AN ABSTRACT OF THE DISSERTATION OF

Delvin E. Peterson for the degree of Doctor of Philosophy in Mechanical Engineering presented on July 11, 2011.

Title: Reduced Order Multi-Legged Mathematical Modeling of Cockroach Locomotion on Inclines

Abstract approved: ______________________________

David P. Cann

While the locomotion performance of legged robots over flat terrain or known obstacles has improved over the past few decades, they have yet to equal the performance of their animal counterparts over variable terrain. This work analyzes a multi-legged reduced order model of cockroach locomotion on variable slopes which will be used as an inspiration for a future sprawled posture legged robot. The cockroach is modeled as a point mass, and each leg of the cockroach is modeled as a massless, tangentially rigid, linearly elastic spring attached at the center of mass. All of the springs are actuated to allow changes in energy to the system. This is accomplished by varying the force free length of each leg in a feed-forward manner without reliance on feedback to change the actuation scheme. Fixed points of the model are found using a numerical solver that varies the velocity and phase shift parameters while leaving all other parameters at fixed values selected to match true cockroach motion. Each fixed point is checked for stability and robustness representing how effective the model is at staying on the predetermined gait, and transport cost as a measure of how efficient this gait is. Stable and robust fixed points were successfully found for the range of heading angles encompassing those of representative cockroach motion at each slope. Cockroaches may select the gait used based on stability or efficiency. Thus, additional fixed points were found in combination with a search routine that varies the leg actuation parameters in order to optimize either stability or metabolic efficiency, gaining insights into why cockroaches use the gaits that they do. Optimized fixed points were found based on four different leg functional combination families depending on whether each leg pushes or pulls. Optimized fixed point gaits exist for every incline slope studied between level ground and vertical slopes, at a range of initial heading angles that encompass those typically used by cockroaches. The selected gaits using both a stability based and an efficiency based optimization on the modeled cockroach are very similar. Both are also similar to gaits used by real cockroaches. The forces generated by the model are qualitatively similar to the experimental forces.

Reduced Order Multi-Legged Mathematical Modeling of Cockroach Locomotion on Inclines

by

Delvin E. Peterson

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 11, 2011
Commencement June 2012

ACKNOWLEDGEMENTS

Several people deserve special acknowledgment for their assistance in this research. First, I would like to thank Dr. John Schmitt for his advice and assistance in this research and his many suggestions that guided the direction of my work. Also, thank you to my graduate committee for their help. I would also like to thank Dr. Ginger Ketting-Weller and the administration of my employer, Walla Walla University, for encouraging me to work on this degree and granting me a leave of absence for its pursuit. Finally, I would like to thank my wife Sara and our three children for their encouragement to keep working and patience as I spent many long hours on this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDICES

Chapter 1 – Introduction

Despite advancements in the design and implementation of legged robots, designing a robot to navigate a variety of difficult terrains remains a difficult task. In particular, a robot which could rapidly and stably navigate inclines of arbitrary slope, obstacles, loose stones, and other perturbations would be extremely useful for disaster response, exploration, and military applications. While the locomotion performance of legged robots over flat terrain or known obstacles has improved over the past few decades, they have yet to equal the performance of their animal counterparts over variable terrain. Ultimately, understanding the fundamental mechanisms by which animals are able to achieve such performance could prove instrumental in the design and control of future legged robot designs.

Since cockroaches are relatively inexpensive to obtain and exhibit interesting dynamic characteristics, their locomotion has been widely studied [1, 3]. Their lateral motion has been shown to be stable both on the level [4] and on vertical slopes [5]. While many creatures run using legs directly under their body, cockroaches use legs to the side in a sprawled posture. This results in significant lateral motion whether the cockroach is on the level [1] or on a cliff [2]. Intuitively, this lateral motion seems energetically inefficient, but it may be important for stability or robustness.

A full model of a cockroach including all of the leg segments, muscles, and body segments would involve many degrees of freedom and be cumbersome to solve. Thus, a reduced order model will be developed based on the motion of cockroaches. Similar reduced order models have effectively predicted motion for cockroaches and other animals while being much simpler to analyze than a full model [6, 4, 7, 2].

Initial investigations with a single-legged, transverse-plane, reduced-order, actuated model on varying slopes have suggested that cockroaches switch from pushing to pulling function based on stability and robustness concerns. Based on optimizing the robustness, the transition from pushing function to pulling function occurs at around a slope of 15 to 40 degrees depending on speed [8]. However, no predictions were made for transitions of individual leg function since it was a single legged model. Additionally, there were no comparisons to actual cockroach force data for slopes other than level or vertical.

This research will investigate a similar transverse-plane, reduced-order, actuated model with the full six legs. This will allow predictions for transitions for each of the legs. Also, the forces generated will be compared with those of cockroaches on varying slopes. A mathematical model which produces cockroach motion with legs that transition from pushing to pulling at slopes similar to those of actual cockroaches will be useful in developing a sprawled posture robot with natural stability like cockroaches have.

This paper will investigate whether or not cockroaches select their gait based on dynamic stability or efficiency concerns, or possibly a combination of both metrics. In particular, optimal gaits will be found based on varying the degree of leg actuation, similar to how a cockroach can select the amount of muscle actuation used. This will be done over a range of initial heading angles encompassing typical values used by cockroaches, and over a range of incline slopes from level ground to vertical slopes. These gaits will be optimized both for stability and separately for metabolic efficiency. By comparing the resulting motion with the actual cockroach motion, we will learn which metric drives the cockroaches gaits. The implementation of both of these optimization metrics could be particularly useful in the development of a sprawled posture multi-legged robot that is similarly stable and efficient.

This document proceeds as follows. Chapter 2 describes the proposed mathematical model for cockroach locomotion after listing similar models in the research. Subsequently, Chapter 3 describes the methods used to gather gaits as well as the metrics used to measure stability, robustness, and metabolic efficiency. Also, the optimization techniques will be developed. The resulting motion from the cockroach locomotion model are analyzed in Chapter 4. Chapter 5 will discuss the resulting gaits from optimization on both metabolic efficiency and dynamic stability. Both the non-optimized and optimized gaits will be compared to forces generated by actual cockroaches in Chapter 6. Chapter 7 describes the importance of the results and suggests some areas for future research.

Chapter 2 – Mathematical Model

2.1 Previous Reduced-Order Modeling Efforts

This work proceeds from a body of research modeling animal motion with simple elastic elements. In the sagittal plane, perpendicular to the ground and parallel to the direction of forward motion, the fundamental template of this type is the Spring-Loaded Inverted Pendulum (SLIP) model. Generally, the running body is modeled as a point mass, and the leg is modeled as a simple inverted pendulum with a spring. The spring allows an exchange between kinetic and potential spring energy in a similar manner to how many animals store elastic energy while running [9]. This model's obvious advantage lies in its simplicity. With a single mass-spring system the equations of motion are very easy to derive and to solve numerically [10].

The motion of many trotting, running, and hopping animals can be represented with a SLIP model [6]. This simplified model has been shown to accurately match the motion of actual legged creatures such as humans, horses, and cockroaches [11]. Studies with guinea fowl using the SLIP model show that the creatures maintain stability after unexpected disturbances in ground height in a passive uncontrolled manner. This is accomplished by rotating the leg towards the ground at a constant rate shortly before expected foot contact, which changes the angle at which the leg is placed on the next step. [12].

While the SLIP model has been shown to accurately describe motion in the sagittal plane, it does not address lateral motion inherent in sprawl posture animals, which deploy legs to the side of the main body. Thus, a similar mass and spring model has been developed for modeling running animals in the transverse plane, parallel to the ground. This model, termed lateral leg-spring (LLS), incorporates a point mass or rigid body with a spring attached in a lateral direction [13]. The spring alternates sides in a manner similar to how a cockroach alternates its tripod gait, causing the body to oscillate back in forth.

Only motion within the transverse plane is modeled, while the hopping motion in the saggital plane is ignored. This means that unlike the SLIP model, legs can be touched down at any time rather than waiting for gravity to return the leg to the ground. Using this model with parameters similar to cockroaches, it has been shown that the resulting motion is naturally stable while also closely matching the lateral translational forces and velocities of real cockroach motion [4]. However the resulting moments and corresponding yaw angles are an order of magnitude different from the actual cockroach data because the single leg force cannot adequately produce the moments of the three legs acting together in a tripod.

To provide a better match to cockroach motion, one multi-legged model replaces each of three legs in a tripod with a spring element similar to the LLS model. The generated leg forces are controlled to approximately match prescribed forcing functions characteristic of actual cockroach motion using a variable equilibrium spring length and a variable hip location on all three legs [14]. Although the controlled parameters were calculated based on a model decoupled from the rotational motion, the resulting forces and moments were very similar to actual cockroach data both qualitatively and quantitatively. Also, the modeled gaits were stable at velocities similar to those preferred by cockroaches.

Another model replaces lateral linear springs with more biologically relevant legs that include both a hip and knee joint while still constraining motion to the lateral plane. This allows rotational motion using torsional springs [15]. Again, leg forces are controlled to match characteristic functions of cockroach forces in this case by setting the moment free angle for each of the torsional springs. Another development of the model more closely approximates the biological system by replacing the torsional springs with actuated muscle pairs. The resulting forces are matched to forces characteristic of cockroaches by adjusting action potential spike trains for the muscles [16]. These models also resulted in translational and rotational motion similar to that measured on cockroaches, with models that are progressively closer to how real cockroaches activate their legs.

All of the preceding LLS models were employed for running on level ground. Some limited work has been done on reduced order modeling of the lateral motion of legged creatures on slopes. Despite significant biological differences, cockroaches and geckos use a dynamically similar gait while running up a vertical slope. This fact has inspired a reduced order model, based on the lateral leg spring model, that produces forces similar to both cockroaches and geckos climbing. This climbing template model is made non-conservative through the addition of a linear actuator in-line with the leg spring. Shortening the actuator length pulls the rigid body mass towards the foot attachment point while simultaneously increasing the potential energy of the spring by extending it [2]. Simulations using this model have shown that a modeled cockroach displays more stability to lateral impulses by placing legs off to the side rather than placing them directly in front. This may provide an answer to why cockroaches employ this lateral motion [5].

Little work has been done with reduced order modeling of legged motion on inclines varying from level ground to vertical slopes. It has been shown that stable and robust gaits exist for a variable slope model based on the lateral leg spring template. This model uses legs actuated by a variable force free length in a single leg, which produces a similar effect to a linear actuator in-line with a spring [8]. The actuator was varied to match an assumed lateral force profile based on interpolating cockroach forces between those observed on the level [1] and on vertical slopes [2]. While gaits were found using both pushing and pulling leg function for all inclines, the stability and robustness of these respective gaits suggest that pushing function is preferred at low slopes, while pulling function is more stable at higher slopes [8]. The current research is based on a multi-legged modification to this model which sets the force free length based on a sinusoidal function with a phase shift.

Many researchers have investigated optimizing robots or mathematical models with respect to either stability or efficiency. For instance, one example varies the mass, inertia, and geometric properties of a modeled one legged hopper to optimize its stability. It uses a numerical solver based on a Nelder-Mead algorithm in an outer loop to minimize the eigenvalues of the Jacobian of a discretized version of the system. To ensure that the optimized parameters produce fixed points, an inner loop is used to find fixed points of the system [17]. Another paper focuses on a model of the human foot during walking to show that the optimal foot length and foot ratio for stability are very close to actual human foot dimensions [18].

One recent work analyzes various possible walking and running gaits for a modeled human leg. By optimizing for metabolic efficiency, the optimization routine settled on both inverted pendulum walking and impulsive running as the optimal gaits depending on what speed was desired. In addition to these two gaits which humans often employ a third gait, termed pendular running, was also discovered for intermediate speeds [19]. Another work analyzed the energy losses of a full model of a hexapedal robot to determine the optimal gait to use during turning of the robot. This turned out to be a tetrapodal gait rather than a tripodal gait [20].

2.2 A Reduced-Order Point Mass Model

The model used is based on a previous point-mass, actuated spring model with a single active leg suitable for variable slopes [8]. However, the model has been extended to a six-legged model more similar to actual cockroach motion. It has been theorized that a single legged model can not accurately describe the yawing motion of the cockroach due to a lack of flexibility to choose moments for one leg rather than a trio of legs [4]. However, initial studies will be done with a point mass model that does not account for this yawing motion. As insights are gained regarding the multi-legged model, this model will be extended to a rigid body which allows rotational motion. The mass of the legs will be neglected since it amounts to only 13% of the total mass [3].

2.3 Alternating Tripod Gait

Cockroaches typically run with an alternating tripod gait where three legs contact the ground, forming a tripod around the cockroach while the opposing legs move [21]. A left tripod is represented in Figure 2.1 where the front left, mid right, and

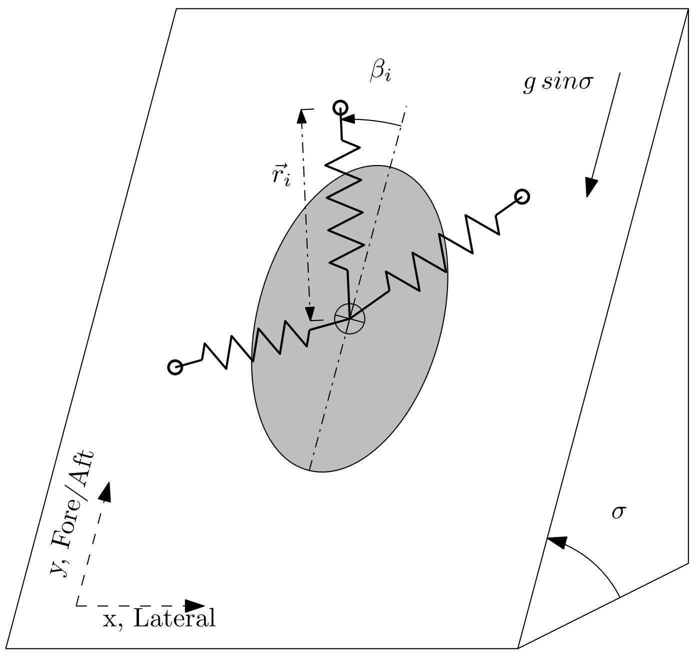

Figure 2.1: Reduced order point mass model for cockroach locomotion with multiple legs represented as linear elastic elements

back left legs contact. In idealized models [14, 2, 16, 8], the right tripod stance is placed at the instant the left tripod stance is lifted. This gives a duty factor, or percentage of time a leg contacts the ground, of 50%. However cockroaches actually employ duty cycles of around 52% to 66% on the level [21] and 50% to 74% on vertical slopes [2].

The stride of each leg is characterized by a stance phase when the leg contacts the ground and a swing phase when the leg swing to its next location. All legs of each tripod start based on a fixed clock cycle, but lift-off occurs when each legs' force returns to zero. This gives a fixed length of time for the stride, but not a fixed length for the stance phase. It also allows for duty cycles, or percentage of time a leg contacts the ground, of other than 50%. Figure 2.2 describes a possible scenario for the modeled alternating tripod gait. Initially, all of the left tripod feet are placed. The timing clock signals the right stance phase to began at t_{des}.

Then, at $2t_{des}$ the left stance phase is signaled to start again. Four complete steps, or two strides for each of a left and right stance phase are depicted. In this example, all legs generate a pushing force. This is apparent since the equilibrium spring length becomes larger than the actual leg length.

Any leg that crosses zero force becomes inactive until the timing clock reactivates it at a new foot placement. This is important since any energy stored in the spring would be lost to the system if the leg was lifted with force. With legs modeled as springs, zero force occurs when the leg length equals the leg force free, or equilibrium, length. The model always starts with a left stance phase. To allow for right stance phase legs that may still be active, the previous stance phase foot placements are calculated as if the modeled roach was traveling at the desired velocity in a stable gait. Each leg that does not produce the desired force direction (pushing or pulling) at the start of the left stance phase is deactivated since it must have already crossed the zero force threshold. Both pushing and pulling leg function are studied, but for this particular simulation (Figure 2.2) pushing gaits are used for all legs. So, the middle left leg and back right leg are still active even though they were part of the previous right stance phase tripod because their forces would still be pushing. However the front right leg has lifted off since its previous foot placement location would have indicated a pulling force.

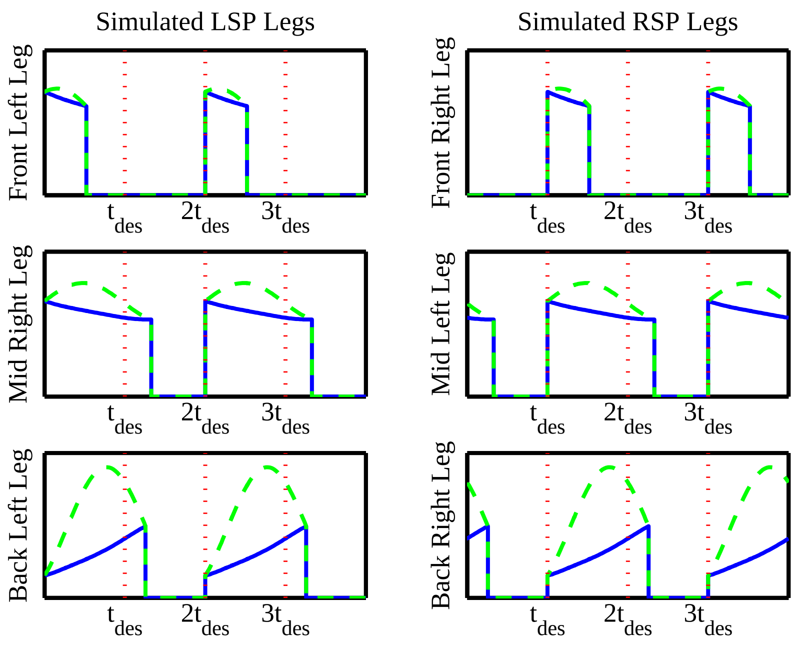

Figure 2.2: Illustration of leg lengths during a fixed stride length simulation of cockroaches with six legs. Lengths are plotted at zero length during the swing phase when the legs are off of the ground. The affect of leg actuation phase shifts on modeled legs is also shown for: $\phi_F = 63°$, $\phi_M = 4°$, $\phi_B = -49°$, with positive phase shifts showing short stance durations and negative phase shifts showing long stance durations

- - - represent leg spring equilibrium length

—— represent actual leg length

2.4 Leg Actuation

Each leg of the cockroach is modeled as a massless, tangentially rigid, linearly elastic spring with a spring force given by:

$$\vec{F}_{leg} = k\left(|\vec{r}_i| - l_{eq}\right)\frac{\vec{r}_i}{|\vec{r}_i|} \tag{2.1}$$

where k is the spring stiffness of the leg, $\vec{r}_i$ is the leg length vector shown in Figure 2.1, and l_{eq} represents the equilibrium length of the spring, or the leg length which would produce zero force. Since the desired model is for variable slope, traditional passive springs would not be adequate by themselves since they cannot add the energy required to climb a slope. However, all of the springs are actuated to allow changes in energy to the system. As in other similar models [14, 2, 8] this is accomplished by changing the force free length of the spring, l_{eq}. Increasing l_{eq} would cause an increased pushing force along the leg, while decreasing it would cause an increased pulling force, either of which would change the kinetic energy of the cockroach mass. In a real system, it would not be feasible to create a spring with a changing equilibrium length, but a similar result could be obtained by attaching a traditional spring to an actuated length leg segment.

Variation in the force free length of each leg will be modeled in a feed forward manner without reliance on feedback to change the actuation scheme. Cockroaches seem to use a similarly uncontrolled method for setting their muscle actuations. When running across a terrain of blocks varying in height up to three times the hip height of the cockroach, they exhibit the same muscle activation patterns as when running across level ground even though the motion experienced is significantly different. These similar muscle patterns are even evident when a foot completely misses contacting the ground [22]. Cockroaches seem to use only their own musculo-skeletal structure, rather than neural control, to maintain stability within a stride since restorative forces act faster than neural control would allow when roaches experience large lateral impulses [23].

