# Modeling and Analysis of a Spatial Compliant Hexapod

Uluç Saranli,
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2110, USA

Martin Buehler,
Centre for Intelligent Machines
McGill University, 480 University St, MC423
Montreal, Quebec, Canada, H3A 2A7

## 1 Introduction

In recent years, there has been increasing interest in legged locomotion, primarily due to the realization that wheeled locomotion has inherent limitations on unstructured environments. The use of legs, adopted by many animals and insects who need to locomote over a wide range of different terrain conditions, seems to be the most natural and effective solution to the problem. In this context, we believe that the analysis and understanding of the principles behind legged locomotion, followed by experimental verification of the results is essential.

### 1.1 Inspiration from Studies of Animals

Biologists have been studying animals and insects for a long time in an attempt to understand the mechanisms, both mechanical and neural, with which animals perform their behavioral repertoire. In the realm of locomotion, we now know that there are certain principles exploited by animals of very different sizes and morphologies. The alternating tripod gait among many different hexapedal insects, for example, is an instance.

There are more striking analogies that have been identified, however. Biologists have found that the center of mass behavior of running animals can be described by a spring mass system, whose parameters depend on size and morphology but whose structure always stays the same. Recently, Robert J. Full observed the same principle in insects as well, including cockroaches and millipedes.

Our inspiration for analyzing and building a hexapod platform mainly comes from these observation, as well as the structural properties of a six legged platform. The alternating tripod gait allows static balance, statically stable walk, dynamical running and possibly leaping. This is a range of behaviors which no single robot has been able to demonstrate before, but almost all six legged animals can perform. Moreover, insects in particular are able to maintain these modes of behavior over extremely irregular "fractal" terrain, without any apparent change in the character of the behavior. Even a simple measure as the large number of different species which adopted six legged locomotion supports our view that this is a natural direction to follow in understanding locomotion.

### 1.2 Contributions of this Report

This report attempts to give a detailed account of the design and modeling efforts for a spatial compliant hexapod robotic platform. Section 2 introduces the concept of *hybrid dynamical systems*, which will be the basis for our hexapod model. Section 3 then describes the simplified hexapod model that we will use in our analysis and design studies. The hybrid dynamical system simulation tool that we have built for

this project, *SimSect*, is described in Section 4, followed by a summary of the simulation results that we have obtained in Section 5.

## 2 Hybrid Dynamical Systems

A large number of dynamical systems that we are interested in analyzing, including the hexapod model that we present in this report, cannot be represented with a single dynamical flow. They are hybrid dynamical systems, which are mixtures of discrete and continuously varying events. This section describes a formal definition of hybrid dynamical systems, and is mostly quoted from the formalism described in [1], with some minor modifications.

We assume that the problem domain is decomposed into the form

$$V = \bigcup_{\alpha \in I} V_\alpha$$

where $I$ is a finite *index set* and $V_\alpha$ is an open, connected subset of $\mathbf{R}^n$. Each element in this union is called a *chart*. Each chart has associated with it a vector field, $f_\alpha : V_\alpha \times \mathbf{R} \to \mathbf{R}^n$. Notice that the charts are not required to be disjoint. Moreover, on the intersection set $V_\alpha \bigcap V_\beta$, continuity, or even agreement of the vector fields are not required for $\alpha, \beta \in I$. For each $\alpha \in I$, the chart $V_\alpha$ must enclose a *patch*, an open subset $U_\alpha$ satisfying $\overline{U}_\alpha \subset V_\alpha$. The boundary of $U_\alpha$ is assumed piecewise smooth and is referred to as the *patch boundary*. Together, the collection of charts and patches is called an *atlas*.

For each $\alpha \in I$ there is a finite set of *boundary functions*, $h_{\alpha,i} : V_\alpha \to R$, $i \in J_\alpha^{bf}$, and real numbers called *target values*, $C_{\alpha,i}$, for $i \in J_\alpha^{\rm bf}$ that satisfy the condition: For $x \in V_\alpha$ where $\alpha \in I$ , we require

$$x \in U_\alpha \text{ if and only if } h_{\alpha,i}(x) - C_{\alpha,i} > 0 \text{ for all } i \in J_\alpha^{\rm bf} \quad .$$

Thus, a patch is to be considered the domain on which a collection of smooth functions are positive. The boundary of a patch is assumed to lie within the set:

$$\bigcup_{i \in J_\alpha^{\rm bf}} h_{\alpha,i}^{-1}(\{C_{\alpha,i}\}) \qquad \text{for } \alpha \in I \, .$$

Conceptually, the evolution of the system is viewed as a sequence of trajectory segments where the endpoint of one segment is connected to the initial point of the next by a transformation. It follows that time may be divided into contiguous periods, called *epochs*, separated by instances where *transition functions* are applied at times referred to as *events*. The transition functions are maps which send a point on the boundary of one patch to a point in another (not necessarily different) patch in the atlas.

Within this framework, an orbit in the flow of a hybrid dynamical system which begins at a time $t_0$ and terminates at $t_f$ may be completely described. A *trajectory*, hence, is a curve $\gamma : [t_0, t_f] \to V \times I$ together with an increasing sequence of real numbers $t_0 < t_1 < \cdots < t_m = t_f$ that satisfies three properties:

- Each time interval $(t_i, t_{i+1})$ corresponds to an epoch and there exists a designated $\alpha$ so that $\gamma(t)$ lies entirely in $\overline{U}_\alpha \times \{\alpha\}$ for all $t \in (t_i, t_{i+1})$.
- For $t \in [t_i, t_{i+1})$ and the unique $\alpha$ specified above, $t \to \pi_1(\gamma(t))$ is an integral curve of the vector field $f_\alpha$.
- $\lim_{t \to t_{i+1}^-} \pi_1(\gamma(t)) = y$ exists, $y \in S_\alpha$ and $T_\alpha(y) = \lim_{t \to t_{i+1}^+} \gamma(t)$.

# 3 The Compliant Hexapod Model

## 3.1 The System Structure

Figure 1 shows the basic structure of the hexapod model. The system consists of a rigid body with six degrees of freedom, whose position and orientation are described by $r_b$ and $R_b$, respectively. Two coordinate frames, $\mathcal{B}$ and $\mathcal{W}$ are defined, the former attached to the hexapod body and the latter is an inertial frame where the dynamics are formulated.

