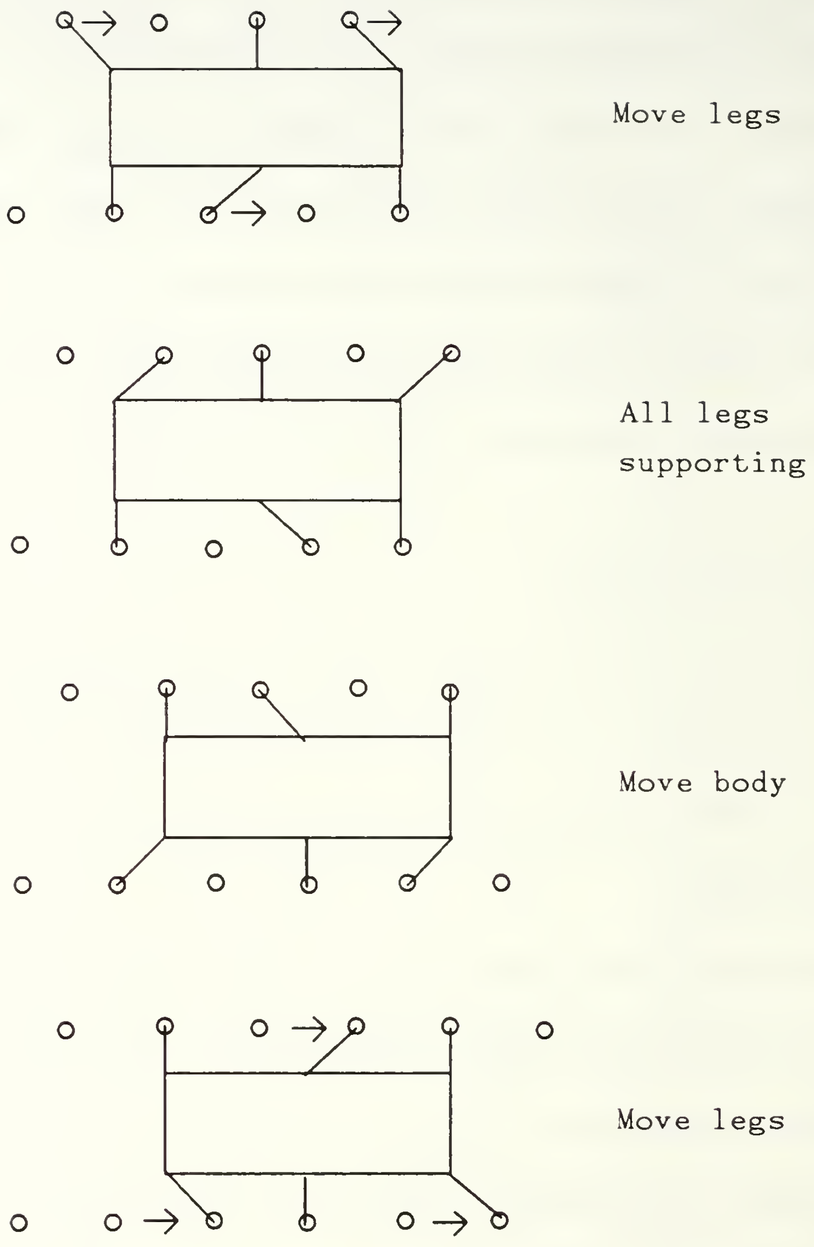


Figure 2.3 Sequence of Stepping for a Tripod FTL with Pitch of 1/3 and Alternating Body and Leg Motion

The most frequently used tripod gait is the limiting form of the *forward wave gait*, where the duty cycle[1], β, approaches 1/2 [Ref. 4,5]. This study introduces the singular FTL tripod gait. Both of these gaits are implemented in the walking algorithms of this simulation and are described further in Chapter III.

It is interesting to note that potentially the fastest forward wave tripod gait for the ASV is an FTL tripod gait with a pitch of one (Fig. 2.4). This of course can only be considered a true FTL gait if the feet are assumed to be dimensionless. In order to prevent the legs from interfering with one another, the duty cycle might be made slightly less than 1/2. This would momentarily leave the vehicle with no supporting legs in contact with the ground. It would also have the disadvantage of not providing sufficient time for possible foothold searches by the leading legs.

C. STEERING

There are several different approaches to steering currently used by ground vehicles. The most familiar method is *articulated*, or automotive style steering [Ref. 15]. With a steering wheel, accelerator and brake, the driver of an automobile can directly control the vehicle's turning radius and forward velocity.

[1] The duty cycle is the fraction of the leg cycle used for supporting the body.

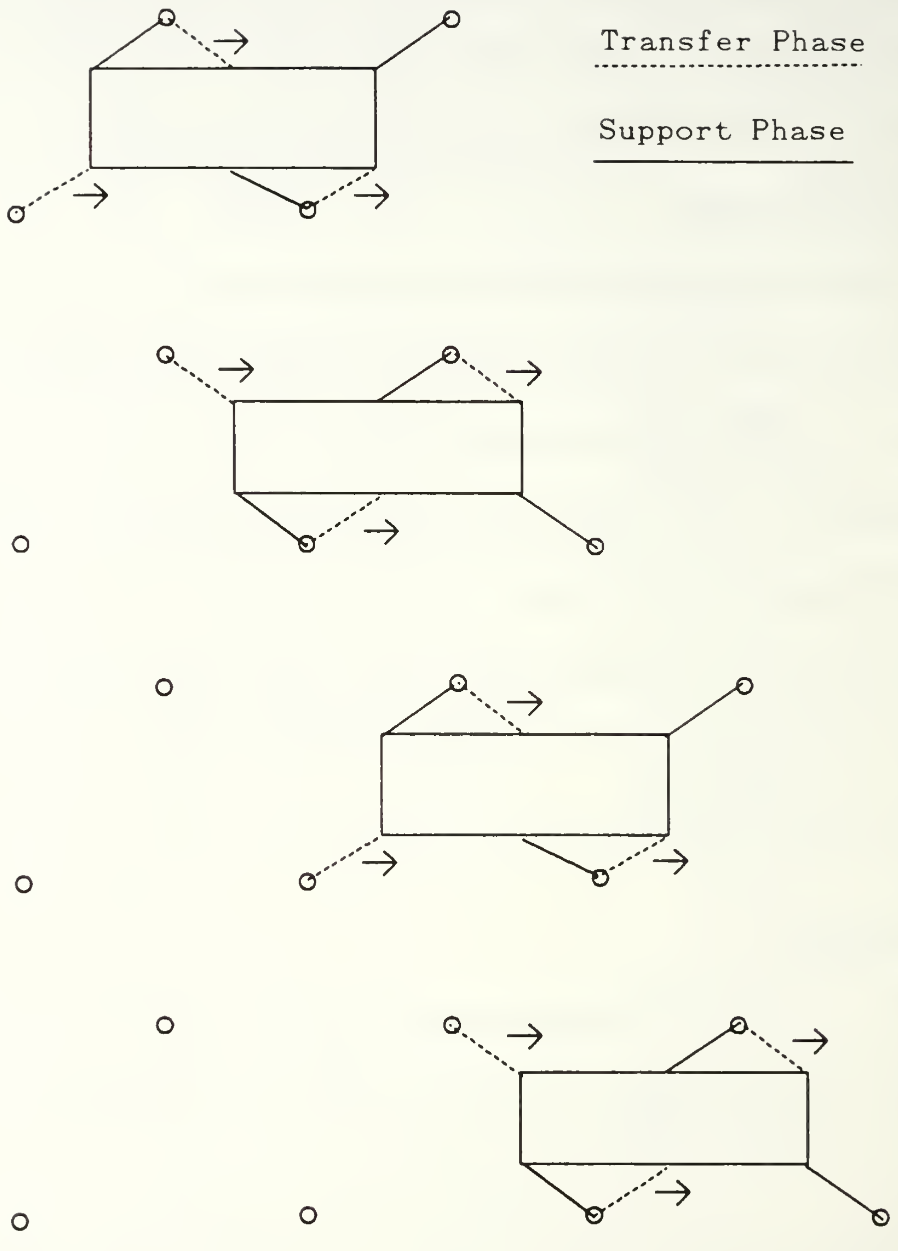


Figure 2.4 Sequence of Stepping for a Tripod FTL with Pitch of 1 and Continuous Body Motion

Tracked vehicles, on the other hand, most frequently utilize *skid* steering. By operating the sets of tracks at differing rates, the driver controls the turning rate and forward velocity of the vehicle.

Tractor-trailers use still another type of steering. Here the driver steers far forward of the vehicle's center of gravity. The trailer follows along in the path of the cab, with the steering of its center of gravity lagging behind the steering of the cab. Furthermore, since the trailer's wheel axle orientation constrains its motion, the trailer is restricted to a larger turning radius than the cab is capable of steering.

Specially designed wheeled vehicles may use *omni - directional* steering [Ref. 16]. This rarely used method allows the driver to specify turning rate and velocity in any horizontal direction.

Legged vehicles have historically used similar steering approaches. McGhee and Iswandhi [Ref. 17], introduced a two-axis joystick control, analogous to articulated steering, in which one axis controlled the turning radius and the other controlled forward velocity. Orin [Ref. 18], applied three-axis joystick control to the Ohio State University Hexapod, a small laboratory scale walking vehicle. This allowed forward, lateral and rotational velocities to be specified by the driver, providing steering control much like that experienced in a helicopter. The current ASV uses a similar three-axis joystick control.

Tractor-trailer style steering has not yet been applied to walking vehicles. This approach, which will be developed in this thesis, should give improved two-axis control to the driver for moving through areas of restricted maneuverability. The driver need only be concerned with maneuvering the front end of the vehicle. The body of the vehicle will follow along the proven path established by the footholds used by the front pair of legs.

D. STABILITY

The problem of vehicle balance is a vital concern for walking machines and has been a focus of many studies. Legged vehicles may maintain their stability using one of two methods, static balancing [Ref. 19] or dynamic balancing [Ref. 20].

Static stability is attained by maintaining the vertical projection of the vehicle's center of gravity within the polygon defined by the supporting legs [Ref. 5]. This method is conceptually simple. It is, however, only valid for stationary or slow moving vehicles, as it neglects the effects of inertia on stability.

Dynamic balancing is a complex process which places fewer restrictions on vehicle velocity. The vehicle may be allowed to momentarily move into a statically unstable configuration, so long as, over time, an adequate base of support is provided [Ref. 20]. This is the mode of balancing normally used by man and most vertebrate animals. It remains an extremely complex process, however, which is difficult to reproduce with legged vehicles.

This model uses only the static criteria for stability. Having the vehicle limited to reasonable velocities and the six legs placed in alternating tripod support patterns ensures a high degree of stability. To guarantee stability, the usable working volume for each leg is reduced. Figure 2.5 shows a worst case situation demonstrating that, if the legs are confined to their respective constrained working volumes, the vertical projection of the center of gravity will always fall within the triangular pattern formed by the supporting legs. A further discussion of the constrained working volume can be found in [Ref. 4].

E. GRAPHICS

There is a wide spectrum of available options from which to choose in the field of graphic displays. Decisions are required as to running the simulation on monochrome or color monitors, the type and number of dimensions for the projection, the use of line or solid figure representation, acceptable display resolution and update time, and whether to employ special hardware options. State of the art graphics machines also offer possibilities which include shading, reflectivity of surfaces, and multiple light sources. A compromise must be made between functionality, visual realism, and cost in order to realize an effective simulation.

Past simulation models featuring the ASV [Ref. 4,21,22] have concentrated on basic functionality in the display. The vehicles and terrain were represented by simplified line drawings on a monochrome monitor. This study attempts to take advantage of recent developments in special hardware and software for graphics workstations, in order to create a more realistic and convincing simulation. It is

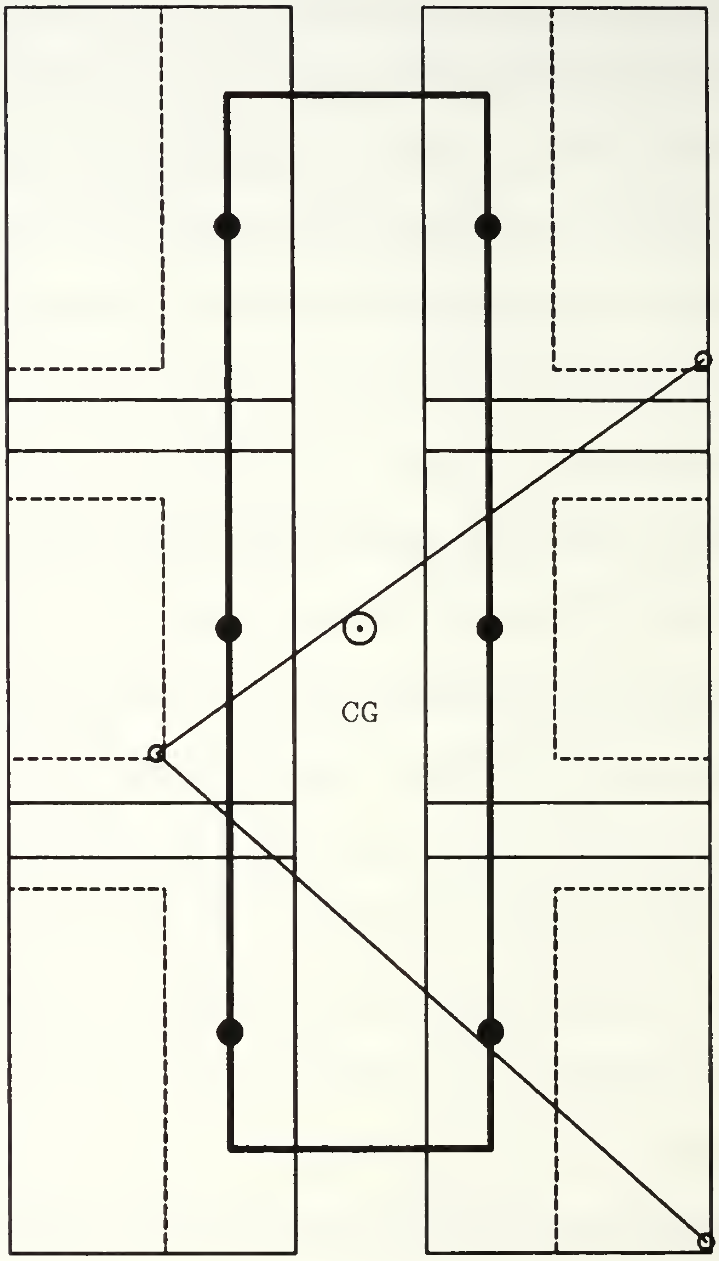


Figure 2.5 Constrained Working Volumes for Adaptive Suspension Vehicle

simulation. It is believed that the IRIS-2400 [Ref. 23] represents a good compromise between state of the art quality, cost, processing time and availability. This system was therefore selected to support the work of this thesis.

F. SUMMARY

This chapter provides background information on previous research leading to this study. Discussions include a brief survey of examples of the state-of-the-art walking machines, follow-the-leader and tripod gaits, vehicle steering, and the question of stability for walking machines. In addition, several concerns are expressed regarding the graphics displays used to portray the action of the walking vehicles.

The following chapter contains a detailed statement of the ASV simulation problem to be solved in this thesis.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

This chapter is intended as a description of the nature of the simulation modeling problem. In it is a discussion of the configuration of the ASV, the mathematics governing its motion, and foothold selection and steering algorithms for two selected gaits. Also covered are the simplifications deemed necessary in the creation of the model. The final section includes a brief description of the modeling facilities.

B. ASV CONFIGURATION

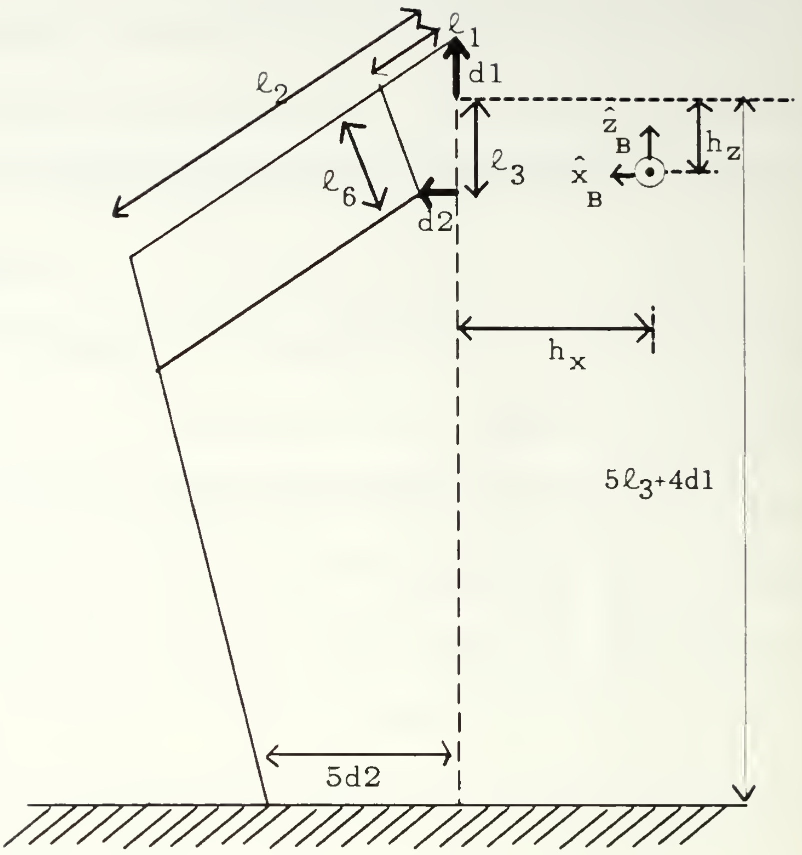
The Adaptive Suspension Vehicle (ASV) is a self-contained, six-legged walking machine designed to traverse uneven terrain. The operator, sitting in a cockpit at the front of the vehicle, controls the vehicle either at a supervisory level by selecting body translational and rotational velocities and allowing the vehicle to automatically place the feet, or by coordinating the individual legs in a precision-footing mode. The various control modes are discussed in [Ref. 10].

The vehicle is equipped with an optical scanning rangefinder, mounted above the cab, for short-range sensing. The laser rangefinder has a range of 10 m and a field of view of 40 degrees on each side of the body axis, and from 15 to 75 degrees below the horizontal. [Ref. 10: p.8]

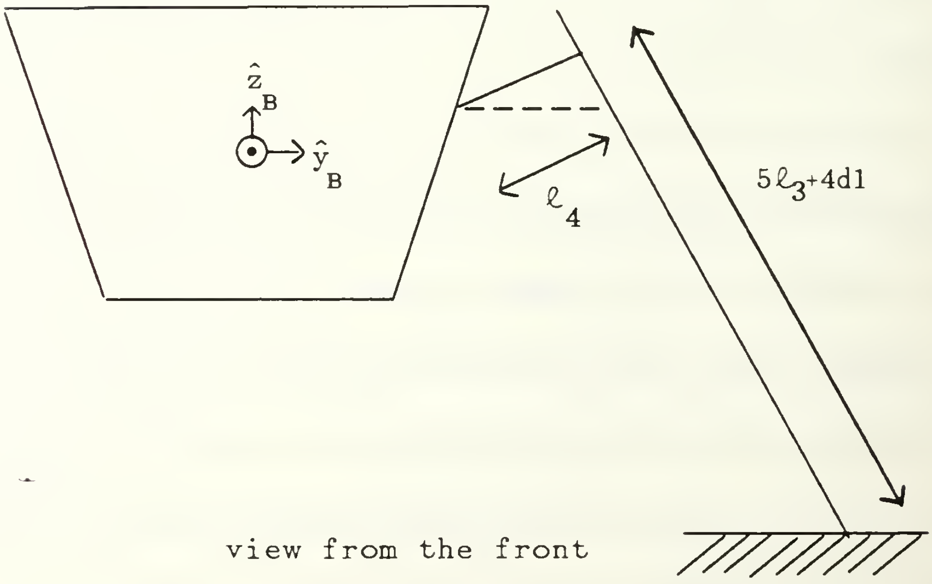
A single 900 cc four-cylinder motorcycle engine is sufficient to power the ASV over sustained periods of time. This is possible due to the aluminum construction of the frame and legs, which make the vehicle relatively light (2700 kg) for its size (3.0 m height, 5.2 m length) [Ref. 10:pp. 8-10]. Power is distributed to eighteen hydraulic actuator pumps through an energy storage flywheel and a series of shafts and toothed belts.

Seventeen Intel 86/30 single-board computers are used for onboard processing and control. One board is dedicated to each leg for motion control and leg sensor data processing. Four more boards compute stability, check actuator motion limits, and generate leg commands based on the operator's control inputs and the internal terrain model. Two additional boards are used for cockpit displays and controls. The terrain model is generated by the remaining five single-board computers using the data gathered from the optical rangefinder. [Ref. 10: pp. 8-10]

The design of the ASV's legs features a two-dimensioned pantograph mounted on a baseplate hinged to the body (Fig. 3.1 and 3.2). This design offers the advantages of energy efficiency resulting from decoupled ground reaction force components, and simplicity of control [Ref. 5,6, and 24]. Vertical and horizontal motion relative to the baseplate are provided by independent actuators mounted to the plate. Abduction and adduction motion is provided by a third actuator mounted on the body.



view from the side



view from the front

Figure 3.1 ASV Leg Configuration (1 of 2)

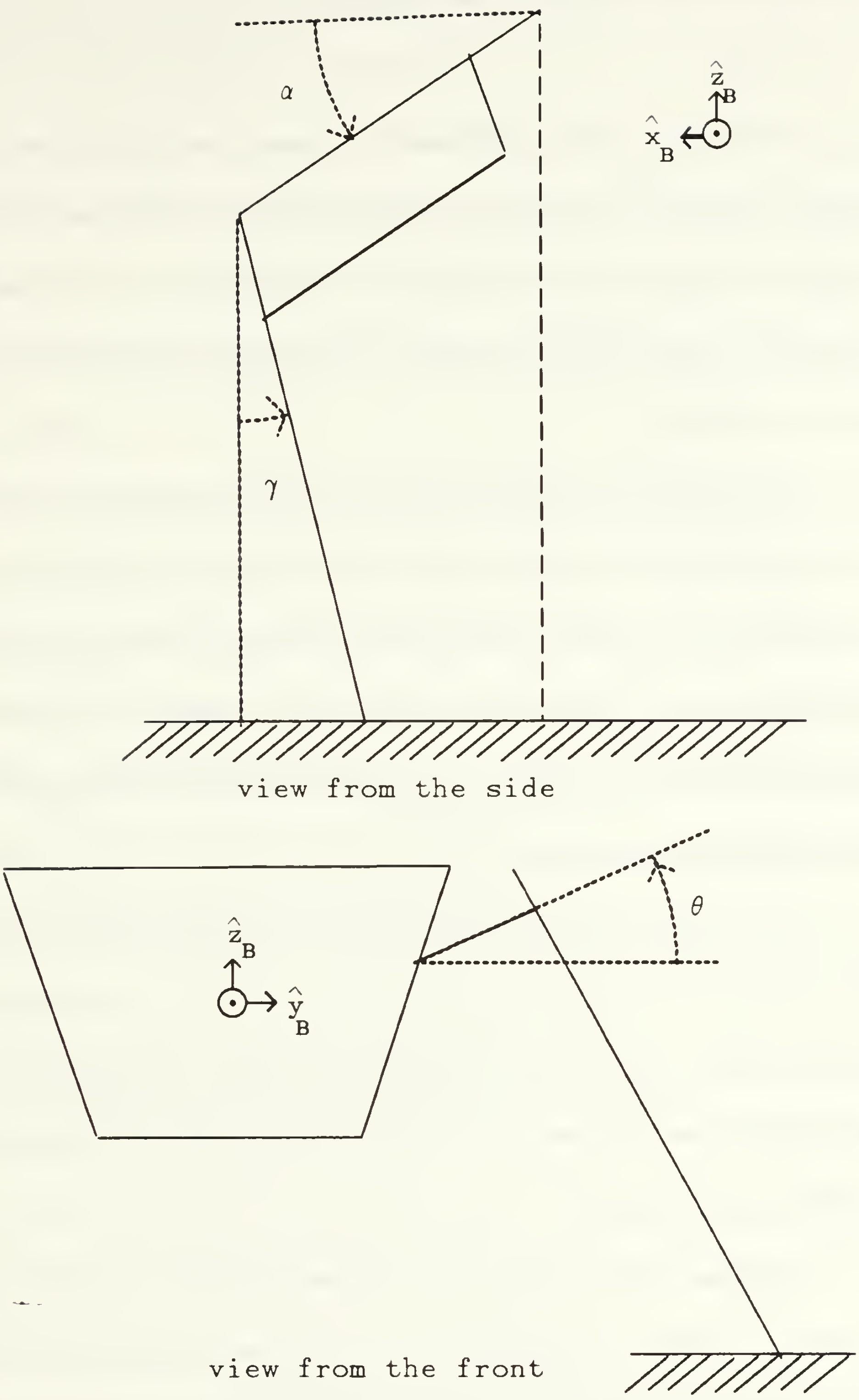


Figure 3.2 ASV Leg Configuration (2 of 2)

C. SELECTED WALKING ALGORITHMS

1. Gaits

The model used in this simulation study currently supports two styles of walking patterns. The first, closely following that used by Lee [Ref. 4] features a periodic tripod *forward wave gait*. The second is a periodic tripod *follow–the–leader* (FTL) gait [Ref. 3]. Both gaits have unique advantages to offer the operator.

The great advantage inherent in the forward wave gait lies in the maneuverability it offers the walking vehicle. The ASV. operating in the forward wave gait mode. is free to place its feet anywhere within a constrained working volume during the leg placement phase of the walking cycle [Ref. 4:pp.59-62]. This freedom allows the vehicle great flexibility in range of movement: even to the point of permitting turning in place.

The price for this freedom of choice for leg placement is that a foothold must be found and tested for each time a leg is placed on the ground. In rough or obscured terrain the process of probing and testing could occupy virtually all of the vehicle's onboard processing capability. Thus, the vehicle's speed over ground could be severely limited.

In this type of terrain the follow-the-leader gait could prove more advantageous. The follow-the-leader gait requires probing and testing only for the forward two legs. Since the following legs step precisely where the leading legs have gone, no further searching is needed. On difficult or dangerous terrain.

where extensive probing and testing of foothold is required, the FTL gait promises both greater ground speeds and more security.

The notable disadvantage of the FTL gait is that its use drastically constrains the vehicle's movement. Maneuvers such as sideways stepping and turning in place are not possible. Turning the vehicle requires a large radius turning circle, similar to that needed by a tractor pulling a long trailer.

2. Steering

The two walking algorithms utilize different control schemes matching the unique gait characteristics. The forward wave gait steering mode allows the operator to independently specify longitudinal velocity, lateral velocity, and azimuth angle rate (ideally using a three-axis joystick). This allows the operator to take fully advantage of the gait's maneuverability. In the absence of a three-axis joystick for this simulation, these body translation and rotation rates are input through three sliding bar controls using a mouse-driven cursor on the display screen.

The vehicle in the follow-the-leader gait mode, with its inherent restriction that the body remain between the two parallel foothold tracks, behaves very much like a tractor and trailer or a wagon. Just as the truck driver steers the cab allowing the trailer to follow in its path, the ASV operator in this mode steers by specifying the desired motion of the vehicle steering point. This point lies just behind the cockpit, mid-way between the two front legs, along the line joining the centers of the two working volumes. In the place of a steering wheel and acceleration pedal, the operator uses a two-axis joystick (simulated with a mouse-driven cursor on a steering pad), to specify the desired magnitude and direction of the cockpit's relative velocity vector.

The truck and trailer or wagon style steering commands are translated into desired longitudinal and lateral translation and azimuth rotation rates in order to maintain compatibility with the wave gait control algorithm in the program. This is done by first transforming the steering point (vehicle head) actual position and desired velocity to Earth coordinates, (x_{hE}, y_{hE}, z_{hE}) and $(\dot{x}_{dhE}, \dot{y}_{dhE}, \dot{z}_{dhE})$ respectively. The desired cockpit position $(x_{dhE}, y_{dhE}, z_{dhE})$ is determined by

$$x_{dhE} = x_{hE} + \dot{x}_{dhE} \cdot \Delta t \tag{3.1}$$

$$y_{dhE} = y_{hE} + \dot{y}_{dhE} \cdot \Delta t \tag{3.2}$$

$$z_{dhE} = z_{hE} + \dot{z}_{dhE} \cdot \Delta t \tag{3.3}$$

where Δt is the program display time increment. Using the desired cockpit position and the centroid of the middle and rear legs' footholds (in Earth coordinates) $(\overline{fh}_x, \overline{fh}_y, \overline{fh}_z)$, the desired azimuth angle Ψ_d is obtained.

$$\Psi_d = \tan^{-1}\left(\frac{y_{dhE} - \overline{fh}_y}{x_{dhE} - \overline{fh}_x}\right) \tag{3.4}$$

The desired new position of the body's center (x_{dE}, y_{dE}, z_{dE}), is then found by

$$x_{dE} = x_{dhE} - \frac{L}{2}\cos\Psi_d \tag{3.5}$$

$$y_{dE} = y_{dhE} - \frac{L}{2}\sin\Psi_d \tag{3.6}$$

$$z_{dE} = z_E \tag{3.7}$$

where L is the length between the center of the working volumes of the forward and rear legs.

The desired Earth translation rates ($\dot{x}_{dE}$ and $\dot{y}_{dE}$) and Euler azimuth angle rate (ω_{dzE}) are determined as

$$\dot{x}_{dE} = \frac{x_{dE} - x_E}{\Delta t} \tag{3.8}$$

$$\dot{y}_{dE} = \frac{y_{dE} - y_E}{\Delta t} \tag{3.9}$$

$$\omega_{dzE} = \frac{\Psi_d - \Psi}{\Delta t} \tag{3.10}$$

with Ψ being the current azimuth angle. These Earth and Euler rates are then translated to body rates ($\dot{x}_{dB}$, $\dot{y}_{dB}$, ω_{dzB}) by

$$\dot{x}_{dB} = \dot{x}_{dE}\cos\Psi + \dot{y}_{dE}\sin\Psi \tag{3.11}$$

$$\dot{y}_{dB} = \dot{y}_{dE}\cos\Psi - \dot{x}_{dE}\sin\Psi \tag{3.12}$$

$$\omega_{dzB} = \omega_{dzE} \tag{3.13}$$

3. Foothold Selection

As indicated in the above section, a new foothold must be selected for each leg while operating in the forward wave gait mode. In order to maximize the foothold's usefulness it should be placed so that the foot remains in the constrained working volume during the leg's support phase for a maximum length of time. The optimal foothold position is determined as the "point on the surface of the constrained working volume such that [the leg's] support trajectory is predicted to pass through the foot reference position" [Ref. 4: p.100]. To simplify the computation, the reference position is taken as the center of the working volume and a straight line is used to approximate the foot trajectory. A line is projected opposite to the direction of the predicted foot velocity vector at the reference point. The intersection of this line and the boundaries of the constrained working volume is then the desired foot position. Subsequent variation of the body velocity will alter the supporting foot trajectory, potentially resulting in a suboptimal foothold.