A sinusoidal actuation pattern will be used to simplify the analysis because the forces generated by cockroaches are generally sinusoidal both on the level [1] and on vertical slopes [2].

$$l_{eq} = l_o - l_{dev} \sin\left(\frac{\pi}{t_{des}}\left(t - t_{start}\right) + \phi\right) \tag{2.2}$$

In Eq. (2.2), l_o represents a baseline length for the leg and is taken from observations of where cockroaches initially place their legs. The actual starting length in the model will differ since the leg must be placed with zero force regardless of the controlled phase shift, ϕ. The amount of deviation in the leg equilibrium length is limited by l_{dev}, also known as the leg actuation parameter. Physical observations of cockroaches on the level show typical variations in the distance to foot placement point of about 10% of the average values [24]. However, in order to provide enough actuation for climbing a slope, an l_{dev} value 50% of l_0 will be used for the initial set of non-optimized gaits. This value was chosen because it avoided allowing the deviation to be too large or small relative to the nominal length. A too small value lead to an ineffective leg on slopes, and a too large value could lead to places where the equilibrium length would have to be negative, which is physically impossible. The optimized gaits will vary this parameter within acceptable limits to optimize certain characteristics of the gait.

A desired step duration, t_{des} can be calculated from the desired velocity of the cockroach (v_{des}) based on a curve fit of measured cockroach data for cockroaches on the level [21]. This relationship between speed and frequency also holds for cockroaches climbing a vertical slope [2]. The time when each leg was initially placed is represented by t_{start}. A similar sinusoidal actuation scheme has been successfully implemented in the design of a legged robot on vertical slopes [5].

Figure 2.2 demonstrates the effect of the ϕ parameter on the actuation of a leg. With a phase shift near zero, as in the middle leg, the leg actuation proceeds as a sinusoidal wave form. The frequency of the wave form is half of the step frequency such that the leg force-free length returns to its original value at t_{des} when the opposing leg is set to set down. Because of the large phase shift on the front leg, the leg equilibrium length at set down is very close to its peak value so it soon returns to the actual leg length and the foot is lifted. On the other hand, the back leg has a negative phase shift so that the leg equilibrium length climbs for longer than half of its period before reaching its maximum.

While all parameter values will be based on observations of cockroach motion, a robot based on the template will need to be built at a larger scale. Fortunately, the scaling of parameters is a well understood and easily solved problem. In past work, cockroach templates have been extended to robots ten times longer in dimension (1000 times more massive). A scaling factor was identified for mass, frequency, stiffness, velocity, damping, and power [5].

2.5 Equations of Motion

Within the plane of motion, the only forces that act are the leg forces and the component of the gravitational force that acts in the plane. The equations of motion, developed using Newton's laws in the inertial frame, are given by:

$$\ddot{x} = \frac{\sum F_x}{m} \tag{2.3}$$

$$\text{for} \quad \sum F_x = \mathbb{C}_x \left(\sum_{i=1}^{6} \vec{F}_{leg_i} \right) \tag{2.4}$$

$$\ddot{y} = \frac{\sum F_y}{m} \tag{2.5}$$

$$\text{for} \quad \sum F_y = \mathbb{C}_y \left(\sum_{i=1}^{6} \vec{F}_{leg_i} \right) - mg \sin \sigma \tag{2.6}$$

In Eqs. (2.4) and (2.6), $\mathbb{C}$ refers to the inertial frame component of the vector and $\vec{F}_{leg_i}$ are the leg forces from Eq. (2.1). The mass of the roach is represented by m and the slope of the incline by σ. Assuming appropriate initial conditions, these differential equations can be solved analytically using a numerical solver. This was done within a Matlab platform using the ode45 function, a numerical ordinary differential equation solver based on an explicit Runge-Kutta 4-5 formula. The events option was used to interrupt the simulation at times when any legs were either placed or lifted off since these discontinuities would otherwise hinder the numerical solution.

Spring mass based models of cockroach locomotion in the transverse plane have shown this model to be inherently stable both on the level [4] and on inclines [25, 8]. It seems that the sprawled posture leg deployment of the cockroach legs, in combination with the natural springiness of the musculo-skeletal system are important for stabilizing cockroaches motion. In fact studies of cockroach response to a large lateral impulse show that they began to generate restorative forces faster than would be possible using neural signals [23]. This indicates that the morphology of the cockroach itself naturally generates these forces to stably restore the insect to its natural gait.

Chapter 3 – Analysis Methods

3.1 Finding Fixed Points

While the motion of the modeled cockroach is continuous, it can be discretized by observing the state of the cockroach $\left(v_i^{TD}, \delta_i^{TD}\right)$ at foot placement of subsequent steps. In this manner, the state of the system is sampled as it passes through a Poincaré section represented by the instant when each foot is placed. Figure 3.1 shows motion typical of actual cockroaches, at an exaggerated scale to show horizontal motion. Cockroaches do not travel in a straight line, but oscillate back and forth with each step. A Poincaré map is a function that outputs the state of the continuous system for the next time the Poincaré section is reached based on the state of the continuous system at the last crossing of the Poincaré section. A fixed point of the mapping is defined by a set of initial conditions and foot placement parameters that result in the modeled cockroach returning to the same velocity and direction heading after each step. This means that the Poincaré map will return the same output states that were input.

Certain conditions have to be met for a gait to be a fixed point of this model. Most importantly, the velocity at the next step must be the same and the heading angle must be equal in magnitude but in the opposite direction because of the assumption of symmetry. Additionally, the average cockroach velocity in the forward direction should match a prescribed desired velocity. Finally, there should not be horizontal drift in the foot placement locations from step to step.

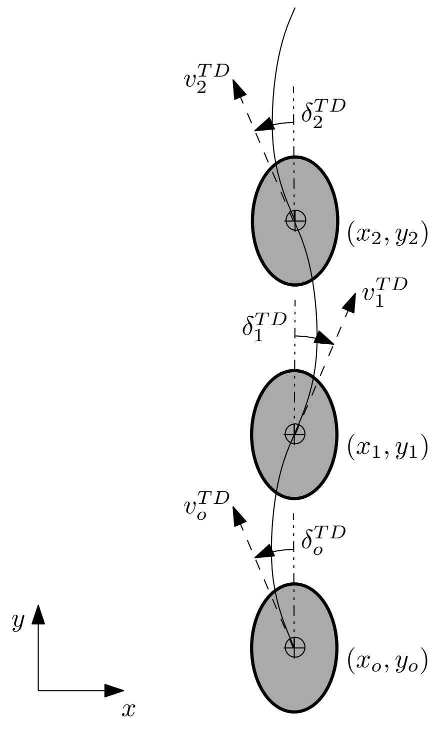

Figure 3.1: Schematic representation of typical cockroach motion. Lateral motion is exaggerated for clarity. Shows criteria used to identify valid fixed point gaits: See equations (3.1) - (3.4)

Mathematically, these conditions are described by Eqs. (3.1) - (3.4).

$$v_{i+1}^{TD} = v_i^{TD} \tag{3.1}$$

$$\delta_{i+1}^{TD} = -\delta_i^{TD} \tag{3.2}$$

$$v_{des} = \frac{y_{i+1}^{TD} - y_i^{TD}}{t_{i+1}^{TD} - t_i^{TD}} \tag{3.3}$$

$$x_{i+1}^{TD} = x_i^{TD} \tag{3.4}$$

Whether or not a gait is a fixed point depends partially on the initial condition of the roach (v_o, δ_o). Additionally, the phase shift parameters of the leg actuation (ϕ) discussed in Eq. (2.2) can be selected to meet the fixed point criteria. To limit the number of parameters required, symmetrical gaits are desired so that only one of each phase shift parameter for each pair of front, mid, and, back legs need to be set. Fixed points can be found using a numerical fixed point solver that varies v_o and each of three ϕ parameters while leaving all other parameters at fixed values selected to match true cockroach motion. Again the Matlab platform provides a handy function for this purpose called fsolve. This function for solving non-linear equations was used with a trust-region dogleg algorithm. The function value tolerance was set to 10^{-8} and the solution tolerance to 10^{-7}.

3.2 Stability

Finding a fixed point ensures that a gait will continue to return to the same state at each step as long as it is not disturbed. However real robots, like real cockroaches, do experience disturbances such as variance in the terrain, slipping feet, and external forces. Fixed points are more valuable if they return to their states when disturbed. A good definition of stability is a system's tendency to return to its original state after a disturbance. While a disturbance may cause the motion to deviate from its path, a stable system will eventually return towards the original path. Contrarily, unstable systems will experience a subsequent growth in the disturbance over time.

As the continuous system is non-linear, determining stability is not straightforward. However if the discretized Poincaré map can be shown to be stable at a given fixed point, then the continuous system must also be stable in a local region. Stability of discrete systems can be checked by analytically forming the Jacobian and observing the eigenvalues. The Jacobian of the Poincaré map is calculated by first making small changes to the initial state of the system at foot placement. Simulating the model forward in time from the start of one left-stance-phase to the start of the next LSP, shows the resulting states after one complete stride. The differences in the resulting states based on the changes in the initial conditions define the Jacobian.

Then the eigenvalues of the Jacobian matrix are calculated. For discrete systems, if all eigenvalues of the Poincaré map lie within a magnitude of one, the system is stable. This ensures that variations from the initial step are not growing in subsequent steps. Those gaits that are stable fixed points will return to the fixed gait after a small disturbance.

3.3 Robustness

Since this is a non-linear system, stability in the local region does not guarantee stability for a sufficiently large disturbance. Robustness provides an even more indicative test of the fixed points' ability to withstand disturbances. Stability describes whether or not a gait can return to its fixed point after a disturbance. Lower eigenvalue magnitudes correspond to faster recovery. However, a non-linear system can be very stable for a small disturbance but not be able to return from a larger disturbance. Robustness quantifies how big of a disturbance can be withstood as well as how quickly it returns. To test for robustness the model must be exposed to disturbances of various magnitudes. In each case, the time for the model to recover to the fixed point should be observed.

As a test for robustness consider a lateral impulse applied during the first step [8]. This test is similar to a test performed on cockroaches which were given a lateral impulse by using chemical propellants to accelerate a small steel ball from a device attached to the cockroach. Despite receiving lateral impulses equal to 85% of the creature's forward momentum they were able to recover to the original gait in two steps on average. Furthermore, restorative forces were generated as soon as 10 ms after the perturbation was applied, faster than neural based response would allow for [23]. For each fixed point, lateral impulses from -80% to 80% of forward momentum are applied over a short period of time. As shown in Figure 3.2, the force begans ramping up at around $\frac{t_{des}}{8}$. The impulse used lasts for 0.004 seconds of which 0.003 seconds are during the ramp up time. The parameters of this theoretical impulse were based on those observed from the above study. The number of steps required to return to the original fixed point within 0.1% of its state values are recorded for each case.

3.4 Transport Cost

It seems natural that cockroaches would select certain gaits over others based on their metabolic efficiency. If a given gait has a high degree of inefficiency, it

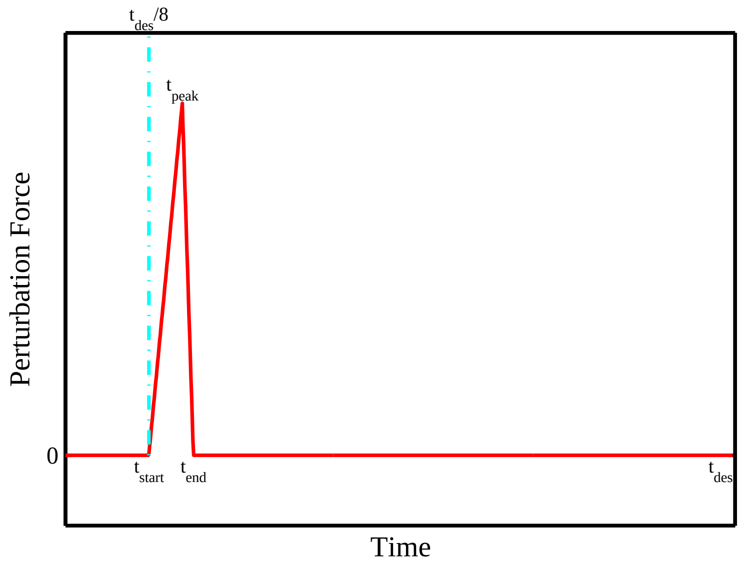

Figure 3.2: Simulated perturbation force throughout a stance phase used to check the robustness of modeled motion. Perturbation is modeled as an impulsive force applied at a time 1/8 of the expected stance phase.

will require more energy and thus require more the cockroach to consume more food. On the other hand, cockroaches may select the gait used based on stability instead of efficiency. For instance, a cockroach is less likely to escape from a predator or catch its prey if its unstable but efficient gait causes it to run in circles. This research will consider gaits optimized based on both stability and transport cost to gain insights into why cockroaches use the gaits that they do.

Metabolic efficiency can be expressed as the cost of transport which calculates the energy expended by an animal during its motion. For legged locomotion, this can include resting costs of energy that an animal uses even when not moving, stance costs describing the energy cost for using the muscles during a stance, and swing costs related to the use of muscles during leg swing [26, 27]. Since optimality is being determined at a fixed desired speed, the resting cost does not matter since it is constant. Also, the swing leg costs are negligible for a massless leg model.

Studies in metabolic efficiency have proven the optimality of the walking and running gaits used by humans and predicted the speeds at which each would occur [19, 26]. A similar approach has been used to determine preferred gaits of horses and humans at certain speeds [28] and to predict the step width of a human during walking [29].

Recent research has shown that minimizing the work done by muscles results in gaits similar to those found by minimizing other metrics of transport cost [27]. One simple equation that describes the cost of work in a muscle is shown below [26, 27].

$$C_w = \int_0^{t_{des}} b_1 \left[F\dot{l}\right]^+ + b_2 \left[F\dot{l}\right]^- dt \tag{3.5}$$

In this equation, b_1 and b_2 represent the contribution of positive work and negative work respectively to the metabolic cost. F is the force generated axially along a leg, and $\dot{l}$ is the time derivative of the length of the leg.

Since each of the forces in the legs are modeled as simple springs it is straightforward to develop an equation for the cost of work done by each leg. Taking $b_1 = b_2 = 1$, the results are shown in Eq. (3.6). This equation is normalized by dividing by the weight and the length of the stride.

$$C_{tran} = \frac{1}{mgL_s} \int_0^{t_1} \sum_{i=1}^{6} \left\{ \left|\vec{F}_i\left(t\right)\right| \left| \frac{\left(x - x_{fp}\right)\dot{x} + \left(y - y_{fp}\right)\dot{y}}{\sqrt{\left(x - x_{fp}\right)^2 + \left(y - y_{fp}\right)^2}} \right| \right\} dt \tag{3.6}$$

L_s and t_s represent the length and time of a step respectively. The force in each leg is given by F_i and x and y represent the motion of the center of mass with $\dot{x}$ and $\dot{y}$ representing their velocities. The original placements of each foot are given by x_{fp} and y_{fp}.

3.5 Optimization

Using the technique described in Section 3.1, fixed points of this model will be found at constant values of l_{dev} representing the amount of deviation allowed in the equilibrium length of the each leg spring. It is also possible to allow these parameters to vary in order to optimize gaits to meet certain design metrics. By increasing the leg actuation parameter, larger amounts of energy can be added to the system since the equilibrium length goes through greater change. Similarly, reducing the leg actuation parameter will typically reduce the amount of work done by that leg. This change in amount of energy is analogous to a cockroach's increased muscle actuation when it requires greater forces.

Fixed points will be found to optimize both stability and transport cost functions. This is done using a Nelder-Mead simplex algorithm which makes small changes in parameters from a starting point. Each of these points is assigned a cost based on either minimizing eigenvalues or minimizing transport cost. Based on the results of these points, a new point is selected in the multi-dimensional direction most likely to produce lower costs [30, 17]. Using a search algorithm, optimal fixed points will be found for each of these conditions for a variety of pushing combinations, slopes, and heading angles.

For the stability optimization method, the cost function was based primarily on minimizing the absolute value of the largest eigenvalue since this serves as the metric for stability. In a similar manner, the cost function for the metabolic efficiency optimization routine was primarily based on minimizing the transport cost metric. Several additional cost metrics were also added to both optimization routines to avoid some non-desirable fixed points as shown in Table 3.1. Typically these constraints describe conditions where the fixed point is close to a transition such that a small change in parameter values could result in drastically different motion.

These additional constraints were divided into two separate groups. The soft constraints consisted of added constraint costs that started at zero as the constraint was crossed and ramped up to higher values as the constrained value extended beyond the constraint. In this way these constraint violations were discouraged, but not prohibited. On the other hand, the hard constraints included a large punitive cost imposed immediately when the constraint was violated in addition to the gradually growing cost. These prevented the search algorithm from continuing to violate these constraints. Hard constraints were imposed in cases where violating the constraint would not produce a valid fixed point.

Constraint	Type	Purpose
Fixed point exists	Hard	Desired gaits must return to their original states at each step
Fixed point exists at 105% of δ_0	Hard	Prevent fixed points right at a barrier of existence
$l_{dev_i} \ngtr 0.9 l_{0_i}$	Hard	Too much actuation could cause an impossible negative equilibrium length
$l_{dev_i} \nless 0.2 l_{0_i}$	Hard	Too little actuation renders the leg force insignificant
$\phi_i \nless -90°$	Hard	Too much negative phase shift can cause the leg to switch between pushing and pulling function
$\phi_i \ngtr 60°$	Soft	Too much positive phase shift renders the leg force insignificant
$tan^{-1} \frac{d}{dt} \lvert l_{eq} - l_{spr} \rvert_{t_{TD}} \nless 10°$	Soft	Too shallow of a departure between the equilibrium and actual spring lengths after leg touch down leads to an insignificant force or a switch in leg function
$tan^{-1} \frac{d}{dt} \lvert l_{eq} - l_{spr} \rvert_{t_{LO}} \nless 10°$	Soft	Too shallow of an approach between the equilibrium and actual spring lengths before leg lift off could lead to a missed lift off if the difference never returns to zero

Table 3.1: Additional constraints implemented during optimization based on stability and metabolic efficiency. Hard constraints can not be violated by the optimizing routine, while soft constraints can be violated with additional penalties added to the function that is being minimized.

Chapter 4 – Non-Optimized Fixed Point Results

A number of fixed points were found as described in Section 3.1 over the set of parameter values found in Table 4.1. Recent analysis of cockroaches indicates that the preferred speed of the cockroach varies in a linear fashion between the endpoints shown in the table (D. Goldman, 2011, personal communication, Georgia Institute of Technology). Thus, the desired velocity (v_{des}) and step duration (t_{des}) were interpolated based on the incline slope (σ). The incline slope was varied in increments of 5°. Additionally, initial heading angles (δ_0) varied as shown from 0 rad to 0.45 rad in increments of 0.05 rad for each slope. Recent research by Dr. Goldman at Georgia Institute of Technology indicates a preferred heading angle of $\delta_0 = 0.15 \pm 0.10$ rad. Additionally, there was an uncertainty of around 0.10 rad on the yaw angle of the cockroaches describing their body orientation in relation to the inertial frame. This yaw angle is not apparent in the point mass model used since it does not allow for moments, however it indicated a need for a greater range of heading angles studied. Parameters for equilibrium length deviation were selected at $l_{dev} = \pm l_0/2$ (see Section 2.4) with positive numbers corresponding to pulling leg function and negative numbers corresponding to pushing leg function.

The resulting fixed point gaits are illustrated in Figure 4.1. Gaits that return to their initial state at each step exist for the majority of attempted combinations of δ_0 and σ. At the lower left corner representing zero slope and zero lateral motion, only a trivial solution was found. Also at high heading angles and intermediate slopes, no valid fixed points were found.