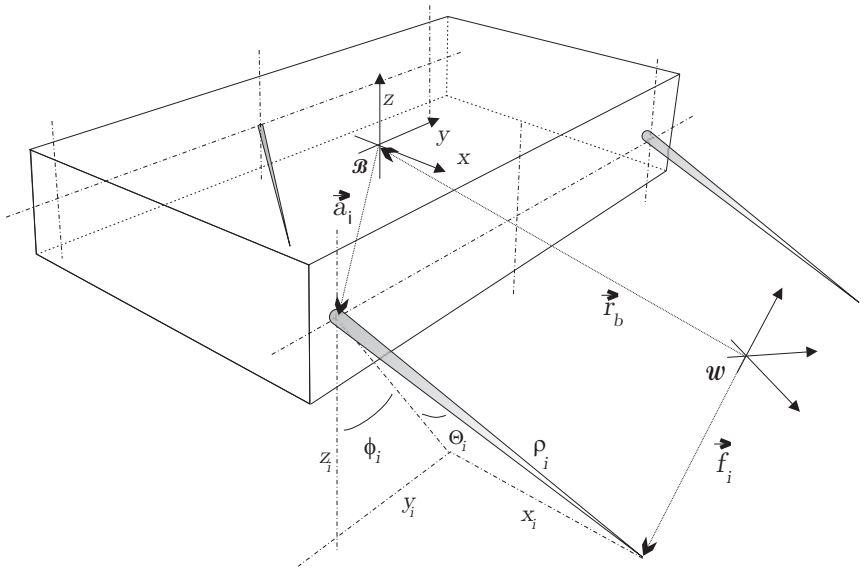


Figure 1: The compliant hexapod model.

The legs are attached to the rigid body, at fixed attachment points $a_i$ in the body frame. Each leg has complete spherical freedom and is assumed to be massless except under certain circumstances where very small toe masses are also incorporated. Note that $(r_b, R_b)$, $v_i$ and $f_i$ are related through a simple coordinate transformation (see Section 3.9).

Associated with each leg, there is a radial and a torsional spring on the $\theta$ direction, as well as torque control on the $\phi$ degree of freedom. These springs and the hip actuation result in forces and torques being applied to the rigid body. In Section 3.2, we derive these forces and torques for a single generalized leg, leading to the formulation of the system dynamics in Section 3.4.

## 3.2 Analysis of a Single Leg

There are several methods that one can use in formulating the dynamics of a mechanical system. The most commonly exercised method is the Euler-Lagrange formulation. However, a simple formulation of the system described in the previous section is not possible in that framework, as a result of the actuation being in a different coordinate system. The choice of the generalized coordinates as the leg states results in a redundant parameterization of the system, which has only six degrees of freedom imposed by the rigid body.

The solution we adopt is to use the rigid body dynamics under force and torque actuation. Section 3.4 details the equations of motion for this formulation. In this section, we formulate the contribution of each leg to the total force and torque to completely specify the rigid body dynamics.

The contribution from each individual leg is independent of the others. Consequently, it will suffice to analyze a generic leg parametrized by its attachment and touchdown points (see Figure 2). The force and torque balance on the massless leg result in the following equalities.

| Coordinate Frames | |
|---|---|
| $\mathcal{W}$ | inertial world frame |
| $\mathcal{B}$ | body frame |
| **States** | |
| $r_b$ | body position |
| $R_b$ | body orientation |
| $\dot{r}_b$ | translational velocity of body |
| $w_b$ | angular velocity of body |
| **Leg states and parameters** | |
| $a_i$ | leg attachment point in $\mathcal{B}$ |
| $f_i$ | toe position in $\mathcal{W}$ |
| $v_i := [\theta_i, \phi_i, \rho_i]^T$ | current leg state in spherical body coordinates |
| $\overline{v}_i := [v_{x_i}, v_{y_i}, v_{z_i}]^T$ | current leg state in Cartesian body coordinates |
| $leg_i$ | stance flag for leg $i$ |
| $\rho_{td}$ | leg length at touchdown |
| $\rho_{lo}$ | leg length at liftoff |
| **Forces and Torques** | |
| $F_{r_i}$ | radial leg spring force |
| $\tau_{\theta_i}$ | leg bending torque in $\theta_i$ direction |
| $\tau_{\phi_i}$ | leg hip torque in $\phi_i$ direction |
| **System Parameters** | |
| $M_0$ | body inertia matrix in body coordinates |
| $M$ | body inertia matrix in world coordinates |
| $m_b$ | body mass |
| **Controller Parameters** | |
| $t_c$ | Period of rotation for a single leg |
| $\phi_s$ | Slice of leg sweep for the slow phase |
| **Actuator Model** | |
| $K_w$ | Motor speed constant |
| $K_\tau$ | Motor torque constant |
| $i_{a_i}$ | Motor armature current |
| $v_{a_i}$ | Motor armature voltage |
| $k_g$ | Motor gear ratio |
| $w_{s_i}$ | Motor shaft speed |
| $\tau_{s_i}$ | Motor shaft torque |

Table 1: Notation used throughout the report

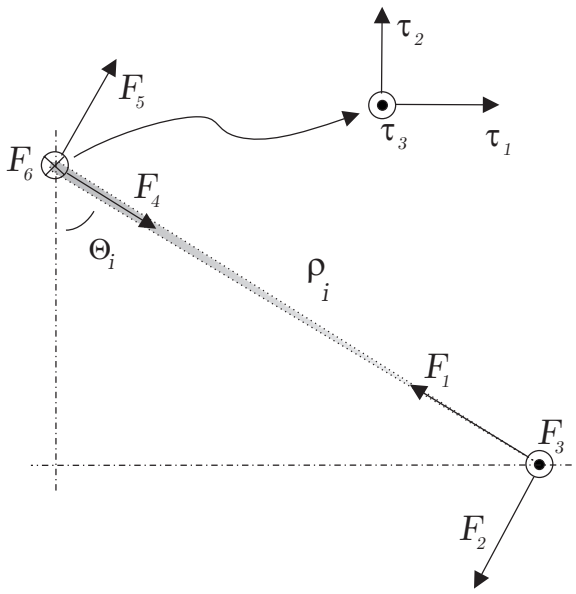


Figure 2: Analysis of a single leg in the plane defined by the leg and the z-axis of $\mathcal{B}$.