The follow-the-leader gait foothold selection process is much different. New footholds for the leading two legs are found by projecting a line along the velocity vector of the vehicle's cockpit. At a set distance (1/12 the length between the forward and rear hip joints), along this line, another line perpendicular to it is projected. This distance is one half the leg stroke of the vehicle while operating with a pitch of 1/3 (Figure 2.2).

The desired foothold is determined by where the second line intersects a line running through the center of the working volume parallel to the body's longitudinal axis (Figure 3.3).

As a front leg abandons its current foothold, that position is recorded for use by the middle leg behind it. In turn, the middle leg foothold positions are saved for use by the rear legs. Thus, during each complete leg cycle, two new foothold positions are computed. This compares favorably to the six new footholds needed while using the forward wave gait.

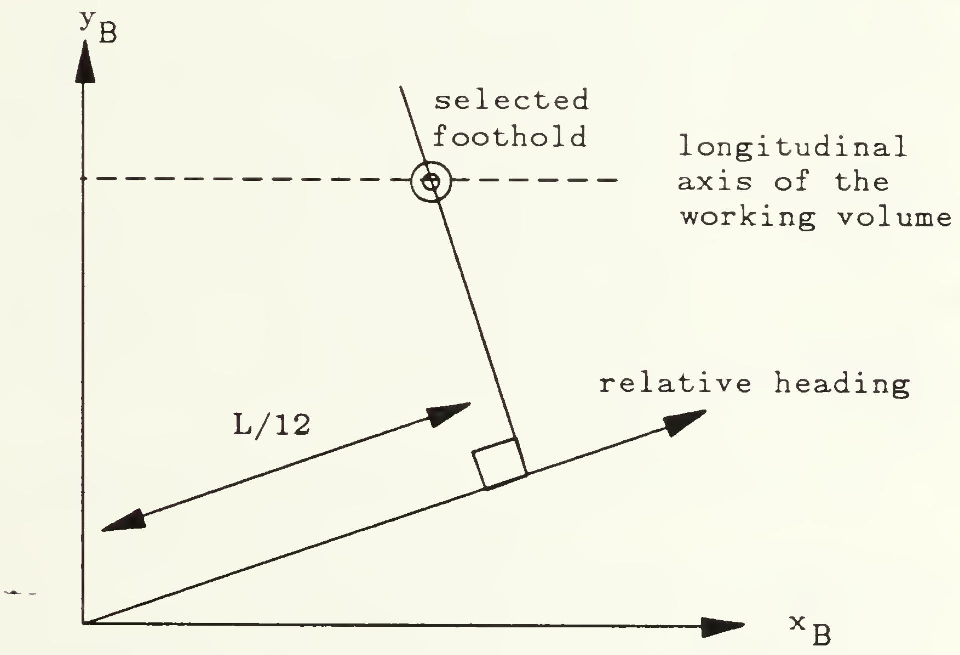


Figure 3.3 New Foothold Location

The current program allows the driver to start operation of the vehicle either using the forward wave gait or the follow-the-leader gait. Once started, the program must be reset before switching modes.

D. MODELING SIMPLIFICATIONS

There are many simplifying assumptions contained within this model of the ASV. These simplifications were made largely in an effort to speed the development of such features as the follow-the-leader gait. However, the program framework was devised with future work in mind. Thus, wherever possible room was left for generalization and expansion.

The most notable simplification in the simulation model deals with terrain. The ground is represented by a smooth, level, checkerboard pattern. Although the ASV was developed to be able to traverse unstructured terrain, there are no obstacles or obstructions in the current model. Uneven terrain will require inclusion of an algorithm for estimating the support plane beneath the vehicle, foot sensors, and a new terrain display routine. A foothold probe and testing routine will also be needed.

As a consequence of the use of flat terrain, the constrained working volume adjustments for uneven terrain [Ref. 4: pp. 109-117] and body regulation plans for varying slopes [Ref. 4: pp. 87-89] were not required. However, the basic structure for body attitude and altitude regulation has been retained in the program modules of this thesis. Consequently, inclusion of sloped terrain should necessitate only minor program changes in these areas.

The model contains only kinematic features of the ASV operation. This means that there are no limits on vehicle acceleration imposed by the model. In order to prevent unrealistic performance on the part of the displayed vehicle, a filter was placed between the commanded inputs and the response of the vehicle. The kinematics and filter for simulating dynamic constraints are described in the section below.

E. MODEL KINEMATICS

The kinematics of the model of the ASV presented here closely follow those developed in the computer simulation of Lee [Ref. 4]. Body motion is specified in terms of translation rates along the body's forward, lateral, and vertical axes (x, y, and z repectively) and rotation rates around these axes. The driver of the vehicle may directly or indirectly control the desired forward and lateral translation rates and the rotation rate around the vertical axis. The remaining three degrees of freedom are automatically regulated to maintain a desired body attitude and altitude with respect to the ground.

Vehicle dynamics are simulated through the use of a simple control filter inserted between the ordered rates and the actual body rates. As a result the body moves with a smooth, exponential transition in response to driver and body regulator control commands.

In order to realize these filtered body rates, the rates are first converted to earth coordinate translation rates and body Euler angle rates. Euler integration is then performed to produce translation distances in earth coordinates and angular displacement around the three body Euler axes.

1. Coordinate Systems

The ASV model makes use of two coordinate systems, earth (x_E, y_E, z_E) and body (x_B, y_B, z_B), in its calculations. The earth coordinate system is used wherever it is required to specify absolute position or velocities of the body, feet, or terrain. The earth coordinate system is defined such that the z_E axis is positive upward and the unit vectors x_E, y_E and z_E are mutually orthogonal.

The body coordinate system is useful in describing operator control and the coordination of body and legs. The origin is defined as the center of the main body section (excluding the cockpit). The z_B axis is projected upward through the top of the body, while the x_B axis is forward along the longitudinal axis and the y_B axis is projected to the body's left, forming the transverse or lateral axis.

Earth coordinates are transformed to body coordinates using the relationship of equation 3.14.

$$\begin{bmatrix} x_E \\ y_E \\ z_E \\ 1 \end{bmatrix} = H \begin{bmatrix} x_B \\ y_B \\ z_B \\ 1 \end{bmatrix} \tag{3.14}$$

where the position vectors $[x_E, y_E, z_E, 1]^T$ and $[x_B, y_B, z_B, 1]^T$ describe the same point in space in earth and body coordinates respectively and H is a 4 x 4 homogeneous transformation matrix [Ref. 25]. The homogeneous transformation matrix can be derived by decomposing the transformation into a translation from the earth coordinate origin and a series of rotations about the Euler axes:

$$H = T_{xyz} \cdot T_{\Psi} \cdot T_{\Phi} \cdot T_{\Theta} \tag{3.15}$$

The homogeneous transformation matrix T_{xyz} represents the translation of the body's center to its current position (d_x, d_y, d_z). The first rotation about the body's vertical axis by the azimuth angle Ψ is represented by the matrix T_{Ψ}. The body is then rotated about its new lateral axis by the elevation angle, Φ, and then about the newly formed longitudinal axis by the roll angle, Θ.[1]

[1] Other notations are sometimes used for these angles. For example, in [Ref. 19], θ signifies elevation angle and ϕ denotes roll angle.

The four homogeneous transformation matrices [Ref. 25: p30] are:

$$T_{xyz} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$T_{\Psi} = \begin{bmatrix} \cos\Psi & -sin\Psi & 0 & 0 \\ \sin\Psi & \cos\Psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$T_{\Phi} = \begin{bmatrix} \cos\Phi & 0 & \sin\Phi & 0 \\ 0 & 1 & 0 & 0 \\ -sin\Phi & 0 & \cos\Phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$T_{\Theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta & 0 \\ 0 & -sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

Substituting equations 3.16, 3.17, 3.18, and 3.19 into 3.15 yields:

$$H=\begin{bmatrix} \cos\Psi\cos\Phi & \cos\Psi\sin\Phi\sin\Theta-\sin\Psi\cos\Theta & \sin\Psi\sin\Theta+\cos\Psi\sin\Phi\cos\Theta & d_x \\ \sin\Psi\cos\Phi & \cos\Psi\cos\Theta+\sin\Psi\sin\Phi\sin\Theta & \sin\Psi\sin\Phi\cos\Theta-\cos\Psi\sin\Theta & d_y \\ -\sin\Phi & \cos\Phi\sin\Theta & \cos\Phi\cos\Theta & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

2. Body Regulation

A simple control algorithm is used in this and Lee's model to maintain the attitude of the vehicle and its height above the ground. The inputs are the estimated support plane and the plane formed by the body's lateral and longitudinal axes.

Body attitude regulation is accomplished by rotating the present body plane towards the desired body plane. The desired plane can be expressed as a function of the terrain slope and be adjusted to suit the driver[3].

The unit vector B_k along the rotation axis, (Fig. 3.4) is given by

$$B_k = \frac{\hat{z}_B \times \hat{z}_D}{|\hat{z}_B \times \hat{z}_D|} = [k_x \;\; k_y \;\; k_z]^T \tag{3.21}$$

where $\hat{z}_B$ and $\hat{z}_D$ are the unit normal vectors of the current and desired body

[3] In the current level terrain model, the desired body plane angle is set equal to zero.

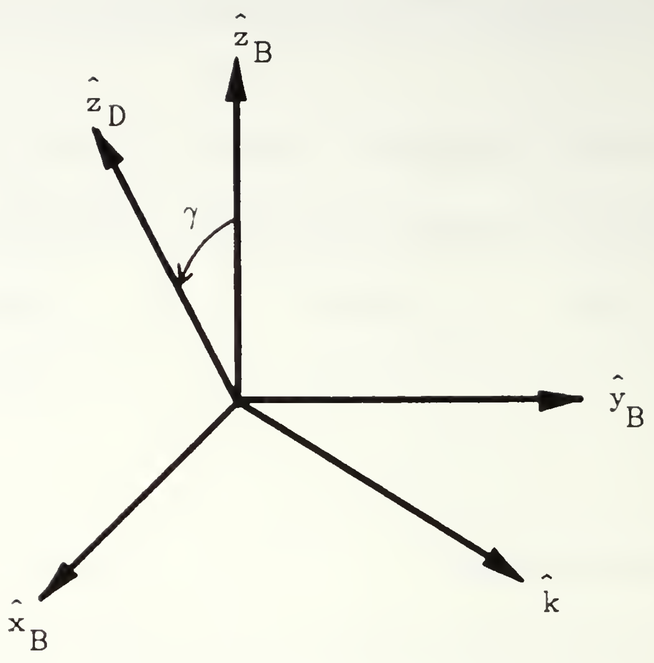


Figure 3.4 Rotation Axis and Angle

planes and $k_z = 0$. The rotation angle, γ, is given by

$$\gamma = \cos^{-1}(\hat{z}_B \cdot \hat{z}_D) \tag{3.22}$$

These values are used in the control function to obtain the rotation rates around the body's longitudinal axis, ω_x, and about its transverse axis ω_y.

Body altitude is defined as the distance along the body plane's unit normal from the estimated support plane to the body's center of gravity.

A mapping function similar to that used for the body plane can be used to relate the desired altitude. h_D. to the current terrain slope. h.[4]

3. Rate Computation

The differential equation describing the simulated dynamics of the control filter is

$$\dot{y}(t) = -\frac{1}{\tau}\, y(t) \tag{3.23}$$

where τ is the time constant of motion and y(t) is difference between the desired and actual position variable. Integrating both sides of equation 3.10 yields an exponential response

$$y(t) = e^{-\frac{1}{\tau} t} \tag{3.24}$$

The control filter for altitude is then

$$\dot{z}_B = -\frac{1}{\tau_1}\,(h_D - h). \tag{3.25}$$

Similarly the equation producing the attitude control response is

$$\dot{\gamma} = -\frac{1}{\tau}\,\gamma. \tag{3.26}$$

[4] In the current model the desired height is set to a constant value

The rotational vector γ_k decomposes into rotation vectors about the forward and lateral axes, yielding:

$$\omega_x = -\frac{1}{\tau_1} k_x \gamma \tag{3.27}$$

$$\omega_y = -\frac{1}{\tau_1} k_y \gamma. \tag{3.28}$$

Velocity is related to acceleration using the same filter. This is accomplished by letting

$$y(t) = \dot{x}(t) \tag{3.29}$$

and substituting into equation 3.23, yielding:

$$\ddot{x}(t) = -\frac{1}{\tau} \dot{x}(t). \tag{3.30}$$

The accelerations for the remaining three rates are

$$\ddot{x}_B = -\frac{1}{\tau_2}(\dot{x}_{B\ commanded} - \dot{x}_{B\ current}) \tag{3.31}$$

$$\ddot{y}_B = -\frac{1}{\tau_2}(\dot{y}_{B\ commanded} - \dot{y}_{B\ current}) \tag{3.32}$$

$$\dot{\omega}_z = -\frac{1}{\tau_2}(\omega_{z\ commanded} - \omega_{z\ current}). \tag{3.33}$$

Using the linear approximation,

$$\Delta velocity \approx \Delta time \cdot acceleration, \tag{3.34}$$

the rates are determined by equations 3.35 through 3.37.

$$\dot{x}_{B\ new} = \frac{\Delta t}{\tau_2}(\dot{x}_{B\ commanded} - \dot{x}_{B\ current}) + \dot{x}_{B\ current} \tag{3.35}$$

$$\dot{y}_{B\ new} = \frac{\Delta t}{\tau_2}(\dot{y}_{B\ commanded} - \dot{y}_{B\ current}) + \dot{y}_{B\ current} \tag{3.36}$$

$$\omega_{z\ new} = \frac{\Delta t}{\tau_2}(\omega_{z\ commanded} - \omega_{z\ current}) + \omega_{z\ current} \tag{3.37}$$

where Δt is the time increment and τ_2 is the time constant of motion.

Body positioning in this computer model is achieved by translating the body center to its proper earth coordinate position and then successively rotating the body about its vertical. transverse and longitudinal axes. In order to do this. body rates are first transformed into earth coordinate translation rates and body Euler angle rates using the method presented by Frank and McGhee [Ref. 19].

$$\begin{bmatrix} \dot{\Theta} \\ \dot{\Phi} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan\Phi\sin\Theta & \tan\Phi\cos\Theta \\ 0 & \cos\Theta & \sin\Theta \\ 0 & \sec\Phi\sin\Theta & \sec\Phi\cos\Theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.38}$$

Roll, elevation and azimuth angles and translation distances are then found through simple Euler integration:

$$y_{new} = y_{old} + \dot{y} \cdot \Delta time. \tag{3.39}$$

4. Leg Kinematics

The ASV's pantograph leg contruction yields relatively simple kinematic and inverse kinematic equations. These equations differ slightly from those presented by Lee [Ref. 4]. This can be attributed to the use of more accurate dimension measurement than those assumed by Lee.

For the front left leg (leg number one) shown in (Fig. 3.1 and 3.2), the foot position is given by

$$x_f = 5d_2 + h_x \tag{3.40}$$

$$y_f = (5l_3 - 4d_1)\sin\Theta + l_4\cos\Theta + h_y \tag{3.41}$$

$$z_f = l_4\sin\Theta + (5l_3 - 4d_1)\cos\Theta \tag{3.42}$$

where the hip position $(h_x. h_y. h_z)$ and foot position $(x_f. y_f. z_f)$ are given in body coordinates. and d_1. d_2. and Θ are the joint variables.

The inverse kinematic equations for the joint variable d_2. derived from equation 3.40 is

$$d_2 = \frac{1}{5}(x_f - h_x). \tag{3.43}$$

Rearranging and squaring both sides of equations 3.41 and 3.42 yields,

$$a^2\sin^2\Theta - a\ l_4\sin\Theta\cos\Theta + l_4^2\cos^2\Theta = (y_f - h_y)^2 \tag{3.44}$$

$$l_4\sin^2\Theta + a\ l_4\sin\Theta\cos\Theta + a^2\cos^2\Theta = (z_f - h_z)^2 \tag{3.45}$$

where $a = (5l_3 + 4d_1)$. Solving equations 3.31 and 3.32 gives.

$$d_1 = \frac{1}{4}\left(5l_3 - \sqrt{(y_f - h_y)^2 + z_f^2 - l_4^2}\right) \tag{3.46}$$

$$\Theta = \sin^{-1}\left(\frac{a\,(y_f - h_y) + l_4(z_f - h_z)}{a^2 + l_4^2}\right) \tag{3.47}$$

In addition to the joint parameters, this model requires leg upper (thigh) angle, α, the lower leg (shank) angle, γ, and the knee position in body coordinates (x_k, y_k, z_k). The thigh angle is given in terms of joint variables as

$$\alpha = \frac{\pi}{2} - \tan^{-1}\left(\frac{d_2}{l_3 + d_1}\right) - \cos^{-1}\left(\frac{l_1^2 - l_6^2 + d_2^2 + (d_1 + l_3)^2}{2\,l_1\sqrt{(l_3 + d_1)^2 + d_2^2}}\right) \tag{3.48}$$

and the knee position as

$$x_k = l_2\cos\alpha - h_x \tag{3.49}$$

$$y_k = (l_2\sin\alpha - d_1)\sin\Theta + l_4\cos\Theta + h_y \tag{3.50}$$

$$z_k = l_4\sin\Theta - (l_2\sin\alpha - d_1)\cos\Theta \tag{3.51}$$

The knee angle is

$$\gamma = \tan^{-1}\left(\frac{z_k - z_f}{x_k - x_f}\right) \tag{3.52}$$

All six legs of the ASV share similar geometries. The remaining kinematic and inverse kinematic equations can be obtained from equations 3.40 through 3.52 with appropriate sign changes.

F. SIMULATION FACILITIES

1. Hardware

The computer simulation presented here is designed to run on either of the two Silicon Graphics, Inc. IRIS-2400 workstations currently in the computer graphics laboratory in the Department of Computer Science at the Naval Postgraduate School. The IRIS (Integrated Raster Imaging System) consists of a Geometry Pipeline, a general purpose microprocessor, a raster subsystem, a 60Hz non-interlaced high-resolution RGB display monitor and a keyboard. In addition each unit has been equipped with two 72 megabyte disk drives, a cartridge tape unit, a floating point accelerator, and a three-input mouse. The Geometry Pipeline is a series of ten or twelve custom VLSI chip matrix multipliers. Under the control of the applications graphics processor, it performs matrix transformations, clipping and scaling of coordinates. The output is sent to the raster subsystem which performs functions such as filling in pixels, shading, depth-cueing and hidden surface removal.

The first IRIS system is based on a Motorola MC68010 processor with 5 megabytes of CPU memory and a 1024 x 786 x 8 bit display memory. It is also equipped with a digitizer tablet. The second IRIS system is a more capable Turbo-2400. It is based on a Motorola MC68020 processor and has 4 megabytes of CPU memory and a 1024 x 768 x 32 bit display memory. An Ethernet network connects both workstations to two VAX 11/780's and one VAX 11/750.

2. Software

The IRIS Graphics Library contains a large number of graphics commands and utilities. This allows the user great flexibility in the choice of coordinate systems and display techniques. While the software is written in C, the graphics commands may be called in C, FORTRAN, Pascal, and Lisp. The code for the model presented in this study is written exclusively in C.

G. SUMMARY

The previous sections of this chapter outlined the physical constraints, simplifications, and tools used in the development of this simulation. The next chapter describes the operation and construction of the actual simulation program.

IV. SIMULATION PROGRAM

A. INTRODUCTION

In this chapter the simulation program is presented. The first section consists of a user's guide, with complete instructions on how to use each program feature. The second section introduces the working environment for graphics on the IRIS-2400. The final section describes the internal operation of the simulation program and discusses the flow through the major modules. A complete listing of the program is provided in the appendix.

B. USER'S GUIDE

1. Starting Up

The program *walk.c* is relatively simple to use. It is entirely menu-driven, with a single mouse button and cursor performing all selection functions. To start the program, type the command "*walk*".

Immediately displayed on the monitor is a split screen view of the control panel and the ASV on its terrain (Fig. 4.1). The right half of the screen features a three-dimensional projection of the ASV on a green and white checkerboard plane against a blue backdrop. The user's vantage point is fixed relative to the center of gravity of the vehicle (above and initially to the vehicle's left side), so that the vehicle will continuously remain in view while walking.

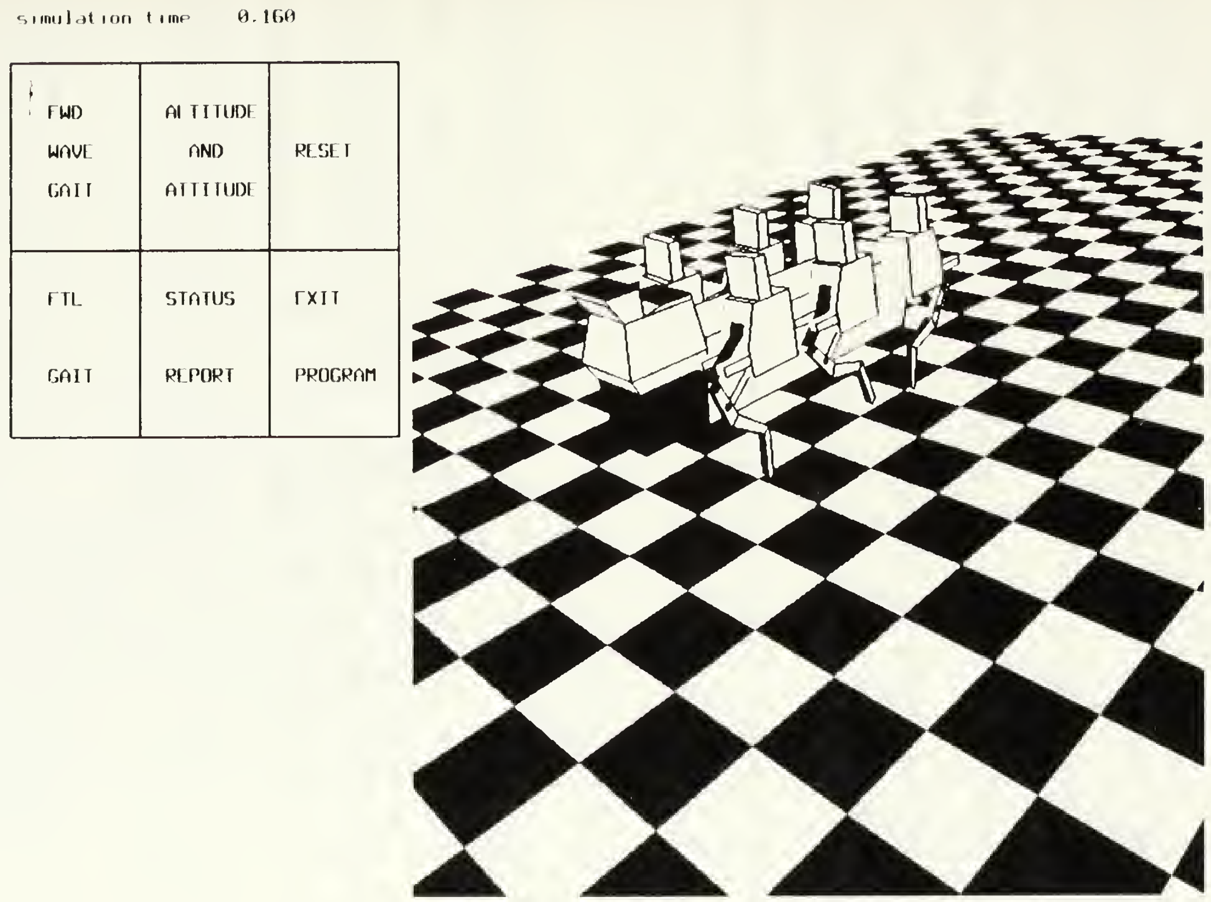


Figure 4.1 View of Initial Screen

The left half of the screen contains a two-dimensional representation of the control panel. Initially it features only the six yellow selection panels of the main menu on a cyan background.

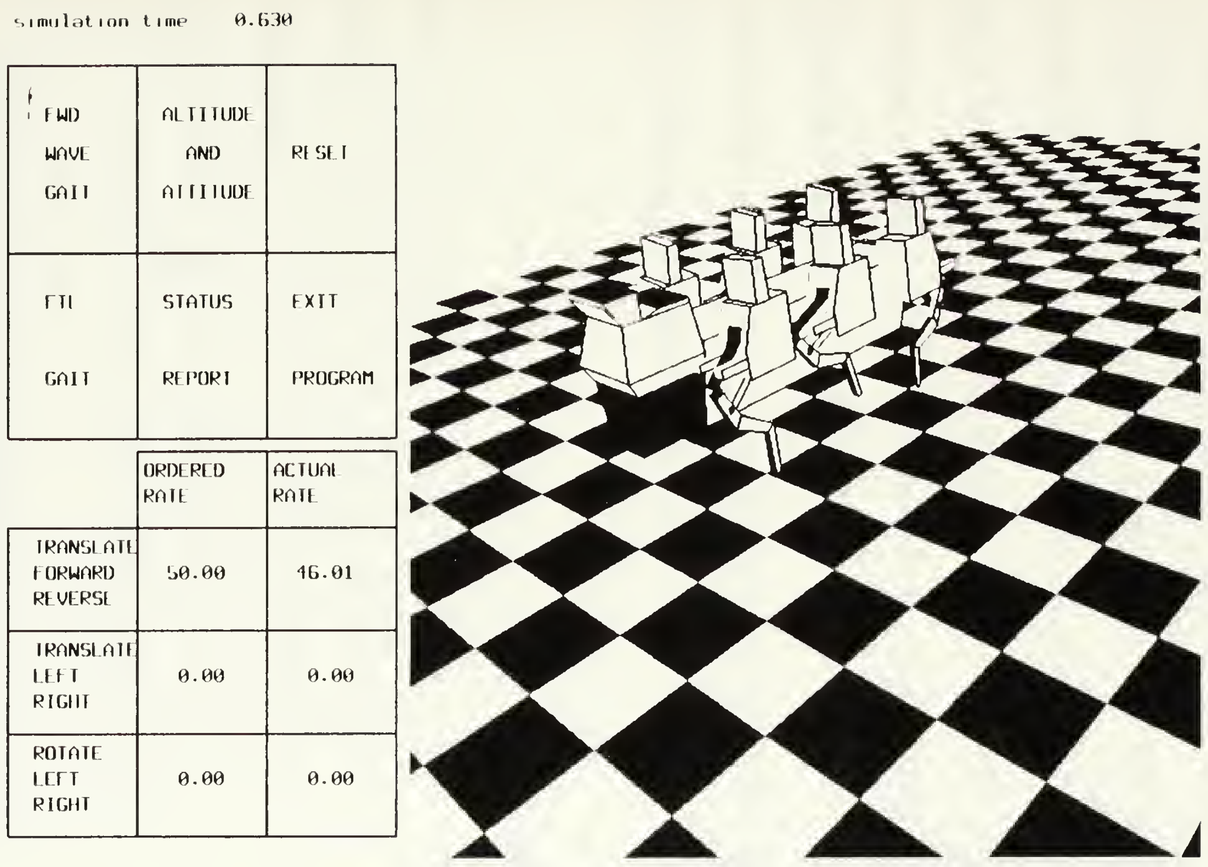
2. Menus

A menu item is selected by placing the cursor over the corresponding panel and clicking the middle mouse button. Pressing the button down will cause the panel beneath the cursor to be highlighted in red, as a potential choice. Releasing the button selects the highlighted menu item. If no changes are desired in the current menu selection, simply move the cursor to a portion of the screen outside the menu selection region and release the mouse button. Selected menu items are highlighted in bright yellow.

3. Forward Wave Gait

In the forward wave gait mode, vehicle velocities are specified in terms of body axis translation and rotation rates. Three of these rates - longitudinal velocity, lateral velocity, and yaw rate, are directly controllable by the operator. The rates for the remaining three degrees of freedom are automatically adjusted by the vehicle in order to maintain proper attitude and altitude. All rates in this mode are defined with respect to the body's center of gravity.

Selecting the forward wave gait panel produces a secondary menu displayed immediately below the main menu (Fig. 4.2). This secondary menu contains three additional panels for use in specifying the vehicle's body rates. The panels are operated in the same manner as those in the main menu. To the right



simulation time 0.630

	ORDERED RATE	ACTUAL RATE
TRANSLATE FORWARD REVERSE	50.00	46.01
TRANSLATE LEFT RIGHT	0.00	0.00
ROTATE LEFT RIGHT	0.00	0.00

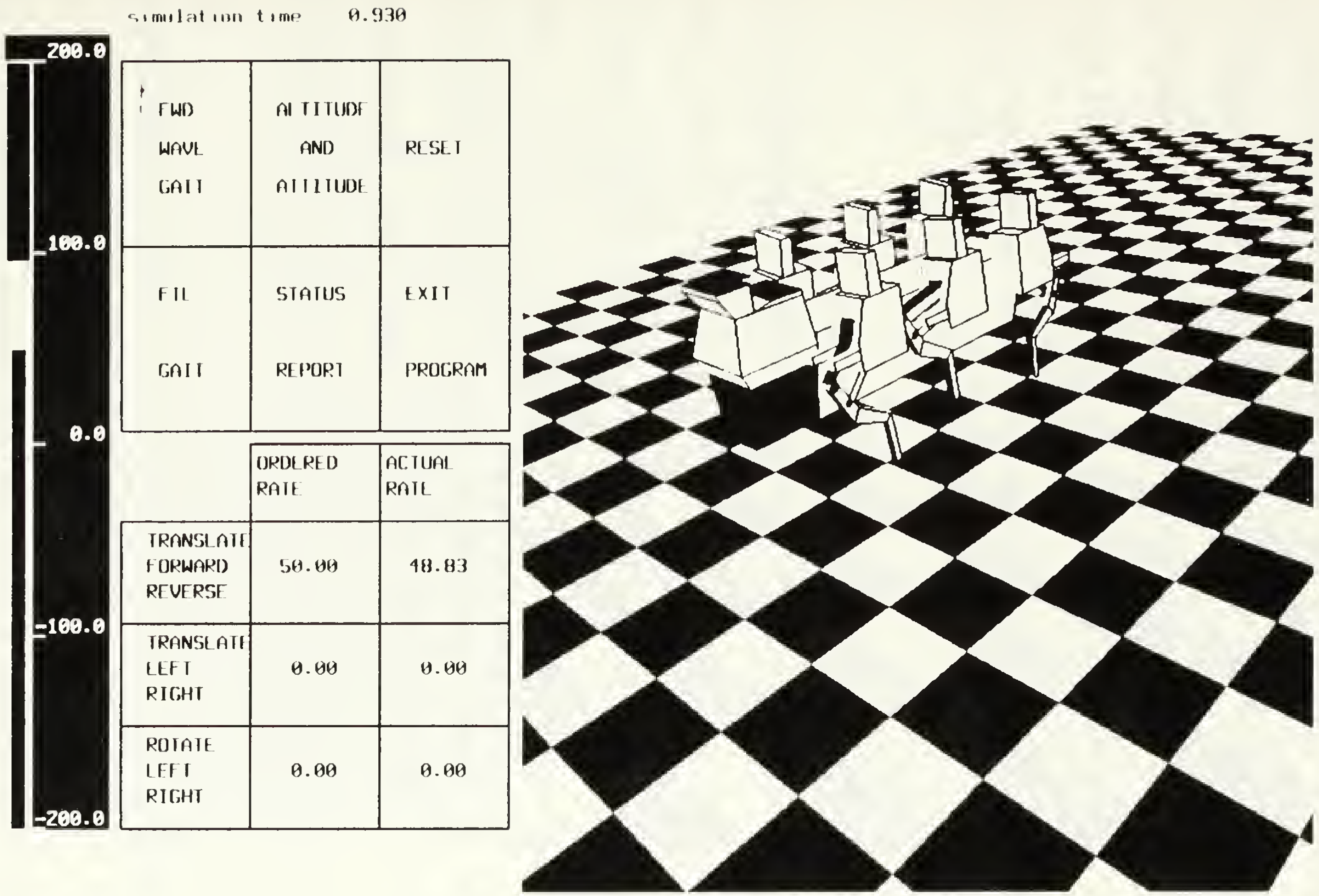
Figure 4.2 View of Forward Wave Gait Screen

of the menu panels are six simulated LED readouts. used for displaying the magnitude of the current and ordered rates.