Parameter	Value
k: Leg Stiffness	$1.5\,N/m$
m: Mass	$2.5\,g$ [31]
σ: Incline Slope	$0° \rightarrow 90°$
v_{des} Desired Velocity	$0.35\,m/s \rightarrow 0.20\,m/s$*
t_{des}: Desired Step Duration	$.0364\,s \rightarrow .0598\,s$ [21, 2]
δ_0: Heading Angle	0.00 rad $\rightarrow$ 0.45 rad*
$\beta_{0,i}$: Foot Placement Angle	F: $\pm 25°$ [24] M: $\mp 55°$ [24] B: $\pm 120°$ [24]
l_{0_i}: Nominal leg length	F: $0.032\,m$ [24] M: $0.026\,m$ [24] B: $0.019\,m$ [24]
l_{dev_i}: Leg Equilibrium Deviation	F: $0.016\,m$ M: $0.013\,m$ B: $0.0095\,m$

*D. Goldman, personal communication, Georgia Institute of Technology

Table 4.1: Selected parameters for reduced order model of cockroach motion based on actual cockroach parameters.

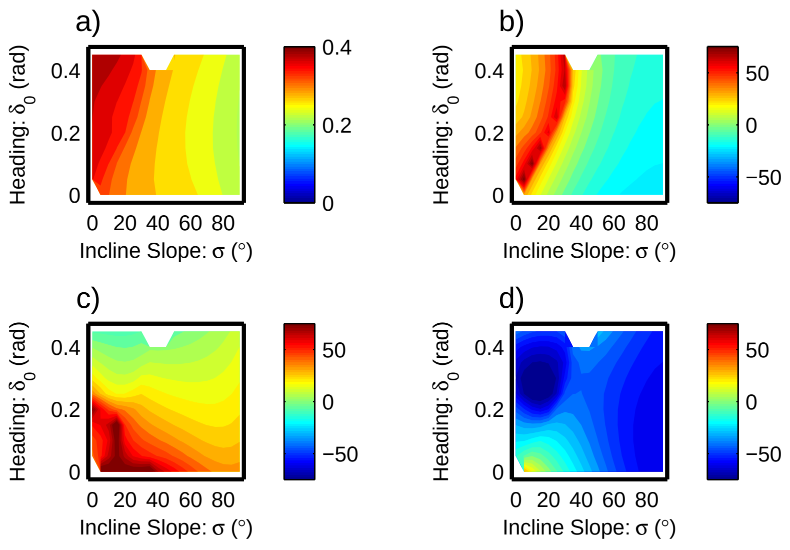

Figure 4.1: Contour plot of parameters resulting from fixed point solver as a function of incline slope and initial heading angle. High phase angles correspond to short durations of the given leg stance period.a) Initial velocity at foot placement (m/s); b)-d) Leg actuation phase shifts (rad) for the front, middle, and back legs

The variation in the initial velocity shown in the upper left graph of Figure 4.1 closely tracks the variation in the desired forward average velocity shown in Table 4.1. In the upper right corner, the graph indicates the variance of the front leg phase shift parameter. The dark red region representing high phase shifts indicates short leg force durations for the front leg (see Section 2.4). This region corresponds with a transition in leg function from pushing at low slopes, to pulling at high slopes. As the incline slope increases the leg pushing force decreases in duration and magnitude until both reach zero. Then continuing on, the leg force increases in duration and magnitude but in a pulling direction instead.

A similar, but less well defined region is shown in the lower left graph for the middle leg. The middle leg pushes only when both incline slope and initial heading angle are low. As either increases, this leg soon transitions to a pulling function. There is no transition for the back leg since all of these gaits use the back leg in a pushing manner. A large part of this graph shows highly negative phase shift angles (ϕ_B) indicating that the back leg tends to be active longer than the other legs (see Section 2.4).

Different types of motion can be produced depending on the direction of the leg force generated in each leg. For a more concise indication of when legs use pushing vs. pulling function, see Figure 4.2. Fixed point gaits at higher incline slopes all used only the back leg for pushing and the other legs for pulling. These gaits are represented by circles on the contour plot. While the back leg continues to push for all slopes, the middle leg (upward triangles), front leg (downward triangles), or both legs (squares) also push at lower slopes.

Gaits with only the back leg pushing are preferred for higher slopes since front or middle legs pushing would hinder the back legs ability to push against gravity in this model. This tendency to use the front and middle legs for pulling has been observed in cockroaches climbing up vertical slopes [2], while cockroaches on the level tend to use all legs for pushing [1]. On level ground and lower slopes the leg function used depends on the heading angle. For instance, gaits with high heading angles have more pronounced lateral motion and thus require greater lateral forces. A pulling middle right leg, combined with pushing front left and back left legs all apply the required lateral forces to the right during a left stance phase.

A similar actuated model of cockroach motion on variable slopes has shown that both pushing and pulling leg functions were possible for each fixed point. However, this was accomplished by using leg placements on the opposite side of the body (for instance on the right side for a left-stance phase) [8]. With fixed leg placement angles, the current model does not allow any overlap between pushing and pulling gaits.

The maximum resulting eigenvalues are plotted in Figure 4.2. Recalling that eigenvalues with a magnitude of less than one are stable, all of the fixed points were found to be stable. This makes these fixed points especially useful in designing a controller since any slope and heading angle can result in a gait that returns to similar states even in the presence of disturbances.

Gaits were more stable in the blue regions of the graph where the incline slope is high. It seems that the sprawled posture arrangement of the legs allows for a self stabilization especially in the presence of gravity. A similar self-stabilizing result was observed for a theoretical model and subsequent physical robot that used two legs deployed off to the side, but in front of the robot [5].

While all gaits were stable at lower slopes, the most stable gaits were found

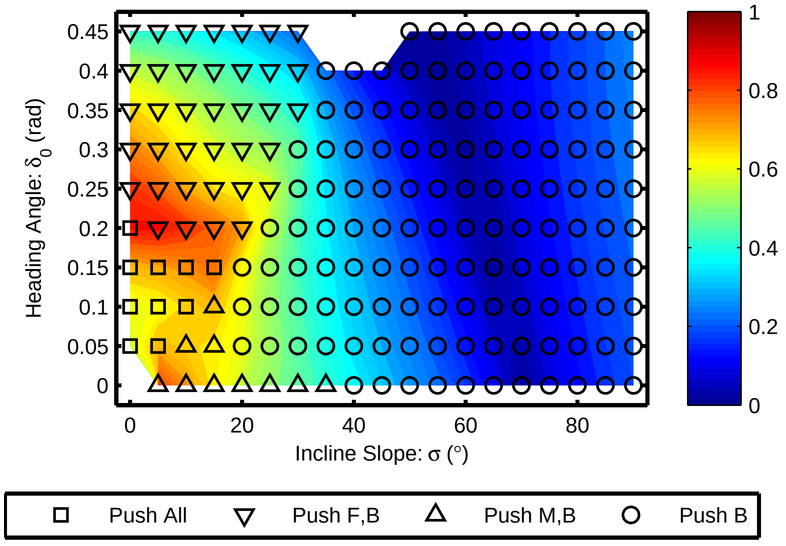

Figure 4.2: Stability of non-optimized fixed point gaits with leg actuation given by $l_{dev} = l_0/2$. Gaits are most stable where eigenvalues are close to zero, represented by the dark blue regions. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

for very large heading angles ($\delta_0 = 0.3 - 0.4$ rad) where the lateral motion is very high. Another stabler region appears at lower heading angles ($\delta_0 = 0.05 - 0.15$ rad). The least stable region appears to be at around $\delta_0 = 0.20$ rad. Here the gaits are transitioning between all legs pushing gaits (indicated by squares on the graph) and front and back legs pushing gaits (indicated by downward triangles). Since the middle leg is transitioning in function, its forces are smaller. Clearly, this middle leg is required to maximize the stability of the cockroach model. This result is not surprising since this leg acts on the opposite side of the body from the other two legs.

The robustness of the fixed points is shown in Figure 4.3. This figure shows the response of the fixed points to a lateral impulse equal to 50% of the cockroach momentum. Recovery in fewer step numbers corresponds to gaits that are robust.

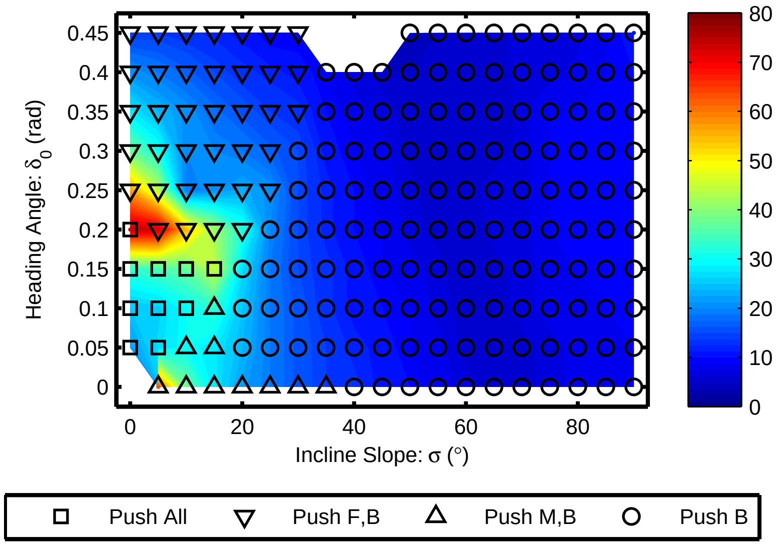

Figure 4.3: Robustness of non-optimized fixed point gaits with leg actuation given by $l_{dev} = l_0/2$. Gaits are most robust where recovery times are short, represented by the dark blue regions. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

Comparison with Figure 4.2 shows that stability and robustness behave similarly for this model. A gait with high stability is also likely to be highly robust, while a marginally stable gait is also marginally robust.

For the majority of gaits, the modeled cockroach is able to recover to the original gait within ten steps of a large lateral perturbation. This is especially surprising considering that the model does not include any active control and relies only on the passive dynamics of the system. Similar to the stability results, the longest recovery times occur at low slopes when the middle leg is transitioning between pushing and pulling. While the original gait is still recovered, it takes over seventy steps to do so. In this region, only the front and back legs show significant force. With both of these legs being placed to the same side of the modeled cockroach, it is harder to generate the required stabilizing forces.

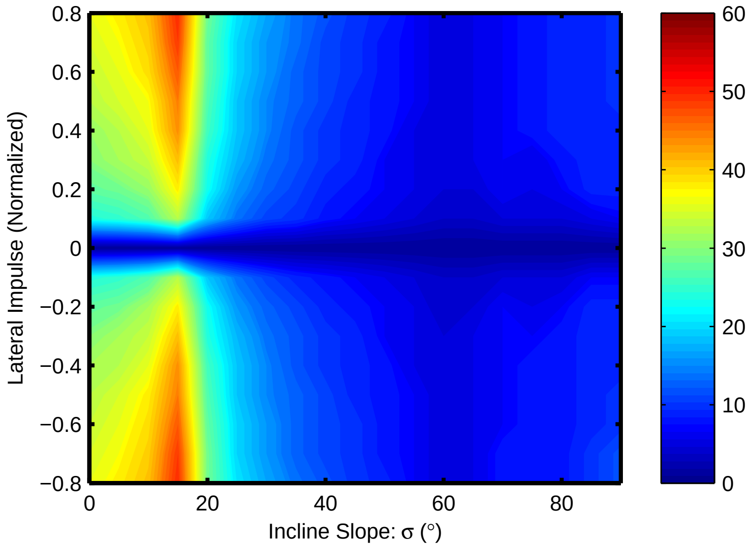

Figure 4.4: Robustness, as defined by number of steps to recover from a lateral impulse of non-optimized fixed point gaits with leg actuation given by $l_{dev} = l_0/2$. Modeled gaits operating at fixed points at the preferred heading angle ($\delta_0 = 0.15$ rad) as a function of incline slope and lateral impulse normalized as a percentage of forward momentum. Gaits are most robust where recovery times are short, represented by the dark blue regions.

Figure 4.4 shows the response of the model to varying magnitudes of lateral impulse. Only variations in slope are shown since the heading angle is fixed at $\delta_0 = 0.15$ rad consistent with the heading angle typically used by cockroaches at all slopes (D. Goldman, 2011, personal communication, Georgia Institute of Technology). The plotted lateral impulses are normalized by dividing by the cockroaches average forward momentum. As should be expected, large impulses generally cause the model to take longer to recover. All of the gaits at $\delta_0 = 0.15$ rad were able to recover from even the highest lateral impulses equal to 80% of the forward momentum.

For incline slopes grater than $\sigma = 20°$, recovery was exceptionally quick regardless of the magnitude of the lateral disturbance. At lower slopes, the time to recovery increased markedly with the magnitude of the disturbance. However, all were still able to eventually recover. The least robust gaits are found at incline slopes of $\sigma = 15°$. For the heading angle used in this plot ($\delta_0 = 0.15$) rad, this is approximately the region at which both the front and middle legs are simultaneously transitioning between pushing and pulling function (see Figure 4.3. This indicates that the back leg, being the primary active force, is not as capable of maintaining a robust gait as all three legs together would be.

The transport costs of all of the fixed points are shown in Figure 4.5. Recall that the transport cost is a representation of the work done to the cockroach by its legs, normalized by the weight of the cockroach and the forward distance traveled. On level ground, transport could be achieved with no work added or removed from the system corresponding to a transport cost of zero. Alternatively, on a vertical slope there must be at least enough work to raise the gravitational potential energy of the cockroach. This corresponds to a transport cost of one. On slopes in between, the ideal transport cost is given by:

$$C_{tran,ideal} = \sin(\sigma) \tag{4.1}$$

for an incline slope σ.

For gaits, where only the back leg pushes, the transport cost tracks the ideal transport cost exactly. This indicates that the legs do not generate negative power at any time. Any negative power would need to be counteracted by positive power somewhere else in the stride, both of which would raise the transport cost calculation above the ideal value. This means that in all cases, leg are contracting when the leg is pulling (as the front and middle legs do) and expanding when the leg is pushing (as the back legs do).

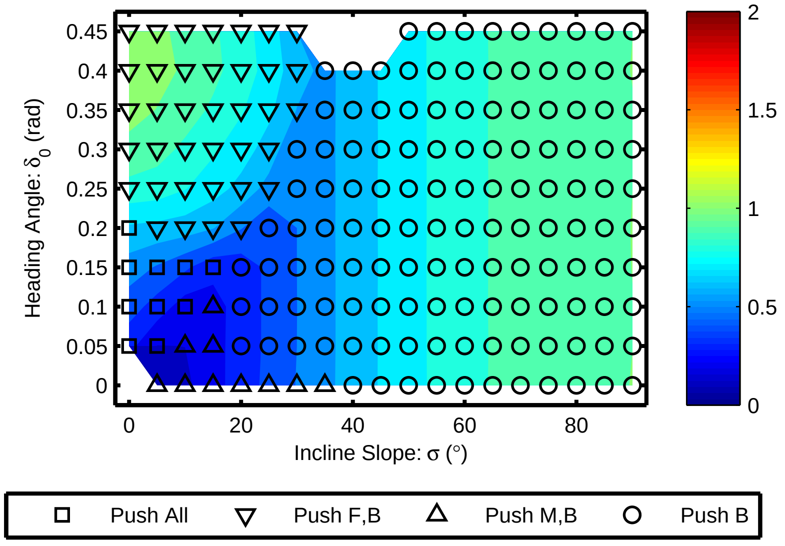

Figure 4.5: Normalized transport costs of non-optimized fixed point gaits with leg actuation given by $l_{dev} = l_0/2$. Smaller transport cost values correspond with more efficient gaits. The minimum possible transport cost can be found from $(C_{tran,min} = \sin\sigma)$. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

The other three categories of gaits do not show this ideal transport cost tracking. In general, the lowest transport costs for low incline slopes occur at low heading angles. As the heading angle increases, the transport cost goes up as well since more work is wasted in lateral motion. There is a marked increase in the transport cost as the mid leg transitions from pushing (gaits marked with squares and upward triangles) to pulling function (gaits marked with downward triangles). While Figure 4.2 shows that the gaits are more stable for higher heading angles, they do not have a lower transport cost. The fact that cockroaches tend to employ heading angles of about $\delta_0 = 0.15$ rad (D. Goldman, 2011, internal communication, Georgia Institute of Technology), seems to indicate that they select this gait based on minimizing transport cost rather than just maximizing stability. This location allows a low transport cost corresponding to low energy usage, while maintaining a very stable gait.

Chapter 5 – Opimization Results

In addition to the constant leg actuation gaits, fixed points were found and optimized for both stability and metabolic efficiency using the parameters in Table 4.1. As before, the incline slope was varied in increments of 5° from level ground to a vertical slope. Additionally, initial heading angles (δ_0) varied as shown from 0 rad to 0.45 rad in increments of 0.05 rad for each slope. The leg actuation parameters (l_{dev}) were varied to find the optimal values for stability and efficiency.

Previously for the non-optimized gaits, this model had developed fixed points that fall into distinct regions based on whether each leg pulls or pushes. This resulted in a unique fixed point solution for each combination of incline slope and heading angle. On the other hand, when the leg deviations are varied to optimize either stability or efficiency, multiple solutions can exist. Optimized fixed points were found based on four different leg functional combination families. These four families are: all legs pushing; front and back legs pushing while middle leg pulls; middle and back legs pushing while front leg pulls; and back legs pushing while front and middle legs pull. No gaits were found with the back leg used for pulling. The regions where fixed points were found for each of the families are shown in Figure 5.1.

Gaits where only the back leg push are found over the widest range of incline slopes and heading angles among the leg functional families. In fact they only do not exist for flat or nearly flat slopes. The middle and back leg pushing gaits are similarly prevalent for higher slopes, but are not as easy to find at higher initial heading angles. In order to climb slopes, the cockroaches need to use both the

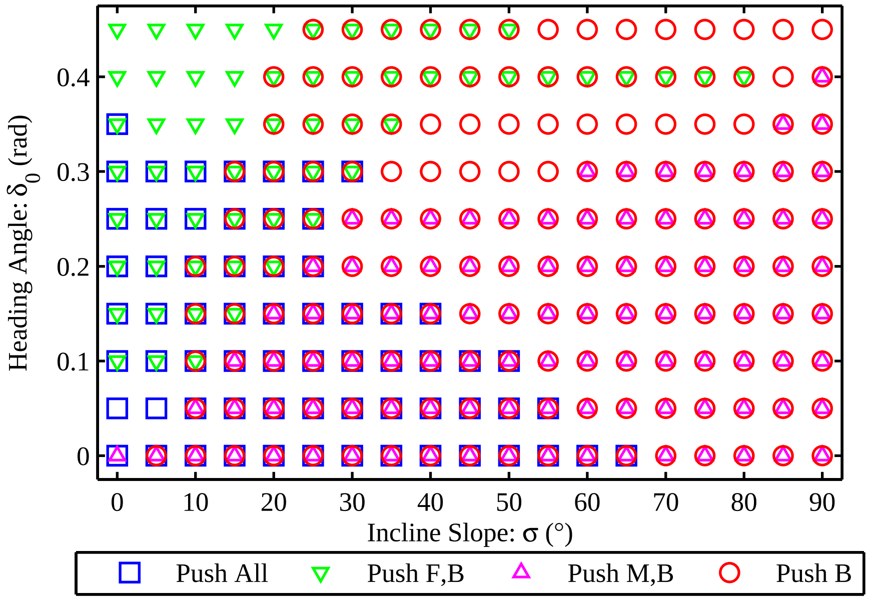

Figure 5.1: Existence of fixed points at various heading angles, δ_0 and incline slopes, σ organized by pushing or pulling function of each leg. Because of the flexibility in leg actuation parameters (l_{dev}), there is some overlap between gait existence of different leg functional sets.

front and back legs to generate positive vertical force, which means that the front legs must pull for these higher slope fixed points.