$$
\begin{aligned}
F_1 = F_4 &= F_{r_i} \\
F_2 = F_5 &= \frac{\tau_{\theta_i}}{\rho_i} \\
F_3 = F_4 &= \frac{\tau_{\phi_i}}{\rho_i \cos\theta_i} \\
\tau_1 &= \tau_{\phi_i} \\
\tau_2 &= \tau_{\phi_i} \tan\theta_i \\
\tau_3 &= \tau_{\theta_i}
\end{aligned}
$$

The rigid hexapod body experiences the opposite of the force and torque on the leg at the attachment point. Projecting these vectors back to $\mathcal{B}$, we can write

$$
\begin{aligned}
F_i &= \begin{bmatrix} -\sin\theta_i & -\cos\theta_i & 0 \\ -\cos\theta_i \sin\phi_i & \sin\theta_i \sin\phi_i & -\cos\phi_i \\ \cos\theta_i \cos\phi_i & -\sin\theta_i \cos\phi_i & -\sin\phi_i \end{bmatrix} \begin{bmatrix} F_{r_i} \\ \tau_{\theta_i}/\rho_i \\ \tau_{\phi_i}/(\rho_i \cos\theta_i) \end{bmatrix} \\
\tau_i &= \begin{bmatrix} -\tau_{\phi_i} \\ \tau_{\phi_i} \tan\theta_i \sin\phi_i + \tau_{\theta_i} \cos\phi_i \\ -\tau_{\phi_i} \tan\theta_i \cos\phi_i + \tau_{\theta_i} \sin\phi_i \end{bmatrix} + a_i \times F_i
\end{aligned}
$$

which are the force and torque contributions of a single leg to the system dynamics. Note that the leg itself, specifically the small mass at the toe, experiences the opposite of this force.

### 3.3 Total Force and Torque on the Body

The cumulative effect of all the legs on the body is simply the sum of the individual contributions, together with the gravitational force. Expressed in $\mathcal{W}$, the force and torque vectors are given by

$$F_T = \begin{bmatrix} 0 \\ 0 \\ -m_b g \end{bmatrix} + R_b \sum_{i=1}^{6} leg_i F_i \quad (1)$$

$$\tau_T = R_b \sum_{i=1}^{6} leg_i \tau_i \quad (2)$$

where we define

$$leg_i := \begin{cases} 0 & leg\ i\ is\ in\ flight \\ 1 & leg\ i\ is\ in\ stance \end{cases}$$

### 3.4 Rigid Body Dynamics

The dynamics of a rigid body under external force and torque actuation is governed by the following equations [2].

$$\begin{aligned} \ddot{r_b} &= \frac{F_T}{m_b} \\ M\dot{w}_b &= -J(w_b)Mw_b + \tau_T \\ \dot{R}_b &= J(w_b)R_b \end{aligned}$$

where we have

$$J(\begin{bmatrix} w_x & w_y & w_z \end{bmatrix}^T) := \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

$$M := R_b M_0 R_b^{-1}$$

### 3.5 Putting it All Together : Hybrid Hexapod

This section describes the remaining components of the compliant hexapod model, which are essential to complete the definition of the hybrid system structure.

- Charts

  Our formulation of the compliant hexapod model has $2^6 = 64$ charts corresponding to all possible combinations of discrete leg states, each in flight or in stance. Each of those charts have different dynamics, parametrized by the leg state flags $leg_i$. Note that the dependence of Equations (1) and (2) on the leg flags is sufficient to incorporate the discrete leg states into the system dynamics.

- Boundary Functions

  These functions are used to identify transitions between different charts. The hexapod model has several boundary functions for each chart, corresponding to touchdown and liftoff conditions for each leg. These conditions are mostly extensions to their counterparts in the spring-loaded inverted pendulum model(SLIP) [4].

  In every chart, there are two boundary functions associated with each stance leg, corresponding to two different liftoff conditions. First is the leg length reaching the radial spring rest extension, at which point, it is assumed to be restricted from extending further and hence lifts off. The corresponding boundary function is $h_{\alpha,l1} := -\rho_i$ with target value $C_{\alpha,l1} = -\rho_{0_i}$.

The second liftoff condition involves the vertical component of the force that the small toe mass experiences (see Section 3.2). For a leg in stance, this force must remain negative. The leg lifts off when it changes sign. Consequently, the boundary function $h_{\alpha,l2} := -F_{T_i,x}$ with target value 0, defines the second liftoff transition.

If a leg is in flight, however, there is only one touchdown boundary function associated with it. For all types of terrains, it is given by the z coordinate of its toe, $f_{i_z}$. The associated target value depends on the terrain and is the terrain height under the toe.

There is, however, an exception to the touchdown condition. There are cases when the condition that the normal component of the force exerted on the toe must be negative, might not hold at touchdown. This is an artifact of the plastic collision of the toe with the ground, combined with the very small toe mass that governs the flight behavior of the legs. We address this problem, by skipping the touchdown transition when the normal force to the toe is positive upon touching the ground. Note that the positive normal force on the toe makes sure that the foot does not go underground, but continues upwards.

- Transition Functions

  Whenever a touchdown or liftoff event is detected, an appropriate transition function is applied to the current state of the system. These transition functions always modify parts of the state space related to the particular leg which initiated the event.

  Even though there are a very large number of possible transitions between charts in the hexapod model, they can be represented easily by considering transitions of each leg separately. In that case, there are only two possible transition types (although there might be several of them occurring simultaneously), the liftoff and touchdown transitions.

  The liftoff transition does not involve any coordinate changes or modifications to the system state. Consequently, the liftoff transition function is the identity map.

  At touchdown, on the other hand, two things happen. First, the toe mass loses all of its energy in a plastic collision with the ground. Second, the z coordinate of the toe is set to be slightly above the ground (1e-5) , mainly for numerical stability of the oncoming transitions for the same leg. The resulting transition function has many effects on both the resulting trajectory for the hexapod body as well as the leg behavior. A more detailed account of the leg dynamics is presented in the next section.

### 3.6 Leg Dynamics

The state space for the hexapod model consists of the position and the orientation of the body , as well as the position of each toe in the world frame. In the preceding sections, we described the dynamical flow concerning the dynamics of the rigid body. This section presents the leg dynamics, which are of considerable importance in the system behavior due to the hybrid nature of the system, even though the body dynamics are not affected by the leg motions in flight.