Releasing the middle mouse button while the cursor is inside the bounds of one of the secondary menu panels results in a sliding bar control panel being displayed on the left edge of the screen (Fig. 4.3). Velocity commands are input by placing the cursor within the black center region of the bar control area. A yellow bar level indicator will rise or fall to match the cursor level, indicating the commanded velocity value. No clicking of the mouse button is required. To set the commanded input at the desired level, move the cursor to the desired height and then slide the cursor horizontally until it is outside the center region of the sliding bar panel. A red bar level indicator displays the current velocity of the vehicle.

4. Follow-the-Leader Gait

Control while in the follow-the-leader gait mode is achieved by specifying the desired relative velocity vector of the ASV's steering point. The operator, in essence, points the vector in the direction in which the steering point should travel. relative to the body longitudinal axis. As stated in the previous chapter, the steering is very much like that of a long tractor-trailer type of vehicle. The control algorithm factors in the magnitude of the desired velocity, footholds and current velocity and automatically regulates the body's motion.



	ORDERED RATE	ACTUAL RATE
TRANSLATE FORWARD REVERSE	50.00	48.83
TRANSLATE LEFT RIGHT	0.00	0.00
ROTATE LEFT RIGHT	0.00	0.00

Figure 4.3 View of Forward Wave Gait Screen with Sliding Bar Control

The follow-the-leader gait control mode is invoked by using the lower left panel of the main menu. When this panel is selected, a white rectangular control area appears directly beneath the main menu (Fig. 4.4). If the middle mouse button is held down while the cursor is within this region, the cursor controls a simulated two-axis joystick. The vertical axis represents the magnitude of the relative velocity vector and the horizontal axis represents the direction. A solid yellow line is used to indicate the current joystick position, and thus the input values. The vehicle's actual relative cockpit velocity vector is indicated by a solid red line.

5. Status and Warnings

The status menu option exists to provide the operator with numerical data on leg and body position and movements. Selecting this item causes a yellow and black display panel to appear below the main menu (Fig. 4.5). Featured on this panel are the translation and rotation rates (with respect to the body axes), the position of the vehicle's center of gravity (in Earth coordinates), the vehicle's orientation (in Euler angles), the walking cycle period, the position of each foot (in body coordinates) and the angles of various components of the legs. The values are updated each display cycle.

During the operation of the vehicle, checks are made on operating parameters. If a leg become positioned so that the foot is outside its corresponding constrained working volume, a red warning box is flashed in the lower left corner of the screen. Similarly, if the walking cycle period becomes too

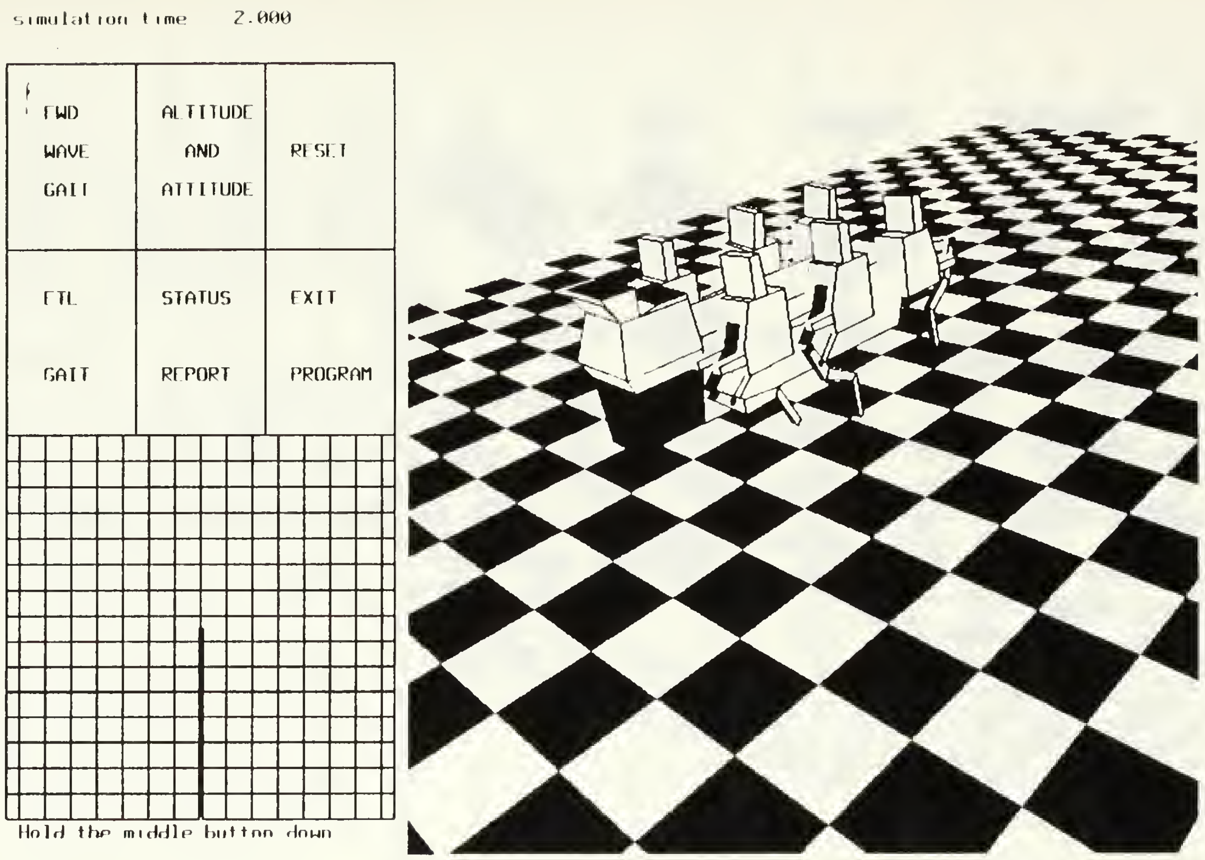
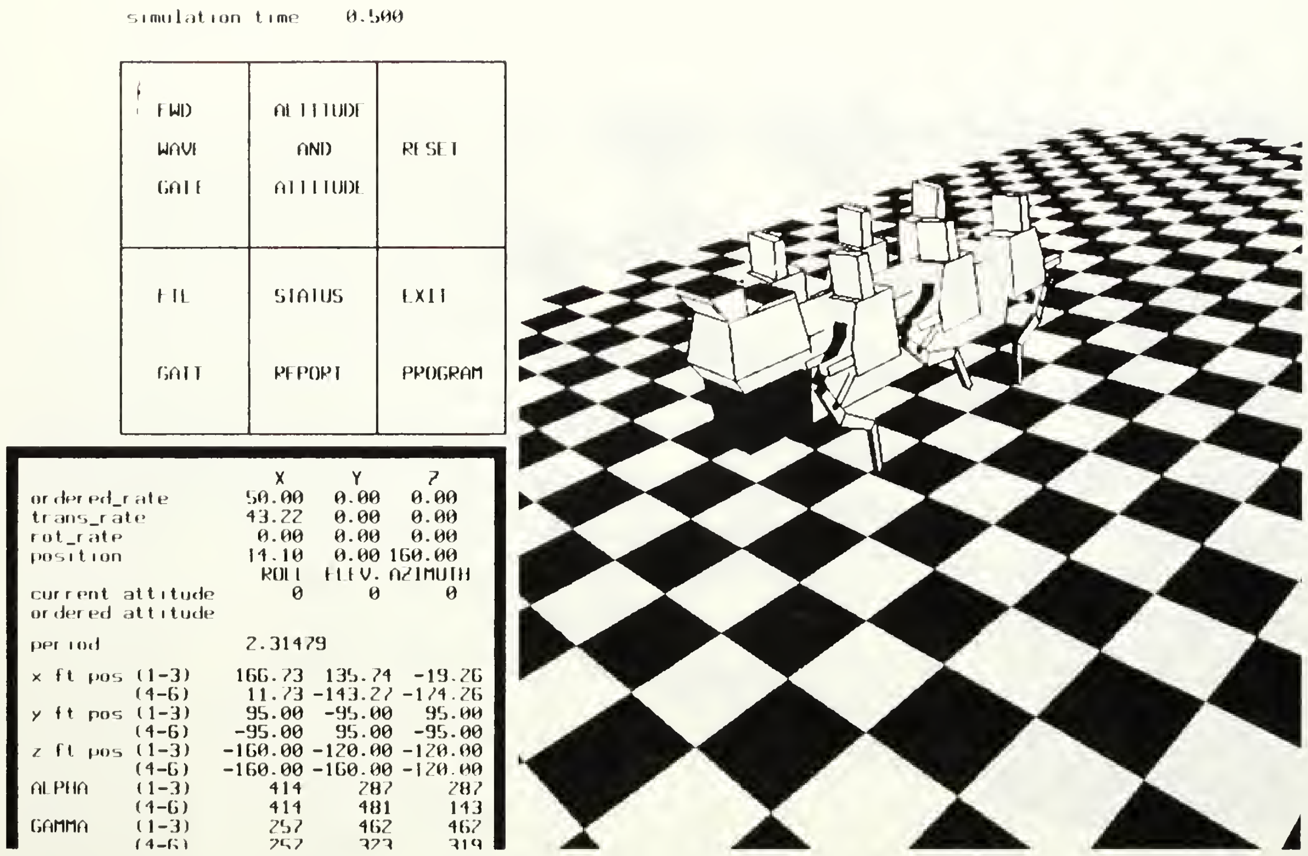


Figure 4.4 View of Follow-the-Leader Gait Screen



```
                          X       Y       Z
ordered_rate          50.00    0.00    0.00
trans_rate            43.22    0.00    0.00
rot_rate               0.00    0.00    0.00
position              14.10    0.00  160.00
                       ROLL   ELEV. AZIMUTH
current attitude          0       0       0
ordered attitude

period               2.31479

x ft pos (1-3)       166.73  135.74  -19.26
         (4-6)        11.73 -143.27 -174.26
y ft pos (1-3)        95.00  -95.00   95.00
         (4-6)       -95.00   95.00  -95.00
z ft pos (1-3)      -160.00 -120.00 -120.00
         (4-6)      -160.00 -160.00 -120.00
ALPHA    (1-3)          414     287     287
         (4-6)          414     481     143
GAMMA    (1-3)          257     462     462
         (4-6)          257     323     319
```

Figure 4.5 View of Status Display Screen

small. a yellow warning box is displayed. In addition to the warning. a deceleration routine is activated to slow the vehicle until the period comes up to an acceptable level [Ref. 4: pp. 66-71].

6. Reset and Exit

The reset option returns all vehicle parameters, including position. to their original values. This feature was included to save time when making a series of test runs. The exit option ends the program. clears the screen and returns the user to the current UNIX shell.

C. GRAPHICS ON THE IRIS-2400

Figures are displayed on the IRIS-2400 by calling a series of short graphics commands. called primatives. The primatives are interpreted into graphical displays by the software and special hardware of the IRIS system. These include commands for specifying color. drawing lines. circles. irregular polygons. and printing text characters on the screen. There are also a series of primatives designed to manipulate coordinate transformation matrices for the purpose of scaling, rotating and translating figures.

A sequence of graphics commands may be grouped into a listing called an object. This object list may then be conveniently executed using a single call. Once created. the object list may at any time be edited as desired through the use of object tags. The object. in essence. functions as a reconfigurable graphics subroutine.

Objects and figures in this program are displayed in the reverse order in which they are called. The last object is overlaid in front (with respect to the viewer's vantage point). of the previously called objects. An important exception to this is the treatment of the back face of polygons. The reverse side of a polygon, in the back face polygon removal mode. is considered transparent and is automatically removed from the image as a hidden surface by the IRIS hardware. This feature enables the display of more realistic appearing three-dimensional objects.

In the double buffer display mode utilized by this program. the special display memory is divided into two sets of bit plane buffers. As one buffer is having display data written into it, the other is used to refresh the monitor. Once the writing is complete. the functions of the buffers are swapped. and a new cycle of writing commences. This display mode provides for a smooth simultaneous update of the entire screen.

D. PROGRAM ORGANIZATION

The simulation program can be divided into three general sections; initialization. simulation loop. and termination. The heart of the program. the simulation loop. cycles through an input phase. which serves as the operator's control interface for the vehicle; the calculation phase. in which the parameters for the position and orientation of the ASV's body and legs are determined; and a display phase. The initialization section performs tasks, such as defining coordinate systems and creating object lists, required to start up the loop. The termination section clears the screen and buffers in preparation for the next IRIS user. A flow chart featuring the program's primary modules is shown in Figure 4.6.

Since the order in which objects are called is critical in this display mode, special provisions are needed to create a full 360 degree viewing coverage of the maneuvering vehicle. Specifically, four separate ASV objects lists are created in the object construction module of the initialization section. Each object has the vehicle components ordered for proper viewing from one of four viewing quadrants. The display section therefore needs only to determine from which quadrant the vehicle is to be viewed and call the appropriate object.

The simulation loop is the dominant part of the code, containing the overhelming majority of the program modules. The loop begins with a call to the driver's *command interface* module. This module controls the operation and display of the menu system, the status panel, and the sliding bar and joystick controls, as well as processing F.T.L. gait steering commands. The organization of the control module is shown in Figures 4.7 and 4.8.

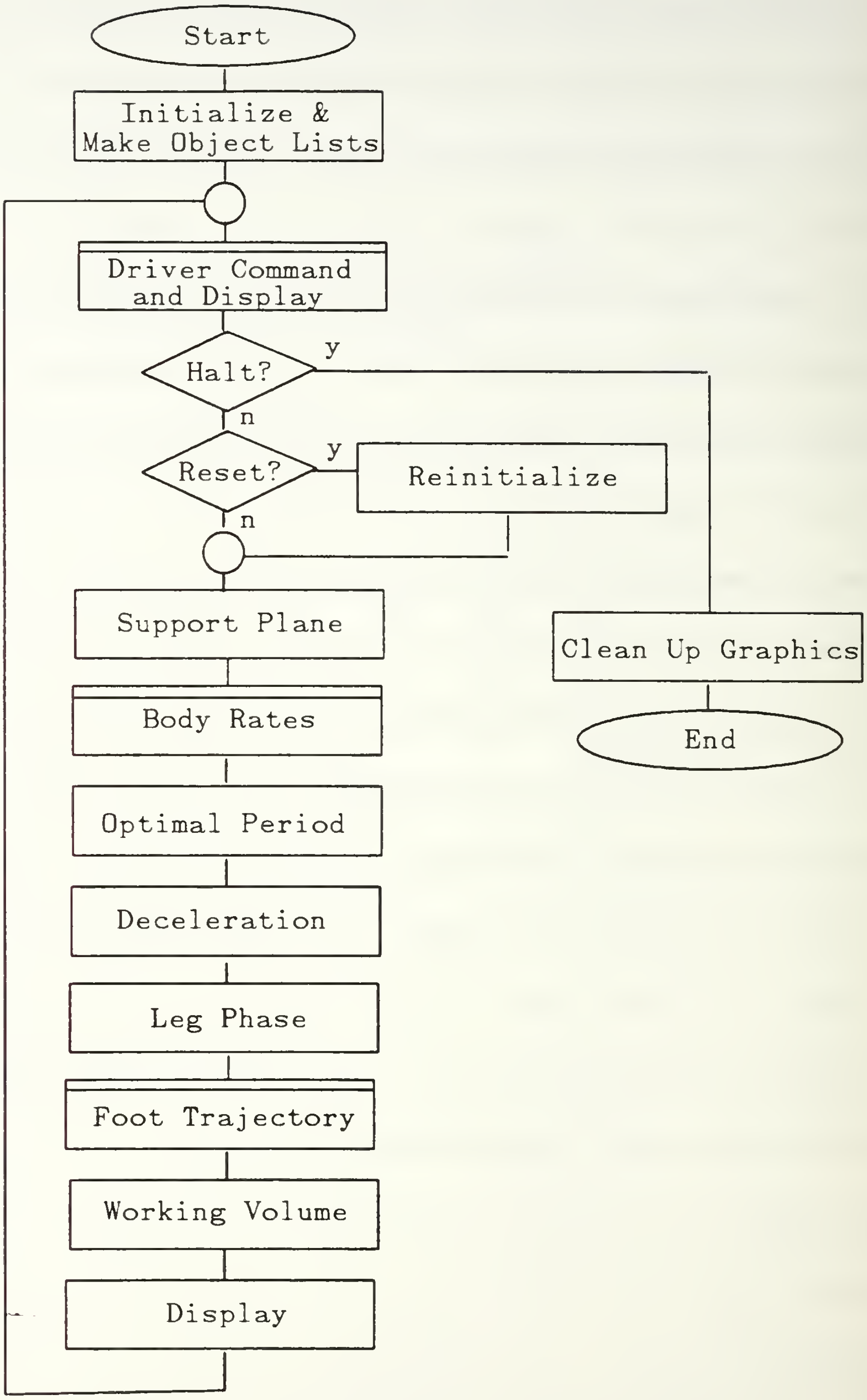


Figure 4.6 Main Simulation Flowchart

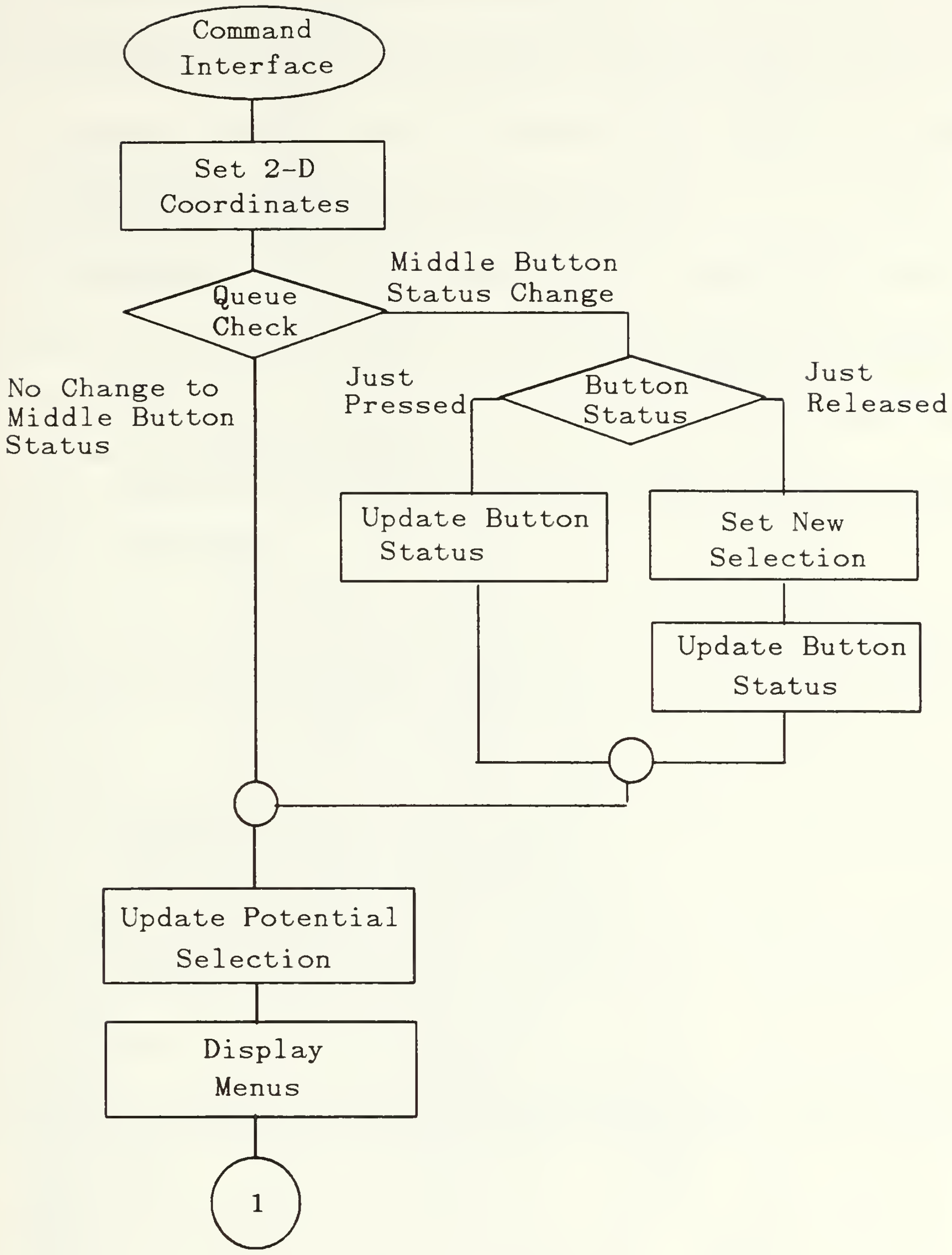


Figure 4.7 Command Interface Module (1 of 2)

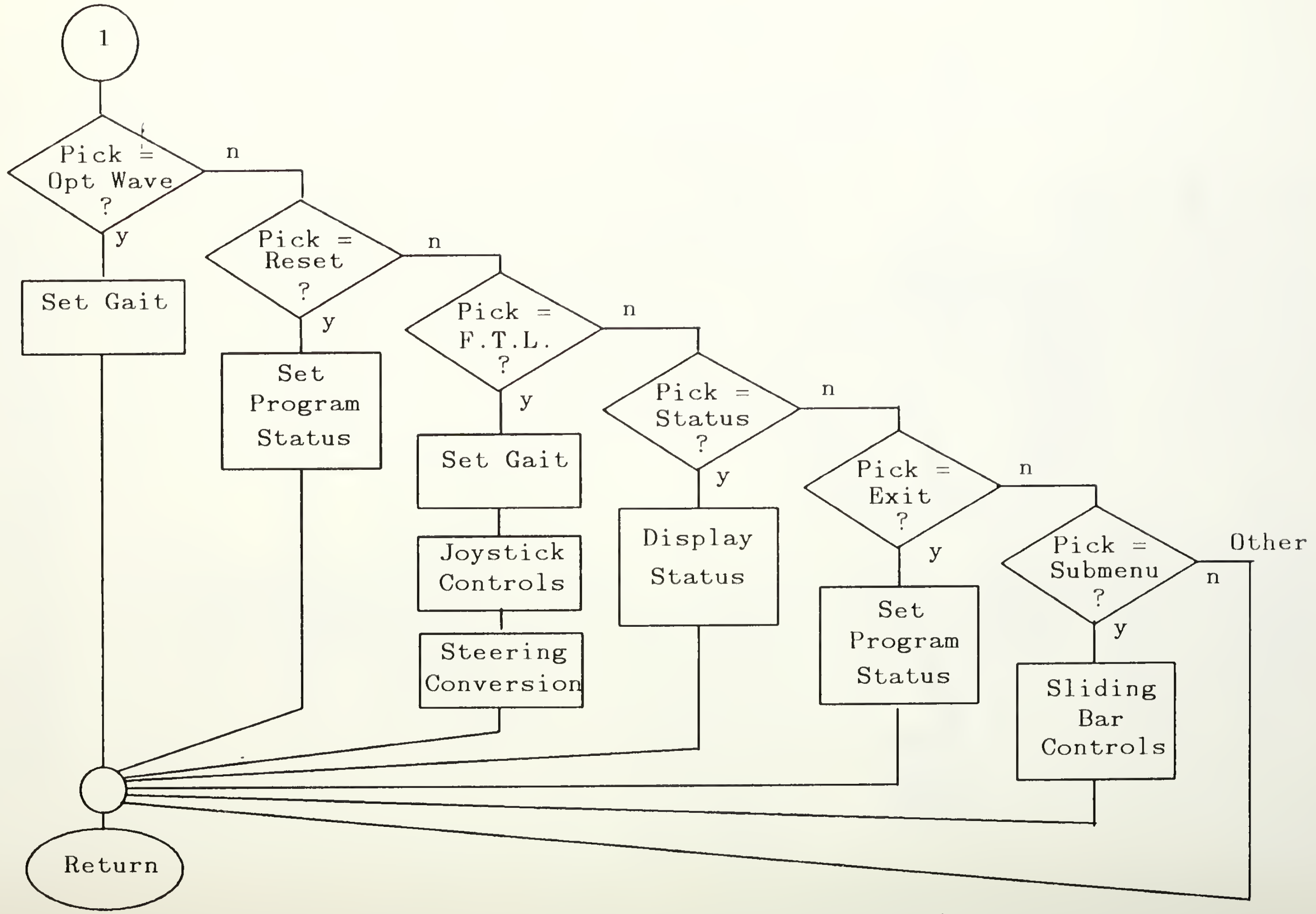


Figure 4.8 Command Interface (2 of 2)

Immediately following the command module are two checks on the program status. If the exit option was selected in the command module, the loop is interrupted and the program enters the termination stage. The reset option causes a module call to re-initialize all working parameters.

The calculation phase of the loop is begins with the *support* module, where a determination of the estimated support plane directly beneath the vehicle is made. The position and velocity of the ASV's body is then calculated in the *body rates* module (Figs. 4.9 and 4.10). The body kinematics used in this module are discussed in section III.E.

The gait period is next calculated in the *optimal period* module. This module uses a optimal period control algorithm which considers the kinematic limit of the supporting legs. In this algorithm, a period is calculated for each leg, based on the time required for its foot to reach the limits of its corresponding constrained working volume. The minimum of all of the supporting leg's periods is chosen as the vehicle's optimal period. No foot should therefore be required leave its constrained working volume while supporting the body. A backward gait period is also computed for the use of the wave gait walking in the reverse direction. [Ref. 4: pp. 63-69]

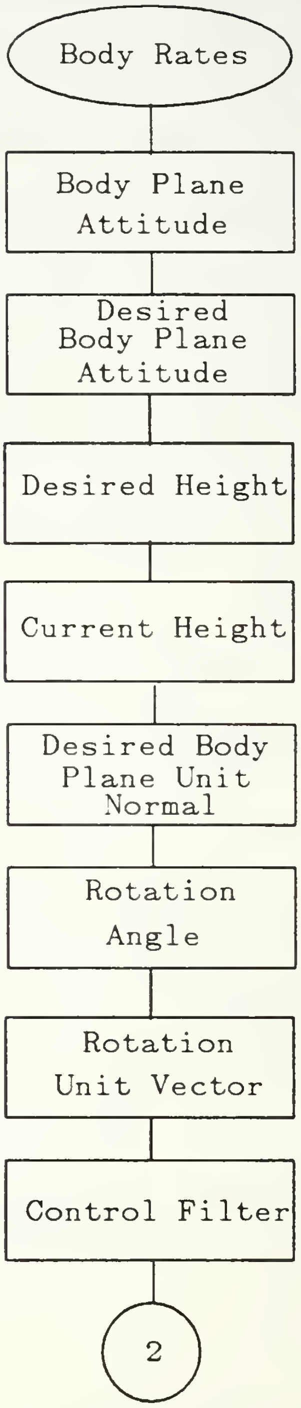


Figure 4.9 Body Rates Module (1 of 2)

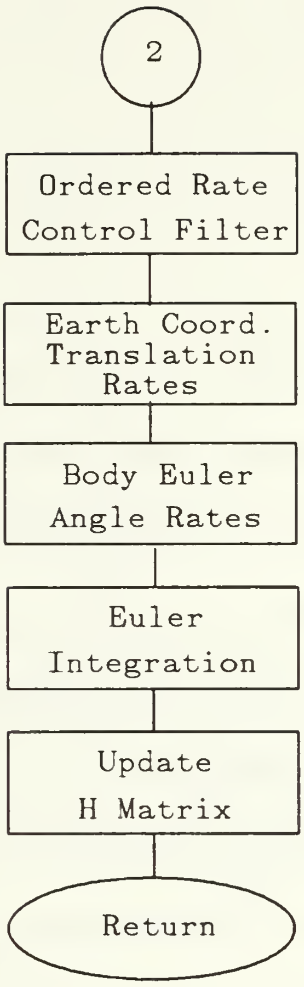


Figure 4.10 Body Rates Module (2 of 2)

The *deceleration* module checks the output of the previous module. If the period is below the set threshold, the vehicle is slowed. Each time the period falls below the minimum value, longitudinal body velocity is cut ten percent and lateral velocity and yaw rate are cut twenty percent. This slowing occurs in each pass until the walking period rises to acceptable limits.

The *leg phase* module is used to update the movement phase of each leg. The phase, expressed as a modulo one floating point number, indicates at what point the leg is in its cycle of supporting, lifting off from the ground, being transferred toward the desired foothold, and being placed onto the ground. The relative phases of the legs in this simulation are set to move the legs in two, 180 degrees out of phase tripods.

The *foot trajectory* module uses the leg phase information and the period in calculating the position of the feet relative to the body. The algorithm is shown in the flow chart of Figure 4.11. The transfer time is the length of time allotted for moving the foot from one foothold to the next. This determines the speed in which the transfer is made.