With all legs pushing, the fixed points are generally found only for low incline slopes and low heading angles. On the other hand, the front and back leg pushing gaits are more easily found for high heading angles and low incline slopes. High initial heading angles are synonymous with gaits that have high lateral motion (see Figure 3.1). This high lateral motion can only be achieved when all legs act in the same horizontal direction, which is true of the front and back leg pushing but middle leg pulling gaits since the middle leg of the tripod is on the opposite side of the body from the other legs.

There are a number of initial heading and incline slope combinations where three different fixed points are possible depending on leg function. However, in no region can all four leg functional families be used.

5.1 Stability Optimized Gaits

When optimizing based on stability, fixed points are found with maximum eigenvalues as small as possible, subject to the constraints described in Section 3.5. For each leg functional family, the resulting leg actuation parameters are shown in Figures 5.2, 5.3, and 5.4 for the front, middle, and back legs respectively. In general, higher leg actuation parameters correspond to more actuation in the given leg (however, when the leg also has a very high phase shift the leg is not actuated for long enough to have a significant effect on the motion of the cockroach). Regions where leg actuations are high are shown in red, while less actuated regions are shown in blue.

Not surprisingly, the front leg shows significantly more actuation as the slope

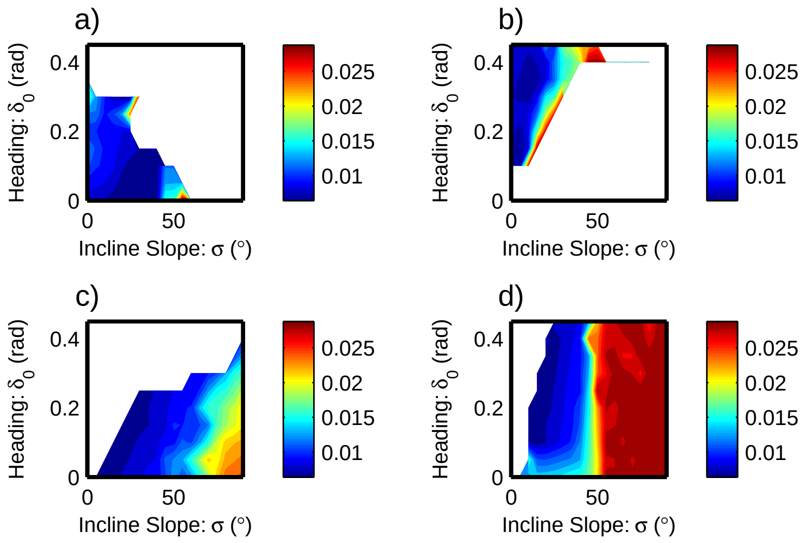

Figure 5.2: Contour plot of front leg actuation parameter ($l_{dev,F}$) from fixed point solver, optimized for stability. The front leg is generally more strongly actuated as slope increases. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

increases since more energy is required to ascend the slope (See Figure 5.2). In fact, for the gaits where only the back leg pushes, the front leg actuation parameter reaches its maximum value of $90\% l_{dev}$ at a slope of around $\sigma = 50°$.

A similar trend of actuation increasing with slope is evident in Figure 5.3 for the middle leg. Further, Figures b) and d) show that the middle leg increases actuation with increasing heading angle when it is used for pulling. This is because the increased heading angle results in increased lateral motion which requires higher lateral forces, specifically from the middle leg. Also of note is that for these stability optimized gaits, the middle leg is used as little as possible for the all legs pushing case (Figure a)) regardless of heading angle or incline slope.

The back leg actuation parameters resulting from the stability optimization

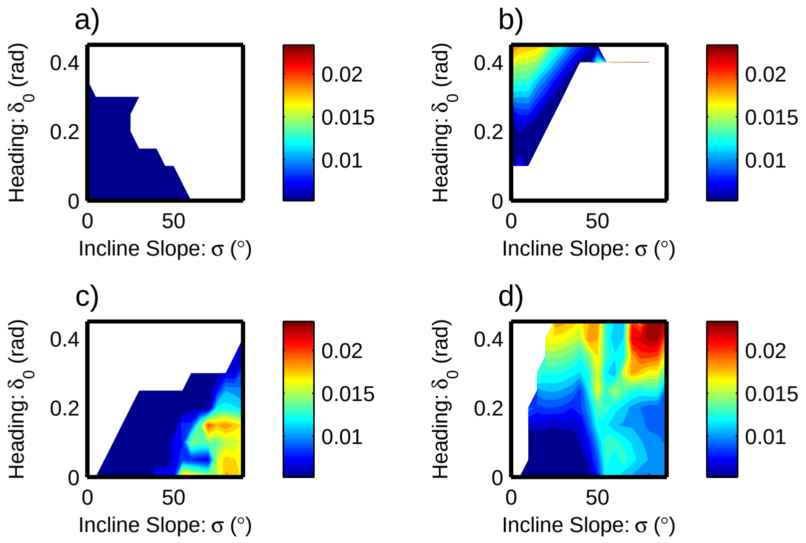

Figure 5.3: Contour plot of middle leg actuation parameter ($l_{dev,M}$) from fixed point solver, optimized for stability. The middle leg is generally not strongly actuated at low slopes unless the leg is pulling and the heading angle is high. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

are shown in Figure 5.4. As with the other two legs, the actuation increases with increasing slope. Also, there is an increase in back leg actuation for increasing heading angle in cases where the middle leg is pushing (Figures a) and c)). It appears that the back leg, as opposed to the front leg, is the preferred way to increase the lateral motion when the middle leg pushes against this lateral motion.

The resulting phase shifts from the stability optimized fixed points are shown in Figures 5.5, 5.6, and 5.7 for the front, middle, and back legs respectively. Large positive phase shifts generally correspond to short stance durations because the start of the leg actuation function is moved close to its peak value. These regions are shown in red on the contour plots. On the other hand, large negative phase shifts generally correspond to long stance durations because the leg actuation function is moved backward from the peak value (see Section 2.4). These regions

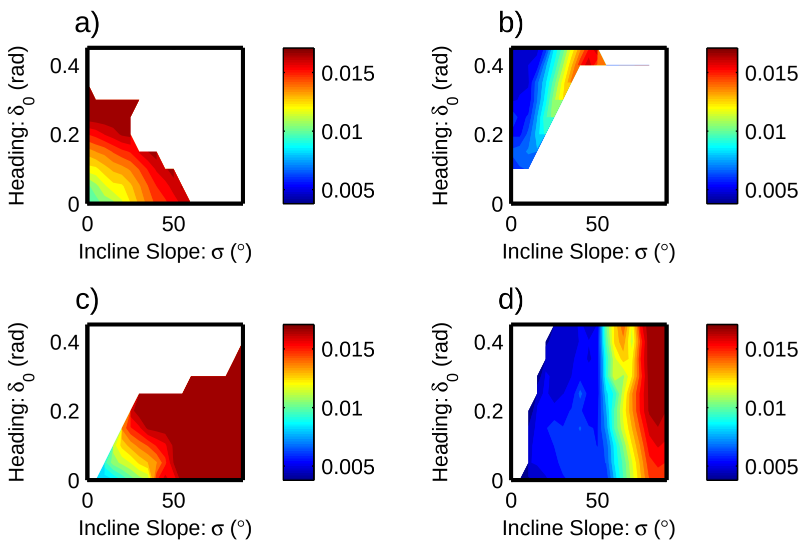

Figure 5.4: Contour plot of back leg actuation parameter ($l_{dev,B}$) from fixed point solver, optimized for stability. The back leg increases in actuation as either heading angle or incline slope increase. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

are shown in blue on the contour plots. For phase shift values near zero, the stance phase is generally close to the desired step duration (t_{des}).

As seen in Figure 5.5, the optimally stable gaits with the front legs used for pushing (Figures a) and b)) have a high phase shift, implying a short stance duration. This is particularly true as the slope increases beyond flat ground. This occurs because the modeled cockroach cannot easily ascend slopes when the front leg is pushing against the motion. Unlike the non-optimized gaits of Chapter 4, high phase shifts do not always correspond to regions where a leg is transitioning between pushing and pulling, since these regions overlap for the optimized case.

The resulting phase shifts for the middle leg in Figure 5.6 indicate a particularly high phase shift for the middle and back leg pushing gaits (Figure c)). Additionally, the phase shift is high for back leg pushing gaits (Figure d)) at high

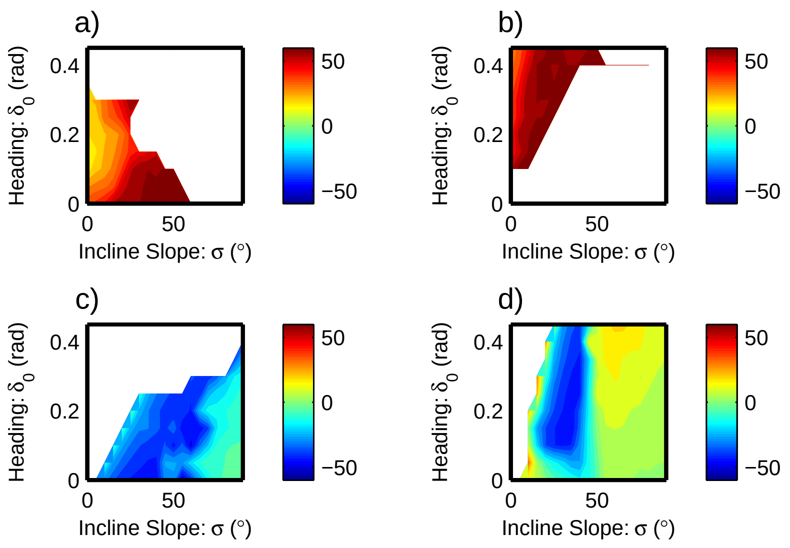

Figure 5.5: Contour plot of front leg phase shifts (ϕ_F °) from fixed point solver, optimized for stability. High phase angles correspond to short durations for the front leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

slopes. Regardless of the leg actuation parameters exhibited, these high phase shifts represent a short stance duration for the leg indicating that the leg is not effective for very long.

On the other hand, the phase shifts of the back leg when optimized for stable gaits are almost entirely negative (see Figure 5.7). This indicates that the back leg is used for a longer stance duration regardless of incline slope, heading angle, or leg function of the other two legs.

Recall from Section 3.5 that optimizing for stability primarily involves minimizing the eigenvalues of the Jacobian associated with the discretized system. The resulting minimized values are shown in Figure 5.8. Since all values are less than one, the model is stable regardless of incline slope, heading angles, or leg function. This is not surprising since all gaits were stable before the gaits were

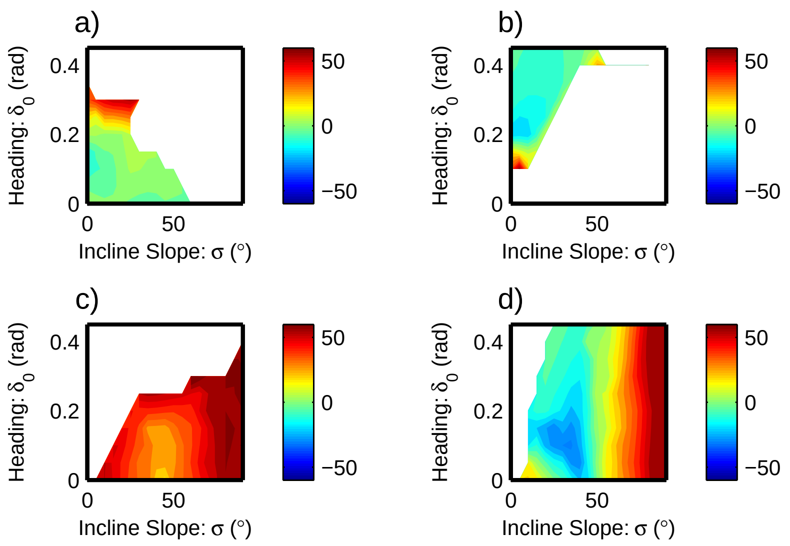

Figure 5.6: Contour plot of middle leg phase shifts (ϕ_M °) from fixed point solver, optimized for stability. High phase angles correspond to short durations for the middle leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

optimized for stability. The most stable regions are shown in blue where the maximum eigenvalue is close to zero, while the red regions indicate areas that are stable but only marginally so.

It is interesting to note that the gaits with a pulling front leg (Figures 5.8 c) and d)) show ideal stability everywhere except for low slopes. This suggests that the pulling front leg serves as a passive aid to stability for the cockroach on slopes. A similar self-stabilizing result was observed for a theoretical model and subsequent physical robot that used two legs deployed off to the side, but in front of the robot [5].

Fixed point gaits that use the front leg in a pushing manner, have stability characteristics that are highly dependent on the initial heading angle. When all legs are pushing, as in Figure 5.8 a), stability decreases as the initial heading angle

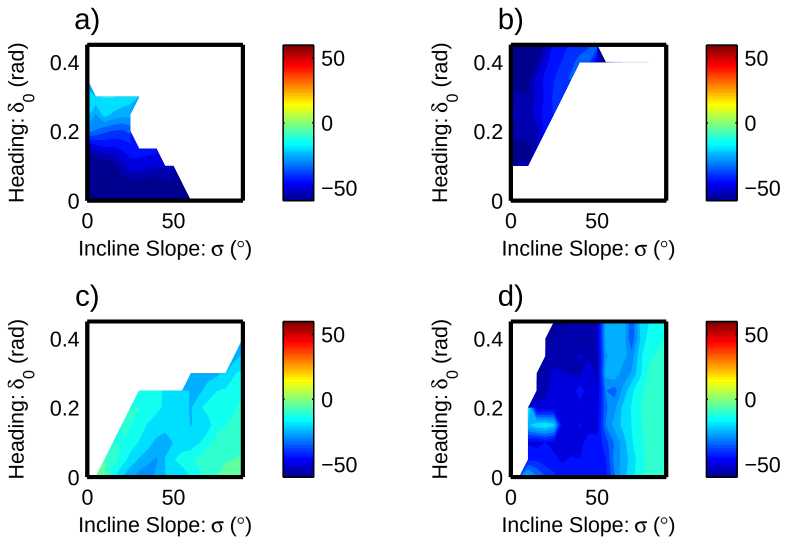

Figure 5.7: Contour plot of back leg phase shifts (ϕ_B °) from fixed point solver, optimized for stability. High phase angles correspond to short durations for the back leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

increases. On the other hand, when only the front and back legs are pushing, as in Figure 5.8 b), stability increases as the initial heading angle increases. On low slopes, it is preferred to use the middle leg for pushing at low heading angles and for pulling at high slopes from a stability standpoint. High initial heading angles are associated with a large degree of lateral motion, which works better when all legs are acting in the same lateral direction. This is true of the front and back leg pushing with middle leg pulling gaits since the middle leg is on the opposite side of the body as the other two.

The next step in the stability optimization is to select between the available leg functional families for each combination of heading angle and incline slope. The results of this selection are shown in Figure 5.9. As explained in the legend, the selection at each heading angle and incline slope is shown with a marker on

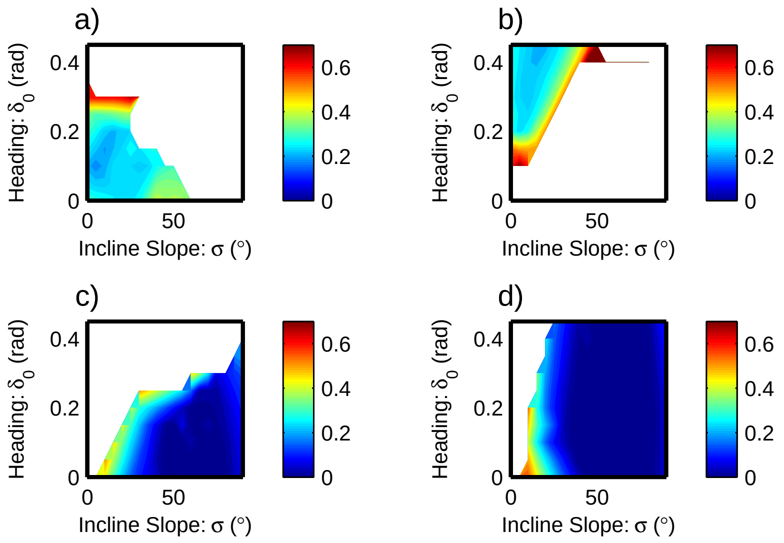

Figure 5.8: Optimal stability of fixed point gaits based on varying leg actuation (l_{dev}). Gaits are most stable where eigenvalues are close to zero, represented by the dark blue regions. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

the plot. For incline slopes of $\sigma = 20°$ and above, nearly all of the selected gaits use only the back leg for pushing. At lower incline slopes, the selected gait family depends on heading angle, with the all legs pushing gaits selected for heading angles less than $\delta_0 = 0.2$ rad. Higher heading angles use gaits with only the front and back legs pushing. The middle and back legs pushing gaits are not very often selected since they generally are stable in the same region as the back legs only pushing gaits which are even more stable.

While these gaits were optimized based on stability rather than efficiency, the transport cost was also calculated. The resulting transport costs are shown in Figure 5.10 for each leg functional family and in Figure 5.11 for the final selected gaits. In both cases, the minimum possible transport cost is simply the energetic cost of lifting the modeled cockroach up the slope given in Eq. (4.1).

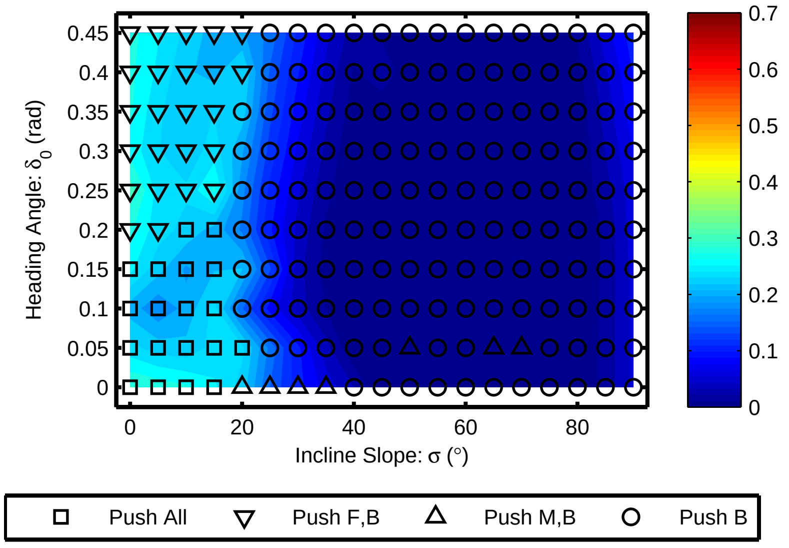

Figure 5.9: Optimal stability of fixed point gaits based on varying leg actuation (l_{dev}). Gaits are most stable where eigenvalues are close to zero, represented by the dark blue regions. Gaits selected based on most stable leg function combination.

Although these gaits were optimized for efficiency, some of the gaits exhibit both the lowest possible eigenvalues and the lowest possible transport costs. This is primarily true for the back leg only pushing gaits. As was discussed in Chapter 4, the back leg pushing gaits are optimally efficient since no negative work is done by any of the legs throughout the stride. This indicates that an optimization based on metabolic efficiency should result in gaits similar to the stability optimized case for the back leg pushing gaits.