The two main leg phases incorporate different dynamical flows for the toe positions of each leg. The simplest case is the leg in flight, where the small toe mass experiences the opposite of the forces computed in Section 3.2. Each toe mass then becomes a second order dynamical system with the computed external force input, in the frame $\mathcal{W}$.

$$\ddot{f}_i \quad = \quad -R_b F_i$$

For the legs in stance, we incorporate a simple coulomb friction model. In this phase, we project the toe force vector onto the tangent plane to the terrain at the touchdown point. The resulting force is then used to drive a first order dynamical system for the toe motion, also incorporating coulomb friction. This simple scheme approximates "sliding" of the toe, also constraining its motion to lie in the terrain surface until liftoff.

This model yields the following leg dynamics

$$\dot{f}_i = \begin{cases} 0 & \| F_t \| \leq K \| F_n \| \\ \frac{F_t(\|F_t\| - K\|F_n\|)}{\|F_t\|} & \| F_t \| > K \| F_n \| \end{cases}$$

where $F_t$ is the projection of $R_b F_i$ on the tangent plane to the terrain surface at the toe position and $F_n$ is the remaining normal component.

### 3.7 Actuator Model

The compliant hexapod model incorporates a very simple actuator model of the hip torque controls. Actuation at each hip is through a brushed DC motor with no electrical dynamics. Consequently, this model component does not have any impact on the mechanical operation of the hexapod. The principal reasons for its implementation are to model the electrical properties of the actuation, including the motor voltage and current, the power consumption of the overall system, as well as the torque limitations of hip actuation.

The simplest component of the model is the torque limits imposed by the torque-speed curves of DC motors. Figure 3 is an example for an off-the-shelf DC motor. These torque limits are imposed on the PD control torques as necessary.

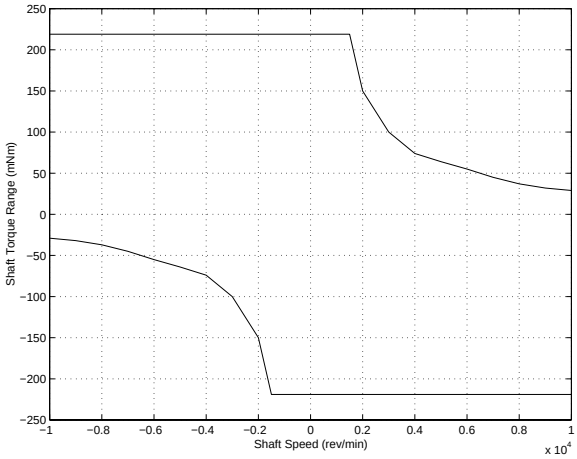


Figure 3: Torque-speed curve for the Maxon RE118751 20W DC motor.

The next component is the modeling of the electrical operation of the DC actuator. The dynamics of the hexapod are formulated as a function of external hip torques. This is based on the assumption that the DC motors operate in torque mode, and instantly provide the desired torque subject to the torque limits, without any electrical dynamics. The leg torque commands of the locomotion controller and the leg rotation speeds can then be converted to DC motor terminal voltages and armature currents for each leg.

$$\begin{aligned} i_{a_i} &= \tau_{s_i}/K_\tau \\ v_{a_i} &= i_{a_i} r_a + K_w w_{s_i} \\ w_{s_i} &:= w_{\phi_i}/k_g \\ \tau_{s_i} &:= \tau_{\phi_i}/k_g \end{aligned}$$

### 3.8 Battery Model

In addition to the actuator model described in Section 3.7, we also incorporate a simple battery discharge model in the hexapod model. The combined use of these components provide an estimate of battery life under different operating regimes and controllers.

The discharge characteristics of off-the-shelf small batteries are in general given by plots of discharge time vs constant discharge current. Figure 4 is an example of such a discharge curve for a lead-acid 12V battery..

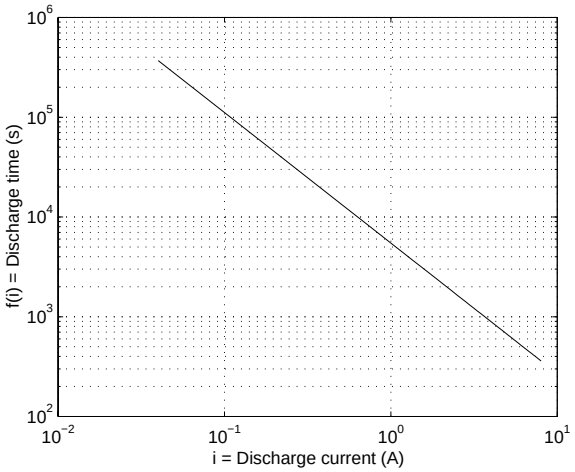


Figure 4: Battery discharge curve for the Yuasa NP2.6-12 battery.

In our model, however, the current drawn from the battery is not constant. Hence, it is not possible to directly use the discharge curve for estimating battery life for hexapod operation. Moreover, the fact that the armature voltage is also a function of time and not the constant nominal voltage of the battery imposes other problems. Finally, the fact that there are six legs, each drawing different currents at different voltages further complicates the model.

Luckily, each of these problems can be addressed in simple and conservative ways to obtain a worst-case estimate of battery life. For this purpose, we first consider a single motor drawing a variable current under constant voltage. We then extend this scheme to handle variable armature voltages assuming a PWM DC motor operation. Finally, we consider the case with multiple motors and their combined load on the battery.

For a certain value of the discharge current, we can use the data of Figure 4 to compute an approximation to the instantaneous percent discharge of the battery with the following formula

$$\frac{dC(t)}{dt} = -\frac{1}{f(i_a(t))}$$

where $C(t)$ is the percent "energy" left in the battery, and $f(i)$ is the battery discharge curve in functional form. Consequently, an estimate for the battery life is given by the solution to the following equation.

$$\int_0^t \frac{1}{f(i_a(\tau))} d\tau = 1 \tag{3}$$

Our model assumes the control of the DC motor is achieved through a PWM servo drive, effectively obtaining a time varying armature voltage from a constant voltage battery. We model this scheme by

observing that the PWM duty factor should be approximately proportional to the armature voltage we get from the actuator model of Section 3.7. The following modification to (3) incorporates this factor.

$$\int_0^t \frac{v_a(\tau)}{v_{nom} f(i(\tau))} d\tau = 1$$

Finally, we adopt a conservative approach to combine the effects of six different actuators on the battery drain. First of all, we need to combine the currents drawn by each of the actuators to obtain the total current waveform on the battery terminals. The output stages of PWM servo amplifiers have full H-bridges to support bidirectional motor operation from a single power supply. Consequently, the total current drawn from the battery is given by

$$i_T(t) = \sum_{i=1}^{6} |i_{a_i}(t)|$$

The worst case in terms of the voltage scaling occurs when the PWM output of all the motor drives are turned on at the same time. Consequently, a conservative estimate of its effect can be incorporated by taking the maximum duty factor of all the motors and scaling the instantaneous battery drain accordingly.