The phase of the leg relative to the beginning of foot liftoff is referred to as the transfer phase. When the leg's transfer phase is negative, corresponding to being on the ground in a supporting role, the foot's relative position is determined by the *support trajectory* module (Fig. 4.12). When the leg's transfer phase is greater than zero but less than the liftoff-transfer transition value, the relative foot position is returned by the *liftoff trajectory* module (Fig. 4.13). Likewise a

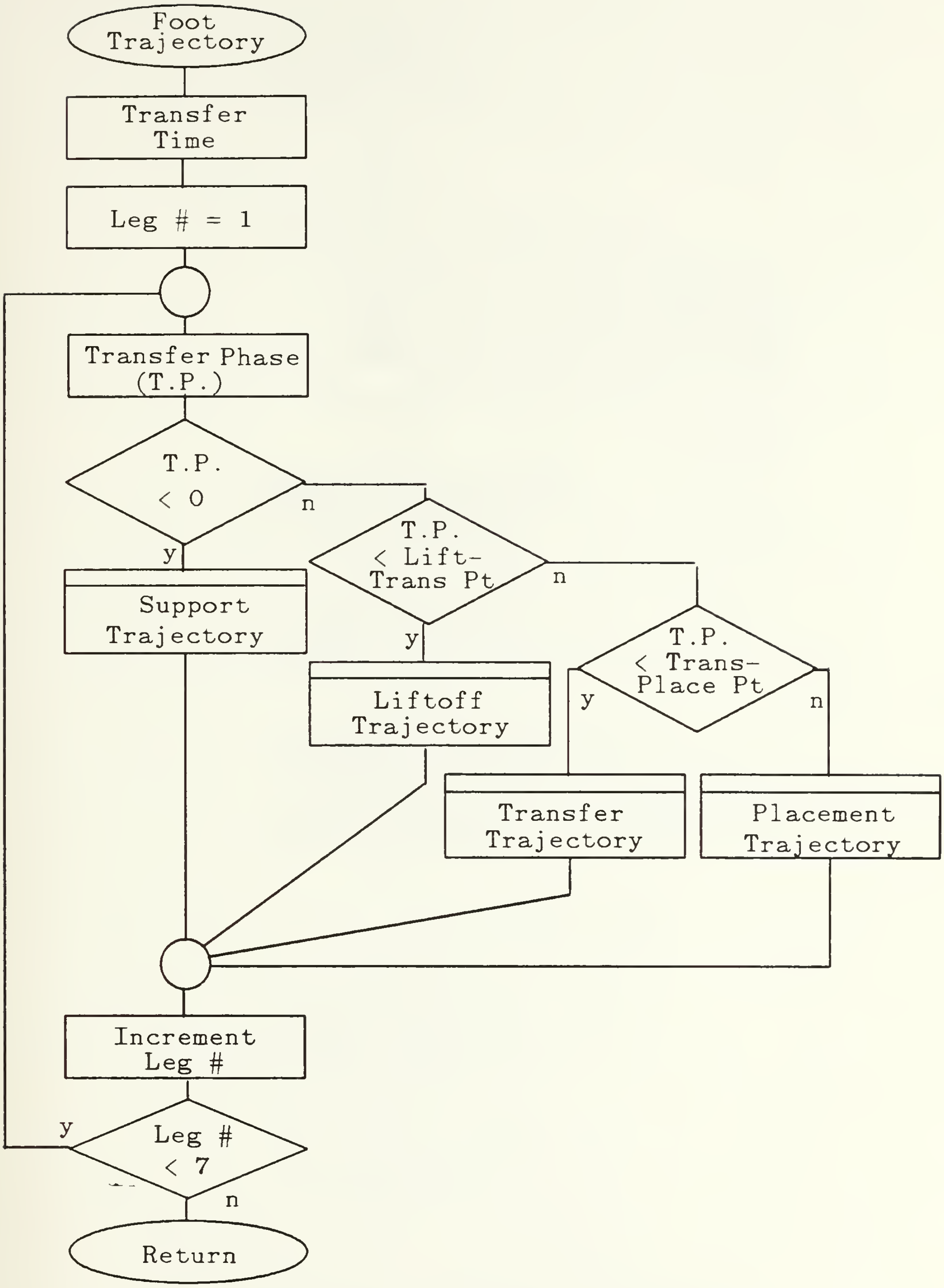


Figure 4.11 Foot Trajectory Module

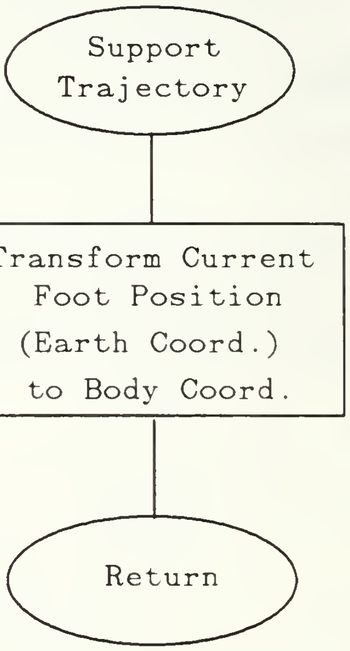

Support Trajectory
Transform Current Foot Position (Earth Coord.) to Body Coord.
Return

Figure 4.12 Support Trajectory Module

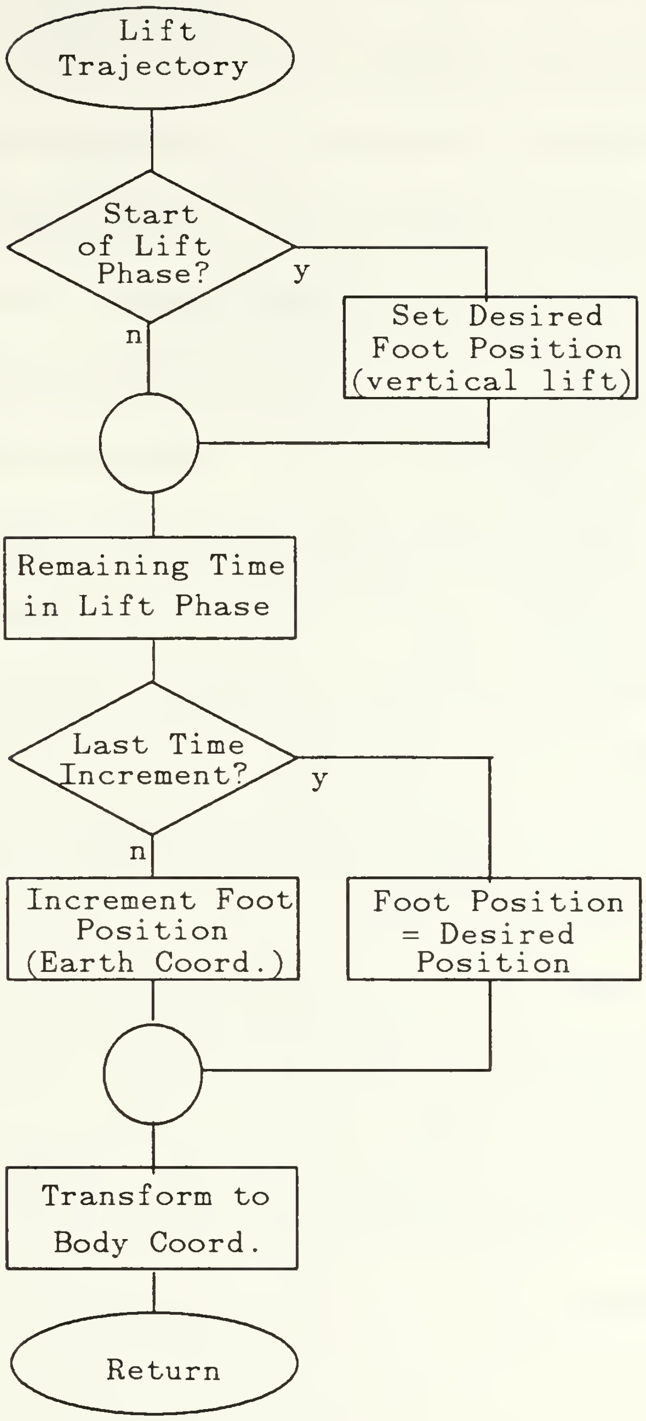

Lift Trajectory
Start of Lift Phase?
y
n
Set Desired Foot Position (vertical lift)
Remaining Time in Lift Phase
Last Time Increment?
y
n
Increment Foot Position (Earth Coord.)
Foot Position = Desired Position
Transform to Body Coord.
Return

Figure 4.13 Lift Trajectory Module

transfer phase value between the two transition point values yields a response from the *transfer trajectory* module (Fig. 4.14 and 4.15). and a phase value greater than the transfer-placement transition value yields a relative foot position calculation from the *placement trajectory* module (Fig. 4.16).

The foothold selection algorithms contained in the *transfer trajectory* module are discussed in section III.C.3. Note that within this module the desired end foot position is treated differently in the follow-the-leader and forward wave gait modes. The forward wave gait. with its high degree of maneuverability, has a considerable greater probability that the projected ideal position toward the end of the transfer phase will be much different from that at the start. Therefore the desired foot position in the case of the forward wave gait is updated on each pass. In the follow-the-leader gait case it is only calculated during the first time through the the module.

The results of the calculation phase are the position and orientation of the body and the relative position of each of the feet. These values are used. with the inverse kinematic relations derived in section III.E.4. in the display phase to obtain the rotation angles and translation distances required to position the ASV's component parts.

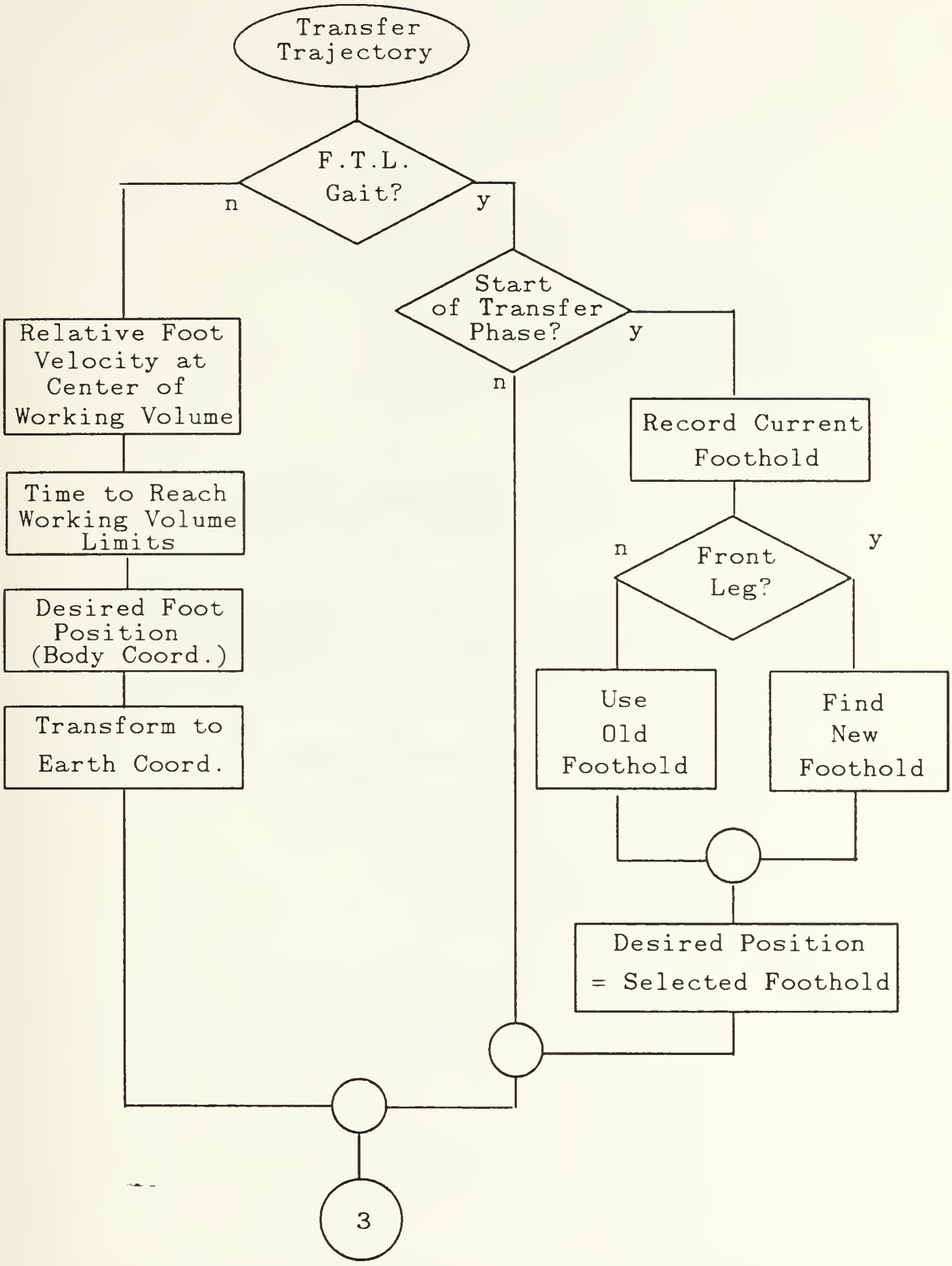

Transfer Trajectory
F.T.L. Gait?
n
y
Start of Transfer Phase?
y
n
Relative Foot Velocity at Center of Working Volume
Time to Reach Working Volume Limits
Desired Foot Position (Body Coord.)
Transform to Earth Coord.
Record Current Foothold
Front Leg?
n
y
Use Old Foothold
Find New Foothold
Desired Position = Selected Foothold
3

Figure 4.14 Transfer Trajectory Module (1 or 2)

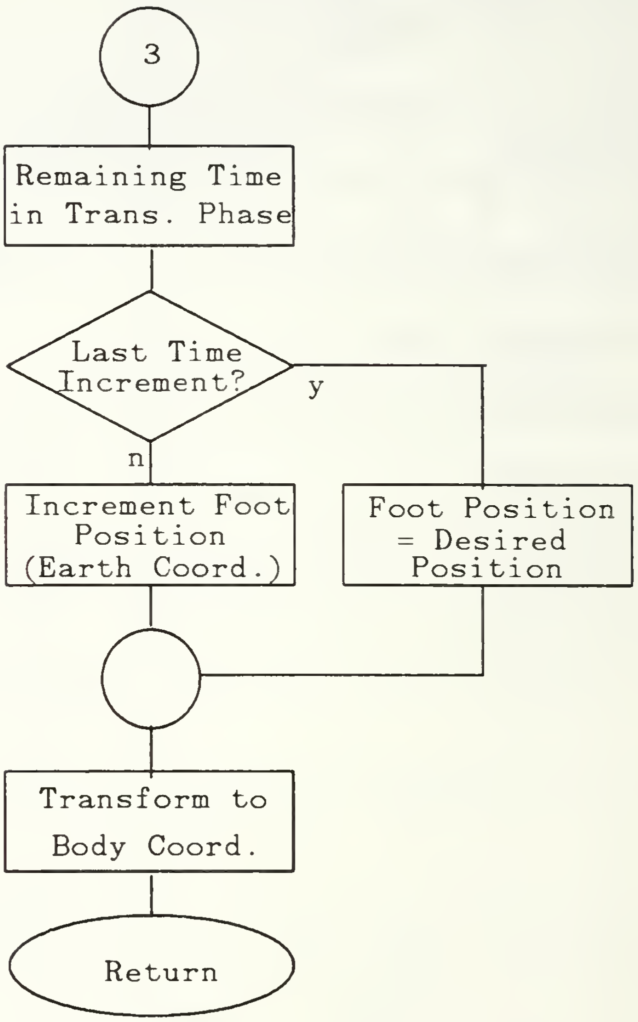

3
Remaining Time in Trans. Phase
Last Time Increment?
y
n
Increment Foot Position (Earth Coord.)
Foot Position = Desired Position
Transform to Body Coord.
Return

Figure 4.15 Transfer Trajectory Module (2 of 2)

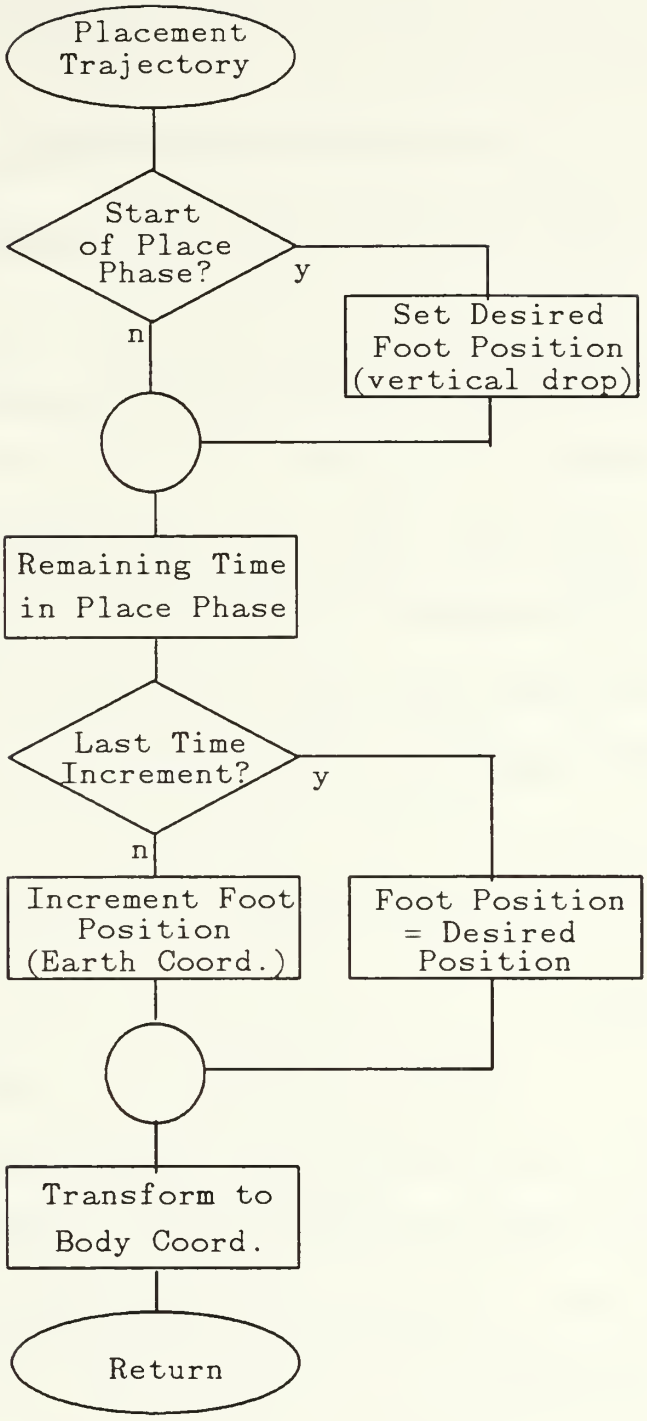

Placement Trajectory
Start of Place Phase?
y
n
Set Desired Foot Position (vertical drop)
Remaining Time in Place Phase
Last Time Increment?
y
n
Increment Foot Position (Earth Coord.)
Foot Position = Desired Position
Transform to Body Coord.
Return

Figure 4.16 Placement Trajectory Module

The ASV object lists are then edited and the updated parameters inserted into their corresponding rotation and translation commands. Once this is completed, the display calls are made for the background, the terrain object and then the properly ordered ASV object. Swapping display buffers completes the loop.

The ASV simulation program presented here consists of fifteen separate files linked. together with the graphics, math, and standard input-output libraries, using the UNIX *make* utility. The program files and Makefile listings are presented in the appendix. The routines were created in a modular fashion for ease of development and testing and to assist in future program changes. Constants are grouped into a single header file *walk.h*, for convenient reference and modification.

E. SUMMARY

This chapter describes the ASV simulation program. The first section is a guide for the operation of the program. It details the use of the menu system and the operator controls. The following section discusses the display of graphics on the IRIS-2400. The final section covers the organization and flow of the main routine and its modules.

V. SIMULATION PERFORMANCE

This chapter provides a brief review of the performance of the ASV simulation program. The review is largely subjective and is based on the author's experience with the operation of the simulation.

A. MODELING FIDELITY

The image of the vehicle on the screen appears to be a reasonable likeness of the actual ASV. This is believed, to a great extent, to be due to the proper scaling of dimensions of component parts, based on available blueprints of the ASV. Details such as the leg hydraulic actuator housings and the optical scanning radar, mounted on the cab of the vehicle, add to the visual effect. The color scheme of the simulation vehicle has been altered to enhance the visibility of the vehicle and its parts.

The walking motion of the model is very similar to that of the real vehicle. This observation is based on viewing of videotapes produced at the Ohio State University. A notable difference is that the simulation model is perceived to operate at a much slower speed. A simulation time increment of 1/100th of a second yields a display time to real time speed ratio of 1:30. Operating the simulation with a simulation time increment much greater than 1/100th of a second to compensate for this causes problems related to the optimum period and leg phase modules of the program. This leads to gross errors in the foot trajectory planning algorithms.

B. FORWARD WAVE TRIPOD GAIT

Driving in the forward wave tripod gait mode is a very simple task. Although a three-axis joystick would be prefered, the mouse-driven sliding bar control is easy to use and effective. Switching between control bars for forward, lateral, or rotational control can be accomplished with reasonable ease.

The external vantage point of the vehicle causes very little problem for maneuvering the vehicle. This may change as the model's speed increases and obstacles are introduced into the environment.

Overall, maneuverability of the ASV in the forward wave tripod gait mode is clearly demonstrated with this model.

C. FOLLOW-THE-LEADER TRIPOD GAIT

The follow-the-leader tripod gait appears to work especially well for forward, straight-line locomotion. Turning, however, is extremely restricted. Preliminary investigations indicate a minimum turning radius of 18 times the body length, using the constrained working volumes depicted in Figure 2.4. This is far greater than expected. An estimated envelope for turning, based on initial simulation trials, is shown in Figure 5.1. Steering commands falling outside of this envelope result in faults within the foot trajectory planning algorithms. These faults

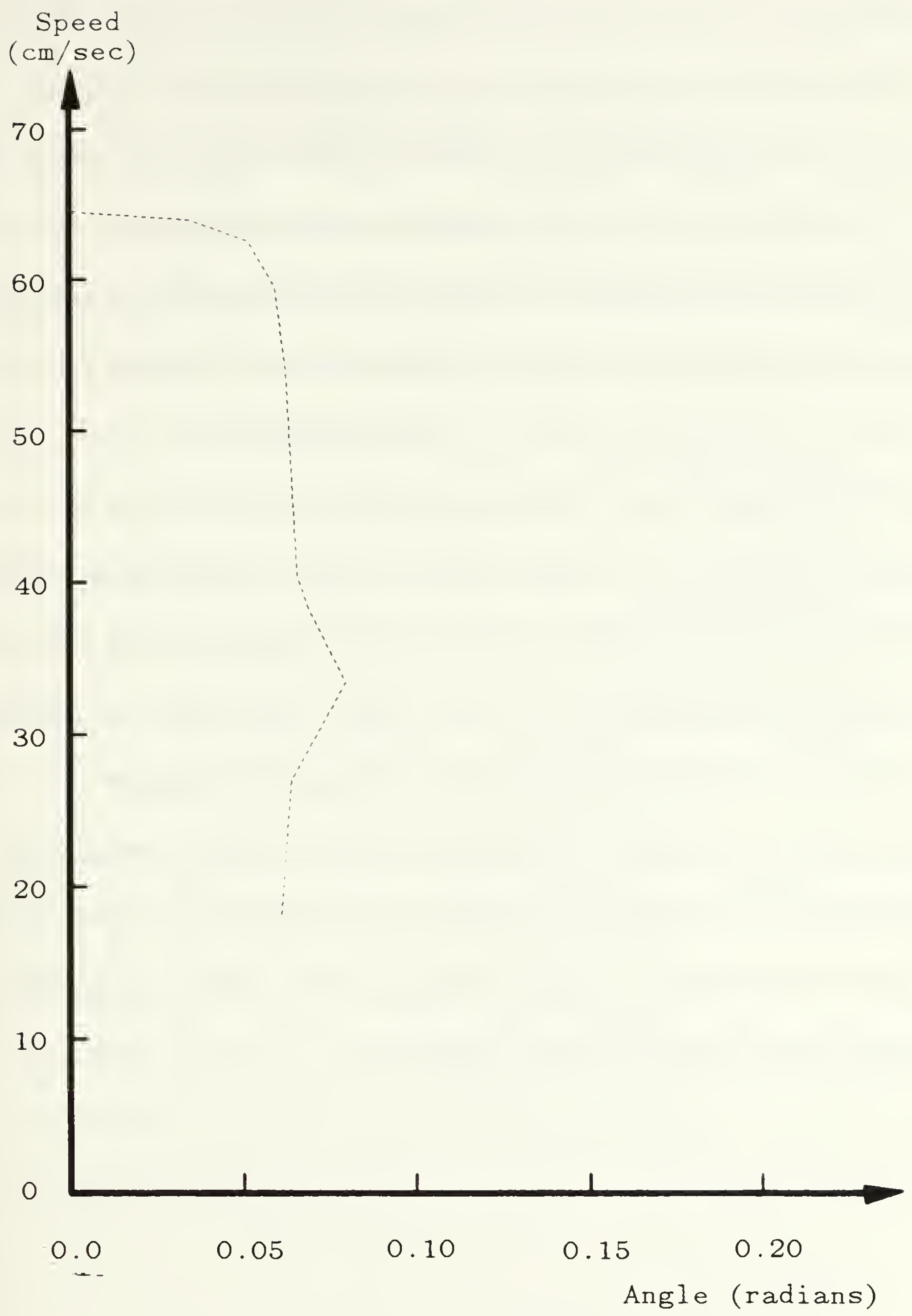


Figure 5.1 Steering Envelope using Constrained Working Volume

usually occur within the first four degrees of the turn attempt. Decreasing the display time interval extends the maximum magnitude of the command envelope. but only marginally improves the permitted relative direction command input.

The shape of the steering envelope is rather unexpected, and as of yet, unexplained. Factors likely to have the greatest influence on the envelope are the geometry of the leg's constrained working volume and the implementation of the optimum period and foot trajectory planning algorithms.

Expanding the constrained working volume to the full working volume has a remarkable effect on the maneuverability of the vehicle, while operating in the follow-the-leader gait mode. By doing so, the minimum turning radius improves to approximately five times the body length. This indicates a great potential advantage in utilizing dynamic stability algorithms for future gaits.

Overall the follow-the-leader gait and tractor-trailer steering appear to be successful for level, relatively obstruction-free terrain. Further research is needed to determine the nature of the limitations and the means to expand the vehicle's maneuverability while operating in this mode.

VI. SUMMARY AND CONCLUSIONS

In this thesis a tripod follow-the-leader gait class is introduced for use by six-legged walking vehicles. The class represents an extension of previously defined follow-the-leader gaits and should prove useful for legged vehicles traveling in rough or treacherous terrain conditions.

A new style of steering is also developed for follow-the-leader gaits. This steering mode exhibits a general response similar to that found in steering a wheeled tractor-trailer vehicle. With this mode, the driver is concerned only with specifying the velocity of the front of the vehicle. The algorithm ensures that the body of the vehicle follows along the path of the front.

An improved simulation model constructed to study the gait and steering algorithms is also presented in this thesis. The vehicle selected as a physical reference for the model is the Adaptive Suspension Vehicle (ASV), which is currently undergoing testing and development at the Ohio State University. The model developed is intended as a general tool for analyzing a variety of walking control algorithms for legged vehicles.

A. RESEARCH CONTRIBUTIONS

Previous research on follow-the-leader gaits [Ref. 3]. has concentrated on gaits that are temporally oriented. Since footholds are used by the following legs immediately after being abandoned by the lead leg. this produces a creeping motion with alternating leg and body movement.

Extending the class of follow-the-leader gaits into the spatial domain relieves the requirement of immediately utilizing a foothold as soon as it is abandoned. This gives a greater degree of freedom to leg movement and allows the possibility of smooth. continuous body motion with shorter leg stroke.

The nature of a follow-the-leader gait greatly constrains the maneuverability of the walking vehicle. The vertical projection of the vehicle's center of gravity is required to fall within the support pattern of the legs and is therefore confined by the history of footholds produced by the lead legs. The similarity of this problem to that of a trailer pulled by a tractor cab has inspired the adoption of the term "tractor-trailer" steering. With tractor-trailer style steering. the driver controls the path of the front of the vehicle. As long as the driver does not turn too sharply (possibly causing a wheeled tractor-trailer to jack-knife). the vehicle's body follows along this path.

The selection of footholds for the leading legs is based on projecting the relative heading vector provided by the operator. The location of recently abandoned footholds is retained within the control algorithm for use by the middle and rear legs.

The simulation presented in this study models the kinematics of the ASV. The model incorporates many of the simulation features presented by Lee [Ref. 4], including omnidirectional control, automatic body altitude and attitude regulation, leg motion planning, body deceleration, and filters between the control inputs and reaction which provides the operator with the "feel" of vehicle dynamics. A simplified variation of constrained working volumes is also used.

The simulation program has a modular design which creates a flexible environment for studying various gaits and control algorithms. The program as currently configured has two modes of operation. The first features a forward wave tripod gait with three-axis control for steering in body coordinates. The second mode utilizes the follow-the-leader tripod gait and two-axis control tractor-trailer style steering, developed in this study. The program's displays and controls are operated with a mouse-driven menu package using a single mouse button.

The graphics presentation is greatly improved over Lee's monochrome line-drawing representation. Three-dimensional, solid body, color graphics are made possible through the implementation of the model on the special purpose software and hardware of the IRIS-2400 system. This provides a notable enhancement of realism for the vehicle simulation.

B. RESEARCH EXTENSIONS

It has become clear, through the work of developing this study, that there are many directions in which future research could be pursued. Four major areas to be considered for extension are: quantitative measurement of the FTL tripod gait performance, improvement of program features, improvement of display speed, and expansion of upper level control algorithms using artificial intelligence.

Developing performance criteria for the simulated ASV is critical if one is to effectively use the program as an aid for developing and evaluating walking algorithms for the actual machine. Initial research might well concentrate on measuring turning radii, steering reaction times, stability margins and basic parameters of mobility.

As with any simulation model, there are many desired features which could be added to enhance realism. Perhaps the most important improvement for this type of mobile vehicle would be the inclusion of rough or uneven terrain into the model. Provisions were made in the development of this model for that eventuality. A few new algorithms, for functions such as estimating the support plane beneath the vehicle and adjusting the constrained working volume to conform to the terrain slope, will need to be written. It should be possible to follow the work of Lee [Ref. 4], at least initially, in improving the simulation in this direction.

Because the ASV is designed for rough terrain locomotion, developing a good foothold search algorithm is important. In addition, the inclusion of foothold search into the simulation model would enable the FTL gait to be better evaluated with respect to other types of gaits in various terrain conditions. This would quantify the advantages of reduced foothold probing requirements for the FTL gait.

This simulation could also be used to further develop steering mechanisms for the ASV. Most notably, the algorithm for the tractor-trailer style steering uses a simple method for body positioning based on the centroid of the established footholds. A different method for body steering which minimizes the maximum leg excursions might improve the vehicle's turning radius.