5.2 Efficiency Optimized Gaits

A set of fixed point gaits was also found over the same range of incline slopes and initial heading angles by optimizing for metabolic efficiency as measured by minimizing the transport cost. The resulting leg actuation parameters are shown

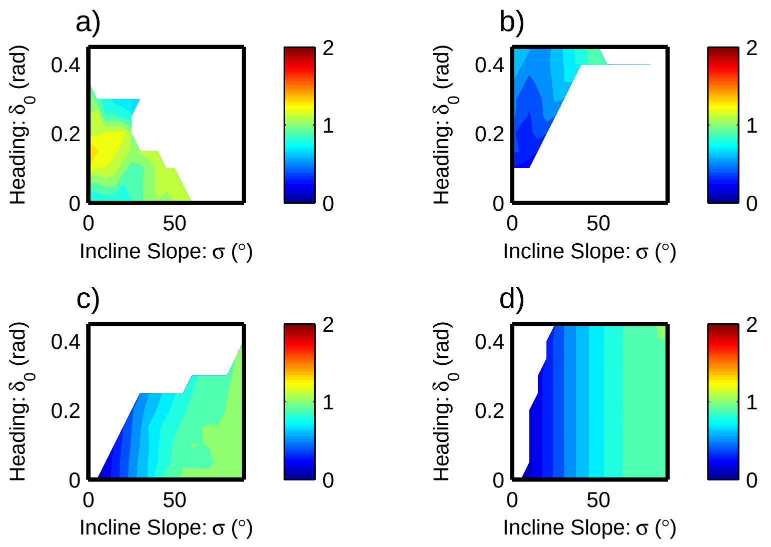

Figure 5.10: Normalized transport costs of fixed points, optimized for stability. Smaller transport cost values correspond with more efficient gaits. The minimum possible transport cost can be found from ($C_{tran,min} = \sin\sigma$). a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

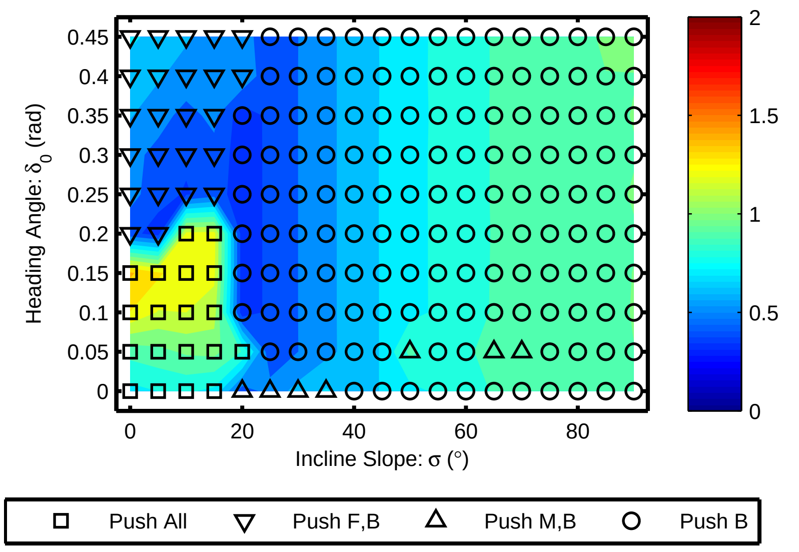

Figure 5.11: Normalized transport costs of fixed points, optimized for stability. Smaller transport cost values correspond with more efficient gaits. The minimum possible transport cost can be found from ($C_{tran,min} = \sin\sigma$). Gaits selected based on most stable leg function combination.

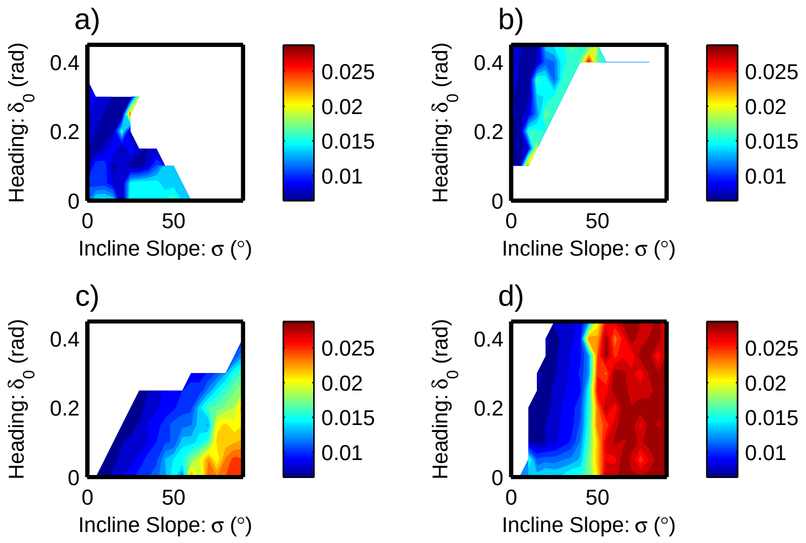

Figure 5.12: Contour plot of front leg actuation parameter ($l_{dev,F}$) from fixed point solver, optimized for metabolic efficiency. Trends are similar to those of the stability optimized gaits. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

in Figures 5.12, 5.13, and 5.14 for the front, middle, and back legs respectively.

These leg actuation values show similar trends to the leg actuation parameters from the stability optimized cases, but the two cases are not numerically equivalent in most cases. For instance the front leg is generally slightly more actuated for the efficiency optimized cases (Figure 5.12) as opposed to those optimized based on stability (Figure 5.2). Also, the middle leg shows some areas above the minimum actuation level in Figure 5.13 which were not present in Figure 5.3. The back leg shows a reduction in leg actuation for the transport cost optimized gaits (Figure 5.14) versus the stability optimized gaits (Figure 5.4). This suggests that optimizing for metabolic efficiency gives a slight preference to using the front leg, while the stability optimized gaits slightly prefer to use the back leg.

Figures 5.15, 5.16, and 5.17 are contour plots of the phase shifts for the

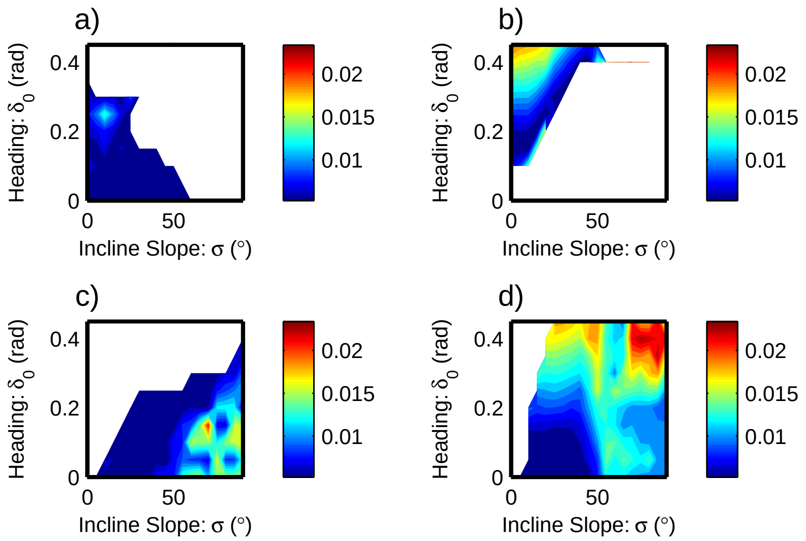

Figure 5.13: Contour plot of middle leg actuation parameter ($l_{dev,M}$) from fixed point solver, optimized for metabolic efficiency. Trends are similar to those of the stability optimized gaits. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

front, middle, and back legs respectively, as found from a metabolic efficiency optimization. The front leg phase shifts in Figure 5.15 a) and b) show regions of high phase shift for front leg pushing gaits similar to those of the eigenvalue optimized gaits. However, the effect is even more pronounced in the low slope regions of the all legs pushing gaits (Figure a)).

When optimizing for metabolic efficiency, the middle leg shows larger regions of high phase shift for the middle leg pushing gaits (Figures 5.16 a) and c)) as opposed to the stability optimized gaits (Figure 5.6). This is especially pronounced in the lower slope regions. This suggests that the middle leg is used for very short durations at low slopes when the middle leg is pushing and gaits are optimized for efficiency. This is not surprising since a pushing middle leg works against the pushing back leg, which causes more muscle work to be wasted. The phase shifts

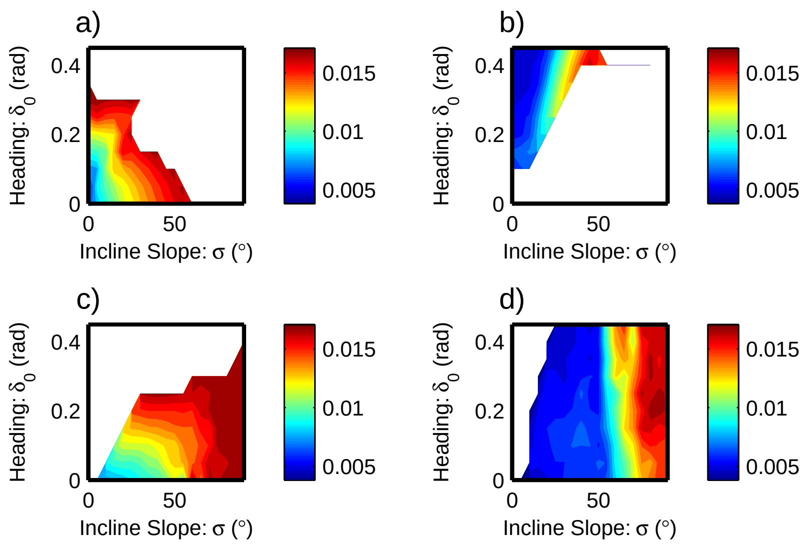

Figure 5.14: Contour plot of back leg actuation parameter ($l_{dev,B}$) from fixed point solver, optimized for metabolic efficiency. Trends are similar to those of the stability optimized gaits. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

shown in Figure 5.17 are similar to those of the stability optimized gaits in Figure 5.7.

When optimizing for metabolic efficiency, the indicative parameter is the transport cost. The resulting minimum transport costs are shown by contour plot in Figure 5.18. As expected, the back leg only pushing gaits (Figure d)) continue to match the lowest possible transport cost as given by Eq. (4.1). A transport cost optimization is not strictly necessary for these gaits since it is known that the back leg only pushing gaits never do negative work regardless of the leg actuation parameters. The middle and back leg pushing gaits (Figure c)) are also very close to the minimum values after the efficiency optimization.

For both the all legs pushing gaits and the front and back legs pushing gaits (Figures 5.18 a) and b)) the efficiency is dependent on both slope and heading

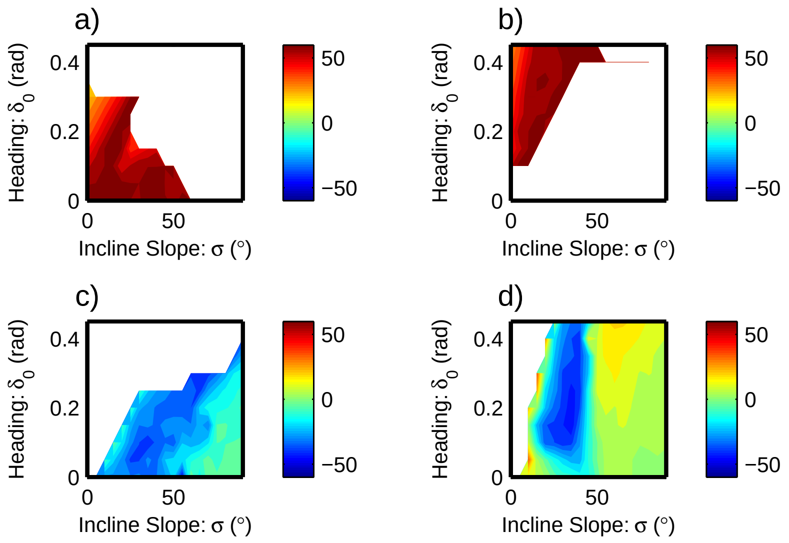

Figure 5.15: Contour plot of front leg phase shifts (ϕ_F °) from fixed point solver, optimized for metabolic efficiency. High phase angles correspond to short durations for the front leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

angles. Increases in either parameter increase the transport cost, thus decreasing the efficiency. When comparing between the two for a common heading angle, the gaits with the middle leg pulling generally have a preferable transport cost.

Figure 5.19 shows the results of selecting the optimally efficient leg functional family for each combination of incline slope and heading angle. As was also true of the stability optimized gaits in Figure 5.9, the back leg pushing gaits are generally preferred at all but the low slopes. At low slopes, the all legs pushing gaits are preferred for lower incline angles with front and back leg pushing gaits preferred at higher incline angles. Again, the middle and back leg pushing gaits are generally not preferred anywhere. However with the efficiency optimized gaits, the front and back leg pushing gaits are preferred down to a heading angle of $\delta_0 = 0.15$ rad instead of 0.25 rad. Also, the back leg pushing gaits are preferred

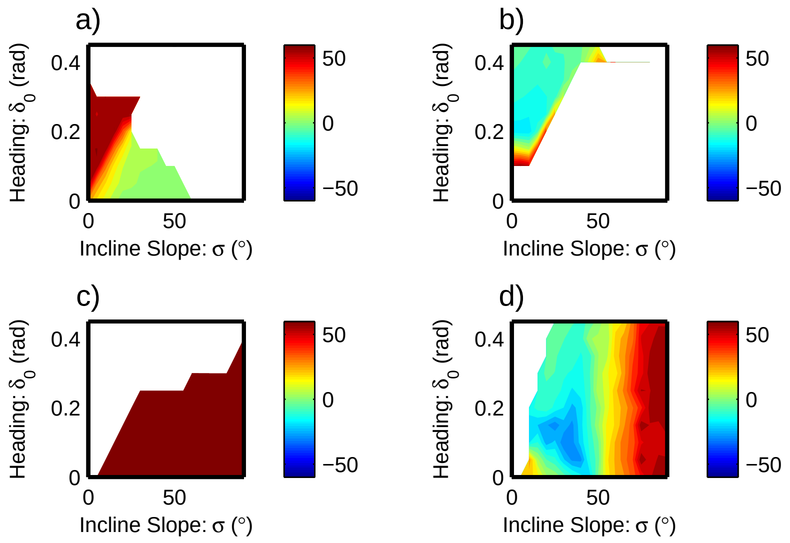

Figure 5.16: Contour plot of middle leg phase shifts (ϕ_M °) from fixed point solver, optimized for metabolic efficiency. High phase angles correspond to short durations for the middle leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

at even lower slopes than was true of the stability optimized gaits.

The resulting eigenvalues of the metabolic efficiency optimized gaits are shown in Figure 5.20 for each leg functional family and Figure 5.21 for the selection between leg functional families. In comparing Figures 5.20 and 5.8, there is no noticeable difference between the resulting eigenvalues between the efficiency optimized gaits and the stability optimized gaits when the middle leg pulls (Figures b) and d)). This means that both optimizations result in similar gaits for these leg functional families. This result is confirmed in comparing the optimized transport costs of Figures 5.18 b) and d) to Figures 5.10 b) and d).

There is some reduction in stability for the middle and back leg pushing gaits when they are optimized for efficiency (Figure 5.20 c)) instead of stability (Figure 5.8 c)) at low slopes. However, this is not an important difference since this gait

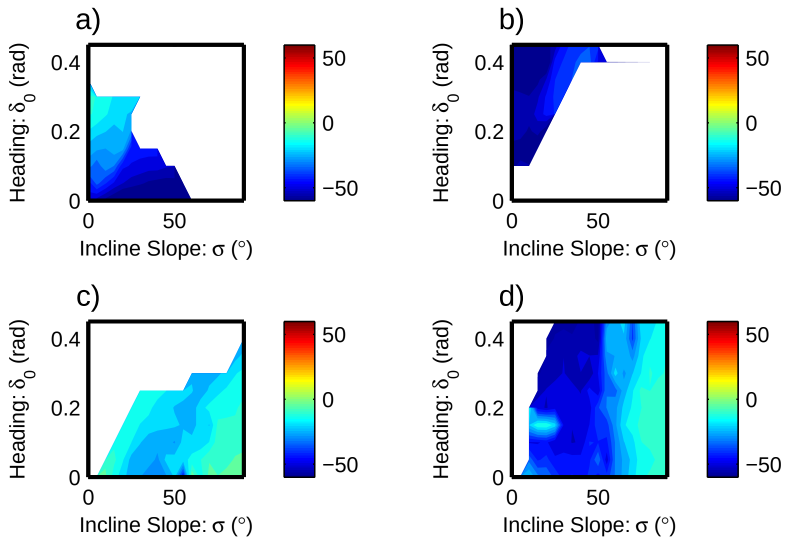

Figure 5.17: Contour plot of back leg phase shifts (ϕ_B °) from fixed point solver, optimized for metabolic efficiency. High phase angles correspond to short durations for the back leg stance period. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

functional family is not preferred based on either optimization scheme.

There is a significant discrepancy between the eigenvalues for the different optimization scheme for the all legs pushing gaits at low slope. Optimizing for stability results in maximum eigenvalues of around 0.3 at low slopes (See Figure 5.8 a)), but results in high transport costs of around 1.2 (See Figure 5.10 a)). However, optimizing for efficiency gives much higher maximum eigenvalues of around 0.65 (See Figure 5.20 a)) along with the improved transport costs of around 0.6 (See Figure 5.18 a)). This suggests that the two optimization routines give differing results and different gaits could be selected depending on which optimization scheme was being used in this region.

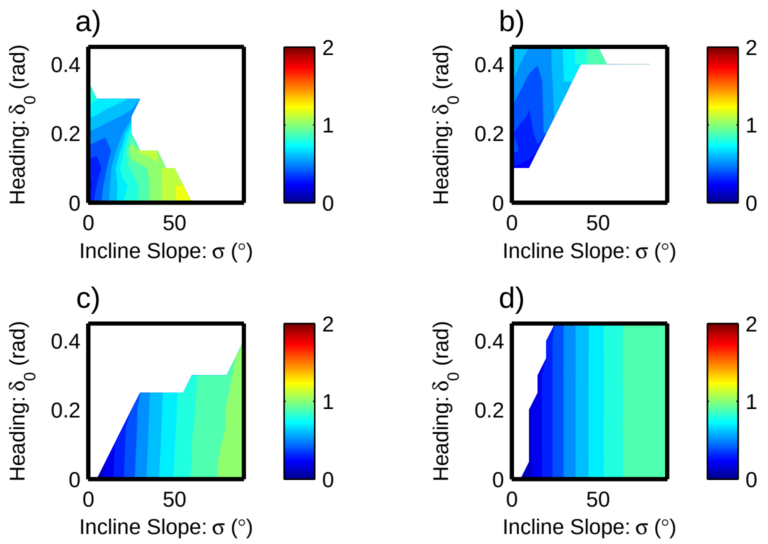

Figure 5.18: Optimal transport costs based on varying leg actuation (l_{dev}). Smaller transport cost values correspond with more efficient gaits. The minimum possible transport cost can be found from ($C_{tran,min} = \sin\sigma$). a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

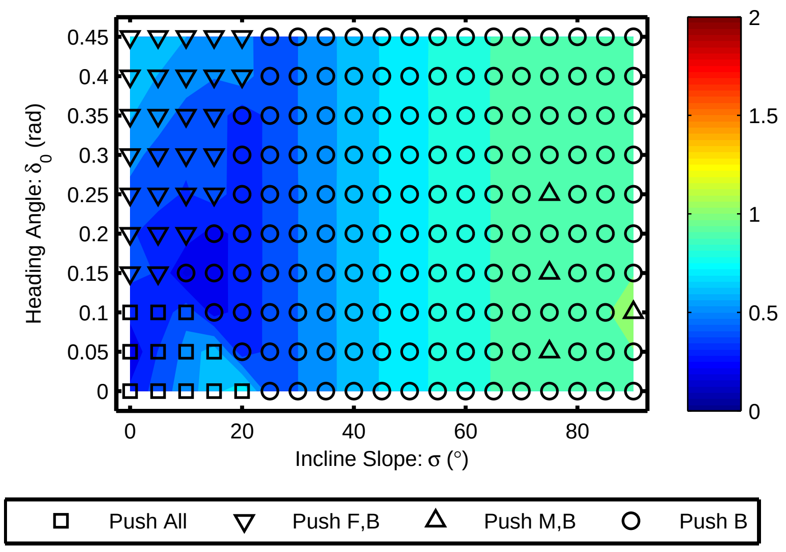

Figure 5.19: Optimal transport costs based on varying leg actuation (l_{dev}). Smaller transport cost values correspond with more efficient gaits. The minimum possible transport cost can be found from ($C_{tran,min} = \sin\sigma$). Gaits selected based on most efficient leg function combination.