All these modification result in the following final form of the battery lifetime equation

$$\int_0^t \frac{\max\limits_{1 \le i \le 6} (v_{a_i}(\tau))}{v_{nom} f(\sum\limits_{i=1}^{6} i_{a_i}(\tau))} d\tau = 1$$

### 3.9 Relevant Coordinate Transformations

- Positional leg states in the body frame

$$\begin{aligned} \overline{v}_i &= [v_{x_i}, v_{y_i}, v_{z_i}]^T = R_b^{-1}(f_i - r_b) - a_i \\ v_i &= [\theta_i, \phi_i, \rho_i]^T = \begin{bmatrix} \arctan(x_i/\sqrt{y_i^2 + z_i^2}) \\ \arctan 2(y_i, -z_i) \\ \sqrt{x_i^2 + y_i^2 + z_i^2} \end{bmatrix} \end{aligned}$$

- Leg velocities in the body frame

$$\begin{aligned} \dot{\overline{v}}_i &= [\dot{v_{x_i}}, \dot{v_{y_i}}, \dot{v_{z_i}}]^T = R_b^{-1}\left(\dot{f}_i - \dot{r_b} - \dot{R}_b(\overline{v}_i + a_i)\right) \\ \dot{v}_i &= [\dot{\theta}_i, \dot{\phi}_i, \dot{\rho}_i]^T = \begin{bmatrix} (D + Av_{z_i})/(\sqrt{C}F) \\ (-v_{z_i}\dot{v_{y_i}} + v_{y_i}\dot{v_{z_i}})/C \\ E/\sqrt{F} \end{bmatrix} \end{aligned}$$

- Leg accelerations in the body frame

$$\begin{aligned} \ddot{\overline{v}}_i &= [\ddot{v_{x_i}}, \ddot{v_{y_i}}, \ddot{v_{z_i}}]^T = R_b^{-1}\left(\ddot{f}_i - \ddot{r_b} - \ddot{R}_b(\overline{v}_i + a_i) - 2\dot{R}_b\dot{\overline{v}}_i\right) \\ \ddot{v}_i &= [\ddot{\theta}_i, \ddot{\phi}_i, \ddot{\rho}_i]^T \\ &= \begin{bmatrix} \frac{-2v_{x_i}(Av_{z_i}+D)^2 + (C+v_{x_i}^2)(3B^2 v_{x_i} - 2BC\dot{v_{x_i}} + C^2\ddot{v_{x_i}} - Cv_{x_i}(\dot{v_{y_i}}^2 + \dot{v_{z_i}}^2 + v_{y_i}\ddot{v_{y_i}} + v_{z_i}\ddot{v_{z_i}}))}{C^{3/2}(C+v_{x_i}^2)^2} \\ 2B(v_{z_i}\dot{v_{y_i}} - v_{y_i}\dot{v_{z_i}}) - C(v_{z_i}\ddot{v_{y_i}} - v_{y_i}\ddot{v_{z_i}})/C^2 \\ -(4E^2 + 4F(\dot{v_{x_i}}^2 + \dot{v_{y_i}}^2 + \dot{v_{z_i}}^2 + v_{x_i}\ddot{v_{x_i}} + v_{y_i}\ddot{v_{y_i}} + v_{z_i}\ddot{v_{z_i}}))/(4F^{3/2}) \end{bmatrix} \end{aligned}$$

where

$$\begin{aligned} A &:= v_{z_i}\dot{v_{x_i}} - v_{x_i}\dot{v_{z_i}} \\ B &:= v_{y_i}\dot{v_{y_i}} + v_{z_i}\dot{v_{z_i}} \\ C &:= v_{y_i}^2 + v_{z_i}^2 \\ D &:= v_{y_i}^2\dot{v_{x_i}} - v_{x_i}v_{y_i}\dot{v_{y_i}} \\ E &:= v_{x_i}\dot{v_{x_i}} + v_{y_i}\dot{v_{y_i}} + v_{z_i}\dot{v_{z_i}} \\ F &:= v_{x_i}^2 + v_{y_i}^2 + v_{z_i}^2 \end{aligned}$$

and

$$\begin{aligned} \ddot{r_b} &= \frac{F_T}{m_b} \\ \dot{w_b} &= M^{-1}(-J(w_b)Mw_b + \tau_T) \\ \dot{R_b} &= J(w_b)R_b \\ \ddot{R_b} &= J(\dot{w_b})R_b + J(w_b)\dot{R_b} \end{aligned}$$

- Toe coordinates in the world frame

$$\begin{aligned} \overline{v}_i &= \begin{bmatrix} \rho_i \sin\theta_i \\ \rho_i \cos\theta_i \sin\phi_i \\ -\rho_i \cos\theta_i \cos\phi_i \end{bmatrix} \\ f_i &= R_b\left(\overline{v}_i + a_i\right) + r_b \end{aligned}$$

# 4 SimSect: A General Purpose Hybrid Dynamics Simulation Environment

The hexapod model described in Section 3.1 evolved in parallel with *SimSect*, a general purpose hybrid dynamical system simulation software we have developed for analyzing the model. In this section, we describe the simulation environment, along with implementation details and description of usage.

## 4.1 The System Architecture

SimSect incorporates an approach widely used in simulation software, separating the integration engine from the dynamical system definition making it much easier to define and integrate different flows without the need to modify and of the integration procedures.

Defining a dynamical system model consists of providing the hybrid components described in Section 2. The software interface for these definitions closely mirrors the structure of the hybrid DsTool [1], where the model definition provides individual functions for the following tasks

- Initialize the partition structure, the initial state and the initial chart.
- Define the properties of a particular chart, and the boundary functions that will be used.
- Compute the vector field for the individual charts.
- Compute boundary functions, identified by a certain index that the chart initialization determines for the current chart.
- Perform chart transitions by computing the next system state and chart.