Dynamic modeling and supplementing the model's kinematics would greatly improve the realism of vehicle movement. Moreover, it should also notably increase mobility, as the vehicle would be free to utilize its leg's full working volumes. This should also lead towards the development of a great number of new gaits, which are dynamically, but not statically, stable.

Graphics techniques can be improved to enhance the realism of the displayed image. Features such as shading, depth-cueing, reflectivity of surfaces, terrain definition, and increased vehicle detail are all possible using current state-of-the-art techniques. Higher resolution monitors and an enlarged number of bit planes in the display hardware are also highly desirable.

Adding additional features to the model has the decided disadvantage of requiring more cpu time for the simulation. As the program now exists, the simulated vehicle moves and reacts markedly slower than the actual vehicle.

There are, however, ways to improve the display time for the model. The prime means is of course to upgrade the hardware, using newly developed and more capable machines. The code may also be streamlined for efficiency. Possibly several of the interactive features could be reduced or eliminated.

Another possibility for improving display time is the replacement of the integration routines within the body kinematics section of the program with an incremental homogeneous transformation matrix technique used by Lee [Ref. 4]. Integration is used here because of the simplicity of the technique and the author's familiarity with the IRIS-2400 special hardware commands for rotation and translation. It may be that the homogeneous transformation matrix could also be used directly with the special hardware to provide the full transformation with fewer trigonometric computations. This possibility has not been investigated by the author.

An interesting avenue of research to explore is to automate the upper levels of the control hierarchy. It may be possible to use an expert system shell running on a special purpose LISP machine to provide driving commands to this simulation. As of this writing, efforts are underway by others to establish communications between the IRIS-2400 system and a Symbolics 3675 LISP machine.

Extensions to the work presented in this thesis are possible and will likely prove very fruitful. It is hoped that this line of research will lead to more efficient and practical gaits and control algorithms for legged walking vehicles in rough terrain.

LIST OF REFERENCES

1. Anon., *Logistical Vehicle Off-Road Mobility*, Project TCCO 62-5. U.S. Army Transportation Combat Developments Agency. Fort Eustis. VA, Febuary 1967.

2. Bekker, M. G., *Introduction to Terrain-Vehicle Systems,* Ann Arbor, MI, University of Michigan Press. 1969.

3. 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. pp. 134-146.

4. Lee, W.J., *A Computer Simulation Study of Omnidirectional Supervisory Control for Rough-Terrain Locomotion by a Multilegged Robot Vehicle*, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, March 1984.

5. McGhee, Robert B., "Vehicular Legged Locomotion." *Advances in Automation and Robotics*. edited by G.N. Saridis, Jai Press., 1985, pp. 259-284.

6. Hirose, S., "A Study of Design and Control of a Quadruped Walking Vehicle," *The International Journal of Robotics Research*. Vol. 3. No. 2. Summer 1984. pp. 113-133.

7. Raibert, M.H., *Legged Robots that Balance*, The MIT Press, Cambridge, MA, 1986.

8. Russell, Jr., M., "ODEX I: The First Functionoid," *Robotics Age*, Vol. 5, No. 5, September 1983, pp. 12-18.

9. Anon., *Annual Report*, Odetics, Inc., Corporate Headquarters. 1515 S. Manchester Ave., Anaheim, CA, 92802-2907, March 1986.

10. Waldron, K.J., and McGhee, R.B., "The Adaptive Suspension Vehicle," *IEEE Control Systems Magazine*, December 1986, pp. 7-12.

11. McGhee. R.B.. and Jain. A.K.. "Some Properties of Regularly Realizable Gait Matricies." *Mathematical Biosciences.* Vol. 13. No. 1. Feburary 1972. pp. 179-193.

12. McGhee. R. B.. "Some Finite State Aspects of Legged Locomotion." *Mathematical Biosciences*, Vol. 2, No. 1. 1968. pp. 67-84.

13. Tsai, S.J., *A 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, March 1983.

14. Tomovic. R.. "A General Theoretical Model of Creeping Displacement," *Cybernetica*, Vol. 4, No. 2, 1961. pp. 98-107.

15. Waldron. K.J.. "Mobility and Controlability Characteristics of Mobile Robotic Platforms." *Proceeding on the 1985 IEEE International Conference on Robotics and Automation.* March 25-28. 1985. St.Louis. MO. pp. 237-243.

16. Moravec. H. P.. "The Stanford Cart and the CMU Rover." *Proceedings of the IEEE*, July 1983. pp. 872-884.

17. McGhee. R. B., and Iswandhi. G. I., "Adaptive Locomotion of a Multileggged Robot over Rough Terrain." *IEEE Transactions on Systems. Man. and Cybernetics.* SMC-9 (4), 1979, pp. 176-182.

18. Orin. D. E.. "Supervisory Control of a Multilegged Robot," *The International Journal of Robotics Research*, Vol. 1, No. 1, Spring 1982, pp. 79-90.

19. Frank, A.A., and McGhee, R.B., "Some Considerations Relating to the Design of Autopilots for Legged Vehicles," *Journal of Terramechanics*, Vol. 6. No. 1, 1969, pp. 23-35.

20. Raibert. M.H., and Sutherland. I.E., "Machines that Walk," *Scientific American*. Vol. 248, No. 2, January 1983, pp. 44-53.

21. Messuri. D.A., and Klein. C.A., "Automatic Body Regulation for Maintaining Stability of a Legged Vehicle During Rough-Terrain Locomotion," *IEEE Journal of Robotics and Automation*. Vol. RA-1. No. 3, September 1985, pp. 132-141.

22. Kwak, S.H., *A Computer Simulation Study of a Free Gait Motion Coordination Algorithm for Rough-Terrain Locomotion by a Hexapod Walking Machine*, Ph.D. dissertation, The Ohio State University, Columbus, Ohio, August 1986.

23. *IRIS User's Guide*, Silicon Graphics, Inc., Mountain View, CA, 1986.

24. Song, M.S., Lee, J.K., and Waldron, K.J., "Motion Study of Two- and Three-Dimensional Pantograph Mechanisms," *Procedings of 9th Applied Mechanisms Conference*. Kansas City, MO, October 1985, Sess. III.A, Paper I.

25. Lee, C.S.G., "On the Robotic Manipulator Control," *Advances in Automation and Robotics*, edited by G.N. Saridis, Jai Press., 1985. pp. 21-63.