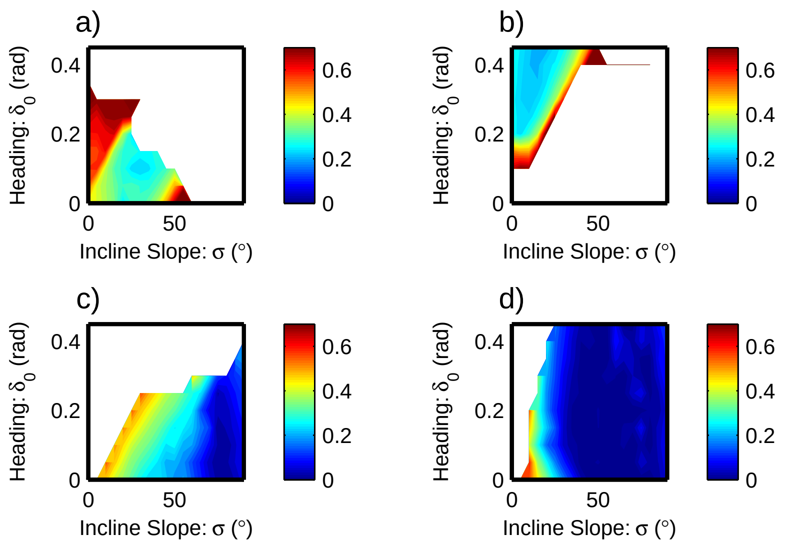

Figure 5.20: Stability of fixed point gaits, optimized for metabolic efficiency. . Gaits are most stable where eigenvalues are close to zero, represented by the dark blue regions. a) Gaits with all legs pushing; b) Gaits with front and back legs pushing; c) Gaits with middle and back legs pushing; d) Gaits with back leg pushing.

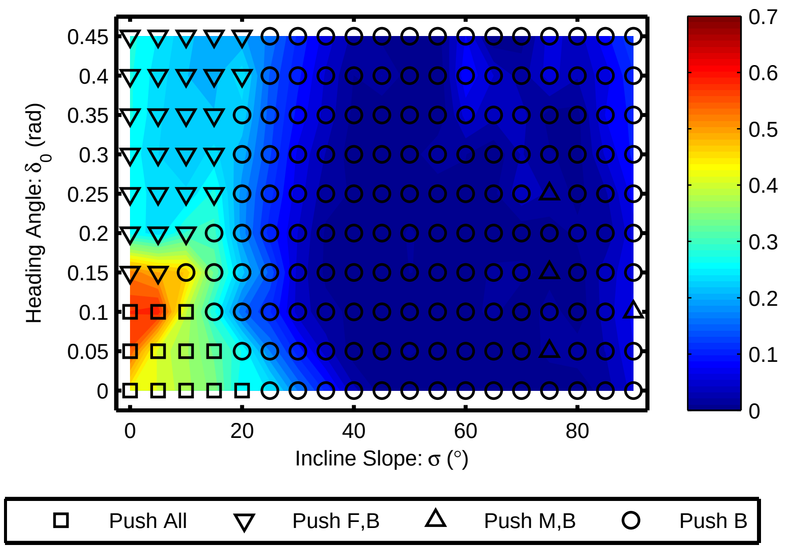

Figure 5.21: Stability of fixed point gaits, optimized for metabolic efficiency. Gaits are most stable where eigenvalues are close to zero, represented by the dark blue regions. Gaits selected based on most efficient leg function combination.

Chapter 6 – Force Comparisons

Figures 6.1 and 6.2 show the range of peak forces in the lateral and fore/aft direction respectively. These were generated from observations of cockroaches running at their preferred speed. The data for varied slopes is based on preliminary unpublished research by Dr. Daniel Goldman at Georgia Institute of Technology. For this research, data has not yet been taken for the middle leg forces. These forces are harder to measure since it is more difficult to isolate these legs on a force plate.

All measured forces are based on a left stance phase as illustrated in Figure 2.1. Lateral forces are measured positive to the right, while fore/aft forces are measure positive forward. This means that the front leg should have a positive lateral force and a negative fore/aft force when pushing. Similarly, the middle leg should have a negative lateral force and fore/aft force when pushing. Finally, the back leg should have a positive lateral force and fore/aft force when pushing. For pulling gaits the opposite conditions would be true.

These graphs indicate that the cockroach uses all legs in a pushing manner when on a level slope. When climbing a vertical slope however, the front and middle legs are used for pulling, while the data on the back leg is inconclusive. The measured fore/aft forces strongly indicate that the back leg is pushing on vertical slopes as well, but the lateral forces are too close to zero to make any definitive conclusion. Until more data is taken it is difficult to say at what slope the transition between pushing and pulling occurs for the front and middle legs. At least for the front leg it appears to happen some time around a slope of

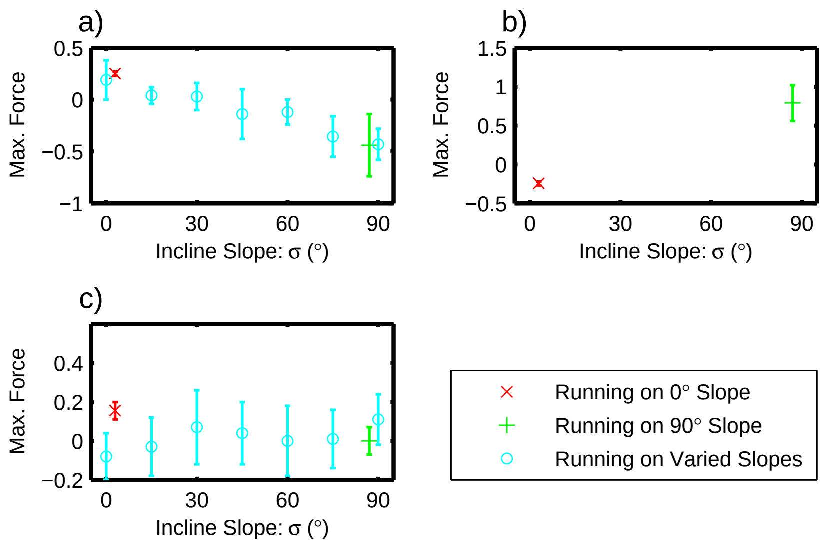

Figure 6.1: Experimental maximum lateral forces of cockroaches obtained from force plates. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle leg; c) Back leg. $\sigma = 0°$ data [1]; $\sigma = 90°$ data [2]; Variable slope data from D. Goldman, 2011, personal communication, Georgia Institute of Technology

$\sigma = 15° \rightarrow 30°$.

Since this model is an idealized version of the real cockroach motion it is not expected to match experimental forces exactly. However, a qualitative match between the experimental model and the measured data would provide some validation of the model. Of particular interest, is whether the model predicts leg transitions between pushing and pulling at similar slopes to those observed in real cockroaches.

For comparison, the lateral and fore/aft forces of the fixed points from the simulated cockroach model are shown as contour plots in Figures 6.3 and 6.4. The transition from pushing to pulling function for the function is visible on both graphs as a rapid change in color ranging from around 10° to 30° with the transition moving higher as the heading angle increases. Based on the upper right

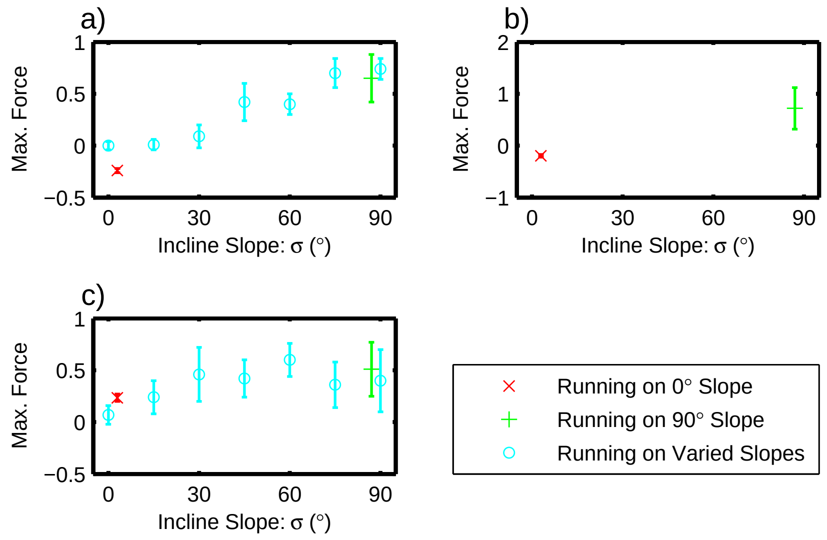

Figure 6.2: Experimental maximum lateral forces of cockroaches obtained from force plates. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle leg; c) Back leg. $\sigma = 0°$ data [1]; $\sigma = 90°$ data [2]; Variable slope data from D. Goldman, 2011, personal communication, Georgia Institute of Technology

graphs, the middle leg transitions at varied slope levels. For low initial heading angles it occurs around 35° while for higher heading angles the leg pulls at all slopes. The exact transition levels are shown using the leg functional gait family symbols plotted in Figures 4.2, 4.3, and 4.5. The back leg forces in the Figures c) are fairly uniform regardless of heading angle or incline slope.

Lateral and fore/aft forces were also calculated from the modeled motion of the fixed points optimized for stability. These are shown in Figures 6.5 and 6.6 respectively. As is also shown more concisely in Figure 5.9, this data shows that the front leg is pushing for slopes less than $\sigma = 20°$, but pulling for higher slopes. The middle leg also pulls for slopes higher than $\sigma = 20°$, but pushes at low slopes only for initial heading angles of less than $\delta_0 = 0.20$ rad while continuing to pull at higher heading angles. The back leg pushes for all combinations of incline

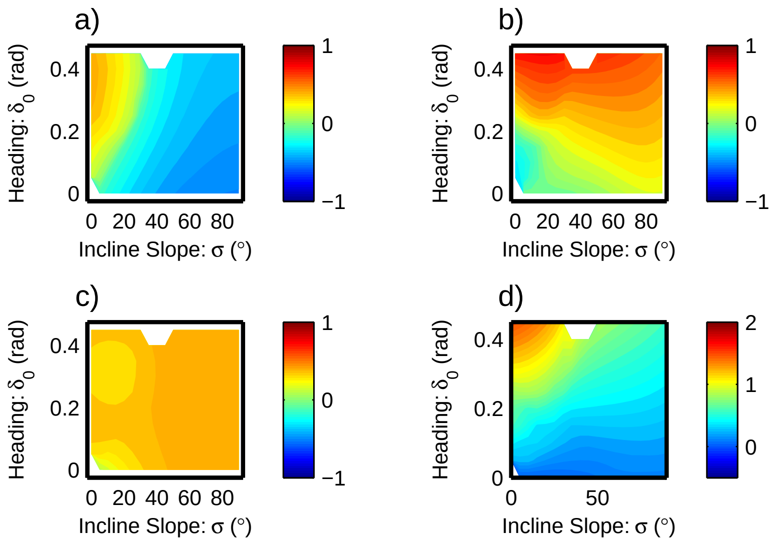

Figure 6.3: Maximum lateral forces of modeled cockroaches based on gaits with leg actuation held constant at $l_{dev} = l_0/2$. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

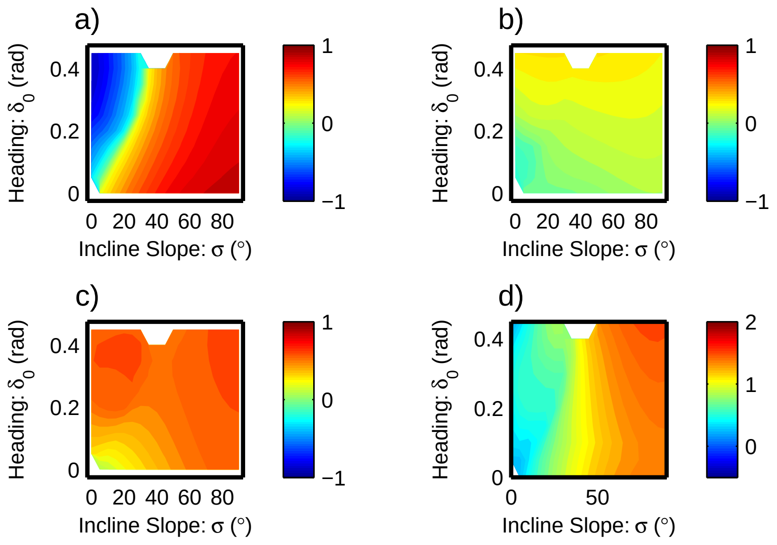

Figure 6.4: Maximum fore/aft forces of modeled cockroaches based on gaits with leg actuation held constant at $l_{dev} = l_0/2$. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

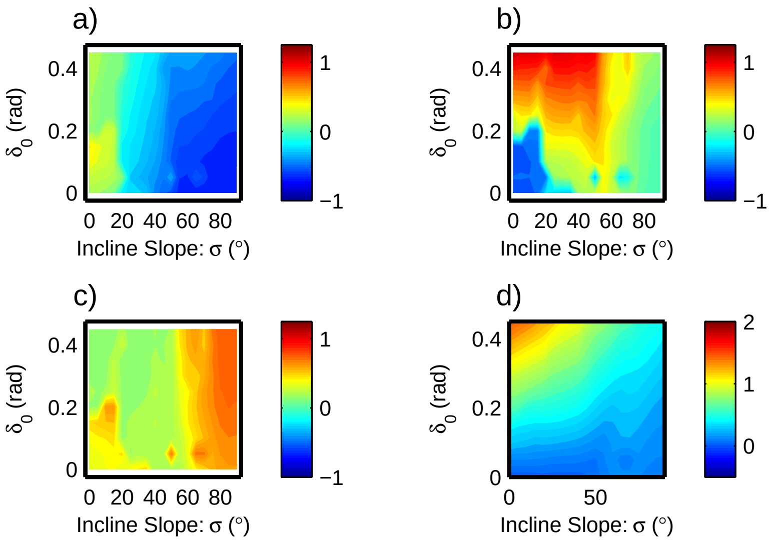

Figure 6.5: Maximum lateral forces of modeled cockroaches based on optimal stability gaits. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

slopes and heading angles.

A similar analysis was done on the gaits optimized for metabolic efficiency. Figures 6.7 and 6.8 represent the lateral and fore/aft forces respectively. Along with Figure 5.19, this data indicates a front leg pushing function for slopes less than $\sigma = 10 \rightarrow 20^\circ$, depending on heading angle, and a pulling function for higher slopes. The middle leg also is used for pulling at most incline slopes and heading angles except for those less than $\sigma = 20^\circ$ and $\delta_0 = 0.10$ rad. The back leg again pushes for all combinations of incline slopes and heading angles.

For both stability optimized gaits and gaits optimized to metabolic efficiency, the transitions between pushing and pulling function are similar to those evidenced by real cockroaches on variable slopes. However, more accurate data from the physical cockroaches is needed to verify this conclusion. This is especially true for the middle leg for which little data exists.

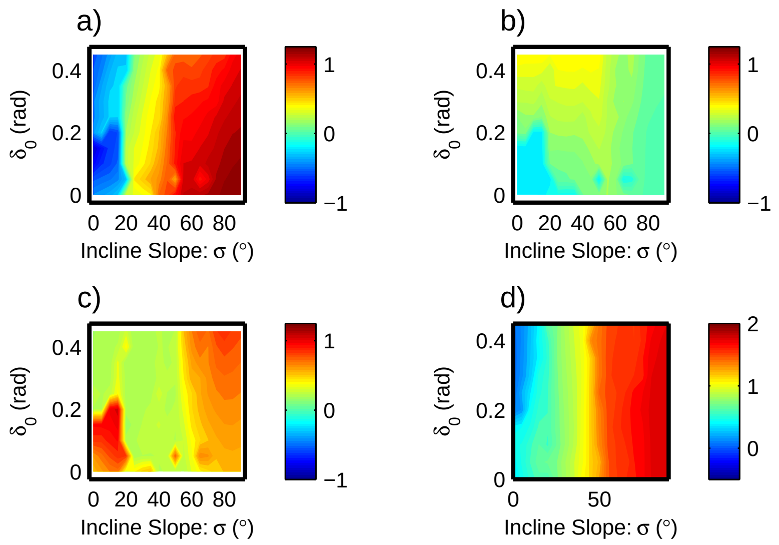

Figure 6.6: Maximum fore/aft forces of modeled cockroaches based on optimal stability gaits. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

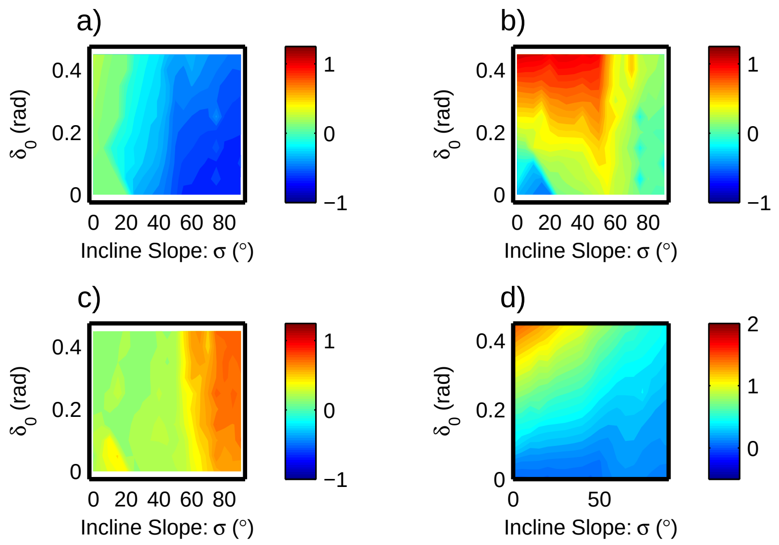

Figure 6.7: Maximum lateral forces of modeled cockroaches based on optimal metabolic efficiency gaits. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

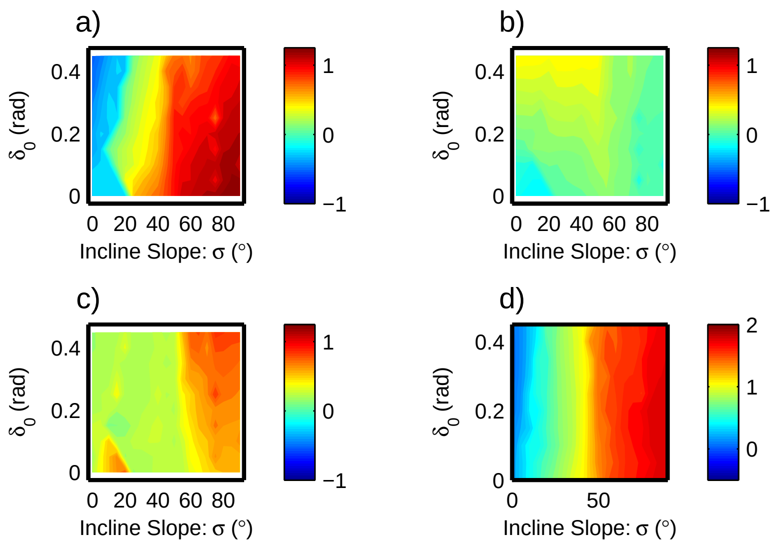

Figure 6.8: Maximum fore/aft forces of modeled cockroaches based on optimal metabolic efficiency gaits. Forces are normalized by dividing by the weight of the cockroach. a) Front leg; b) Middle Leg; c) Back Leg; d) Net combined force of all legs.