- Validate a chart by checking whether a given trajectory point lies in the chart.
- Compute an auxiliary function, mainly used for data collection purposes.
- Compute the homogeneous transformations for visualization of the system trajectory using *Geomview* (see Section 4.3).

These tasks, once implemented in appropriate functions, are then used by the integration engine to compute the system trajectory flow.

## 4.2 The Integration Engine

### 4.2.1 The Iterator Approach

Most numerical algorithms make heavy use of some form of iteration, where a particular procedure is repetitively applied to update certain values, until a predetermined condition is satisfied. Numerical integration of dynamical systems is no exception.

In many levels, SimSect makes use of an abstract *iterator* concept, under which many of the components in the system are defined. An iterator consists of an initialization procedure, an iterating function which also checks for the termination condition and a wrap-up procedure invoked after the termination of the iteration.

SimSect implements a hierarchy of several iterators building different components of the integration. The following sections describe various iterators in SimSect detailing their initialization, iteration and wrap-up procedures.

### 4.2.2 The Chart Iterator

The chart iterator is the topmost level iterator in the integration. It cycles through successive charts until the simulation stops.

There is no specific initialization for this iterator because the system initialization takes care of all the state and chart setup.

The iteration procedure consists of integrating the current chart until a chart crossing is detected, in which case the appropriate transition function is called and the integration continues from the next chart. It also calls the model function defining the chart properties (such as the boundary function indices) at every iteration. The details of the chart integration are hidden in the iteration on level below, which is the flow iteration.

Currently, there are two termination conditions for this iterator. The final time crossing and the maximum chart count. In the occurrence of either event, chart iteration stops and the integration stops.

### 4.2.3 The Flow Iterator

Situated below the chart iterator, the flow iterator steps through successive time steps in the integration inside the current chart. The initialization consists of determining the states of every boundary function in the current chart, determined by the model definition.

The main task of the iteration function is to invoke the Runge Kutta iterator to compute the next flow trajectory point. Several other tasks are also carried out, including

- Validation of the flow point to lie in the current chart
- Recording of the current flow point in the flow data memory for data collection.
- Update of the visualization subsystem state.

The termination condition for this iterator is the detection of a boundary function crossing of the appropriate type. The wrap-up function, then, invokes the stopping iterator to determine the earliest exact transition point, triggered by one, or possibly more than one, boundary functions. The iterator then exits and the chart iterator proceeds with the next chart.

### 4.2.4 Stopping Functions and the Stopping Iterator

Central to the definition of hybrid systems and the operation of the SimSect hybrid integrator are the boundary functions and the concept of *stopping functions*. Boundary functions and their properties are briefly explained in Section 2. The stopping functions are a generalized implementation of boundary functions, very precisely detecting state space triggered events. They are scalar valued functions of the state associated with a particular target value. SimSect incorporates mechanisms to make sure that the discrete numerical integration of the dynamical system computes a point at the target value crossing of these functions, up to a certain numerical precision.

It can easily be seen that the chart transitions, which are target value crossings of the boundary functions can easily be detected using this scheme. Other possible uses of this mechanism are the detection of the final time point to stop the simulation, periodic measurements of the system state, as well as measurements from arbitrary sections of the state space. Currently, only the boundary function and the final time crossing stopping functions are implemented.

The stopping functions are also implemented using an iterator. Once the flow iterator detects the target value crossing of one of the boundary functions, it invokes the stopping function iterator, which then refines the last integration time step until the exact crossing point is determined. SimSect utilizes a midpoint subdivision algorithm to refine the time step, using the leftmost data point as the initial point in evaluating the intermediate trajectory points.

The initialization of the iterator consists of saving the data points to the left and right of the crossing at the point when it is detected. Some additional bookkeeping is also done at this stage.

The iteration procedure for the stopping function detection is fairly complicated. At each iteration, the procedure computes the system state at the midpoint between the most recent left and right data points. Depending on the sign of the stopping function at that state, either the left or the right data point is replaced by the new state and the iteration proceeds with the next step. One very important detail to note is that, in computing the midpoint states, the left data point saved in the initialization phase, which does not change during the iterations is used. When the most recent left data point, updated at each step is used, it leads to cumulative numerical errors, which lead to failure of the detection in certain cases. Discontinuities in the stopping functions are typical instances of such instances, even though they should not occur with a carefully implemented system model.

The iteration terminates upon the detection of the crossing up to a certain numerical precision, or when the maximum number of stopping iterations is exceeded. The wrap-up procedure computes the system state at the detected intersection point and exits.

### 4.2.5 The Runge-Kutta Iterator

The final and the lowest level iterator in SimSect is the Runge-Kutta iterator. It implements an adaptive time step third order Runge-Kutta integration algorithm, where the iteration occurs in the adaptation of the time step. This iterator is called by both the flow and stopping iterators in computing the next discrete trajectory point and abstracts away the adaptation of the time step and the details of the Runge-Kutta algorithm.

In this case, no significant initializations take place. The iteration routine then computes the next data point given the current state and the current time step. Moreover, it suggests a new time step, based on the magnitude of the change in state and the maximum tolerance parameter of the simulation system. Please refer to the source code for the details of the algorithm.

The iteration continues until the norm of the change is smaller than the integration tolerance, in which case it terminates and returns the computed system state, together with the new suggested time step for the next invocation of the iterator.

## 4.3 Visualization

SimSect provides a visualization framework through the use of *Geomview*, a programmable mathematical visualization tool [3]. Through the interface SimSect provides, the model implementation can perform

various visualization tasks ranging from displaying three dimensional environments to plotting and manipulating trajectory data. This section briefly describes how the interface is designed and can be used by the model.

The current SimSect implementation only supports the visualization of three dimensional animations of the simulated system, through a model-supplied function mapping the system state to homogeneous transformation matrices of the virtual environment. The model programmer needs to supply a OOGL scene definition file, where the transformation matrices of the relevant objects in the scene are assigned specific names. The simulation initialization invokes Geomview and load this scene definition file before integration. Then, during integration, the updated homogeneous transformation matrices are computed at a rate determined by the frame rate setting of the simulation, and update messages are sent to Geomview. The result is an animation of the dynamical system at the same speed as the integration. The two components that the model builder needs to provide are the function for computing the homogeneous transformations and the OOGL scene definition file. Please refer to the example hexapod implementation for further details.