APPENDIX

PROGRAM LISTING

```
/*******************************************************
   This program is written for the iris-2400
              walk.c
              ------
   This is the main program for the simulation.
   Relle Lyman          04 May  1987
*******************************************************/


#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include < math.h >

main()
{
   Object machineobject[4],leg[7][4],textobj,vertextobj ,thighobj[7][2][4],
          actuatorobj[7][2][4],shinobj[7][2][4], walker[4],groundobject;

 /* NOTE: this program uses only elements 1-6 of arrays and vectors.
    Legs are numbered to remain consistent with original research .  */

   Tag transrot_tag[4],tr_end_tag[4], legmovetag[7][4],
       actmovetag[7][2][4],bodytag1[4],
       thighmovetag[7][2][4],shinmovetag[7][2][4];

   Colorindex wmask ;

   int   i,j,k,n,
       program_status, /* desired status of program: RUN, HALT or RESET */
       selected_gait,  /* indicates which tripod gait is to be used */
       slow_flag,      /* flag indicating deceleration is needed */
       warning,        /* flag indicating supporting leg is outside of working volume */
       leg_status[7];  /* status of leg (supporting, liftoff, transfer, placement) */

   static float
      hx[7]= {0,155.,155.,0.,0.,-155.,-155.}, /* Leg attachment points */
      hy[7] = {0,50.,-50.,50.,-50.,50.,-50.},
      hz[7]= {0,23.,23.,23.,23.,23.,23.},
      l4[7] = {0,L4,-L4,L4,-L4,L4,-L4};

   static Angle  theta[7]={0,0,0,0,0,0,0},     /* Leg component angles */
                alpha[7]={364,364,364,364,364,-364,-364},
                gamma[7]={317,317,317,317,317,-317,-317};
```

```
/* walk.c */

float  temp,temp1,temp2,temp3,top,bottom, /* Temporary variables */
       alpharad[7],        /* Leg component angles in radians */
       thetarad,
       legcoord_x[7],      /* Foot position in leg coordinates */
       legcoord_y[7],
       legcoord_z[7],
       azimuth,elev,roll,  /* Body Euler angles (rads) */
       ordered_vel_mag,    /* Ordered velocity of the cockpit (magnitude) */
       ordered_vel_dir,    /* Ordered velocity of the cockpit (direction) */
       d1[7],              /* Joint variables */
       d2[7],
       knee[7][2],         /* Relative position of knee */
       foot[7][2],         /* Relative position of foot */
       h[4][4],            /* Homogeneous transformation matrix */
       invh[4][4],         /* Inverse homogeneous transformation matrix */
       legphase[7],        /* Phase of individual legs */
       rel_legphase[7],    /* Phase of individual legs relative to leg one*/
       period,             /* Period of leg cycle */
       min_period,         /* Minimum allowed period */
       tx,ty,tz;           /* Earth coordinates of body position */

vector  rot_rate,   /* Body rotation rates */
        trans_rate, /* Body translation rates */
        ordered_rate,  /* Ordered lateral and longitudinal translation and yaw rates */
        footpos[7],    /* Position of foot in earth coordinates */
        b_footpos[7],  /* Position of foot in body coordinates */
        fh[7],         /* selected footholds (earth coordinates) */
        oldfh[7];      /* old selected footholds (earth coordinates) */

work_vol  cwv[7];   /* Constrained working volumes */

plane   spe;      /* Estimated support plane */


/* Initialize the IRIS graphics */

ginit() ;          /* standard IRIS graphics initialization */

doublebuffer() ;   /* double buffering mode */

gconfig() ;        /* configure the IRIS (use the above commands */

wmask=(1<<getplanes())-1 ; /* enable all the bit planes for writing */
                           /* set to 2**(getplanes())minus one      */
                           /* all bit planes on                     */
writemask(wmask) ;

backface(TRUE);            /* set backface polygon removal on */
```

```
qdevice(MIDDLEMOUSE);        /* set up the queue for the menu */
tie(MIDDLEMOUSE,MOUSEX,MOUSEY);

mapcolor(LTYELLOW,225,225,0);        /* create new colors */
mapcolor(WHITE1,230,230,230);

viewport(410,1023,0,767) ;                /* set world view */
perspective(600,(614.0/768),0.0,1023.0) ;

/*   make the ground   */
makeground(&groundobject);

/*   make the robot    */
makewalker(machineobject,d1,d2,theta,knee,gamma,alpha,transrot_tag,
        tr_end_tag,walker,leg,thighobj,actuatorobj,shinobj,
        legmovetag,thighmovetag,actmovetag,shinmovetag,tx,ty,tz,roll,
        elev,azimuth,hx,hy,hz,l4) ;

/* Initialize the ASV walking routine parameters. */
initialize(h,invh,&rot_rate,&trans_rate,&ordered_rate,&spe,&period,
        leg_status,legphase,rel_legphase, footpos,b_footpos,cwv,fh,
        oldfh,&selected_gait,&ordered_vel_mag,&ordered_vel_dir,
        &min_period,&program_status,&tx,&ty,&tz,&roll,&elev,&azimuth);

while(TRUE)      /* Main program loop */
{
   /* Input the driver's commands. */
   driver_command(&ordered_rate,&rot_rate,&trans_rate,&program_status,
         b_footpos,&period,alpha,gamma,theta,&slow_flag,&roll,&elev,
         &azimuth,&tx,&ty,&tz,&ordered_vel_mag,&ordered_vel_dir,fh,
         &selected_gait);

   if (program_status == HALT)
   {
      /* Quit program. */
      break;
   }
   if (program_status == RESET)
   {
      /* Reinitialize the ASV walking parameters. */
      initialize(h,invh,&rot_rate,&trans_rate,&ordered_rate,&spe,&period,
         leg_status,legphase,rel_legphase, footpos,b_footpos,cwv,fh,
         oldfh,&selected_gait,&ordered_vel_mag,&ordered_vel_dir,
         &min_period,&program_status,&tx,&ty,&tz,&roll,&elev,&azimuth);
   }
```

```
    /* Calculate the estimated support plane. */
    /* Future revision needed for rough terrain. */
    support_plane(&spe);


    /* Calculate the body rotation and translation rates. */
    body_rates(&rot_rate,&trans_rate,&spe,h,invh,&ordered_rate,
            &tx,&ty,&tz,&roll,&elev,&azimuth);

    /* Calculate the constrained working volume for the legs. */
    con_work_vol(cwv,b_footpos,leg_status,&warning);

    /* Calculate the optimal period for walking. */
    optimal_period(legphase,b_footpos,&rot_rate,&trans_rate,cwv,
                leg_status,&period);

    /* Decelerate if necessary. */
    decelerate(&trans_rate,&rot_rate,&period,&slow_flag,&min_period);

    /* Calculate the phase of each leg. */
    leg_phase(legphase,rel_legphase,&period);

    /* Calculate the new position for each foot. */
    foot_trajectory(legphase,&period,leg_status,footpos,b_footpos,fh,oldfh,
                invh,h,cwv,&trans_rate,&rot_rate,&selected_gait);


    /* Display the ASV on the screen. */

    /* This section computes the new parameters to position the legs
       relative to the body, based on the relative position of the feet.
       It then check to ensure that no actuator positions exceed the limits. */

    /* Convert foot position to leg coordinates. */
    for(i=1; i<5; i++)
    {
     legcoord_x[i] = b_footpos[i].x - hx[i];
     legcoord_y[i] = b_footpos[i].y - hy[i];
     legcoord_z[i] = b_footpos[i].z - hz[i];
    }

    /* The foot position of the rear legs are changed to compensate for
      the 180 degree rotation used in the leg construction routine. */
    for(i=5; i<7; i++)
    {
      legcoord_x[i] = hx[i] - b_footpos[i].x;
      legcoord_y[i] = b_footpos[i].y - hy[i];
     legcoord_z[i] = b_footpos[i].z - hz[i];
    }
    for(i=1; i<7 ; i++)
    {
        /* generate required parameters d1,d2, theta */

        d2[i] = legcoord_x[i]/5.0;
        temp= legcoord_y[i] * legcoord_y[i];
        temp2=legcoord_z[i]*legcoord_z[i];
        d1[i] = -(5.0*L3-sqrt(temp+temp2-L4*L4))/4.0;
        temp1=5.0*L3+4.0*d1[i] ;

        switch (i)
        {
          case 1:
          case 3:
         case 5: temp3 = temp1*legcoord_y[i] + L4*legcoord_z[i];
               break;
          case 2:
         case 4:
         case 6: temp3 = temp1*legcoord_y[i] - L4*legcoord_z[i];
        }
       thetarad = asin(temp3/(temp1*temp1 + L4*L4));
       theta[i] = thetarad * 573 + 0.5;
    }

    for(i=1 ;i<7 ; i++)       /* prepare parameters for graphics  */
    {                   /* update on all 6 legs             */
      temp = L3+d1[i] ;
     temp1 = d2[i]*d2[i] + temp*temp;
     temp2 = (L1*L1 - L6*L6 + temp1)/(2.0*L1*sqrt(temp1));

      alpharad[i]=((PI/2)-atan(d2[i]/temp)-acos(temp2)) ;

     /* Note: One half of a degree has been added to all angles */
      alpha[i]=(alpharad[i]*573+.5);

      knee[i][0]=(L2*cos(alpharad[i])+.5); /* relative to baseplate */
      knee[i][1]= -((L2*sin(alpharad[i])- d1[i])+0.5);/* relative to baseplate */

      foot[i][0]= (5.0*d2[i]+.5);              /* relative to baseplate */
      foot[i][1]= -(5.0*L3+4.0*d1[i]+.5) ;  /* relative to baseplate */
      top=(knee[i][0]-foot[i][0]);
      bottom=(knee[i][1]-foot[i][1]);

      gamma[i]=(atan(top/bottom)*573+.5) ;
```

```
/* walk.c */
        for (n=0; n<4; n++) /* The walker is updated in each quadrant */
        {
          editobj(thighobj[i][0][n]) ;        /* edit each leg to new   */
            objreplace(thighmovetag[i][0][n]) ; /* location           */
            rotate(alpha[i],'Y') ;
             closeobj() ;

          editobj(thighobj[i][1][n]) ;
            objreplace(thighmovetag[i][1][n]) ;
            translate(0.0,0.0,d1[i]) ;
            closeobj() ;

          editobj(actuatorobj[i][0][n]) ;
            objreplace(actmovetag[i][0][n]) ;
            rotate(alpha[i],'Y') ;
            closeobj() ;

          editobj(actuatorobj[i][1][n]) ;
            objreplace(actmovetag[i][1][n]) ;
            translate(d2[i],0.0,-L3) ;
            closeobj() ;

          editobj(shinobj[i][0][n]) ;
            objreplace(shinmovetag[i][0][n]) ;
            rotate(gamma[i],'Y') ;
            closeobj() ;

          editobj(shinobj[i][1][n]) ;
            objreplace(shinmovetag[i][1][n]) ;
            translate((float)(knee[i][0]),0.0,(float)knee[i][1]) ;
            closeobj() ;

          editobj(leg[i][n]) ;
            objreplace(legmovetag[i][n]) ;
            rotate(theta[i],'X') ;
            closeobj() ;
        }  /* end quadrant loop  */
      } /* end for leg loop i=1 ... */

      for (n=0; n<4; n++)
      {
        editobj(machineobject[n]) ;
          objdelete(transrot_tag[n],tr_end_tag[n]);
          objinsert(transrot_tag[n]);
            translate(tx,ty,tz);
            rotate((int) (azimuth*573),'Z');
            rotate((int)(elev*573),'Y');
            rotate((int)(roll*573),'X');
          closeobj() ;
      } /* end of quadrant loop */
```

```
                                /* set up the background   */
   color(BLUE);
   clear();

   /* Keep the viewing relationship constant. */
   perspective(600,(614.0/768),0.0,1023.0) ;
   lookat(800.0-tx,800.0+ty,550.0,tx,ty,-50.0,1100);

                                /* CALL THE GROUND    */
   callobj(groundobject);

   /* Display the ASV in the correct quadrant configuration */
   if (azimuth < 0.0)
   {
   {
     azimuth += 2.0 * PI;
   }
   if (azimuth > 2.0 * PI)
   {
     azimuth -= 2.0 * PI;
   }
   if (azimuth < 0.25*PI)
   {
     callobj(machineobject[0]);
    }
   if ((azimuth >= 0.25*PI)&&(azimuth < 0.75*PI))
   {
     callobj(machineobject[3]);
   }
   if ((azimuth >= 0.75*PI)&&(azimuth < 1.25*PI))
   {
     callobj(machineobject[2]);
   }
   if ((azimuth >= 1.25*PI)&&(azimuth < 1.75*PI))
   {
     callobj(machineobject[1]);
   }
   if (azimuth >= 1.75*PI)
   {
     callobj(machineobject[0]);
   }

   swapbuffers() ;
 }                              /* end of main program loop */
```

```
  /* Clean up the screen.  */

  color(BLACK) ;
  clear() ;
  swapbuffers();
  color(BLACK);
  clear();
  swapbuffers();
  finish() ;
  gexit() ;


} /* END OF MAIN PROGRAM */
```

```
/*****************************************************************
    This is the header file for the program walk.c.
                walk.h
                ------
                Relle Lyman
                14 May 1987
*****************************************************************/

#define BETA                   0.5
#define DELTA_TIME             0.010
#define TIME_CONSTANT_1 0.1
#define TIME_CONSTANT_2 0.25
#define TIME_CONSTANT_3 0.5
#define FTL_GAIT                1
#define FWD_WAVE_GAIT           2
#define FORWARD                 1
#define BACKWARD                0
#define END_LIFT_PHASE          0.2
#define BEGIN_PLACE_PHASE 0.8
#define SUPPORTING              0
#define LIFTOFF                 1
#define TRANSFER_FORWARD 2
#define PLACEMENT               3
#define ON                     1
#define OFF                    0
#define LENGTH                 310.0   /* The length between the forward
                                          and aft hip joints */
#define HALFLENGTH             155.0   /* Half the length between the forward
                                          and aft hip joints */
#define FOOTLIFTHEIGHT         40.0
#define LONG_TIME              1000000
#define H0                     160.0   /* Desired body height (cm)*/
#define OUTER_LIMIT            6.08    /* cm/sec  */
#define INNER_LIMIT            1.52    /* cm/sec  */
#define RUN                    0
#define HALT                   1
#define RESET                  2
#define NORMAL                 0
#define SLOW                   1
#define PI                     3.14159

#define UP                     1
#define DOWN                   2
#define IN                     1
#define OUT                    0
#define LTYELLOW               100
#define WHITE1                 107
#define TEXTCOLOR              BLACK
#define NOHIGHLIGHT            LTYELLOW
#define ACTIVEHIGHLIGHT        RED
#define INACTIVEHIGHLIGHT YELLOW
```

```
/* walk.h */

#define L1      20.0
#define L2      102.0
#define L3      24.0
#define L4      32.0
#define L6      30.0


struct mag_in_xyz    /* magnitude along x, y, and z axes */
{
  float x,y,z;
};
typedef struct mag_in_xyz vector;

struct plane_coefficients   /* plane coefficients */
{
  float a,b,c,d;
};
typedef struct plane_coefficients plane;

typedef struct
{
  float  min,
         max,
         center;
}  dimensions;

typedef struct
{
  dimensions  x,
              y,
              z;
}  work_vol;


typedef struct
{
  int left,right,top,bottom,x0,y0,x1,y1,x2,y2;
  char *text0,*text1,*text2;
}  menubox;
```

```
/*************************************************************************

   This is a function for the iris 2400 program walk.c.
                    init.c
                  ------------
           Relle Lyman                   27 Apr 1987
*************************************************************************/


#include "gl.h"
#include "device.h"
#include "walk.h"

initialize(h,invh,rot_rate,trans_rate,ordered_rate,spe,period,leg_status,
     legphase,rel_legphase,footpos,b_footpos,cwv,fh,oldfh,selected_gait,
     ordered_vel_mag,ordered_vel_dir,program_status,tx,ty,tz,roll,elev,azimuth)

/*  This function computes the body rotation and translation rates. */


vector *rot_rate,        /* rotation rate */
     *trans_rate,        /* translation rate */
     *ordered_rate,     /* ordered x translation, y translation,
                        and z rotation rates  */
     fh[7],              /* selected footholds (earth coord.) */
     oldfh[7],           /* old selected footholds (earth coord.) */
     footpos[7],         /* position of the foot in earth coord. */
     b_footpos[7];      /* position of the foot in body coord. */

plane  *spe;           /* support plane in earth coord */

work_vol cwv[7];        /* constrained working volume */

float  h[4][4],         /* homogeneous transformation matrix */
     invh[4][4],        /* inverse of transformation matrix */
     legphase[7],     /* phase of the phase */
     rel_legphase[7], /* phase of the leg relative to leg one */
     *period,          /* body support period */
     *tx,*ty,*tz,     /* position of body in earth coordinates */
     *roll,*elev,*azimuth, /* body euler angles */
     *ordered_vel_mag, /* ordered velocity of the steering pt (magnitude)*/
     *ordered_vel_dir;  /* ordered velocity of the steering pt (direction)*/

int    leg_status[7],         /* status of the leg */
     *program_status,        /* desired status of program */
     *selected_gait;       /* type of tripod gait to be used */

{

  int   i,j;

  float  modulus_one();      /* modulus one function */
```

```
/* init.c */

  /* initialize the transformation matrix */
  h[0][0] = 1.0;
  h[0][1] = 0.0;
  h[0][2] = 0.0;
  h[0][3] = 0.0;

  h[1][0] = 0.0;
  h[1][1] = 1.0;
  h[1][2] = 0.0;
  h[1][3] = 0.0;

  h[2][0] = 0.0;
  h[2][1] = 0.0;
  h[2][2] = 1.0;
  h[2][3] = H0;    /* initial height of the center of the body */

  h[3][0] = 0.0;
  h[3][1] = 0.0;
  h[3][2] = 0.0;
  h[3][3] = 1.0;

  /* initialize the inverse transformation matrix */
  for (i=0; i<3; i++)
  {
    for (j=0; j< 3; j++)
    {
      invh[i][j] = h[j][i];
    }
    invh[3][i] = 0.0;
    invh[i][3] = -(h[0][i]*h[0][3] + h[1][i]*h[1][3] +
             h[2][i]*h[2][3]);
  }
  invh[3][3] = 1.0;

  /* initialize the body rotation and translation rates */
  rot_rate->x = 0.0;
  rot_rate->y = 0.0;
  rot_rate->z = 0.0;
  trans_rate->x = 0.0;
  trans_rate->y = 0.0;
  trans_rate->z = 0.0;

  /* initialize the commanded body rates */
  ordered_rate->x = 0.0; /* translation */
  ordered_rate->y = 0.0; /* translation */
  ordered_rate->z = 0.0; /* rotation */

  *period = LONG_TIME;

  *selected_gait = FWD_WAVE_GAIT;
```

```
/* init.c */
/* initialize the relative leg phase */
rel_legphase[1] = 0.0;
rel_legphase[2] = 0.5;
rel_legphase[3] = BETA;
rel_legphase[4] = BETA-0.5;
rel_legphase[5] = 2*BETA - 1.0;
rel_legphase[6] = modulus_one(2*BETA - 0.5);

/* initialize the leg status and phase */
for (i=1; i<7; i++)
{
   leg_status[i] = SUPPORTING;
   legphase[i] = rel_legphase[i];
}

/* initialize the constrained working volume for each leg */
cwv[1].x.min    =   95.0;
cwv[1].x.max    =  215.0;
cwv[1].x.center =  155.0;

cwv[1].y.min    =   60.0;
cwv[1].y.max    =  131.0;
cwv[1].y.center =   95.0;

cwv[1].z.min    = -240.0;
cwv[1].z.max    = - 80.0;
cwv[1].z.center = -160.0;

cwv[2].x.min    =   95.0;
cwv[2].x.max    =  215.0;
cwv[2].x.center =  155.0;

cwv[2].y.min    = -131.0;
cwv[2].y.max    = - 60.0;
cwv[2].y.center = - 95.0;

cwv[2].z.min    = -240.0;
cwv[2].z.max    = - 80.0;
cwv[2].z.center = -160.0;

cwv[3].x.min    = - 60.0;
cwv[3].x.max    =   60.0;
cwv[3].x.center =    0.0;

cwv[3].y.min    =   60.0;
cwv[3].y.max    =  131.0;
cwv[3].y.center =   95.0;

cwv[3].z.min    = -240.0;
cwv[3].z.max    = - 80.0;
cwv[3].z.center = -160.0;
```

```
/* init.c */

   cwv[4].x.min    = - 60.0;
   cwv[4].x.max    =   60.0;
   cwv[4].x.center =    0.0;

   cwv[4].y.min    = -131.0;
   cwv[4].y.max    = - 60.0;
   cwv[4].y.center = - 95.0;

   cwv[4].z.min    = -240.0;
   cwv[4].z.max    = - 80.0;
   cwv[4].z.center = -160.0;

   cwv[5].x.min    = -215.0;
   cwv[5].x.max    = - 95.0;
   cwv[5].x.center = -155.0;

   cwv[5].y.min    =   60.0;
   cwv[5].y.max    =  131.0;
   cwv[5].y.center =   95.0;

   cwv[5].z.min    = -240.0;
   cwv[5].z.max    = - 80.0;
   cwv[5].z.center = -160.0;

   cwv[6].x.min    = -215.0;
   cwv[6].x.max    = - 95.0;
   cwv[6].x.center = -155.0;

   cwv[6].y.min    = -131.0;
   cwv[6].y.max    = - 60.0;
   cwv[6].y.center = - 95.0;

   cwv[6].z.min    = -240.0;
   cwv[6].z.max    = - 80.0;
   cwv[6].z.center = -160.0;

   /* initialize the selected foothold positions */
   fh[1].x = cwv[1].x.center + LENGTH/12.0;
   fh[2].x = cwv[2].x.center - LENGTH/12.0;
   fh[3].x = cwv[3].x.center - LENGTH/12.0;
   fh[4].x = cwv[4].x.center + LENGTH/12.0;
   fh[5].x = cwv[5].x.center + LENGTH/12.0;
   fh[6].x = cwv[6].x.center - LENGTH/12.0;

   for (i=1;i<7;i++)
   {
     fh[i].y = cwv[i].y.center;
     fh[i].z = 0.0;
   }
```

```
  /* initialize the earth relative foot positions */
  for (i=1;i<7;i++)
  {
    footpos[i].x = fh[i].x;
    footpos[i].y = fh[i].y;
    footpos[i].z = fh[i].z;
  }

  /* initialize the old selected foothold positions */
  for (i=1;i<7;i++)
  {
    oldfh[i].x = fh[i].x - LENGTH/3.0;
    oldfh[i].y = cwv[i].y.center;
    oldfh[i].z = 0.0;
  }

  /* initialize the body relative foot positions */
  for (i=1;i<7;i++)
  {
    b_footpos[i].x = cwv[i].x.center;
    b_footpos[i].y = cwv[i].y.center;
    b_footpos[i].z = cwv[i].z.center;
  }

  /* initialize the estimated support plane */
  spe->a = 0.0;
  spe->b = 0.0;
  spe->c = 1.0;
  spe->d = 0.0;

  /* initialize the ordered velocity of the steering point */
  *ordered_vel_mag = 0.0;
  *ordered_vel_dir = 0.0;

  /* initialize the body attitude and position */
  *roll = 0.0;
  *elev = 0.0;
  *azimuth = 0.0;
  *tx = 0.0;
  *ty = 0.0;
  *tz = H0;

  /* initialize desired program status */
  *program_status = RUN;

} /* end of initialize */
```

```
/**********************************************************************
  This is a function for the iris 2400 program walk.c.
                    driver.c
                    ------------
           Relle Lyman                    13 May 1987
***********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

  menubox   box[]={
    0.0,0,0,0,0,0,0,0,0,0,"not","used","here",
    100,200,670,525,120,567,120,597,120,627,"GAIT","WAVE","FWD",
    200,300,670,525,220,567,220,597,220,627,"ATTITUDE","AND","ALTITUDE",
    300,400,670,525,320,567,320,597,320,627," ","RESET"," ",
    100,200,525,380,120,422,120,452,120,482,"GAIT"," ","FTL",
    200,300,525,380,220,422,220,452,220,482,"REPORT"," ","STATUS",
    300,400,525,380,320,422,320,452,320,482,"PROGRAM"," ","EXIT",

    100,200,310,230,120,250,120,270,120,290,"REVERSE","FORWARD","TRANSLATE",
    100,200,230,150,120,170,120,190,120,210,"RIGHT","LEFT","TRANSLATE",
    100,200,150,70,120,90,120,110,120,130,"RIGHT","LEFT","ROTATE",

    100,200,310,230,120,250,120,270,120,290," "," "," ",
    100,200,230,150,120,170,120,190,120,210," "," "," ",
    100,200,150,70,120,90,120,110,120,130," "," "," "};

driver_command(ordered_rate,rot_rate,trans_rate,program_status,
      b_footpos,period,alpha,gamma,theta,slow_flag,roll,elev,
      azimuth,tx,ty,tz,ordered_vel_mag,ordered_vel_dir,fh,selected_gait)

/* This function inputs the driver's commands using a menu and
   the mouse. */

vector *ordered_rate, /* ordered x translation, y translation, and z rotation rates */
      *rot_rate,       /* actual rotation rate vector */
      *trans_rate,     /* actual translation rate vector */
      b_footpos[7],    /* position of foot in body coordinates */
      fh[7];           /* selected footholds (in earth coordinates) */

int   *program_status, /* desired status of the program RUN/HALT/RESET */
     *selected_gait,  /* desired tripod gait */
     *slow_flag;      /* flag indicating deceleration is required */

float *period,         /* body support period */
     *tx,*ty,*tz,     /* body translation distance (Earth coord) */
     *azimuth,*elev,*roll, /* body Euler angles  */
     *ordered_vel_mag, /* ordered cockpit velocity (magnitude) */
     *ordered_vel_dir; /* ordered cockpit velocity (direction) */
```

```
/* driver.c */

Angle alpha[7],         /* thigh angle */
      gamma[7],          /* shin angle */
      theta[7];         /* leg lateral angle */

{
  Device     dummy,x,y;

  static int buttonflag = UP,pick,potentialpick,mainmenupick,submenu,slidebar;

  int          i;

  float       barvalue;

  static float  time;

  char        str_orx[100],str_ory[100],str_orz[100],
           str_trx[100],str_try[100],str_rrz[100],str_time[100];



  pushmatrix();
  pushviewport();

  viewport(0,500,0,767);
  ortho2(0.0,500.0,0.0,767.0);

    color(CYAN);      /* screen background color */
    clear();

   /* Display simulation time on top of screen */
  color(TEXTCOLOR);
  time += DELTA_TIME;
  sprintf(str_time,"simulation time %8.3f",time);
  cmov2i(105,700);
  charstr(str_time);

    if (qtest() == MIDDLEMOUSE)   /* Button just pressed or released */
    {
      qread(&dummy);
      qread(&x);
      qread(&y);

      if (buttonflag == DOWN)   /* Button was just released. */
      {
        buttonflag = UP;

        if (potentialpick == 0)
        {
          /* No change */
        }
        else if (potentialpick < 7)    /* Main menu chosen */
        {
          mainmenupick = potentialpick;
          pick = 0;
          pick = potentialpick;
          potentialpick = 0;
        }
        else                /* submenu  chosen */
        {
          pick = potentialpick;  /* no change to main menu pick */
          potentialpick = 0;
        }
      }
      else                   /* Button was just pressed. */
      {
        buttonflag = DOWN;
      }
    } /* end of qtest */

    if (buttonflag == DOWN)     /* Find the box over which the
                                    cursor lies for highlighting. */
    {
      x = getvaluator(MOUSEX);
      y = getvaluator(MOUSEY);

      potentialpick = 0;

       for (i=1;i<7;i++)    /* Check the main menu. */
      {
        if (x < box[i].right && x > box[i].left &&
          y < box[i].top  && y > box[i].bottom)
         {
          potentialpick = i;
        }
      }
      if (submenu == 1)  /* Check submenu #1.  */
      {
         for (i=7;i<10;i++)
        {
          if (x < box[i].right && x > box[i].left &&
          y < box[i].top  && y > box[i].bottom)
           {
            potentialpick = i;
          }
        }
      }
      if (submenu == 2)  /* Check submenu #1. */
      {
         for (i=10;i<13;i++)
         {
           if (x < box[i].right && x > box[i].left &&
           y < box[i].top  && y > box[i].bottom)
            {
              potentialpick = i;
            }
         }
      }
    }
    else    /* button is up */
    {
      potentialpick = 0;
    }

    /* Display the menu. */

    for (i=1;i<7;i++)
    {
      if (i == potentialpick)
      {
        color(ACTIVEHIGHLIGHT);
       }
      else if (i == mainmenupick)
      {
        color(INACTIVEHIGHLIGHT);
      }
      else
      {
        color(NOHIGHLIGHT);
      }

      rectfi(box[i].left, box[i].bottom, box[i].right, box[i].top);
      color(TEXTCOLOR);
      recti(box[i].left, box[i].bottom, box[i].right, box[i].top);
      cmov2i(box[i].x0,box[i].y0);
      charstr(box[i].text0);
      cmov2i(box[i].x1,box[i].y1);
      charstr(box[i].text1);
      cmov2i(box[i].x2,box[i].y2);
      charstr(box[i].text2);
    }
```

```
  if (submenu == 1)       /* Display submenu #1. */
  {
    for (i=7;i<10;i++)
    {
      if (i == potentialpick)
      {
        color(ACTIVEHIGHLIGHT);
      }
      else if (i == pick)
      {
        color(INACTIVEHIGHLIGHT);
      }
      else
      {
        color(NOHIGHLIGHT);
      }

      rectfi(box[i].left, box[i].bottom, box[i].right, box[i].top);
      color(TEXTCOLOR);
      recti(box[i].left, box[i].bottom, box[i].right, box[i].top);
      cmov2i(box[i].x0,box[i].y0);
      charstr(box[i].text0);
      cmov2i(box[i].x1,box[i].y1);
      charstr(box[i].text1);
      cmov2i(box[i].x2,box[i].y2);
      charstr(box[i].text2);
    }

   color(WHITE);           /* Draw LED gages. */
   rectfi(200,70,400,370);
   color(BLACK);
   recti(200,70,300,150);
   recti(300,70,400,150);
   recti(200,150.300,230);
   recti(300,150,400,230);
   recti(200,230,300,310);
   recti(300,230,400,310);
   recti(200,310,300,370);
   recti(300,310,400,370);
   color(RED);
   cmov2i(205,350);
   charstr("ORDERED");
   cmov2i(205,330);
   charstr("RATE");
   cmov2i(305,350);
   charstr("ACTUAL");
   cmov2i(305,330);
   charstr("RATE");
```

```
    /* Display the parameter values. */
    sprintf(str_orx,"%7.2f",ordered_rate->x);
    sprintf(str_ory,"%7.2f",ordered_rate->y);
    sprintf(str_orz,"%7.2f",ordered_rate->z);
    sprintf(str_trx,"%7.2f",trans_rate->x);
    sprintf(str_try,"%7.2f",trans_rate->y);
    sprintf(str_rrz,"%7.2f",rot_rate->z);

    cmov2i(205,270);
    charstr(str_orx);
    cmov2i(205,190);
    charstr(str_ory);
    cmov2i(205,110);
    charstr(str_orz);
    cmov2i(305,270);
    charstr(str_trx);
    cmov2i(305,190);
    charstr(str_try);
    cmov2i(305,110);
    charstr(str_rrz);

  }

  if (submenu == 2)      /* Display submenu #2. */
  {
    for (i=10;i<13;i++)
    {
      if (i == potentialpick)
      {
        color(ACTIVEHIGHLIGHT);
      }
      else if (i == pick)
      {
        color(INACTIVEHIGHLIGHT);
      }
      else
      {
        color(NOHIGHLIGHT);
      }

      rectfi(box[i].left, box[i].bottom, box[i].right, box[i].top);
      color(TEXTCOLOR);
      recti(box[i].left, box[i].bottom, box[i].right, box[i].top);
      cmov2i(box[i].x0,box[i].y0);
      charstr(box[i].text0);
      cmov2i(box[i].x1,box[i].y1);
      charstr(box[i].text1);
      cmov2i(box[i].x2,box[i].y2);
      charstr(box[i].text2);
    }
```

```
        color(WHITE);           /* Draw LED gages. */
        rectfi(200,70,400,370);
        color(BLACK);
        recti(200,70,300,150);
        recti(300,70,400,150);
        recti(200,150,300,230);
        recti(300,150,400,230);
        recti(200,230,300,310);
        recti(300,230,400,310);
        recti(200,310,300,370);
        recti(300,310,400,370);
        color(RED);
        cmov2i(205,350);
        charstr("ORDERED");
        cmov2i(205,330);
        charstr("ANGLE");
        cmov2i(305,350);
        charstr("ACTUAL");
        cmov2i(305,330);
        charstr("ANGLE");

    }

    /* Action! */

    switch (pick)
    {
     case 1:    submenu = 1;
             *selected_gait = FWD_WAVE_GAIT;
             break;

     case 2:    submenu = 2;
             break;

     case 3:    submenu = 3;
             *program_status = RESET;
             break;

     case 4:    submenu = 4;
             joystick(trans_rate,rot_rate,ordered_vel_mag,ordered_vel_dir,&buttonflag);
             steering_conv(ordered_rate,ordered_vel_mag,ordered_vel_dir,
                    azimuth,tx,ty,fh);
             *selected_gait = FTL_GAIT;
             break;
    case 5:     submenu = 5;
                status_report(ordered_rate,trans_rate,rot_rate,
                        b_footpos,period,alpha,gamma,theta,
                        slow_flag,roll,elev,azimuth,tx,ty,
                        tz);
                break;

    case 6:     /* exit */
                *program_status = HALT;
                break;

    case 7:     bar(-200.0,200.0,&slidebar,&barvalue,trans_rate->x);
                if (slidebar == IN)
                {
                 ordered_rate->x = barvalue;
                }
                break;

    case 8:     bar(-100.0,100.0,&slidebar,&barvalue,trans_rate->y);
                if (slidebar == IN)
                {
                 ordered_rate->y = barvalue;
                }
                break;

    case 9:     bar(-1.0,1.0,&slidebar,&barvalue,rot_rate->z);
                if (slidebar == IN)
                {
                 ordered_rate->z = barvalue;
                }
                break;

    case 10:    /* Future expansion */
                break;

    case 11:    /* Future expansion */
                break;

    case 12:    /* Future expansion */
                break;

    default:    color(BLACK);
   }

  popviewport();
  popmatrix();

} /* end of driver_command */
```

```
/* driver.c */

/***************************************************************************

bar(minval, maxval,slidebar,barvalue,actualvalue)
float minval, maxval,*barvalue,actualvalue;
int   *slidebar;
{
  register i;
  char     str[20];
  int      x,y;
  static int   barlevel;

  /* Draw the sliding bar. */
  cursoff();
  color(BLACK);
  rectfi(9,69,90,690);
  color(RED);
  recti(10,70,30,670);
  for (i=0;i<5;i++)
  {
    move2i(30,70 + i*150);
    draw2i(40,70 + i*150);
    cmov2i(34,73 + i*150);
    sprintf(str, "%6.1f",minval + i*(maxval-minval)/4.0);
    charstr(str);
  }
  curson();

  /* Check the location of the cursor. If it is inside the
     sliding bar, then calculate the value for its position. */
  x = getvaluator(MOUSEX);
  y = getvaluator(MOUSEY);
  if (10 < x && x < 30 && 70 < y && y < 670)
  {
    barlevel = y;
    *slidebar = IN;
    *barvalue = minval + (maxval - minval)*(y - 70)/600.0;
  }
  else
  {
    *slidebar = OUT;
  }
  /* Draw the bar showing the actual level. */
  color(RED);
  rectf(15.0,70.0,25.0,(370.0+600.0*actualvalue/(maxval-minval)));

  /* Draw the bar showing the ordered value. */
  color(YELLOW);
  recti(11,70,29,barlevel);
} /* end of bar */
```

```
/* driver.c */

/**************************************************************************/
joystick(trans_rate,rot_rate,ordered_vel_mag,ordered_vel_dir,buttonflag)

vector   *trans_rate,   /* translation rates of the center of gravity in body coordinates */
         *rot_rate;     /* body Euler angle rotation rates */

float    *ordered_vel_mag, /* ordered velocity of cockpit (magnitude) */
         *ordered_vel_dir; /* ordered velocity of cockpit (direction) */

int      *buttonflag;   /* indicator for middle mouse button */
{
  int x,y,i;

  float vx,vy,    /*  velocity of cockpit in body coordinates */
        magn,dir; /* magnitude and direction of cockpit velocity vector */


  /* Display the steering box. */
  color(BLUE);
  recti(100,80,400,380);
  /* Display the grid */
  for (i=1;i<15;i++)
  {
    move2(90.0+i*20.0,80.0);
    draw2(90.0+i*20.0,380.0);
  }
  for (i=1;i<15;i++)
  {
    move2(100.0,80.0+i*20.0);
    draw2(400.0,80.0+i*20.0);
  }

  /* Display instructions.  */
  cmov2i(110,65);
  charstr("Hold the middle button down");
  cmov2i(110,50);
  charstr("to control the joystick");

  /* Display the current velocity of the cockpit.  */
  vx = trans_rate->x;
  vy = rot_rate->z * HALFLENGTH + trans_rate->y;
  magn = sqrt(vx*vx + vy*vy);
  dir  = atan2(vy,vx);
  if (vx == 0.0)
  {
    dir = 0.0;
  }
  linewidth(5);
  color(YELLOW);
  move2(250.0,80.0);
  if (vx == 0.0)
  {
    dir = 0.0;
  }
  linewidth(5);
  color(YELLOW);
  move2(250.0,80.0);
  draw2((250.0-400.0*dir),(80.0+magn*3.0));

  /* Check the location of the cursor. */
  x = getvaluator(MOUSEX);
  y = getvaluator(MOUSEY);
  if (*buttonflag == DOWN)
  {
    if (100 < x && x < 400 && 80 < y && y < 380)
    {
       *ordered_vel_mag = (y-80)/3.0;
       *ordered_vel_dir = (250-x)/400.0;
    }
  }
  /* Display the ordered velocity of the cockpit. */
  linewidth(3);
  color(RED);
  move2(250.0,80.0);
  draw2((250.0 - 400.0 * *ordered_vel_dir),(80.0 + *ordered_vel_mag * 3.0));
  linewidth(1);
} /* end of joystick */
```

```
/***********************************************************************
   This is a function for the iris2400 program walk.c.
                       steering.c
                    -----------------
   Relle Lyman                        26 Mar 1987
***********************************************************************/


#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>