Numerical comparisons of the forces exhibited by the cockroaches and the model are shown in Tables 6.1, 6.2, and 6.3 for the front, middle, and back legs respectively. The first column of the tables shows the experimental data (D. Goldman, 2011, personal communication, Georgia Institute of Technology), while the second column is from the non-optimized fixed points. The last two columns are the gaits optimized for stability and efficiency. For each slope in the table, the minimum and maximum forces for heading angles between $\delta_0 = 0.05 \rightarrow 0.40$ are shown, since this is the range of heading angles that consistently produces valid fixed points. The forces generated are similar in magnitude to that of the measured data.

Front Leg				
σ	F_{lat}(exp.)	F_{lat}(model)	F_{lat}(opt. stab.)	F_{lat}(opt. eff.)
0°	$0.00 \rightarrow 0.38$	$0.14 \rightarrow 0.44$	$0.19 \rightarrow 0.46$	$0.11 \rightarrow 0.25$
15°	$-0.04 \rightarrow 0.12$	$-0.18 \rightarrow 0.30$	$0.12 \rightarrow 0.29$	$-0.11 \rightarrow 0.13$
30°	$-0.10 \rightarrow 0.16$	$-0.31 \rightarrow 0.10$	$-0.30 \rightarrow -0.14$	$-0.31 \rightarrow -0.14$
45°	$-0.38 \rightarrow 0.10$	$-0.40 \rightarrow -0.24$	$-0.45 \rightarrow -0.31$	$-0.44 \rightarrow -0.31$
60°	$-0.24 \rightarrow 0.00$	$-0.45 \rightarrow -0.32$	$-0.61 \rightarrow -0.39$	$-0.59 \rightarrow -0.34$
75°	$-0.55 \rightarrow -0.16$	$-0.48 \rightarrow -0.36$	$-0.64 \rightarrow -0.46$	$-0.61 \rightarrow -0.39$
90°	$-0.58 \rightarrow -0.28$	$-0.49 \rightarrow -0.38$	$-0.66 \rightarrow -0.51$	$-0.66 \rightarrow -0.50$
σ	F_{fa}(exp.)	F_{fa}(model)	F_{fa}(opt. stab.)	F_{fa}(opt. eff.)
0°	$-0.04 \rightarrow 0.04$	$-0.87 \rightarrow -0.29$	$-0.81 \rightarrow -0.37$	$-0.49 \rightarrow -0.22$
15°	$-0.04 \rightarrow 0.06$	$-0.62 \rightarrow 0.33$	$-0.55 \rightarrow -0.24$	$-0.27 \rightarrow 0.20$
30°	$-0.02 \rightarrow 0.20$	$-0.21 \rightarrow 0.56$	$0.27 \rightarrow 0.52$	$0.26 \rightarrow 0.54$
45°	$0.24 \rightarrow 0.60$	$0.49 \rightarrow 0.71$	$0.60 \rightarrow 0.80$	$0.59 \rightarrow 0.82$
60°	$0.30 \rightarrow 0.50$	$0.65 \rightarrow 0.79$	$0.81 \rightarrow 1.08$	$0.71 \rightarrow 1.05$
75°	$0.56 \rightarrow 0.84$	$0.73 \rightarrow 0.84$	$0.92 \rightarrow 1.16$	$0.74 \rightarrow 1.13$
90°	$0.64 \rightarrow 0.84$	$0.77 \rightarrow 0.87$	$1.05 \rightarrow 1.23$	$1.02 \rightarrow 1.22$

Table 6.1: Comparison of experimental forces to modeled front leg forces, showing a qualitative similarity in direction and magnitude. Experimental data from D. Goldman, 2011, personal communication, Georgia Institute of Technology. All forces are normalized by dividing by cockroach weight. Ranges represent the range of data for both the experimental data and the modeled data (including all fixed points for a given slope at heading angles between $\delta_0 = 0.05$ rad and 0.40 rad).

Middle Leg				
σ	$\mathrm{F_{lat}}$(exp.)	$\mathrm{F_{lat}}$(model)	$\mathrm{F_{lat}}$(opt. stab.)	$\mathrm{F_{lat}}$(opt. eff.)
0°	N/A	$-0.31 \rightarrow 0.62$	$-0.55 \rightarrow 0.93$	$-0.12 \rightarrow 0.94$
15°	N/A	$-0.04 \rightarrow 0.67$	$-0.50 \rightarrow 0.89$	$-0.39 \rightarrow 0.89$
30°	N/A	$0.07 \rightarrow 0.65$	$0.20 \rightarrow 0.95$	$0.22 \rightarrow 0.94$
45°	N/A	$0.12 \rightarrow 0.62$	$0.32 \rightarrow 0.92$	$0.33 \rightarrow 0.90$
60°	N/A	$0.19 \rightarrow 0.60$	$0.31 \rightarrow 0.47$	$0.29 \rightarrow 0.46$
75°	N/A	$0.24 \rightarrow 0.57$	$0.12 \rightarrow 0.32$	$-0.16 \rightarrow 0.30$
90°	N/A	$0.24 \rightarrow 0.51$	$0.04 \rightarrow 0.14$	$-0.15 \rightarrow 0.16$
σ	$\mathrm{F_{fa}}$(exp.)	$\mathrm{F_{fa}}$(model)	$\mathrm{F_{fa}}$(opt. stab.)	$\mathrm{F_{fa}}$(opt. eff.)
0°	N/A	$-0.18 \rightarrow 0.25$	$-0.19 \rightarrow 0.38$	$-0.07 \rightarrow 0.38$
15°	N/A	$-0.03 \rightarrow 0.27$	$-0.19 \rightarrow 0.37$	$-0.17 \rightarrow 0.37$
30°	N/A	$0.04 \rightarrow 0.27$	$0.09 \rightarrow 0.40$	$0.10 \rightarrow 0.40$
45°	N/A	$0.06 \rightarrow 0.27$	$0.15 \rightarrow 0.41$	$0.15 \rightarrow 0.40$
60°	N/A	$0.10 \rightarrow 0.27$	$0.16 \rightarrow 0.22$	$0.15 \rightarrow 0.22$
75°	N/A	$0.12 \rightarrow 0.26$	$0.07 \rightarrow 0.13$	$-0.10 \rightarrow 0.12$
90°	N/A	$0.12 \rightarrow 0.23$	$0.02 \rightarrow 0.07$	$-0.10 \rightarrow 0.07$

Table 6.2: Range of modeled middle leg forces. Experimental data does not exist for middle legs due to the difficulty in measuring this leg independent of the others. All forces are normalized by dividing by cockroach weight. Ranges represent the range of data for both the experimental data and the modeled data (including all fixed points for a given slope at heading angles between $\delta_0 = 0.05$ rad and 0.40 rad).

Back Leg				
σ	F_{lat}(exp.)	F_{lat}(model)	F_{lat}(opt. stab.)	F_{lat}(opt. eff.)
0°	$-0.20 \rightarrow 0.04$	$0.33 \rightarrow 0.38$	$0.16 \rightarrow 0.48$	$0.15 \rightarrow 0.27$
15°	$-0.18 \rightarrow 0.12$	$0.30 \rightarrow 0.39$	$0.19 \rightarrow 0.65$	$0.16 \rightarrow 0.46$
30°	$-0.12 \rightarrow 0.26$	$0.36 \rightarrow 0.39$	$0.17 \rightarrow 0.23$	$0.18 \rightarrow 0.23$
45°	$-0.12 \rightarrow 0.20$	$0.40 \rightarrow 0.41$	$0.19 \rightarrow 0.25$	$0.19 \rightarrow 0.26$
60°	$-0.18 \rightarrow 0.18$	$0.41 \rightarrow 0.42$	$0.38 \rightarrow 0.58$	$0.39 \rightarrow 0.61$
75°	$-0.14 \rightarrow 0.16$	$0.41 \rightarrow 0.42$	$0.59 \rightarrow 0.70$	$0.64 \rightarrow 0.77$
90°	$-0.02 \rightarrow 0.24$	0.42	$0.67 \rightarrow 0.79$	$0.65 \rightarrow 0.77$
σ	F_{fa}(exp.)	F_{fa}(model)	F_{fa}(opt. stab.)	F_{fa}(opt. eff.)
0°	$-0.02 \rightarrow 0.16$	$0.32 \rightarrow 0.60$	$0.20 \rightarrow 0.96$	$0.20 \rightarrow 0.31$
15°	$0.08 \rightarrow 0.40$	$0.28 \rightarrow 0.61$	$0.26 \rightarrow 1.07$	$0.15 \rightarrow 0.67$
30°	$0.20 \rightarrow 0.72$	$0.37 \rightarrow 0.61$	$0.21 \rightarrow 0.27$	$0.21 \rightarrow 0.27$
45°	$0.24 \rightarrow 0.60$	$0.46 \rightarrow 0.54$	$0.23 \rightarrow 0.28$	$0.23 \rightarrow 0.29$
60°	$0.44 \rightarrow 0.76$	$0.53 \rightarrow 0.57$	$0.39 \rightarrow 0.67$	$0.41 \rightarrow 0.75$
75°	$0.14 \rightarrow 0.58$	$0.57 \rightarrow 0.61$	$0.56 \rightarrow 0.79$	$0.60 \rightarrow 0.86$
90°	$0.10 \rightarrow 0.70$	$0.58 \rightarrow 0.63$	$0.58 \rightarrow 0.78$	$0.58 \rightarrow 0.78$

Table 6.3: Comparison of experimental forces to modeled back leg forces, showing a qualitative similarity in direction and magnitude. Experimental data from D. Goldman, 2011, personal communication, Georgia Institute of Technology. All forces are normalized by dividing by cockroach weight. Ranges represent the range of data for both the experimental data and the modeled data (including all fixed points for a given slope at heading angles between $\delta_0 = 0.05$ rad and 0.40 rad).

Chapter 7 – Discussion and Conclusions

Fixed points were successfully found for nearly all of the combinations of desired velocity (v_{des}), initial heading angle (δ_0), and incline slope (σ) that were attempted. These points were found over a range of incline slopes from level running to cliff climbing as well as a range of initial heading angles that encompass those observed on cockroaches. Gaits were modeled at velocities similar to that of those observed in real cockroaches.

Furthermore, Figure 4.1 shows that each of these fixed points were stable. In fact Figures 4.3 and 4.4 demonstrate that these gaits were also robust even for lateral impulses as large as 80% of the forward momentum. This return to a stable gait from large disturbances was accomplished without an active controller, except to place legs at the locations predetermined to produce a stable gait. Clearly, this control strategy shows significant promise towards developing a robot controller for a sprawled posture cockroach like robot with stable and robust gaits.

As Figures 4.2, 4.3, and 4.5 show, most of the selected gaits for high slopes ($\sigma >= 25°$) had ideal stability and robustness with eigenvalues approaching zero and the number of steps to recovery approaching one. Additionally, these gaits had ideal transport costs. This is an important finding especially since cockroaches also seem to use this back leg pushing gait on large inclines.

At lower incline slopes, multiple gait families are available. For these, stability and robustness are optimized by selecting gaits where the front and back legs push, while the middle legs pull. These gaits use high heading angles, which corresponds with a large degree of lateral motion. A secondary more stable region is also evident at heading angles of around $\delta_0 = 0.10$ rad, with all legs used for pushing function. On the other hand, the efficiency of the gaits as evidenced by the transport cost is most ideal at low heading angles. The transport cost grows consistently higher as the heading angle increases. It is known that cockroaches use heading angles of around $\delta_0 = 0.15$ rad (D. Goldman, 2011, personal communication, Georgia Institute of Technology) with all legs used in a pushing mechanism [1]. This seems to be an acceptable compromise between lowering the cost of transport while maintaining good stability.

The stability and metabolic efficiency of these gaits was further improved using optimization routines for each criteria. The resulting gaits consisted of four overlapping leg functional families which included: all legs pushing; front and back legs pushing while back leg pulls; middle and back legs pushing while front leg pulls; and back leg pushing while front and middle legs pull.

For both types of optimization, the resulting gaits for incline slopes above $\sigma = 20°$ consisted almost entirely of back leg pushing gaits. These gaits are naturally efficient since the leg forces do not generate any negative work. Therefore there is not significant difference between the two optimization schemes in this area. The middle and back leg pushing gaits are seldom used as the optimal gait for a given combination of incline slope and heading angle for either optimization scheme.

There are also similarities between the two optimization routines regarding the leg function used at low incline slopes. In both cases, the gaits with all legs pushing are selected for low initial heading angles with the front and back leg pushing gaits selected for higher heading angles. However, the transition between the two occurs at around $\delta_0 = 0.125$ rad for the efficiency optimized gaits and at a higher value of around $\delta_0 = 0.20$ rad for the stability optimized gaits.

The gaits resulting from the optimization for the front and back leg pushing families are remarkably similar. This suggests that the stability based optimization and the efficiency based optimization goals are met with similar leg actuations and phase shifts for this family of gaits. On the other hand there is a large difference between the gaits resulting from the two optimization for the all legs pushing gaits. For these gaits, stability and efficiency optimizations work against each other. While the stability and transport costs of the two cases are significantly different, there are also differences in the use of the legs. In particular, the middle leg has a very high phase shift for the efficiency optimized gaits, which means that this leg is only used for a very short duration. This means that the middle leg is much less effectual when optimized for metabolic efficiency as opposed to stability.

While a cockroach cannot choose the slope of the ground which it wishes to travel on, it could conceivably choose the leg muscle actuation levels, whether each leg pushes or pulls, and the initial heading angle of its gait. If this selection were based on stability alone, Figure 5.9 would indicate the preferred selection at least for the modeled cockroach. At higher slopes, the selected gait would use only the back leg for pushing. All heading angles are equally stable, so the choice between heading angles would not matter. At slopes below $\sigma = 20°$, the selected gaits would use all legs for pushing and the modeled cockroach would use a heading angle of around $\delta_0 = 0.10 \text{ rad} \rightarrow 0.15$ rad.

The selected gaits using an efficiency optimization on the modeled cockroach would actually be very similar based on Figure 5.19. For higher slopes, the selected gait would again use only the back leg for pushing and the same transport cost would be achieved regardless of the heading angle. At slopes below $\sigma = 10^\circ$, the selected gaits would use all legs for pushing and the modeled cockroach would use a similar heading angle of around $\delta_0 = 0.10$ rad . However, the gaits would use the middle leg for a much shorter duration than those of the stability optimized gaits.

Based on the experimental data that we have, it is hard to determine which of these very similar optimizations are closer to what the real cockroaches do. However, given that cockroaches use heading angles of around $\delta_0 = 0.15$ rad (D. Goldman, 2011, personal communication, Georgia Institute of Technology) and that the front and middle legs transition from pushing to pulling at around $\sigma = 15^\circ \rightarrow 30^\circ$, both of the optimization routines result in gaits remarkably similar to real cockroaches. It seems plausible that cockroaches use gaits based on either stability, efficiency, or a combination of both. It is also telling that the forces generated by the model are qualitatively similar to the experimental forces.

One major step toward continuing this research would be to get more detailed estimates of the forces used by cockroaches, particularly in the middle leg. This would allow a more exact estimation of what slope cockroaches transition their leg function. This information could be valuable information in designing a robot based on this mathematical model.

Another advancement in this research would come from a more realistic model of the cockroach motion which would incorporate rigid body dynamics in rotational motion. This would allow the yawing characteristics of the model to be compared to actual yawing motion of cockroaches. In particular, a model that has leg attachment points at locations other than the center of gravity would be needed. Moving the hip joint from the center of mass would allow the legs to generate torque, causing the body to rotate in the yaw direction.

Bibliography

[1] R. Full, R. Blickhan, and L. Ting, "Leg design in hexapedal runners," *The Journal of Experimental Biology*, vol. 158, pp. 369–390, 1991.

[2] D. Goldman, T. Chen, D. Dudek, and R. Full, "Dynamics of rapid vertical climbing in cockroaches reveals a template," *The Journal of Experimental Biology*, vol. 209, pp. 2990–3000, 2006.

[3] R. Kram, B. Wong, and R. Full, "Three-dimensional kinematics and limb kinetic energy of running cockroaches," *Journal of Experimental Biology*, vol. 200, pp. 1919–1929, 1997.

[4] J. Schmitt, M. Garcia, R. Razo, P. Holmes, and R. Full, "Dynamics and stability of legged locomotion in the horizontal plane: a test case using insects," *Biological Cybernetics*, vol. 86, pp. 343–353, 2002.

[5] J. Clark, D. Goldman, P. Lin, L. G., T. Chen, H. Komsuoglu, R. Full, and D. Koditscheck, "Design of a bio-inspired dynamical vertical climbing robot," in *Proceedings of Robotics: Science and Systems*, 2007.

[6] R. Blickhan and R. Full, "Similarity in multilegged locomotion: Bouncing like a monopode," *Journal of Comparative Physiology A*, vol. 173, pp. 509–517, 1993.

[7] K. Autumn, S. Hsieh, D. Dudek, J. Chen, C. Chitaphan, and R. Full, "Dynamics of geckos running vertically," *The Journal of Experimental Biology*, vol. 209, pp. 260–272, 2006.

[8] J. Schmitt and S. Bonnono, "Dynamics and stability of lateral plane locomotion on inclines," *Journal of Theoretical Biology*, vol. 261, pp. 598–609, 2009.

[9] R. Blickhan, "The spring mass model for running and hopping," *Journal of Biomechanics*, vol. 22, no. 11/12, pp. 1217–1227, 1989.

[10] P. Holmes, R. Full, D. Koditscheck, and J. Guckenheimer, "The dynamics of legged locomotion: Models, analyses, and challenges," *SIAM Review*, vol. 48, no. 2, pp. 207–304, 2006.

[11] M. Srinivasan and P. Holmes, "How well can spring-mass-like telescoping leg models fit multi-pedal sagittal-plane locomotion data?" *Journal of Theoretical Biology*, vol. 255, pp. 1–7, 2008.

[12] M. Daley and A. Biewener, "Running over rough terrain reveals limb control for intrinsic stability," *Proceedings of the National Academy of the Sciences*, vol. 103, no. 42, pp. 15 681–15 686, 2006.

[13] J. Schmitt and P. Holmes, "Mechanical models for insect locomotion: Dynamics and stability in the horizontal plane i. theory," *Biological Cybernetics*, vol. 83, pp. 501–515, 2000.

[14] J. Seipel, P. Holmes, and R. Full, "Dynamics and stability of insect locomotion: A hexapedal model for horizontal plane motions," *Biological Cybernetics*, vol. 91, pp. 76–90, 2004.

[15] R. Kukillaya and P. Holmes, "A hexapedal jointed-leg model for insect locomotion in the horizontal plane," *Biological Cybernetics*, vol. 97, pp. 379–395, 2007.

[16] ——, "A model for insect locomotion in the horizontal plane: Feedforward activation of fast muscles,stability,and robustness," *Journal of Theoretical Biology*, vol. 261, pp. 210–226, 2009.

[17] K. Mombaur, R. Longman, H. Bock, and J. Schlijder, "Stable one-legged hopping without feedback and with a point foot," in *Proceedings of the 2002 IEEE International Conference on Robotics 8 Automation Washington, DC May*, 2002.

[18] Q. Wang, K. Wei, L. Wang, and D. Lv, "Modeling and stability analysis of human normal walking with implications for the evolution of the foot," in *Proceedings of the 2010 3rd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics, The University of Tokyo, Tokyo, Japan, September 26-29*, 2010.

[19] M. Srinivasan and A. Ruina, "Computer optimization of a minimal biped model discovers walking and running," *Nature*, vol. 439, pp. 72–75, 2006.

[20] S. Roy and D. Pratihar, "Study on energy consumption in turning motion of hexapod walking robots," in *Proceedings of the World Congress on Engineering Vol III WCE 2011, July 6 - 8, 2011, London, U.K.*, 2011.

[21] L. Ting, R. Blickhan, and R. Full, "Dynamic and static stability in hexapedal runners," *Journal of Theoretical Biology*, vol. 197, pp. 251–269, 1994.

[22] S. Sponberg and R. Full, "Neuromechanical response of musculo-skeletal structures in cockroaches during rapid running on rough terrain," *Journal of Experimental Biology*, vol. 211, pp. 433–446, 2008.