## 4.4 SimSect Usage

### 4.4.1 Invocation

SimSect requires the presence of a configuration file, which optionally sets values of various configuration parameters as well as the initial conditions of the dynamical system. SimSect accepts only one command line argument to set the config file, which defaults to `SimSect.rc`

```
Usage :  SimSect [-c config_file]
```

### 4.4.2 The Configuration File

As the first step in its initialization, SimSect loads the configuration file, which consists of assignments of string or numerical values to relevant symbols. These symbols correspond to either the names of the system states, or the system parameters as specified in the model definition code, or various configuration parameters for the integration engine. The assignment statement takes the form

$$symbol\_name = symbol\_value; \ \# \text{ Optional comment field.}$$

The semi-colon is mandatory and separates the assignment statements.

Table 4.4.2 describes the SimSect integrator configuration symbols. The initial state and the parameter names are determined by the mode definition file, so please refer to the hexapod implementation code for an example of their use.

### 4.4.3 Data Output Files

Upon completion of integration, SimSect saves the computed data file as well as some configuration information to a sequence of files. The names of these files can be set using the `dataBaseName` symbol in the configuration file. By default, the data output files are `SimSect.data`, `SimSect.param` and `SimSect.initial`, recording the auxiliary variables over time, the model parameters and the initial system state, respectively. These files are plain ascii files each line corresponding to a data point.

# 5 Behavioral Studies of an Open Loop Locomotion Strategy

In this section, we describe simulation studies of a simple open-loop leg control strategy on the hexapod. These studies are more towards exploring the behavioral capabilities of the platform, rather than characterizing the performance of this simple controller, which can only offer limited speed control and possibly some crude directional control.

| Chart iteration | |
|---|---|
| `finalTime` | The final time for integration |
| `maxChartCount` | Maximum number of transitions before termination |
| **Runge-Kutta iteration** | |
| `tolerance` | Runge-Kutta integration tolerance |
| `maxTimeStep` | Upper bound for the adaptive time step |
| `minTimeStep` | Lower bound for the adaptive time step |
| `rkPower` | Exponent for extending the time step |
| **Stopping iteration** | |
| `stopPrecision` | Numerical precision for crossing detection |
| `maxStopIterations` | Max number of iterations before stopping algorithm gives up |
| **General Configuration** | |
| `recordPeriod` | Time period for trajectory data point recording |
| `measurePeriod` | Time period for exact time measurement of system state |
| `dataBaseName` | Base name for the simulation data output files |
| `useGeomview` | Flag to turn on/off the Geomview interface |

Table 2: SimSect configuration symbols and descriptions

### 5.1 The Open-Loop Clocked Alternating Tripod Controller

Our open-loop control strategy consists of time driven reference signals for each leg, in combination with PD controllers for motor torque control. The reference signals are designed to generate an alternating tripod gait, where each tripod goes through two phases of different rotation speeds, corresponding to periods of time where a tripod is in stance or in flight. This scheme results in a two parameter family of reference signals for the overall locomotion, under the symmetry constraints of the alternating tripod gait. Figure 5 illustrates an example of such a reference command.

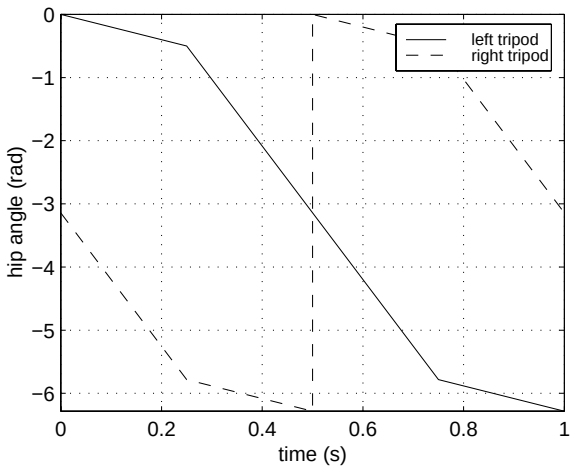


Figure 5: Example alternating tripod reference trajectories. Each leg in a tripod uses the corresponding reference. A tripod is defined as the front and back legs of one side together with the middle leg of the opposite side.

The two parameters of the reference commands are the cycle time and the sweep angle. The cycle time determines the time periodicity of the reference signals and determines the duration of one stride. This cycle time is equally shared between the stance and flight phases of the leg. The difference between the stance and flight phases is introduced by the second parameter. The sweep angle determines the angle span of the stance phase, where the legs are usually much slower and keep the hexapod upright.

## 5.2 Flat Terrain

In this simplest mode of operation, the hexapod invariably settles down on a stable forward running gait at a speed determined by several factors. The upper limit on the locomotion speed is imposed by the actuator limitations of Section 3.7. Below this limit, simple control of the forward velocity can be achieved by using the two controller parameters described in the preceding sections. Figure 6 shows the stable running velocity of the hexapod in response to different controller parameter combinations.

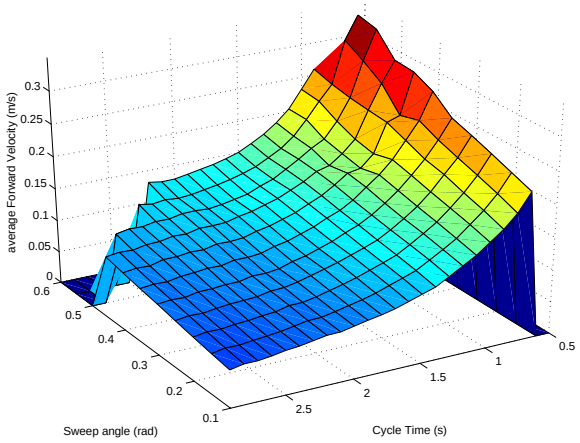


Figure 6: The average forward velocity as a function of the controller parameters $t_c$ and $\phi_s$.

The regular periodic behavior of the hexapod can be observed in the state space trajectories of the system. Figure 7 shows such a plot, where the system settles down on a periodic orbit. Although we do not have any analytical insight on whether the system admits periodic orbits or attractors in phase space, Figure 7 and similar observations suggest their presence.