steering_conv(ordered_rate,ordered_vel_mag,ordered_vel_dir,azimuth,tx,ty,fh)

/* This function calculates converts ordered head velocity to
   ordered body translation and rotation rates. */

float  *ordered_vel_mag,  /* ordered velocity of the cockpit (magnitude) */
       *ordered_vel_dir,  /* ordered velocity of the cockpit (direction) */
       *azimuth,          /* body azimuth angle (radians) */
       *tx, *ty;          /* current position of the body's center of
                             gravity (in earth coordinates) */

vector  *ordered_rate,    /* ordered forward and lateral translation
                             rates and azimuth angle rate */
        fh[7];            /* selected foothold (in earth coordinates) */

{
  float hx,hy,              /* current head (cockpit) position (earth coord.)*/
        dhx,dhy,            /* desired head position (earth coord.) */
        fhcen_x,fhcen_y,    /* foothold centroid (earth coord.) */
        dcgx,dcgy,          /* desired center of gravity (earth coord.) */
        dazimuth,           /* desired azimuth angle (earth coord.) */
        diffazm;            /* difference between desired and current azimuth */

  vector  desired_rate;  /* desired earth translation rates and azimuth
                            angle rate */
/* steering_conv */

  /* Note: This module uses a level terrain assumption. */

  /* Calculate current head position (earth coordinates). */
  hx = *tx + HALFLENGTH * cos( *azimuth);
  hy = *ty + HALFLENGTH * sin( *azimuth);

  /* Calculate the desired head position (earth coord.). */
  dhx = hx + DELTA_TIME * *ordered_vel_mag * cos( *ordered_vel_dir + *azimuth);
  dhy = hy + DELTA_TIME * *ordered_vel_mag * sin( *ordered_vel_dir + *azimuth);

  /* Calculate the foothold centroid. (Forward gaits only) */
  fhcen_x = (fh[3].x+fh[4].x+fh[5].x+fh[6].x)/4.0;
  fhcen_y = (fh[3].y+fh[4].y+fh[5].y+fh[6].y)/4.0;

  /* Calculate the desired azimuth angle. */
  dazimuth = atan2((dhy-fhcen_y),(dhx-fhcen_x));
  diffazm = dazimuth - *azimuth;

  /* Adjust the difference to a value between pi and -pi. */
  if (diffazm < -3.14159)
  {
    diffazm += 6.2831853;
  }
  if (diffazm > 3.14159)
  {
    diffazm -= 6.2831853;
  }

  /* Calculate the desired center of gravity. */
  dcgx = dhx - HALFLENGTH * cos(dazimuth);
  dcgy = dhy - HALFLENGTH * sin(dazimuth);

  /* Calculate the desired rates (earth coord.). */
  desired_rate.x = (dcgx - *tx)/DELTA_TIME;
  desired_rate.y = (dcgy - *ty)/DELTA_TIME;
  desired_rate.z = diffazm/DELTA_TIME;

  /* Convert to body translation and rotation rates. */
  ordered_rate->x = cos( *azimuth) * desired_rate.x
               + sin( *azimuth) * desired_rate.y;
  ordered_rate->y = cos( *azimuth) * desired_rate.y
               - sin( *azimuth) * desired_rate.x;
  ordered_rate->z = desired_rate.z;

} /* end of steering_conv */
```

```
/****************************************************************
   This is a routine for the iris-2400  program walk.c.
                 status.c
                 --------
   This routine creates a status report to be displayed
   on the viewing screen beneath the ASV.
   Relle Lyman        27 Mar  1987
****************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"

status_report(ordered_rate,trans_rate,rot_rate,b_footpos,period,alpha,gamma,
              theta,slow_flag,roll,elev,azimuth,tx,ty,tz)

int     *slow_flag;  /* flag indicating deceleration is needed */

Angle   theta[7],        /* leg component angles */
      alpha[7],
      gamma[7];

float   *period,          /* period of leg cycle */
      *tx,*ty,*tz,       /* position of body in earth coordinates */
      *roll,*elev,*azimuth; /* body Euler angles */

vector  *rot_rate,       /* body rotation rates */
      *trans_rate, /* body translation rates */
      *ordered_rate, /* ordered lateral and longitudinal and yaw rates */
      b_footpos[7];    /* foot position in body coordinates */
{
  int   i,k;

  char  str_fpx[7][100],str_fpy[7][100],str_fpz[7][100],
      str_orx[100],str_ory[100],str_orz[100],
      str_trx[100],str_try[100],str_trz[100],
      str_rrx[100],str_rry[100],str_rrz[100],
      str_alpha[7][100],str_gamma[7][100],str_theta[7][100],
      str_period[100],str_slow[100],
      str_tx[100],str_ty[100], str_tz[100],
      str_roll[100],str_azimuth[100],str_elev[100];
```

```
/* status.c */

sprintf(str_orx,"%7.2f",ordered_rate->x);
sprintf(str_ory,"%7.2f",ordered_rate->y);
sprintf(str_orz,"%7.2f",ordered_rate->z);
sprintf(str_trx,"%7.2f",trans_rate->x);
sprintf(str_try,"%7.2f",trans_rate->y);
sprintf(str_trz,"%7.2f",trans_rate->z);
sprintf(str_rrx,"%7.2f",rot_rate->x);
sprintf(str_rry,"%7.2f",rot_rate->y);
sprintf(str_rrz,"%7.2f",rot_rate->z);
sprintf(str_period,"%9.5f",*period);
sprintf(str_tx,"%7.2f",*tx);
sprintf(str_ty,"%7.2f",*ty);
sprintf(str_tz,"%7.2f",*tz);
sprintf(str_roll,"%7d",((int) (*roll * 573.0)));
sprintf(str_azimuth,"%7d",((int) (*azimuth * 573.0)));
sprintf(str_elev,"%7d",((int) (*elev * 573.0)));

for (k=1;k<7;k++)
{
  sprintf(str_fpx[k],"%7.2f",b_footpos[k].x);
  sprintf(str_fpy[k],"%7.2f",b_footpos[k].y);
  sprintf(str_fpz[k],"%7.2f",b_footpos[k].z);
  sprintf(str_alpha[k],"%7d",alpha[k]);
  sprintf(str_gamma[k],"%7d",gamma[k]);
  sprintf(str_theta[k],"%7d",theta[k]);
}

pushmatrix();
  viewport(0,400,0,767);
  ortho2(0.0,400.0,0.0,767.0);
  color(BLACK);
  rectfi(10,10,400,370);
  color(YELLOW);
  rectfi(20,20,390,360);
  color(BLACK);
  cmov2i(220,340);
  charstr("X");
  cmov2i(280,340);
  charstr("Y");
  cmov2i(340,340);
  charstr("Z");

  cmov2i(30,325);
  charstr("ordered_rate");
  cmov2i(30,310);
  charstr("trans_rate");
  cmov2i(30,295);
  charstr("rot_rate");
  cmov2i(30,280);
  charstr("position");
  cmov2i(210,265);
  charstr("ROLL");
  cmov2i(260,265);
  charstr("ELEV.");
  cmov2i(310,265);
  charstr("AZIMUTH");

  cmov2i(30,250);
  charstr("current attitude");
  cmov2i(30,235);
  charstr("ordered attitude");

  cmov2i(30,210);
  charstr("period");

  if(*slow_flag == SLOW)   /* moving too fast */
  {
   cmov2i(750,220);
   color(RED);
   charstr("TOO FAST");
   color(BLACK);
  }

  cmov2i(30,185);
  charstr("x ft pos (1-3)");
  cmov2i(110,170);
  charstr("(4-6)");
  cmov2i(30,155);
  charstr("y ft pos (1-3)");
  cmov2i(110,140);
  charstr("(4-6)");
  cmov2i(30,125);
  charstr("z ft pos (1-3)");
  cmov2i(110,110);
  charstr("(4-6)");
  cmov2i(30,95);
  charstr("ALPHA    (1-3)");
  cmov2i(110,80);
  charstr("(4-6)");
  cmov2i(30,65);
  charstr("GAMMA    (1-3)");
  cmov2i(110,50);
  charstr("(4-6)");
  cmov2i(30,35);
  charstr("THETA    (1-3)");
  cmov2i(110,20);
  charstr("(4-6)");
```

```
    cmov2i(180,325);
    charstr(str_orx);
    cmov2i(240,325);
    charstr(str_ory);
    cmov2i(300,325);
    charstr(str_orz);

    cmov2i(180,310);
    charstr(str_trx);
    cmov2i(240,310);
    charstr(str_try);
    cmov2i(300,310);
    charstr(str_trz);

    cmov2i(180,295);
    charstr(str_rrx);
    cmov2i(240,295);
    charstr(str_rry);
    cmov2i(300,295);
    charstr(str_rrz);

    cmov2i(180,280);
    charstr(str_tx);
    cmov2i(240,280);
    charstr(str_ty);
    cmov2i(300,280);
    charstr(str_tz);

    cmov2i(180,210);
    charstr(str_period);

    cmov2i(180,250);
    charstr(str_roll);
    cmov2i(240,250);
    charstr(str_elev);
    cmov2i(300,250);
    charstr(str_azimuth);
    for (i=1;i<4;i++)
    {
      k= 110+i*70;
       cmov2i(k,185);
       charstr(str_fpx[i]);
       cmov2i(k,170);
       charstr(str_fpx[i+3]);
       cmov2i(k,155);
       charstr(str_fpy[i]);
       cmov2i(k,140);
       charstr(str_fpy[i+3]);
       cmov2i(k,125);
       charstr(str_fpz[i]);
       cmov2i(k,110);
       charstr(str_fpz[i+3]);
       cmov2i(k,95);
       charstr(str_alpha[i]);
       cmov2i(k,80);
       charstr(str_alpha[i+3]);
       cmov2i(k,65);
       charstr(str_gamma[i]);
       cmov2i(k,50);
       charstr(str_gamma[i+3]);
       cmov2i(k,35);
      charstr(str_theta[i]);
       cmov2i(k,20);
      charstr(str_theta[i+3]);
    }

  popmatrix();

}
```

```
/**********************************************************************
    This is a function for the iris 2400 program walk.c.
                        support.c
                        -------------
       Relle Lyman                    21 Aug 1986
**********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

support_plane(spe)

/* This module will compute a new estimated support plane based on
   the position of the supporting legs.  As a temporary measure it
   is assumed the support plane is flat and at "sea level" (i.e.
   z = 0 ).  The equation for the plane is Ax+By+Cz+D=0.  */

plane  *spe;     /* estimated support plane in earth coordinates */

{
  spe->a = 0.0;
  spe->b = 0.0;
  spe->c = 1.0;
  spe->d = 0.0;
}
```

```
/************************************************************************

  This is a function for the iris 2400 program walk.c.
                 body_rates.c
                 ------------
        Relle Lyman                   19 Apr 1987
************************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

body_rates(rot_rate,trans_rate,spe,h,invh,ordered_rate,tx,ty,tz,
      roll,elev,azimuth)

 /* This function computes the body rotation and translation rates. */


  vector *rot_rate,      /* rotation rate */
         *trans_rate,     /* translation rate */
         *ordered_rate;  /* ordered x translation, y translation,
                    and z rotation rates  */

  plane  *spe;         /* support plane in earth coord */

  float  h[4][4],      /* homogeneous transformation matrix */
        invh[4][4],   /* inverse transformation matrix */
        *tx,*ty,*tz,   /* position of body in earth coordinates */
        *roll,*elev,*azimuth;  /* body Euler angles */
{
  int    i,j;

  float  eta,        /* body plane attitude wrt earth plane */
        eta_d,      /* desired body plane attitude */
        height,     /* distance form CG to est. support plane */
        height_d,   /* desired height */
        gamma,      /* angle between desired and present body unit normal vectors */
        kx,         /* x component of rotation unit vector in body coord */
        ky,         /* y component of rotation unit vector in body coord */
        ka,         /* control law gain */
        a,b,c,     /* body plane desired unit normal in body coord */
        length,    /* rotation vector normalizing factor */
        celev,selev,telev,     /* sine,cosine,tangent of elev */
        croll,sroll,cazim,sazim,  /* sin and cos of roll and azimuth */
        m;

  plane  spb;       /* support plane in body coordinates */

  vector db_unit_norm, /* desired body plane unit normal in earth coordinates */
         trans_rate_e,  /* Translation rates in earth coordinates */
         rot_rate_e;   /* Rotation in body Euler rates */
```

```
/* body_rates.c */

  /* Calculate the body plane attitude. (level ground assumption!)*/
  eta = 0.0;

  /* Calculate desired body plane attitude (level ground assumption!)*/
  eta_d = eta;

  /* Calculate the desired body clearance. (level ground assumption!)*/
  height_d = H0;

  /* Calculate the support plane coefficients in body coordinates. */
  /*   [spb] T = [spe] T * h                                        */
  spb.a = spe->a * h[0][0] + spe->b * h[1][0] + spe->c * h[2][0] + spe->d * h[3][0];
  spb.b = spe->a * h[0][1] + spe->b * h[1][1] + spe->c * h[2][1] + spe->d * h[3][1];
  spb.c = spe->a * h[0][2] + spe->b * h[1][2] + spe->c * h[2][2] + spe->d * h[3][2];
  spb.d = spe->a * h[0][3] + spe->b * h[1][3] + spe->c * h[2][3] + spe->d * h[3][3];

  /* Height of body CG above support plane */
  height = spb.d;

  /* Calculate desired unit normal for the body plane in earth coord. */
  m = sqrt(spe->a * spe->a + spe->b * spe->b);
  /* check for division by zero */
  if (m>0.0)
  {
    db_unit_norm.x = spe->a * sin(eta_d) / m;
    db_unit_norm.y = spe->b * sin(eta_d) / m;
  }
  else
  {
    db_unit_norm.x = 0.0;
    db_unit_norm.y = 0.0;
  }
  db_unit_norm.z = cos(eta_d);
```

```
/* body_rates.c */

 /* Transform the desired unit normal vector of the body plane
    from earth to body coordinates.  [a,b,c]^T=invhr*b_unit_norm   */
  /* Note: invhr is the inverse of the rotational transformation
    submatrix (first three rows and columns of h). */
a = invh[0][0]*db_unit_norm.x + invh[0][1]*db_unit_norm.y +
   invh[0][2]*db_unit_norm.z;
b = invh[1][0]*db_unit_norm.x + invh[1][1]*db_unit_norm.y +
   invh[1][2]*db_unit_norm.z;
c = invh[2][0]*db_unit_norm.x + invh[2][1]*db_unit_norm.y +
   invh[2][2]*db_unit_norm.z;

  /* Control law yielding an exponential response */
  /* d gamma/dt = -ka * gamma                    */
ka = 1/TIME_CONSTANT_1;
gamma = acos(c);
length = sqrt(a*a + b*b);
if (length < .00001)
{
   kx = 0;
   ky = 0;
}
else
{
   /* components of rotation unit vector in body coordinates */
   kx = -b/length;
   ky =  a/length;
}

 /* Calculate rotation and translation rates */
trans_rate->z = -ka * (height_d - height);
rot_rate->x = -ka * kx * gamma;
rot_rate->y = -ka * ky * gamma;

  /* Rate = dt * acceleration + old rate  */
trans_rate->x = DELTA_TIME * (ordered_rate->x - trans_rate->x)/
            TIME_CONSTANT_2 + trans_rate->x;
trans_rate->y = DELTA_TIME * (ordered_rate->y - trans_rate->y)/
            TIME_CONSTANT_3 + trans_rate->y;
rot_rate->z = DELTA_TIME * (ordered_rate->z - rot_rate->z)/
            TIME_CONSTANT_3 + rot_rate->z;

  /* Conversion to Earth coordinate translation rates. */

croll = cos(*roll);
sroll = sin(*roll);
telev = tan(*elev);
celev = cos(*elev);
selev = sin(*elev);
cazim = cos(*azimuth);
sazim = sin(*azimuth);
```

```
/* body_rates.c */

   trans_rate_e.x = trans_rate->x * croll*cazim +
           trans_rate->y * (cazim*sroll*selev - sazim*celev) +
           trans_rate->z * (sazim*selev + cazim*sroll*celev);
   trans_rate_e.y = trans_rate->x * croll*sazim +
           trans_rate->y * (sazim*sroll*selev + cazim*celev) +
           trans_rate->z * (cazim*selev + sazim*sroll*celev);
   trans_rate_e.z = -trans_rate->x * sroll +
           trans_rate->y * croll*selev +
           trans_rate->z * cazim*celev;

   /* Conversion to body Euler rates */
   rot_rate_e.x = rot_rate->x + rot_rate->y * telev * sroll +
          rot_rate->z * telev * croll;
   rot_rate_e.y = rot_rate->y * croll - rot_rate->z * sroll;
   rot_rate_e.z = rot_rate->y * sroll / celev +
          rot_rate->z * croll / celev;

   /* Integration routine */
   *tx += trans_rate_e.x * DELTA_TIME;
   *ty += trans_rate_e.y * DELTA_TIME;
   *tz += trans_rate_e.z * DELTA_TIME;
   *roll += rot_rate_e.x * DELTA_TIME;
   *elev += rot_rate_e.y * DELTA_TIME;
   *azimuth += rot_rate_e.z * DELTA_TIME;

   /* Update the H matrix */

   croll = cos(*roll);
   sroll = sin(*roll);
   telev = tan(*elev);
   celev = cos(*elev);
   selev = sin(*elev);
   cazim = cos(*azimuth);
   sazim = sin(*azimuth);
```

```
/* body_rates.c */

   h[0][0] = croll*cazim;
   h[0][1] = cazim*sroll*selev - sazim*celev;
   h[0][2] = sazim*selev + cazim*sroll*celev;
   h[0][3] = *tx;
   h[1][0] = sazim*croll;
   h[1][1] = cazim*celev + sazim*sroll*selev;
   h[1][2] = sazim*sroll*celev - cazim*selev;
   h[1][3] = *ty;
   h[2][0] = -sroll;
   h[2][1] = croll*selev;
   h[2][2] = croll*celev;
   h[2][3] = *tz;
   h[3][0] = 0.0;
   h[3][1] = 0.0;
   h[3][2] = 0.0;
   h[3][3] = 1.0;

   /* inverse homogeneous transform matrix */
   for (i=0; i<3; i++)
   {
      for (j=0; j<3; j++)
      {
         invh[i][j] = h[j][i];
      }
      invh[3][i] = 0.0;
      invh[i][3] = -(h[0][i]*h[0][3] + h[1][i]*h[1][3] + h[2][i]*h[2][3]);
   }
   invh[3][3] = 1.0;
} /* end of body_rates */
```

```
/***********************************************************************
    This is a function for the iris 2400 program walk.c.
                        con_work_vol.c
                        --------------
       Relle Lyman                 19 Apr 1987
***********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

con_work_vol(cwv,b_footpos,leg_status,warning)

/* This module will compute a new constrained working volume for
   improved stability on rough terrain. Currently all values are
   set for a perfectly flat support plane. Dimensions are
   expressed in cm. */

work_vol  cwv[7]; /* constrained working volume in body coordinates */

vector    b_footpos[7]; /* foot position in body coordinates */

int       leg_status[7],/* status of leg (supporting, liftoff, transfer,
                             placement) */
         *warning;      /* flag indicating supporting leg is outside of
                           the working volume */
{
  int i;




  *warning = OFF;
  for (i=1;i<7;i++)  /* check each leg */
  {
    if (leg_status[i] == SUPPORTING)
    {
      if ((b_footpos[i].x < cwv[i].x.min)||
          (b_footpos[i].x > cwv[i].x.max)||
          (b_footpos[i].y < cwv[i].y.min)||
          (b_footpos[i].y > cwv[i].y.max)||
          (b_footpos[i].z < cwv[i].z.min)||
          (b_footpos[i].z > cwv[i].z.max))   /* outside limits */
       {
         *warning = ON;
       }
    }
  }
  if (*warning == ON)
  {
    pushmatrix();
    pushviewport();
    viewport(0,130,0,80);
    ortho2(0.0,130.0.0.0.80.0);
    color(RED);
    rectfi(10,10.130.70);
    color(BLACK);
    cmov2i(10,30);
    charstr(" OUTSIDE CWV");
    popviewport();
    popmatrix();
  }
}
```

```
 ************************************************************************
   This is a function for the iris2400 program walk.c.
                    opt_period.c
                  ----------------
       Relle Lyman              29 Apr 1987
 ************************************************************************


#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>


optimal_period(legphase,b_footpos,rot_rate,trans_rate,cwv,leg_status,period)

/*  This function computes the optimal period for walking.  */

vector *rot_rate,      /* body rotation rate */
       *trans_rate,    /* body translation rate */
       b_footpos[7];   /* position of foot in body coordinates */

work_vol  cwv[7];      /* constrained working volume */

float  legphase[7],    /* phase of leg */
       *period;        /* body support period */

int    leg_status[7]; /* status of leg  0 = supporting  */

{

  vector b_footvel;      /* foot velocity */

  float  fx,fy,fz,       /* foot position */
         tmin,           /* minimum temporal kinetic margin */
         tx,ty,tz,       /* temporal kinetic margins */
         d,              /* distance to cwv limit */
         speed,          /* magnitude of body velocity */
         fs_period,      /* forward support period */
         bs_period,      /* backward support period */
        min_fs_period, /* minimum forward support period */
        min_bs_period, /* minimum backward support period */
         fs_phase,       /* forward support phase */
         bs_phase,       /* backward support phase */
        mvx,mvy,mvz;

  int    i,
        minleg;

  static int  gait=FORWARD; /*  Wave gait direction */
```

```
/* optimal_period */

  /* Initialize variables */
 tmin = LONG_TIME;
 min_fs_period = LONG_TIME;
 min_bs_period = LONG_TIME;

  /* For each leg compute the maximum instantaneous support period. */
 for (i=1; i< 7; i++)
 {
    /* Support check */
   if (leg_status[i] == SUPPORTING)
   {
      /* Compute foot velocity. */
     b_footvel.x = -(trans_rate->x)+(rot_rate->z * b_footpos[i].y)
                              -(rot_rate->y * b_footpos[i].z);
     b_footvel.y = -(trans_rate->y)-(rot_rate->z * b_footpos[i].x)
                              +(rot_rate->x * b_footpos[i].z);
     b_footvel.z = -(trans_rate->z)+(rot_rate->y * b_footpos[i].x)
                              -(rot_rate->x * b_footpos[i].y);

      /* Check to see if foot is in cwv. */
     fx = b_footpos[i].x;
     fy = b_footpos[i].y;
     fz = b_footpos[i].z;
     if ((fx< cwv[i].x.min) || (fx> cwv[i].x.max)||
        (fy< cwv[i].y.min) || (fy> cwv[i].y.max)||
        (fz< cwv[i].z.min)|| (fz > cwv[i].z.max))    /* outside cwv */
     {
       tmin = 0.1;
     }
     else
     {
       /* Compute distance to x limit and the temporal
         kinetic margin in the x direction. */
       if (b_footvel.x > 0.0)
       {
         d = cwv[i].x.max - fx;
         tx = d / b_footvel.x;
       }
       else if (b_footvel.x < 0.0)
       {
         d = fx - cwv[i].x.min;
         tx = d / -b_footvel.x;
       }
       else
       {
         tx = LONG_TIME;
       }
```

```
        /* Check for minimum value. */
        if (tx<tmin)
        {
          tmin = tx;
        }

        /*  Compute distance to y limit and the temporal
          kinetic margin in the y direction. */
        if (b_footvel.y > 0.0)
        {
          d =  cwv[i].y.max - fy;
          ty = d / b_footvel.y;
        }
        else if (b_footvel.y < 0.0)
        {
          d = fy -  cwv[i].y.min;
          ty = d / -b_footvel.y;
        }
        else
        {
          ty = LONG_TIME;
        }
        /* Check for minimum value. */
        if (ty<tmin)
        {
          tmin = ty;
        }

        /*  Compute distance to z limit and the temporal
          kinetic margin in the z direction. */
        if (b_footvel.z > 0.0)
        {
          d =  cwv[i].z.max - fz;
          tz = d / b_footvel.z;
        }
        else if (b_footvel.z < 0.0)
        {
          d = fz -  cwv[i].z.min;
          tz = d / -b_footvel.z;
        }
        else
        {
          tz = LONG_TIME;
        }
        /* Check for minimum value. */
        if (tz<tmin)
        {
          tmin = tz;
        }
      }  /* end inside cwv */
```

```
/* optimal_period */

      /* Compute the support phase for forward and backward gaits. */
      fs_phase = legphase[i]/BETA;
      bs_phase = (BETA - legphase[i])/BETA;

      /* Compute support period. */
      fs_period = (tmin-0.1)/(1.0 - fs_phase);
      bs_period = (tmin-0.1)/(1.0 - bs_phase);

      /* Find the minimum support period. */
      if (fs_period < min_fs_period)
      {
        min_fs_period = fs_period;
      }

      if (bs_period < min_bs_period)
      {
        min_bs_period = bs_period;
      }
    } /* end support check */
  } /* end leg loop */

  /* Choose gait. */
  speed = sqrt(trans_rate->x * trans_rate->x +
               trans_rate->y * trans_rate->y);
  if ((speed < OUTER_LIMIT)&&(trans_rate->x > INNER_LIMIT))
  {
    gait = FORWARD;
  }
  else if ((speed<OUTER_LIMIT)&&(trans_rate->x< -INNER_LIMIT))
  {
    gait = BACKWARD;
  }
  else
  {
    /* No gait change. */
  }

  if (gait == FORWARD)
  {
    *period = min_fs_period;
  }
  else
  {
    *period = min_bs_period;
  }
} /* end optimal_period */
```

```
/***********************************************************************
   This is a function for the iris 2400 program walk.c.
                  decelerate.c
                  ------------
          Relle Lyman                  04 May 1987
***********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>


decelerate(rot_rate,trans_rate,period,slow_flag,min_period)

/*  This function computes the foot transfer rate and slows the
   vehicle if the maximum rate is exceeded. */


vector *rot_rate,     /* body rotation rate */
       *trans_rate;   /* body translation rate */

float  *period,       /* optimal period for the leg cycle */
       *min_period;   /* minimum leg period */

int    *slow_flag;    /* flag indicating vehicle has been slowed. */

{
  int     i,j;

  float   transfer_time; /* time from liftoff to placement */



  if(*period < *min_period) /* slow down */
  {
    *slow_flag = SLOW;
    *period = *min_period;

    trans_rate->x *= .95;
    trans_rate->y *= .7;
    trans_rate->z *= .95;
    rot_rate->x   *= .95;
    rot_rate->y   *= .95;
    rot_rate->z   *= .7;
```

```
      /* display warning on screen */
      pushmatrix();
      pushviewport();
      viewport(200,330,0,80);
      ortho2(200.0,330.0,0.0,80.0);
      color(YELLOW);
      rectfi(210,10,330,70);
      color(BLACK);
      cmov2i(210,30);
      charstr(" DECELERATION ");
      popviewport();
      popmatrix();
    }
    else
    {
      *slow_flag = NORMAL;
    }


} /* end of decelerate*/
```

```
/**********************************************************************

   This is a function for the iris2400 program walk.c.
                  leg_phase.c
                  -----------

    Relle Lyman                24 Aug 1986
**********************************************************************/
#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

leg_phase( legphase, rel_legphase, period)

/*  This function computes the phase for each leg.  */

float  legphase[7],        /*  phase of leg  */
       rel_legphase[7], /*  relative phase of leg  */
       *period;            /*  body support period   */

{
  static float   bodyphase = 0.0; /* kinematic phase of body */

  float     modulus_one();   /*  modulus one function */

  int          i;


  /* Calculate new body phase. */
  bodyphase = modulus_one(bodyphase + DELTA_TIME/(*period) );

  /*  Calculate new phase for each leg. (mod1) */
  for (i=1; i<7; i++)
  {
    legphase[i] = modulus_one(bodyphase - rel_legphase[i]);
  }

}  /*  end of leg_phase  */
```

```
/*********************************************************************

   This is a function for the iris2400 program walk.c.
                    ft_traj.c
                 ----------------
   Relle Lyman                          19 Apr 1987
*********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

foot_trajectory(legphase,period,leg_status,footpos,b_footpos,fh,
        oldfh,invh,h,cwv,trans_rate,rot_rate,selected_gait)

/* This function calculates the trajectory for each foot. */

float   legphase[7],    /* Phase of individual leg */
       *period,         /* Optimal period            */
        h[4][4],        /* Homogeneous transformation matrix */
        invh[4][4];     /* Inverse transformation matrix     */

vector  footpos[7],
         b_footpos[7],
        fh[7],          /* Foothold selection (earth coordinates) */
        oldfh[7],        /* Old foothold selection (earth coordinates) */
        *trans_rate,     /* Body translation rate */
        *rot_rate;        /* Body rotation rate */

work_vol  cwv[7];

int     leg_status[7],  /* State of individual leg */
       *selected_gait;  /* Desired tripod gait */

{
  float trans_time,        /* Time required to go from leg liftoff to leg touchdown  */
        end_lift_phase,   /* Point in transfer phase where liftoff ends  */
        begin_place_phase,  /* Point in transfer phase where placement begins */
        trans_phase;      /* Leg transfer phase */

  static int   liftoff_flag[7]=OFF,  /* Indicates first time entering the
                                  subroutine in the current leg cycle. */
          transfer_flag[7]=OFF,
            place_flag[7]=OFF;

  static vector  d_footpos[7];   /* Desired foot position */

  int    i;          /* Leg number */
```

```
/* foot_trajectory */

 /* Calculate the time required to move a leg from liftoff to
    touchdown. (Transfer time)  */
trans_time = (1.0 - BETA) * _ABS(*period);

/* Calculate phase points marking change of transfer mode
    (direction). Note: Modify later to account for transfer
    time. */
end_lift_phase = 0.2;
begin_place_phase = 0.8;

/* For each leg */
for (i=1; i<7; i++)
{
  /* Calculate transfer phase. */
  /* (lift = 0.0 place = 1.0 ) */
  if (*period < 0)
  {
    trans_phase = (1.0 - legphase[i])/(1.0 - BETA);
  }
  else
  {
    trans_phase = ( legphase[i] - BETA)/(1.0 - BETA);
  }

  /* Find the leg transfer state. */
  if (trans_phase < 0.0)
  {
    leg_status[i] = SUPPORTING;
    support_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
             b_footpos,invh,i);
  }
  else if (trans_phase < end_lift_phase)
  {
    leg_status[i] = LIFTOFF;
    lift_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
           b_footpos,invh,&trans_time,&trans_phase,&end_lift_phase,i);
  }
  else if (trans_phase < begin_place_phase)
  {
    leg_status[i] = TRANSFER_FORWARD;
    transfer_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
            b_footpos,h,invh,&trans_time,&trans_phase,
           &begin_place_phase,i,cwv,trans_rate,rot_rate,fh,oldfh,
           period,selected_gait);
  }
/* foot_trajectory */

    else      /* end_place_phase < trans_phase < 1.0 */
    {
      leg_status[i] = PLACEMENT;
      placement_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
                b_footpos,invh,&trans_time,&trans_phase,i);
    }
  }

} /* end of foot_trajectory */
```

```
/* foot_trajectory */

/******************************************************************************/

lift_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,b_footpos,
     invh,trans_time,trans_phase,end_lift_phase,leg_number)

/* This function calculates the trajectory of the foot while it is
   being lifted from the ground.  It is called from foot_trajectory().*/

vector    footpos[7],   /* Present foot position in earth coordinates */
          b_footpos[7]; /* Present foot position in body coordinates  */

float     *trans_time,
          *end_lift_phase,
          *trans_phase,
          invh[4][4];    /* Inverse homogeneous transformation matrix */

int       liftoff_flag[7], /* Indicates the first time entering subroutine
                              in the current leg cycle. */
          transfer_flag[7],
          place_flag[7],
          leg_number;
{
  float   lift_time;

  int     i;

  static vector   d_footpos[7];  /* Desired foot position
                                    in earth coordinates */


  i = leg_number;

  /* Calculate the desired footposition. */
  if ( liftoff_flag[i] != ON)
  {
    d_footpos[i].z =  footpos[i].z + FOOTLIFTHEIGHT;
    liftoff_flag[i] = ON;
    transfer_flag[i] = OFF;
    place_flag[i] = OFF;
  }

  /* Calculate the time required to reach the desired height
     from the present foot position.  */
  lift_time = *trans_time * (*end_lift_phase - *trans_phase);
```

```
 /* foot_trajectory */

   /* Calculate the new foot position. (Earth Coordinates) */
   if (DELTA_TIME < lift_time)
   {
      footpos[i].z += (d_footpos[i].z - footpos[i].z)
                      * DELTA_TIME / lift_time;
   }
   else       /* Last increment of time */
   {
      footpos[i].z = d_footpos[i].z;
   }

   /* Transform to body coodinates.  */
   /*   [b_footpos[i]]T = invh * [footpos[i]]T  */
   transform_point(b_footpos,invh,footpos,i);

}    /* end of lift_trajectory  */
```

```
/* foot_trajectory */

/*****************************************************************************/

transfer_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
        b_footpos,h,invh, trans_time,trans_phase,begin_place_phase,
        leg_number,cwv,trans_rate,rot_rate,fh,oldfh,period,selected_gait)

/*  This function calculates the trajectory of the foot during the
   phase in which the foot is transfered forward.  The function
   is called from foot_trajectory().*/

vector   footpos[7],   /*  Present foot position in earth coordinates */
        b_footpos[7], /*  Present foot position in body coordinates  */
       fh[7],        /*  Selected footholds (earth coordinates) */
       oldfh[7],      /*  Old selected footholds (earth coordinates) */
        *trans_rate,  /*  Body translation rate */
        *rot_rate;    /*  Body rotation rate */

work_vol  cwv[7];      /* Constrained working volume in body coordinates */

float    *trans_time,
         *begin_place_phase,
         *trans_phase,
        *period,        /* Optimum period of gait */
         h[4][4],       /* Homogeneous transformation matrix */
         invh[4][4];     /* Inverse transformation matrix */

int      liftoff_flag[7], /* Indicates the first time entering subroutine
                         in current leg cycle */
       transfer_flag[7],
        place_flag[7],
       *selected_gait,   /* Desired tripod gait */
        leg_number;
{
  float   trans_fwd_time, /* Time remaining in the transfer forward phase */
         vx,vy,          /* Velocity of cockpit in body coordinates */
         rel_hd,         /* Relative heading of cockpit velocity */
         proj_dist,      /* Projected distance forward for new footholds */
          min_time;       /* Minimum time to reach any cwv limit */

  int     i;

  vector   cwv_velocity,   /* Instantaneous velocity of the center of
                          the cwv (earth coodinates) */
          time_to_limit, /* Time to reach the cwv limits */
         bfh[7],         /* Selected foothold in body coordinates */
          bd_footpos[7]; /* Desired foot position in body coordinates */

  static vector d_footpos[7]; /* Desired foot position in earth coordinates */
```

```
/* foot_trajectory */

i = leg_number;

if (*selected_gait == FTL_GAIT)
{
  if (transfer_flag[i] != ON)
  {
    transfer_flag[i] = ON;
    liftoff_flag[i] = OFF;
    place_flag[i] = OFF;

    /* Save foothold position. */
    oldfh[i].x = fh[i].x;
    oldfh[i].y = fh[i].y;
    oldfh[i].z = fh[i].z;

    proj_dist = LENGTH * 0.21666;

   switch (i)
   {
     case 1:    /* find new left foothold */
            vx = trans_rate->x;
            vy = trans_rate->y + rot_rate->z * HALFLENGTH;
            rel_hd = atan2(vy,vx);
            bfh[1].y = 82.0;
            bfh[1].x = HALFLENGTH+proj_dist*cos(rel_hd)-(82.0-
                    proj_dist*sin(rel_hd))*tan(rel_hd);
            bfh[1].z = -160.0;
            /* Transform to earth coordinates. */
            /* [fh[i]]T = h * [bfh[i]]T  */
            transform_point(fh,h,bfh,1);
            break;
     case 2:    /* find new right foothold */
            vx = trans_rate->x;
            vy = trans_rate->y + rot_rate->z * HALFLENGTH;
            rel_hd = atan2(vy,vx);
            bfh[2].y = -82.0;
            bfh[2].x = HALFLENGTH+proj_dist*cos(rel_hd)-(-82.0-
                    proj_dist*sin(rel_hd))*tan(rel_hd);
            bfh[2].z = -160.0;
            /* Transform to earth coordinates. */
            /* [fh[i]]T = h * [bfh[i]]T  */
            transform_point(fh,h,bfh,2);
            break;
     default:   /* back leg uses old front leg foothold */
            fh[i].x = oldfh[i-2].x;
            fh[i].y = oldfh[i-2].y;
            fh[i].z = oldfh[i-2].z;
   }
/* foot_trajectory */

        /* determine the desired foot position */
        d_footpos[i].x = fh[i].x;
        d_footpos[i].y = fh[i].y;
        d_footpos[i].z = fh[i].z;
      }
    }
  else      /* FWD_WAVE_GAIT  Calculate a new desired foot position
                              at each time increment. */
  {
    /* Calculate the desired touchdown point. */
    /* Future change note:  Change from cwv center to midstance. */

    /*  Calculate foot velocity at center of cwv (body coordinates) */
    cwv_velocity.x = trans_rate->x - rot_rate->z *  cwv[i].y.center
                              + rot_rate->y *  cwv[i].z.center;
    cwv_velocity.y = trans_rate->y + rot_rate->z *  cwv[i].x.center
                              - rot_rate->x *  cwv[i].z.center;
    cwv_velocity.z = trans_rate->z - rot_rate->y *  cwv[i].x.center
                              + rot_rate->x *  cwv[i].y.center;

    /* Calculate the time to reach the limits of the cwv. */
    if (cwv_velocity.x < 0.0)
    {
      time_to_limit.x = ( cwv[i].x.min -  cwv[i].x.center)/cwv_velocity.x;
    }
    else if (cwv_velocity.x > 0.0)
    {
      time_to_limit.x = ( cwv[i].x.max -  cwv[i].x.center)/cwv_velocity.x;
    }
    else
    {
      time_to_limit.x = LONG_TIME;
    }

    if (cwv_velocity.y < 0.0)
    {
      time_to_limit.y = ( cwv[i].y.min -  cwv[i].y.center)/cwv_velocity.y;
    }
    else if (cwv_velocity.y > 0.0)
    {
      time_to_limit.y = ( cwv[i].y.max -  cwv[i].y.center)/cwv_velocity.y;
    }
    else
    {
      time_to_limit.y = LONG_TIME;
    }
```

```
/* foot_trajectory */

    if (cwv_velocity.z < 0.0)
    {
      time_to_limit.z = ( cwv[i].z.min -  cwv[i].z.center)/cwv_velocity.z;
    }
    else if (cwv_velocity.z > 0.0)
    {
      time_to_limit.z = ( cwv[i].z.max -  cwv[i].z.center)/cwv_velocity.z;
    }
    else
    {
      time_to_limit.z = LONG_TIME;
    }

    /* Determine the minimum time to reach the cwv limit. */
    min_time = time_to_limit.x;
    if (time_to_limit.y < min_time)
    {
      min_time = time_to_limit.y;
    }
    if (time_to_limit.z < min_time)
    {
      min_time = time_to_limit.z;
    }

    /* Calculate the desired touchdown point in body coordinates. */
    /* Note: This point changes if the body is in motion. */
    bd_footpos[i].x =  cwv[i].x.center + cwv_velocity.x * min_time * .9;
    bd_footpos[i].y =  cwv[i].y.center + cwv_velocity.y * min_time * .9;
    bd_footpos[i].z =  cwv[i].z.center + cwv_velocity.z * min_time * .9;

    /* Transform to Earth coodinates. */
    /* [d_footpos[i]]T = h * [bd_footpos[i]]T */
    transform_point(d_footpos,h,bd_footpos,i);
  }


  /* Calculate the time remaining in the transfer forward phase. */
  trans_fwd_time = *trans_time * (*begin_place_phase - *trans_phase);

  /* Calculate the new foot position. (Earth Coordinates) */
  if (DELTA_TIME < trans_fwd_time)
  {
    footpos[i].x += (d_footpos[i].x -  footpos[i].x)
                  * DELTA_TIME / trans_fwd_time;
    footpos[i].y += (d_footpos[i].y -  footpos[i].y)
                  * DELTA_TIME / trans_fwd_time;
    footpos[i].z =  footpos[i].z; /* Level ground assumption! */
  }
```

```
/* foot_trajectory */

  else       /* Last increment of time */
  {
     footpos[i].x = d_footpos[i].x;
     footpos[i].y = d_footpos[i].y;
     footpos[i].z =  footpos[i].z;  /* Level ground assumption! */
  }

  /* Transform to body coodinates. */
  /*  [b_footpos[i]]T = invh * [footpos[i]]T  */
  transform_point(b_footpos,invh,footpos,i);

}  /* end of transfer_trajectory  */
```

```
 /* foot_trajectory */

/*****************************************************************************/

placement_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
        b_footpos,invh,trans_time,trans_phase, leg_number)

/* This function calculates the trajectory of the foot while it is
   being lowered from the ground. It is called from foot_trajectory().*/

vector   footpos[7],    /* Present foot position in earth coordinates */
        b_footpos[7]; /* Present foot position in body coordinates  */

float    *trans_time,
        *trans_phase,
         invh[4][4];     /* Inverse homogeneous transformation matrix */

int       liftoff_flag[7], /* Indicates the first time entering subroutine
                            in current leg cycle */
        transfer_flag[7],
         place_flag[7],
        leg_number;
{
  float   place_time;

  int     i;

  static vector  d_footpos[7]; /* Desired foot position in earth coordinates */

  i = leg_number;

  /* Calculate the desired foot position. */
  if ( place_flag[i] != ON)
  {
    d_footpos[i].z =  footpos[i].z - FOOTLIFTHEIGHT;
     liftoff_flag[i] = OFF;
     transfer_flag[i] = OFF;
     place_flag[i] = ON;
  }

  /* Calculate the time required to reach the desired height
     from the present foot position. */
  place_time = *trans_time * (1.0 - *trans_phase);

  /* Calculate the new foot position. (Earth Coordinates) */
  if (DELTA_TIME < place_time)
  {
     footpos[i].z += (d_footpos[i].z -  footpos[i].z)
                  * DELTA_TIME / place_time;
  }
/*  foot_trajectory */

   else        /* Last increment of time */
   {
      footpos[i].z = d_footpos[i].z;
   }

   /* Transform to body coodinates.  */
   /*  [b_footpos[i]]T = invh * [footpos[i]]T  */
   transform_point(b_footpos,invh,footpos,i);

}  /* end of placement_trajectory  */
```

```
/* foot_trajectory */

/*********************************************************************************/

support_trajectory(liftoff_flag,place_flag,transfer_flag,footpos,
        b_footpos,invh,leg_number)

/* This function calculates the trajectory of the foot during the
   foot support phase.  It is called from foot_trajectory().*/

vector   footpos[7],   /* Present foot position in earth coordinates */
        b_footpos[7]; /* Present foot position in body coordinates  */

float    invh[4][4];    /* Inverse homogeneous transformation matrix */

int      liftoff_flag[7], /* Indicates the first time entering subroutine
                         in current leg cycle */
        transfer_flag[7],
         place_flag[7],
        leg_number;
{

  int     i;


   /* In this phase the foot is kept stationary on the ground.  It
     is assumed that the foot will not slip or move accidently. */

  i = leg_number;

   /* Transform foot position to body coodinates.  */
   /*    b_footpos[i]T = invh * footpos[i]T  */
  transform_point(b_footpos,invh,footpos,i);

   /* Turn off flags. */
   liftoff_flag[i] = OFF;
   transfer_flag[i] = OFF;
   place_flag[i] = OFF;

}  /* end of support_trajectory  */
```

```
/*********************************************************************
   This function is for the program walk.c on the iris-2400.
                    conwalk.c
                    ---------
                         Based in part on J.H.Kessler's
   R. L. Lyman              program "conwalker.c"
   24 Apr 1987
*********************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"

/*    This function calls up the walker from constructwalker (with legs
  already properly positioned) and then rotates and translates it as
  commanded.                                              */

/*  Note:  Due to the limited number of bit planes available
  four separate walkers are constructed, one for each viewing
  quadrant.  The walker for each quadrant is drawn from furthest
  component to nearest.  This provides a quasi- Z buffer effect
  while in double buffer mode.                        */


makewalker(machineobject,d1,d2,theta,knee,gamma,alpha,transrot_tag,
        tr_end_tag,walker,leg,thighobj,actuatorobj,shinobj,
        legmovetag,thighmovetag,actmovetag,shinmovetag ,tx,ty,tz,
       roll,elev,azimuth,hx,hy,hz,l4)

Tag    transrot_tag[4],tr_end_tag[4],legmovetag[][4],
     thighmovetag[][2][4],actmovetag[][2][4],shinmovetag[][2][4];

Object machineobject[4],walker[4],thighobj[][2][4],actuatorobj[][2][4],
     shinobj[][2][4],leg[][4];

int d1[],d2[],knee[][2] ;

Angle theta[],alpha[],gamma[];

float tx,ty,tz,roll,azimuth,elev,
    hx[7],hy[7],hz[7],l4[7];


{
  int n;
/* conwalk.c */

  constructwalker(walker,d1,d2,knee,alpha,gamma,theta,leg,thighobj,
           actuatorobj,shinobj,legmovetag,thighmovetag,actmovetag,
           shinmovetag,hx,hy,hz,l4) ;

  for (n=0; n<4; n++)      /* Rotate and translate the walkers in each
                    quadrant.  */
  {
    machineobject[n]=genobj();
    makeobj(machineobject[n]);
      pushmatrix() ;

       /* Note: Each walker is built on the origin. Rotations are done
          before translating to the proper location. */
      transrot_tag[n]=gentag();
      maketag(transrot_tag[n]);

        translate(tx,ty,tz);
        rotate((int) (elev * 573),'Y');
        rotate((int) (roll * 573),'X');
        rotate((int) (azimuth * 573),'Z');

       tr_end_tag[n]=gentag();
       maketag(tr_end_tag[n]);

       callobj(walker[n]);

       popmatrix() ;
    closeobj();
  }  /* end quadrant loop  */
}  /* end of makewalker   */
```

```
/* conwalk.c */

/*****************************************************************************/
makeground(groundobject)

/*  This function creates a checkerboard groundplane below the ASV object.*/

 Object  *groundobject;
{
  Object  squareobject;
  Tag     transqrtag;

  static int
    ground1[4][3]={{1000,-500,0},{1000,500,0},{-1000,500,0},{-1000,-500,0}},
    ground2[4][3]={{2000,-1000,0},{2000,1000,0},{-2000,1000,0},{-2000,-1000,0}},
    square[4][3]={{0,-100,0},{0,0,0},{-100,0,0},{-100,-100,0}};

  int     i,j;

  float   tx,ty;

  squareobject=genobj();
    makeobj(squareobject);
    color(WHITE);
    polfi(4,square);
  closeobj();

  *groundobject=genobj();
    makeobj(*groundobject);
    color(RED);        /*  fill outer background squares  */
    polfi(4,ground2);
    color(GREEN);      /*  fill inner background squares  */
    polfi(4,ground1);

    for (i=0; i<40; i++)
    {
     for (j=0; j<20; j++)
       {
        if ((i+j)%2 < 1)
        {
          tx=(i-20)*(-100.0);
          ty=(j-10)*(-100.0);
          pushmatrix();
           translate(tx,ty,0.0);
          callobj(squareobject);  /* place the white squares */
          popmatrix();
        } /* end if */
      } /* end for j */
    }  /* end for i  */
  closeobj();
}  /*  end makeground */
```

```
/* conwalk.c */

/*************************************************************************************/

constructwalker(walker,d1,d2,knee,alpha,gamma,theta,leg,thighobj,
          actuatorobj,shinobj,legmovetag,thighmovetag,actmovetag,
         shinmovetag,hx,hy,hz,l4)

/*  This is where the walker is made. Here each part is assembled
   and then the parts are put together. This assembled walker is
   then rotated and translated in makewalker which is called by
   the main program.                                   */

Tag legmovetag[][4],thighmovetag[][2][4],actmovetag[][2][4],
    shinmovetag[][2][4];

Object walker[4],leg[][4],thighobj[][2][4],actuatorobj[][2][4],shinobj[][2][4];

int d1[],d2[],knee[][2];

Angle alpha[],gamma[],theta[];

float hx[7],hy[7],hz[7],        /* leg pivot position */
      l4[7];

{
  Object body,head,eye,boxobj[7];

  static float legx[7]={0.0,155.0,155.0,0.0,0.0,-155.0,-155.0},
               legy[7]={0.0,82.0,-82.0,82.0,-82.0,82.0,-82.0},
               legz[7]={0.0,0.0,0.0,0.0,0.0,0.0,0.0};

  Coord x,y,z ;

  int i,j,k,n,legnum ;

  static int
     /* Coordinates for building the body of the asv */
     blackbody[4][3]={{206,50,22},{206,-50,22},{206,-30,-101},{206,30,-101}},
     lbodyarry[4][3]={{-200,30,-101},{-200,50,22},{206,50,22},{206,30,-101}},
     rbodyarry[4][3]={{-200,-30,-101},{206,-30,-101},{206,-50,22},{-200,-50,22}},
     tbodyarry[4][3]={{-200,50,22},{-200,-50,22},{206,-50,22},{206,50,22}},
     bbodyarry[4][3]={{206,-30,-101},{-200,-30,-101},{-200,30,-101},{206,30,-101}},
     backbodyarry[4][3]={{-200,30,-101},{-200,-30,-101},{-200,-50,22},{-200,50,22}},
/* conwalk.c */

    /* Coordinates for building the hydraulics housing structure */
    front_rt_top[4][3]={{27,-25,16},{38,-25,-13},{38,-13,-13},{27,-13,16}},
    front_rt_bttm[4][3]={{38,-25,-13},{38,-25,-46},{38,-13,-46},{38,-13,-13}},
    rt_interior[5][3]={{20,-25,38},{38,-25,-13},{38,-25,-46},{-38,-25,-46},{-38,-25,38}},
    rt_side[5][3]={{-38,-25,38},{-38,-25,-46},{38,-25,-46},{38,-25,-13},{20,-25,38}},
    lt_interior[5][3]={{-38,25,38},{-38,25,-46},{38,25,-46},{38,25,-13},{20,25,38}},
    lt_side[5][3]={{20,25,38},{38,25,-13},{38,25,-46},{-38,25,-46},{-38,25,38}},
    top_box[4][3]={{20,-25,38},{20,25,38},{-38,25,38},{-38,-25,38,}},
    back_box[4][3]={{-38,25,-46},{-38,-25,-46},{-38,-25,38},{-38,25,38}},
    front_top[4][3]={{20,25,38},{20,-25,38},{27,-25,13},{27,25,13}},
    front_lt_top[4][3]={{27,13,16},{38,13,-13},{38,25,-13},{27,25,16}},
    front_lt_bttm[4][3]={{38,13,-13},{38,13,-46},{38,25,-46},{38,25,-13}},
    bttm_lt[4][3]={{38,25,-46},{38,13,-46},{-38,13,-46},{-38,25,-46}},
    bttm_rt[4][3]={{38,-25,-46},{38,-13,-46},{-38,-13,-46},{-38,-25,-46}},

    highbox_top[4][3]={{-8,-25,88},{8,-25,88},{8,25,88},{-8,25,88}},
    highbox_front[4][3]={{8,25,88},{8,-25,88},{10,-25,38},{10,25,38}},
    highbox_back[4][3]={{-8,-25,88},{-8,25,88},{-10,25,38},{-10,-25,38}},
    highbox_rt[4][3]={{8,-25,88},{-8,-25,88},{-10,-25,38},{10,-25,38}},
    highbox_lt[4][3]={{-8,25,88},{8,25,88},{10,25,38},{-10,25,38}},

    rt_spar_front[4][3]={{79,-13,-20},{79,-25,-20},{79,-25,-30},{79,-13,-30}},
    rt_spar_top[4][3]={{79,-13,-20},{38,-13,-19},{38,-25,-19},{79,-25,-20}},
    rt_spar_bttm[4][3]={{38,-13,-32},{79,-13,-30},{79,-25,-30},{38,-25,-32}},
    rt_spar_rt[4][3]={{38,-25,-32},{79,-25,-30},{79,-25,-20},{38,-25,-19}},
    rt_spar_lt[4][3]={{79,-13,-30},{38,-13,-32},{38,-13,-19},{79,-13,-20}},

    lt_spar_front[4][3]={{79,25,-20},{79,13,-20},{79,13,-30},{79,25,-30}},
    lt_spar_top[4][3]={{79,25,-20},{38,25,-19},{38,13,-19},{79,13,-20}},
    lt_spar_bttm[4][3]={{38,25,-32},{79,25,-30},{79,13,-30},{38,13,-32}},
    lt_spar_rt[4][3]={{38,13,-32},{79,13,-30},{79,13,-20},{38,13,-19}},
    lt_spar_lt[4][3]={{79,25,-30},{38,25,-32},{38,25,-19},{79,25,-20}},

/* cab construction arrays */
cab_bottom[4][3]={{305,-30,-101},{206,-30,-101},{206,30,-101},{305,30,-101}},
cab_top[4][3]={{250,33,74},{206,33,74},{206,-33,74},{250,-33,74}},

cab_fwd_support[4][3]={{305,30,-101},{305,41,-16},{305,-41,-16},{305,-30,-101}},
cab_fwd_lower[4][3]={{305,41,-16},{318,48,8},{318,-48,8},{305,-41,-16}},
cab_fwd_upper[4][3]={{318,48,8},{302,33,68},{302,-33,68},{318,-48,8}},
cab_fwd_ovhd[4][3]={{275,33,68},{250,33,74},{250,-33,74},{275,-33,68}},

cab_rt_support[4][3]={{305,-30,-101},{305,-41,-16},{206,-41,-16},{206,-30,-101}},
cab_rt_lower[4][3]={{305,-41,-16},{318,-48,8},{206,-48,8},{206,-41,-16}},
cab_rt_upper[4][3]={{318,-48,8},{302,-33,68},{206,-33,68},{206,-48,8}},
cab_rt_ovhd[4][3]={{275,-33,68},{250,-33,74},{206,-33,74},{206,-33,68}},
cab_lt_support[4][3]={{206,30,-101},{206,41,-16},{305,41,-16},{305,30,-101}},
cab_lt_lower[4][3]={{206,41,-16},{206,48,8},{318,48,8},{305,41,-16}},
cab_lt_upper[4][3]={{206,48,8},{206,33,68},{302,33,68},{318,48,8}},
cab_lt_ovhd[4][3]={{206,33,68},{206,33,74},{250,33,74},{275,33,68}},

cab_aft_support[4][3]={{206,-30,-101},{206,-41,-16},{206,41,-16},{206,30,-101}},
cab_aft_lower[4][3]={{206,-41,-16},{206,-48,8},{206,48,8},{206,41,-16}},
cab_aft_upper[4][3]={{206,-48,8},{206,-33,68},{206,33,68},{206,48,8}},
cab_aft_ovhd[4][3]={{206,-33,68},{206,-33,74},{206,33,74},{206,33,68}},

scanner_fwd_lower[4][3]={{302,33,68},{322,33,95},{322,-33,95},{302,-33,68}},
scanner_fwd_upper[4][3]={{322,33,95},{322,33,101},{322,-33,101},{322,-33,95}},
scanner_rt[5][3]={{302,-33,68},{322,-33,95},{322,-33,101},{275,-33,101},{275,-33,68}},
scanner_lt[5][3]={{302,33,68},{275,33,68},{275,33,101},{322,33,101},{322,33,95}},
scanner_aft[4][3]={{275,33,101},{275,33,68},{275,-33,68},{275,-33,101}},
scanner_top[4][3]={{322,33,101},{275,33,101},{275,-33,101},{322,-33,101}};

 /* The making of the leg is quite complicated. Each leg consists of an
 upper link (thigh), lower link (actuator). and a shank (shin). These
 segments are first defined in a standardized orientation, and are then
 rotated and translated into the proper position. This is done by using
 2 objects for each segment. The first object is the correctly rotated
 segment, and the second object is the correctly translated first
 object. Thus the segment is then in the proper position. To hold the
 screen coordinate system fixed the matrix is pushed before each translation
 or rotation and then popped after the object is constructed or called. */

for (n=0; n<4; n++)   /* Make a set of legs for each viewing quadrant
                 Each quadrant must have unique tags.        */
{
  for(legnum=1 ;legnum<7;legnum++)
  {
    /* Each segment is constructed and positioned  */
    buildthigh(n,legnum,d1,alpha,thighobj,thighmovetag) ;
    buildactuator(n,legnum,d2,alpha,actuatorobj,actmovetag) ;
    buildshin(n,legnum,knee,gamma,shinobj,shinmovetag) ;

    leg[legnum][n]=genobj();
    makeobj(leg[legnum][n]);
    pushmatrix();
      /* translate(legx[legnum],legy[legnum],legz[legnum]) ; */
      translate(hx[legnum],hy[legnum],hz[legnum]) ;

      legmovetag[legnum][n]=gentag();  /* The leg is assembled from  */
      maketag(legmovetag[legnum][n]);  /* its parts and the entire leg is */
                        /* then rotated to the proper angle. */
      rotate(theta[legnum],'X');
     translate(0.0,l4[legnum],0.0); /* extend leg outward *
    if (((n > 1)&&(legnum < 5))||
      ((n < 2)&&(legnum > 4)))        /* Build the left side first. */
    {
     if (legnum > 4)                /* Reverse the back legs. */
     {
       pushmatrix();
       rotate(1800,'Z');
     }
      color(BLACK);
      polfi(5,lt_interior);

      color(GREEN);
     polfi(5,lt_side);
      polfi(4,front_lt_top);
      polfi(4,front_lt_bttm);
     polfi(4,bttm_lt);

     polfi(4,lt_spar_front);
     polfi(4,lt_spar_bttm);
     polfi(4,lt_spar_lt);
     polfi(4,lt_spar_rt);

     color(BLUE);
     polfi(4,lt_spar_top);

     color(BLACK);
     polyi(4,lt_spar_rt);

      color(CYAN) ;
      callobj(thighobj[legnum-1][n]);
      callobj(actuatorobj[legnum-1][n]);
      callobj(shinobj[legnum-1][n]);

      color(GREEN) ;
     polfi(4,rt_spar_front);
     polfi(4,rt_spar_bttm);
     polfi(4,rt_spar_lt);
     polfi(4,rt_spar_rt);

     color(BLUE);
     polfi(4,rt_spar_top);

      color(GREEN);
      polfi(4,front_rt_bttm);
     polfi(4,front_rt_top);
     polfi(4,front_top);
     polfi(4,bttm_rt);
     polfi(4,back_box);
     polfi(4,top_box);
     polfi(5,rt_side);
```

```
        color(BLACK);
        polyi(4,top_box);
        polyi(5,rt_side);
        polyi(4,rt_spar_rt);

         color(GREEN);
        polfi(4,highbox_front);
        polfi(4,highbox_lt);
        polfi(4,highbox_back);
        polfi(4,highbox_rt);
        polfi(4,highbox_top);

         color(BLACK);
        polyi(4,highbox_top);
        polyi(4,highbox_rt);
        polyi(4,highbox_back);

        if (legnum > 4)      /* For reversing the back legs. */
        {
           popmatrix() ;
        }

      }
      else        /* Build the right side first. */
      {
        if (legnum > 4)                /* Reverse the back legs. */
        {
          pushmatrix();
          rotate(1800,'Z');
        }

         color(BLACK);
         polfi(5,rt_interior);

         color(GREEN);
        polfi(5,rt_side);
         polfi(4,front_rt_top);
         polfi(4,front_rt_bttm);
        polfi(4,bttm_rt);

        polfi(4,rt_spar_bttm);
        polfi(4,rt_spar_rt);
        polfi(4,rt_spar_lt);
        polfi(4,rt_spar_front);

        color(BLUE);
        polfi(4,rt_spar_top);
        color(BLACK);
        polyi(4,rt_spar_lt);

         color(CYAN) ;
         callobj(thighobj[legnum][1][n]);
         callobj(actuatorobj[legnum][1][n]);
         callobj(shinobj[legnum][1][n]);

         color(GREEN) ;
         polfi(4,back_box);
        polfi(4,bttm_lt);
        polfi(4,front_top);
         polfi(4,front_lt_bttm);
        polfi(4,front_lt_top);

        polfi(4,lt_spar_bttm);
        polfi(4,lt_spar_rt);
        polfi(4,lt_spar_lt);
        polfi(4,lt_spar_front);

        color(BLUE);
        polfi(4,lt_spar_top);

         color(GREEN);
        polfi(4,top_box);
        polfi(5,lt_side);

        color(BLACK);
        polyi(4,top_box);
        polyi(5,lt_side);
        polyi(4,lt_spar_lt);

         color(GREEN);
        polfi(4,highbox_back);
        polfi(4,highbox_rt);
        polfi(4,highbox_front);
        polfi(4,highbox_lt);
        polfi(4,highbox_top);

         color(BLACK);
        polyi(4,highbox_top);
        polyi(4,highbox_lt);
        polyi(4,highbox_front);
        if (legnum > 4)       /* For reversing the back legs. */
        {
           popmatrix() ;
        }
       }
       popmatrix();
     closeobj() ;
   }  /* end of leg loop */
}  /* end of quadrant loop */

   body=genobj() ;                    /* The body is constructed */
   makeobj(body);

     color(LTYELLOW) ;
     polfi(4,lbodyarry) ;
     polfi(4,backbodyarry) ;
     polfi(4,bbodyarry) ;
     polfi(4,rbodyarry) ;

     color(YELLOW);
     polfi(4,tbodyarry);

     color(BLACK) ;
     polfi(4,blackbody) ;
   closeobj() ;

   head=genobj() ;                    /* construct the head */
   makeobj(head) ;
     color(YELLOW);
     polfi(4,cab_top);
     polfi(4,cab_fwd_ovhd);
     polfi(4,cab_rt_ovhd);
     polfi(4,cab_lt_ovhd);
     polfi(4,cab_aft_ovhd);

     color(BLACK);
     polfi(4,cab_bottom);
     polfi(4,cab_fwd_support);
     polfi(4,cab_rt_support);
     polfi(4,cab_lt_support);
     polfi(4,cab_aft_support);

     color(WHITE1);
     polfi(4,cab_fwd_lower);
     polfi(4,cab_rt_lower);
     polfi(4,cab_lt_lower);
     polfi(4,cab_aft_lower);
```

```
/* conwalk.c */

    color(WHITE) ;
    polfi(4,cab_fwd_upper);
    polfi(4,cab_rt_upper);
    polfi(4,cab_lt_upper);
    polfi(4,cab_aft_upper);

    color(BLACK) ;
    polfi(4,cab_top);
    polyi(4,cab_fwd_lower);
    polyi(4,cab_fwd_upper);
    polyi(4,cab_fwd_ovhd);
    polyi(4,cab_rt_lower);
    polyi(4,cab_rt_upper);
    polyi(4,cab_rt_ovhd);
    polyi(4,cab_lt_lower);
    polyi(4,cab_lt_upper);
    polyi(4,cab_lt_ovhd);
  closeobj() ;

  eye=genobj() ;                    /* contruct the radar (eye)*/
  makeobj(eye) ;
    color(RED);
    polfi(4,scanner_fwd_upper);
    polfi(4,scanner_fwd_lower);
    polfi(5,scanner_rt);
    polfi(5,scanner_lt);
    polfi(4,scanner_aft);

    color(BLACK) ;
    polfi(4,scanner_top);

    color(BLUE) ;
  closeobj() ;

  walker[0]=genobj();           /*  assemble all the parts for quad I */
  makeobj(walker[0]);           /* back and right first */
    callobj(leg[6][0 );
    callobj(leg[4][0 );
    callobj(leg[2][0 );
    callobj(body);
    callobj(head);
    callobj(eye);
    callobj(leg[5][0 );
    callobj(leg[3][0 );
    callobj(leg[1][0 );
  closeobj() ;
/* conwalk.c */

  walker[1]=genobj();          /* assemble all the parts for quad II */
  makeobj(walker[1]);          /* front and right first */
    callobj(leg[2][1]);
    callobj(leg[4][1]);
    callobj(leg[6][1]);
    callobj(head);
    callobj(eye);
    callobj(body);
    callobj(leg[1][1]);
    callobj(leg[3][1]);
    callobj(leg[5][1]);

  walker[2]=genobj();          /* assemble all the parts for quad III*/
  makeobj(walker[2]);          /* front and left first */
    callobj(leg[1][2]);
    callobj(leg[3][2]);
    callobj(leg[5][2]);
    callobj(head);
    callobj(eye);
    callobj(body);
    callobj(leg[2][2]);
    callobj(leg[4][2]);
    callobj(leg[6][2]);

  walker[3]=genobj();          /* assemble all the parts for quad IV */
  makeobj(walker[3]);          /* back and left first */
    callobj(leg[5][3]);
    callobj(leg[3][3]);
    callobj(leg[1][3]);
    callobj(body);
    callobj(head);
    callobj(eye);
    callobj(leg[6][3]);
    callobj(leg[4][3]);
    callobj(leg[2][3]);
}
```

```
/* conwalk.c */

/**************************************************************************/
buildthigh(n,legnum,d1,alpha,thighobj ,thighmovetag)

/*  this function constructs the thigh (upper link) and rotates, then
   translates it into the proper position                        */

Tag thighmovetag[][2][4];
int n,legnum,d1[] ;
Angle alpha[] ;
Object thighobj[][2][4];
{
 static int
     thighltside[4][3]={{0,10,7},{102,10,7},{102,10,-7},{0,10,-7}},
     thighrtside[4][3]={{0,-10,-7},{102,-10,-7},{102,-10,7},{0,-10,7}},
     thighfront[4][3]={{0,-10,7},{102,-10,7},{102,10,7},{0,10,7}} ,
     thighbttm[4][3]={{0,10,-7},{102,10,-7},{102,-10,-7},{0,-10,-7}};


   thighobj[legnum][0][n]=genobj();
   makeobj(thighobj[legnum][0][n]);

     pushmatrix() ;

     thighmovetag[legnum][0][n]=gentag();    /*  rotate thigh  */
     maketag(thighmovetag[legnum][0][n]);
     rotate(alpha[legnum],'Y') ;

     if(legnum>4)     /* Build the left side first. */
     {
       color(CYAN);
       polfi(4,thighbttm);
       polfi(4,thighltside);
       polfi(4,thighrtside);

       color(RED);
       polfi(4,thighfront);

       color(BLACK);
       polyi(4,thighrtside);
     }
```

```
    else            /* Build the right side first. */
    {
      color(CYAN);
      polfi(4,thighbttm);
      polfi(4,thighrtside);
      polfi(4,thighltside);

      color(RED);
      polfi(4,thighfront);

      color(BLACK);
      polyi(4,thighltside);
    }

    popmatrix() ;

  closeobj() ;

  thighobj[legnum][1][n] =genobj() ;
  makeobj(thighobj[legnum][1][n]) ;

    pushmatrix() ;

    thighmovetag[legnum][1][n] =gentag();
    maketag(thighmovetag[legnum][1][n]);          /* translate thigh  */
    translate(0.0,0.0,(float)(-d1[legnum])) ;

    callobj(thighobj[legnum][0][n]);
    popmatrix() ;

  closeobj() ;

}
```

```
/* conwalk.c */

/*************************************************************************/
buildactuator(n,legnum,d2,alpha,actuatorobj,actmovetag)

/* construct the actuator (lower link) */

Tag actmovetag[][2][4];

int n,legnum,d2[];

Angle alpha[];

Object actuatorobj[][2][4];
{

  static int actltside[4][3]={{0,10,5},{83,10,5},{83,10,-5},{0,10,-5}},
          actfront[4][3]={{0,-10,5},{83,-10,5},{83,10,5},{0,10,5}} ,
          actrtside[4][3]={{0,-10,-5},{83,-10,-5},{83,-10,5},{0,-10,5}},
          actbttm[4][3]={{0,10,-5},{83,10,-5},{83,-10,-5},{0,-10,-5}} ;


  actuatorobj[legnum][0][n]=genobj();
  makeobj(actuatorobj[legnum][0][n]);

    pushmatrix();

    actmovetag[legnum][0][n]=gentag();
    maketag(actmovetag[legnum][0][n]);   /* rotate actuator */
    rotate(alpha[legnum],'Y') ;

    if(legnum>4)   /* Build the left side first. */
    {
      color(CYAN);
      polfi(4,actbttm);
      polfi(4,actltside);
      polfi(4,actrtside);

      color(RED);
      polfi(4,actfront);

      color(BLACK);
      polyi(4,actrtside);
    }
```

```
      else          /* Build the right side first. */
      {
        color(CYAN);
        polfi(4,actbttm);
        polfi(4,actrtside);
        polfi(4,actltside);

        color(RED);
        polfi(4,actfront);

        color(BLACK);
        polyi(4,actltside);
      }
      popmatrix();
    closeobj();

    actuatorobj[legnum][1][n]=genobj();
    makeobj(actuatorobj[legnum][1][n]);

      pushmatrix();
        actmovetag[legnum][1][n]=gentag();
        maketag(actmovetag[legnum][1][n]);          /* translate actuator */
        translate((float)(d2[legnum]),0.0,(float)(-L3));
        callobj(actuatorobj[legnum][0][n]);
      popmatrix();
    closeobj();
} /* end of buildactuator */
```

```
/* conwalk.c */

/*************************************************************************/
buildshin(n,legnum,knee,gamma,shinobj,shinmovetag)

/*  construct the shank (shin ) */

Tag shinmovetag[][2][4];
int n,legnum,knee[][2];
Angle gamma[];
Object shinobj[][2][4];
{
  static int
shinltside[6][3]={{6.5.3},{10,5,-59},{-7,5,-50},{-6,5,3}.{-3,5.6},{3.5,6}},
shankltside[4][3]={{10,5,-59},{-23,5,-102},{-36,5,-100},{-7,5,-50}} ,
shinfront[4][3]={{6,5,3},{6,-5,3},{10.-5,-59},{10,5,-59}},
shankfront[4][3]={{10,-5,-59}.{-23,-5,-102},{-23.5,-102}.{7,5,-59}},
ankleltside[6][3]={{-23.5,-102},{3.5,-153}.{2.5,-157},{-3,5,-158},{-6.5.-158},{-36,5,-100}},
shinrtside[6][3]={{3,-5,6},{-3.-5,6}.{-6,-5.3},{-7,-5.-50}.{10,-5.-60}.{6,-5,3}},
shankrtside[4][3]={{-7,-5,-50},{-36,-5,-100},{-23,-5,-102},{10,-5,-59}} ,
anklertside[6][3]={{-36,-5,-100},{-6,-5,-158},{-3.-5.-158},{2,-5.-157}.{3,-5,-153}.{-23,-5,-102}}.
anklefront[4][3]={{-23,5.-102},{-23,-5,-102},{3,-5,-153},{3,5,-153}},
shinback[4][3]={{-7,-5,-50},{-6,-5,3},{-6,5,3}.{-7,5,-50}},
shankback[4][3]={{-36,-5,-100},{-7,-5,-50},{-7.5,-50},{-36,5.-100}},
ankleback[4][3]={{-6,-5,-158},{-36,-5,-100},{-36,5,-100},{-6,5,-158}},
bottom_fwd[4][3]={{3,5,-153},{2, 5,-157},{2,-5,-157},{3,-5,-153}},
bottom_mid[4][3]={{2,5,-157},{-3,5,-158},{-3,-5,-158},{2,-5,-157}},
bottom_aft[4][3]={{-3,5,-158},{-6,5,-158},{-6,-5,-158},{-3,-5,-158}};

   shinobj[legnum][0][n]=genobj();
    makeobj(shinobj[legnum][0][n]);
    pushmatrix() ;

    shinmovetag[legnum][0][n]=gentag();
    maketag(shinmovetag[legnum][0][n]);     /* rotate shank */
      rotate(gamma[legnum],'Y');

      if(legnum>4)   /* Build the left side first. */
      {
         color(BLACK);
         polfi(4,bottom_fwd);
         polfi(4,bottom_mid);
         polfi(4,bottom_aft);

         color(CYAN);
         polfi(4,ankleback);
         polfi(6,ankleltside);
         polfi(6,anklertside);

         color(RED);
         polfi(4,anklefront) ;
```

```
        color(CYAN);
        polfi(4,shankback) ;
        polfi(4,shankltside);
        polfi(4,shankrtside);

        color(RED);
        polfi(4,shankfront) ;

        color(CYAN);
        polfi(4,shinback);
        polfi(6,shinltside);
        polfi(6,shinrtside);

        color(RED);
        polfi(4,shinfront) ;

        color(BLACK);
        polyi(6,anklertside);
        polyi(4,shankrtside);
        polyi(6,shinrtside);
    }
    else           /* Build the right side first. */
    {
        color(BLACK);
        polfi(4,bottom_fwd) ;
        polfi(4,bottom_mid) ;
        polfi(4,bottom_aft) ;

        color(CYAN);
        polfi(4,ankleback);
        polfi(6,anklertside);
        polfi(6,ankleltside);

        color(RED);
        polfi(4,anklefront);

        color(CYAN);
        polfi(4,shankback) ;
        polfi(4,shankrtside);
        polfi(4,shankltside);

        color(RED);
        polfi(4,shankfront);

        color(CYAN);
        polfi(4,shinback);
        polfi(6,shinrtside);
        polfi(6,shinltside);
        color(RED);
        polfi(4,shinfront);

        color(BLACK);
        polyi(6,ankleltside);
        polyi(4,shankltside);
        polyi(6,shinltside);
      }
      color(BLACK);

      pushmatrix();
      rotate(-900,'X');
      translate(0.0,0.0,5.0);
      circf(0.0,0.0,7.0) ;
      circf(0.0,32.0,5.0) ;
      popmatrix();

      pushmatrix();
      rotate(900,'X');
      translate(0.0,0.0,5.0);
      circf(0.0,0.0,7.0) ;
      circf(0.0,-32.0,5.0) ;
      popmatrix();

      popmatrix();
     closeobj();

    shinobj[legnum][1][n]=genobj();
    makeobj(shinobj[legnum][1][n]);

      pushmatrix();

      shinmovetag[legnum][1][n]=gentag();
      maketag(shinmovetag[legnum][1][n]);           /* translate shank  */
        translate((float)knee[legnum][0],0.0,(float)knee[legnum][1]) ;

        callobj(shinobj[legnum][0][n]);

      popmatrix();
    closeobj();


} /* end of buildshin */
```

```
/*************************************************************************
   This is a function for the iris2400 program walk.c.
                        toolbox.c
                        ---------
     Relle Lyman                    25 Aug 1986
**************************************************************************/

#include "gl.h"
#include "device.h"
#include "walk.h"
#include <stdio.h>
#include <math.h>

/****************************************************************************/
transform_point(p2,m,p1,i)

/*  This function changes the coordinate system for a point vector
    using a homogeneous transformation submatrix.      p2 = m * p1   */

int    i;        /* Leg number */

float  m[4][4]; /* Homogeneous transformation submatrix */

vector p1[7],    /* Vector represented in first coordinate system */
       p2[7];    /* Vector represented in transformed coordinate system  */
{
  p2[i].x = m[0][0]*p1[i].x + m[0][1]*p1[i].y + m[0][2]*p1[i].z + m[0][3];
  p2[i].y = m[1][0]*p1[i].x + m[1][1]*p1[i].y + m[1][2]*p1[i].z + m[1][3];
  p2[i].z = m[2][0]*p1[i].x + m[2][1]*p1[i].y + m[2][2]*p1[i].z + m[2][3];

} /* end of transform_point */

/****************************************************************************/
float modulus_one(temp)

/*  This function performs the modulus one operation on numbers of type float. */

float temp;
{
  while (temp >= 1.0)
  {
    temp -= 1.0;
  }
  while (temp < 0.0)
  {
    temp += 1.0;
  }
  return temp;
}  /* end of modulus_one */
```

```
/* Makefile */

#  This is Makefile.  It is used in the utility make to speed
#    compilation of walk.c.  To use it, just type "make".     */

   CFLAGS = -Zf -Zg -g

   SRCS =      walk.c
          conwalk.c
          support.c
          toolbox.c
          steering.c
          body_rates.c
          ft_traj.c
          opt_period.c
          leg_phase.c
          con_work_vol.c
          driver.c
          status.c
          decelerate.c
          init.c

   OBJS =      walk.o
          conwalk.o
          support.o
          toolbox.o
          steering.o
          body_rates.o
          ft_traj.o
          opt_period.o
          leg_phase.o
          con_work_vol.o
          driver.o
          status.o
          decelerate.o
          init.o

  walk :  (OBJS)
          cc -o walk (OBJS)  -Zg -Zf

  (OBJS) :     walk.h
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library (Code 0142) Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman (Code 62) Department of Electronics and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Curricular Officer (Code 32) Department of Electronics and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
5.	Prof. Robert B. McGhee (Code 52Mz) Department of Computer Science Naval Postgraduate School Monterey, California 93943	20
6.	Prof. Roberto Cristi (Code 62Cx) Department of Electronics and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
7.	Lt. Relle L. Lyman, Jr. Naval Sea Systems Command SEA 90G Washington, D.C. 20362-5101	2
8.	Prof. Kenneth J. Waldron Department of Mechanical Engineering 206 W. 18th Avenue Ohio State University Columbus, Ohio 43210	1

9. Russel L. Werneth (Code 69Wh) 1
Department of Mechanical Engineering
Naval Postgraduate School
Monterey, California 93943

10. William J. Butler 1
Naval Sea Systems Command SEA 90G
Washington, D.C. 20362-5101

11. Cdr. Bart Everett 1
Naval Ocean System Center (Code 442)
San Diego, California 92152

12. Research Administration (Code 012) 1
Naval Postgraduate School
Monterey, California 93943

13. Center for Naval Analyses 1
2000 N. Beauregard St.
Alexandria, VA 22311

Thesis
L9535 Lyman
c.1 A computer simulation
study of tripod follow-
the-leader gait coordina-
tion for a hexapod walk-
ing machine.

20 AUG 92 57415

Thesis
L9535 Lyman
c.1 A computer simulation
study of tripod follow-
the-leader gait coordina-
tion for a hexapod walk-
ing machine.