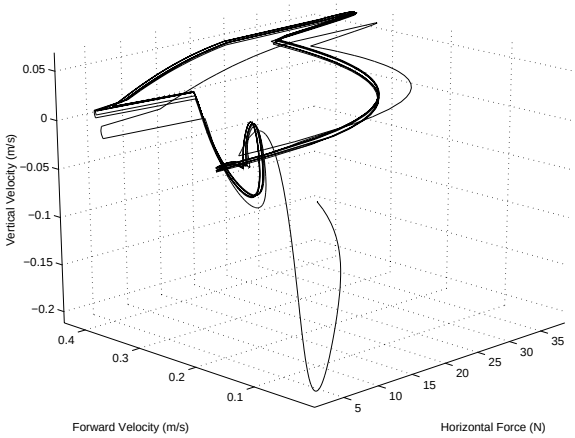


Figure 7: The cyclic behavior of the hexapod running on flat terrain.

This periodic locomotion is also robust to various initial conditions and disturbances. The system is able to recover from initial conditions with lateral and fore-aft velocities of up to $1m/s$, settling down on its periodic running pattern. Higher initial velocities cause the hexapod to topple over, which is quite natural given the small size of the body.

Another important characteristic of the system is that the periodic behavior of the system is sensitive to the initial conditions. Figure 8 shows two different runs with different initial conditions. They have different periodicities, even though the controller parameters are identical in both cases.

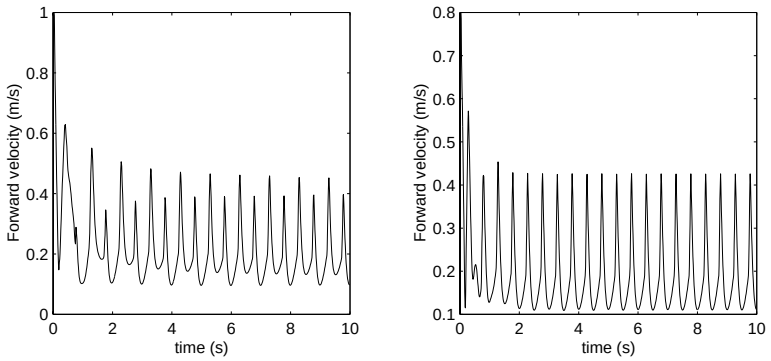


Figure 8: Different periodic forward velocity profiles resulting from two different initial conditions. (left) $v_{x0} = -0.6m/s$, $v_{y0} = -0.8m/s$. (right) $v_{x0} = -0.8m/s$, $v_{y0} = 0.2m/s$.

### 5.3 Climbing Slopes

Among different terrain conditions that we explored is a range of slopes with constant elevation. Figure 9 shows the average forward velocity of the hexapod as a function of the terrain slope. Note that due to the simple open-loop nature of the controller, there is considerable variation in the forward velocity for different slopes. One interesting detail is the increase in speed for positive slopes. This, however, is expected because the slope shifts the phase of the actual stance phase relative to the two-stroke open-loop controller, including more of the fast swing control in stance. This results in faster locomotion.

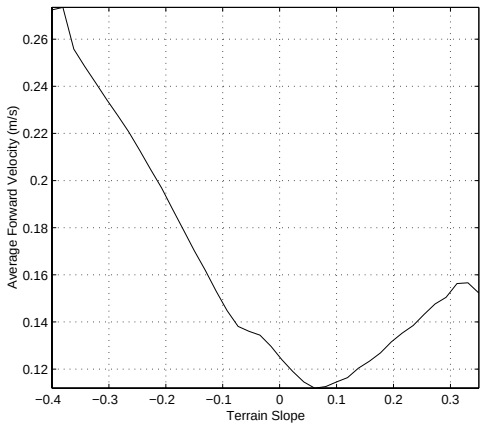


Figure 9: The average forward velocity as a function of the terrain slope for $t_c = 1.0s$ and $\phi_s = \pi/10$.

Another important study with sloped terrain is its effect on the battery lifetime. Figure 10 shows the dependence of the battery lifetime to the terrain slope, computed with the battery model described in Section 3.8. As expected, the battery lifetime decreases as the slope increases. Note, however, that in an actual system, downhill slopes would actually charge the battery, increasing the lifetime, although this effect is not implemented in our model. Even this conservative model predicts hexapod operation of up to 45 minutes with commercially available low-end batteries.

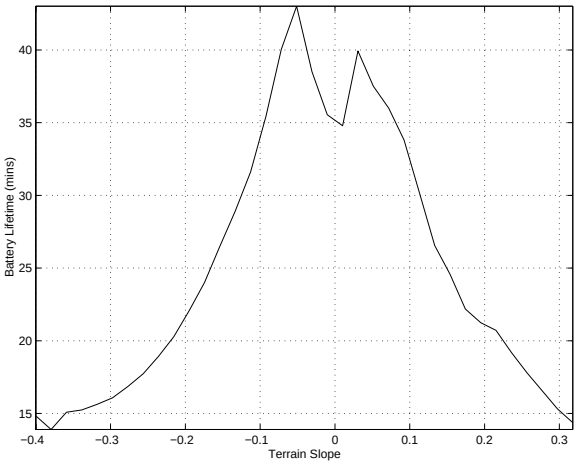


Figure 10: Battery lifetime as a function of the terrain slope.

# 6 Conclusion

In this report, we described our modeling and simulation results for a compliant hexapod robot, inspired by examples of hexapedal locomotion in insects. Even though our model is complicated enough to make complete analysis of the system impossible, it is simple enough that we can attempt to understand certain aspects of its operation. In this context, our simulation results indicate that the static stability properties of the hexapedal structure persist to some extent in dynamical modes of operation and yield stable behaviors. Our investigations of the practical feasibility of the platform under actuation and power limitations also indicate that such a platform can be built with the present technology, having some of the behavioral capabilities that we would like to implement.

# 7 Acknowledgements

We thank Prof. Robert J. Full for his insight to many aspects of locomotion as well as the large body of data that his laboratory provided to help our understanding of hexapedal locomotion. We also thank Prof. John Guckenheimer whose experience was invaluable in our building of the simulation environment for the hexapod model.

# References

[1] J. G. Allen Back and M. Myers. A dynamical simulation facility for hybrid systems. DsTool documentation.

[2] P. C. Hughes. *Spacecraft Attitude Dynamics*. John Wiley & Sons, New York, 1986.

[3] M. Phillips. Geomview software manual.

[4] W. J. S. Uluc Saranli and D. E. Koditschek. Toward the control of a multi-jointed, monoped runner. In *Proceedings of the IEEE International Conference On Robotics and Automation*, Leuven, Belgium, May 1998.