[23] D. Jindrich and R. Full, "Dynamic stabilization of rapid hexapedal locomotion," *The Journal of Experimental Biology*, vol. 205, pp. 2803–2823, 2002.

[24] A. Wickramasuriya and J. Schmitt, "Improving horizontal plane locomotion via leg angle control," *Journal of Theoretical Biology*, vol. 256, pp. 414–427, 2009.

[25] S. Bonnono, "Dynamics and stability of lateral plane locomotion on variable inclines," Master's thesis, Oregon State University, 2009.

[26] M. Srinivasan and A. Ruina, "Idealized walking and running gaits minimize work," *Proc. Roy. Soc. A*, vol. 463, pp. 2429–2446, 2007.

[27] M. Srinivasan, "Fifteen observations on the structure of energy-minimizing gaits in many simple biped models," *Journal of the Royal Society: Interface*, p. doi: 10.1098/rsif.2009.0544, 2010.

[28] A. Ruina, J. Bertram, and M. Srinivasan, "A collisional model of the energetic cost of support work qualitatively explains leg-sequencing in walking and galloping, pseudo-elastic leg behavior in running and the walk-to-run transition," *Journal of Theoretical Biology*, vol. 237, pp. 170–192, 2005.

[29] J. Donelan, R. Kram, and A. Kuo, "Mechanical and metabolic determinants of the preferred step width in human walking," *Proc. R. Soc. Lond. B*, vol. 268, pp. 1985–1992, 2001.

[30] J. Lagarias, J. Reeds, M. Wright, and P. Wright, "Convergence properties of the nelder-mead simplex method in low dimensions," *SIAM Journal on Optimization*, vol. 9, pp. 112–147, 1998.

[31] T. Kubow and R. Full, "The role of the mechanical system in control: a hypothesis of self-stabilization in hexapedal runners," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 354, pp. 849–861, 1999.

APPENDICES

Appendix A – Matlab Code

A.1 Robot Simulation

A.1.1 simgait_multileg.m

This is the main code for the simulation. Based on parameters set as global values and initial conditions passed in as inputs, it simulates and returns the motion of the cockroach.

```
function [xtot,xdtot,ytot,ydtot,thetatot,thetadtot,...
  timetot,indbrktot,Fxtot,Fytot,Mtot,fpxtot,fpytot,...
  leqtot,lsprtot,tot_ener,Ctrantot] = ...
  simgait_multileg(X0,numperiod,plotflag)
% This function simulates the number of gaits specified
% in numperiod based on the initial conditions X0. States
% are x, xdot, y, ydot, theta, and thetadot. Plots are
% included if plotflag = 1.


global cval tstartstance fpx fpy FootAct
%The following values are input as global variables but
%local copies need to be stored since their values get
%overwritten
tstartstance0 = tstartstance;
fpx0 = fpx;
```

```
fpy0 = fpy;
FootAct0 = FootAct;
options = odeset('Events',@stancebreak,...
  'InitialStep',.000001);
%Read in global cval
kval = cval(1);
m = cval(2);
Ival = cval(3);
% sigma = cval(4);
tdes = cval(5);
% vdes = cval(6);
% Omega = pi/tdes;
% g = 9.81;

% Initialize storage variables here
xtot = [];    %States
xdtot = [];
ytot = [];
ydtot = [];
thetatot = [];
thetadtot = [];
timetot = [];
Fxtot = [];   %Forces and Moments
Fytot = [];
Mtot = [];
```

```
leqtot = [];  %Spring length parameters
lsprtot = [];
Ctran0 = 0;   %Transport Cost calculator
Ctrantot = [];
fpxtot = fpx; %Foot placements
fpytot = fpy;
indbrktot = zeros(1,6); %Indices of step breaks
%Set stride counters (-1 for not just placed)
stride = -ones(1,6);
end_ft = 1; %Which foot counter is used for termination
stride(tstartstance == tstartstance(end_ft)) = 0;
tsimstart = tstartstance(end_ft);
indbrktot(tstartstance == tsimstart) = 1;

% Simulate stance phases
while stride(end_ft) < numperiod
  tspan = [tsimstart tsimstart+tdes];
  [T,X,te,˜,ie] = ...
    ode45(@stancedynamics,tspan,[X0 Ctran0],options);
  Ctran = X(:,7);
  X = X(:,1:6);
  % Store relevant data
  xtot = [xtot;X(:,1)];
  xdtot = [xdtot;X(:,2)];
  ytot = [ytot;X(:,3)];
```

```
ydtot = [ydtot;X(:,4)];
thetatot = [thetatot;X(:,5)];
thetadtot = [thetadtot;X(:,6)];
timetot = [timetot;T];
Ctrantot = [Ctrantot;Ctran];
%Store index of events
indbrk = zeros(1,6);
for n = 1:length(ie)
  if te(n) == T(length(T))
    if ie(n) <= 6
      %Store a negative value for liftoff stance-breaks
      indbrk(ie(n)) = -length(timetot);
    end
  end
end
tstarttol = 1e-10;
indbrk((tstartstance + 2*tdes >= T(length(T)))&...
  (tstartstance + 2*tdes < T(length(T))+tstarttol)) ...
  = length(timetot);
indbrktot = [indbrktot;indbrk];
%Calculate parameters that need stored at every time
for n = 1:length(T)
  [Fx,Fy,M,leq,lspr] = legforces(T(n),X(n,:));
  leq(~FootAct) = 0;
  lspr(~FootAct) = 0;
```

```
    Fxtot = [Fxtot;Fx];
    Fytot = [Fytot;Fy];
    Mtot = [Mtot;M];
    leqtot = [leqtot;leq];
    lsprtot = [lsprtot;lspr];
  end
  %Set up for next stride
  ind = length(T);
  X0 = X(ind,:);
  Ctran0 = Ctran(ind);
  tstartstance(indbrk > 0) = T(ind);
  FootAct(indbrk > 0) = true;
  FootAct(indbrk < 0) = false;
  [fpxtemp,fpytemp] = footplace(X0);
  fpx(indbrk > 0) = fpxtemp(indbrk > 0);
  fpy(indbrk > 0) = fpytemp(indbrk > 0);
  tsimstart = T(ind);
  %Store new foot placement points
  fpxtot = [fpxtot;fpx];
  fpytot = [fpytot;fpy];
  stride(indbrk > 0) = stride(indbrk > 0) + 1;
end
%Reset initial values
FootAct = FootAct0;
fpx = fpx0;
```

```
fpy = fpy0;
tstartstance = tstartstance0;

%Calculate energy for plotting
kin_ener = m*(xdtot.^2 + ydtot.^2)/2+Ival*thetadtot.^2/2;
pot_ener = sum(kval*(lsprtot - leqtot).^2/2,2);
tot_ener = kin_ener+pot_ener;
% Plot data if required
if plotflag
  figure, hold on
  N = size(fpxtot,1);
  for n = 1:N
    %Foot placement points
    plot(fpxtot(n,:),fpytot(n,:),'x',...
      'MarkerEdgeColor',[(N-n)/(N-1) 0 (n-1)/(N-1)])
  end
  %Center of mass trajectory
  plot(xtot,ytot,'k-')
  axis image,axis equal
  markstyle = ['b+';'gx';'r*';'c+';'yx';'m*'];
  %Stance break markers
  for n = 1:6
    indt = indbrktot(indbrktot(:,n)>0,n);
    plot(xtot(indt),ytot(indt),markstyle(n,:),...
      'LineWidth',1,'MarkerSize',8)
```

```
end
figure
%Important states
subplot(2,2,1),plot(timetot,xtot), hold on
title('x')
subplot(2,2,2),plot(timetot,ytot), hold on
title('y')
subplot(2,2,3),plot(timetot,thetatot*180/pi), hold on
title('\theta')
ylabel('\circ')
for n = 1:6
  indt = indbrktot(indbrktot(:,n)>0,n);
  subplot(2,2,1),plot(timetot(indt),xtot(indt),...
    markstyle(n,:),'LineWidth',2,'MarkerSize',8)
  subplot(2,2,2),plot(timetot(indt),ytot(indt),...
    markstyle(n,:),'LineWidth',2,'MarkerSize',8)
  subplot(2,2,3),plot(timetot(indt),thetatot(indt)*...
    180/pi,markstyle(n,:),'LineWidth',2,'MarkerSize',8)
end
figure
%Forces and Moments
COL_ORD=[0 0 1;0 1 0;1 0 0;0 1 1;1 1 0;1 0 1;.6 .6 .6];
subplot(2,2,1), hold on, title('Fx')
plot(timetot,sum(Fxtot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
```

```
subplot(2,2,2), hold on, title('Fy')
plot(timetot,sum(Fytot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
subplot(2,2,3), hold on, title('M')
plot(timetot,sum(Mtot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
subplot(2,2,4), hold on
for n = 1:6
  indt = indbrktot(indbrktot(:,n)>0,n);
  subplot(2,2,1), plot(timetot,Fxtot(:,n),...
    'Color',COL_ORD(n,:))
  plot(timetot(indt),Fxtot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,2), plot(timetot,Fytot(:,n),...
    'Color',COL_ORD(n,:))
  plot(timetot(indt),Fytot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,3), plot(timetot,Mtot(:,n),...
    'Color',COL_ORD(n,:));
  plot(timetot(indt),Mtot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,4), plot(0,0,'Color',COL_ORD(n,:))
end
plot(0,0,'Color',COL_ORD(7,:),'LineStyle','-.')
set(gca,'Visible','off')
```

```
legend('1','2','3','4','5','6','sum','Location','West')
figure
%Leg lengths
subplot(3,2,1)
plot(timetot,lsprtot(:,1),timetot,leqtot(:,1),':')
title(['Actual and Equilibrium Spring Lengths'...
  ' of a Simulated Cockroach'])
ylabel('Front lng. (m)')
subplot(3,2,3)
plot(timetot,lsprtot(:,2),timetot,leqtot(:,2),':')
ylabel('Mid lng. (m)')
subplot(3,2,5)
plot(timetot,lsprtot(:,3),timetot,leqtot(:,3),':')
xlabel('Time (s)')
ylabel('Back lng. (m)')
subplot(3,2,2)
plot(timetot,lsprtot(:,4),timetot,leqtot(:,4),':')
title(['Actual and Equilibrium Spring Lengths'...
  ' of a Simulated Cockroach'])
ylabel('Front lng. (m)')
subplot(3,2,4)
plot(timetot,lsprtot(:,5),timetot,leqtot(:,5),':')
ylabel('Mid lng. (m)')
subplot(3,2,6)
plot(timetot,lsprtot(:,6),timetot,leqtot(:,6),':')
```

```
  xlabel('Time (s)')
  ylabel('Back lng. (m)')
  figure
  %Energy
  plot(timetot,tot_ener,timetot,kin_ener,'r:',...
    timetot,pot_ener,'b-.')
  legend('Total Energy','Kinetic Energy',...
    'Potential Energy','Location','NorthWest')
  title('Total Energy of a Simulated Cockroach')
  xlabel('Time (s)')
  ylabel('Energy (J)')
end
end

%%% end main function

%%% stancedynamics function

function Xd = stancedynamics(t,X)

global cval

%Call cval
% kval = cval(1);
m = cval(2);
```

```
Ival = cval(3);
sigma = cval(4);
tdes = cval(5);
vdes = cval(6);
% Omega = pi/tdes;
g = 9.81;

% read current state
x = X(1);
xd = X(2);
y = X(3);
yd = X(4);
% theta = X(5);
thetad = X(6);

% solve equations of motion
[Fx,Fy,M] = legforces(t,X);
xdd = sum(Fx)/m;
ydd = (sum(Fy)-m*g*sin(sigma))/m;
thetadd = sum(M)/Ival;
Xd = [xd;xdd;yd;ydd;thetad;thetadd];

%Calculate transport cost
global fpx fpy
Xd(7) = sum(sqrt(Fx.^2+Fy.^2).*...
  abs(((x-fpx)*xd + (y-fpy)*yd)./sqrt((x-fpx).^2 ...
  + (y-fpy).^2)))/(m*g*tdes*vdes);
if isnan(Xd(7)) %remove a discontinuity
  Xd(7) = 0;
end
end

%%% end stancedynamics function

%%% stancebreak function

function [delta, isterminal, direction] ...
  = stancebreak(t,X)
% Determines when a front leg force returns to zero

global cval FootAct tstartstance
tdes = cval(5);

%Calculate foot ending stance break
[~,~,~,l,dr] = legforces(t,X);
deltal = dr - l;
deltal(~FootAct) = 1;
%Calculate foot beginning stance break
deltat = tstartstance + 2*tdes - t;
```

```
%Set return values
delta = [deltal';deltat'];
isterminal = [FootAct';true(6,1)];
%Setting any to zero would ignore the corresponding leg
direction = [zeros(6,1);-ones(6,1)];
minstep = tdes/50;
for n = 1:6
  if abs(t-tstartstance(n)) < abs(minstep)
    isterminal(n) = 0;
  end
end
end

%%% end stancebreak function
```

A.1.2 legforces.m

This code is called by simgait_multileg and is used to calculate the forces of the leg springs in the inertial frame.

```
function [Fx,Fy,M,l,dr] = legforces(t,v)
% Calculate forces and moments for each leg (fixed frame)
global cval tstartstance fpx fpy FootAct

%Call cval
kval = cval(1);
```

```
% m = cval(2);

% Ival = cval(3);

% sigma = cval(4);

% tdes = cval(5);

% vdes = cval(6);

% dx = cval(7:12);

% dy = cval(13:18);

l0 = cval(19:24);

ldev = cval(25:30);

% beta0 = cval(31:36);

Tdrv = cval(37:39);

Omega = pi./Tdrv;

Omega = [Omega Omega];

phi = cval(40:42);

phi = [phi phi];


% Read states

x = v(1);

% xd = v(2);

y = v(3);

% yd = v(4);

% theta = v(5);

% thetad = v(6);


%Calculate hip positions in the inertial frame
```

```
[hpx,hpy] = locatehips(v);
%Calculate position vectors
l = l0 - ldev.*sin(Omega.*(t-tstartstance)+phi);
rx = fpx - hpx;
ry = fpy - hpy;
r = [rx;ry];
%Initialize vectors
dr = zeros(1,6);
F = zeros(2,6);
M3 = zeros(3,6);
for n = 1:6
  if FootAct(n)
    %Calculate forces
    dr(1,n) = norm(r(:,n));
    F(:,n) = kval*(dr(n)-l(n))*r(:,n)/dr(n);
    r3 = [fpx(n)-x;fpy(n)-y;0];
    F3 = [F(:,n);0];
    M3(:,n) = cross(r3,F3);
  end
end
Fx = F(1,:);
Fy = F(2,:);
M = M3(3,:);
end
```

```
%%% end legforces function
```

A.1.3 footplace.m

At each new step, this function finds the placement of feet.

```
function [fpx, fpy] = footplace(X)
% Calculates foot placement vector. Use only values
% relevant for the appropriate stance phase. First
% position for front leg, second position for middle leg
% (opposite side), third position for back leg.  This
% function returns all three leg positions, but can be
% called if only one leg is being placed.  The calling
% function must be careful to only reassign the relevant
% foot.

% Modified 5-13, calculate all six legs

%read cval
global cval
% dx = cval(7:12);
% dy = cval(13:18);
l0 = cval(19:24);
ldev = cval(25:30);
beta0 = cval(31:36);
phi = cval(40:42);
```

```
l0C = l0 - ldev.*sin([phi phi]);

% read initial state
% x = X(1);
% xd = v(2);
% y = X(3);
% yd = v(4);
theta = X(5);
% thetad = v(6);

% Calculate position
% Use the following conventions: theta (body orientation)
% is +CCW for LSP and RSP from straight up beta (leg
% placement angle) is +CCW for LSP and RSP from body axis
% (so legs on the right side are expected to have
% negative values)
[hpx,hpy] = locatehips(X);
fpx = hpx - l0C.*sin(beta0+theta);
fpy = hpy + l0C.*cos(beta0+theta);
end

%%% end footplace function
```

A.1.4 locatehips.m

This function calculates the location of hips. For point mass models, the hips are located at the center of mass so the function returns zeros.

```
function [hpx,hpy] = locatehips(X)
%Used to calculate the hip locations of the roach at any
%given point in time

%read cval
global cval
dx = cval(7:12);
dy = cval(13:18);

%read state
x = X(1);
% xd = v(2);
y = X(3);
% yd = v(4);
theta = X(5);
% thetad = v(6);

%Calculate hip positions in the inertial frame
hpx = x + dx*cos(theta)-dy*sin(theta);
hpy = y + dx*sin(theta)+dy*cos(theta);
end
```

```
%%% end locatehips function
```

A.2 Parameter Initialization

A.2.1 find_fp.m

This script is used to set the initial parameters of a single fixed point. It also calls fsolve on the fp_multileg function to find a fixed point.

```
clear all, close all,%clc
format long, format compact
global cval tstartstance fpx fpy FootAct fp_guess act_var
global act_con ldev IsPush
%Select pushing legs and desired slope
IsPush = [true true true];
sig = 0;

%These are the options for the fsolve routine
act_var = [1 13 14 15];
%var order = [v0 delta0 tdes 4-6=>ldev1-3 7-9=>beta1-3
%10-12=>Tdrv1-3 13-15=>phi1-3 ldev% beta%]
act_con = [1 2 3 4];
%con order = [Dv Ddelta vdes xdrift]
TypX = [.25 .5 .05 .01*ones(1,3) pi/2*ones(1,3) ...
  .05*ones(1,3) pi/4*ones(1,3) 1 1]';
options = optimset('MaxIter',30,'MaxFunEvals',1000,...
  'Display','iter','TolX',1e-6,'TolFun',1e-6,...
  'TypicalX',TypX(act_var),'ScaleProblem','Jacobian');
% warning('ScaleProblem off for older version of Matlab')

%Set up parameters
% Parameters from Seipel, Holmes, and Full, "Dynamics and
% Stability of Insect Locomotion: A3 Hexapedal Model for
% Horizontal Plane Motions", Biological Cybernetics, 2004
cval = [1.5 0.0025 2.04e-7 sig*pi/180 .05 .25];
%cval = [k m I sig tdes vdes]
%Interpolate to find vdes
vdes0 = .35;
vdes90 = .20;
vdes = vdes0 + sig/90*(vdes90 - vdes0)
cval(6) = vdes;
k = cval(1);
% m = cval(2);
% Ival = cval(3);
% sigma = cval(4);
% tdes = cval(5);
vdes = cval(6);
% vary tdes to fit vdes
tdes = inv(freq_fitter(vdes))/2;
cval(5) = tdes;
```

```
% The following parameters are in leg order
% FL-MR-BL-FR-ML-BR. Parameters from Kukillaya and Holmes
%, "A Hexapedal Jointed-Leg Model for Insect Locomotion
% in the Horizontal Plane", Biological Cybernetics, 2007.
% dx = [-.0035 .0035 -.0035 .0035 -.0035 .0035];
% dy = [.014 .007 0 .014 .007 0];
%%%%% d = 0 case (hip over C.M.)
dx = zeros(1,6)
dy = zeros(1,6)
%%%%%


% Calculate the foot length and angle based on the
% parameters from the paper.
% fx = [-.011 .013 -.013 .011 -.013 .013];
% fy = [.02 .007 -.01 .02 .007 -.01];
% r = [fx;fy]-[dx;dy]
% for n = 1:6
%     l0(n) = norm(r(:,n));
%     bet(n) = atan2(-r(1,n),r(2,n))*180/pi;
% end
% l0
% bet
% Use approximate values close to those calculated in the
% commented out section
l0 = [.01 .01 .014 .01 .01 .014];
```

```
%Updated from Arun's data
l0 = [.032 .026 .019 .032 .026 .019];
%Starting guess
% ldev = [.015 .015 .015];
ldev = l0(1:3)/2
ldev = abs(ldev);
ldev(IsPush) = -ldev(IsPush);
ldev = [ldev ldev];

%Using Arun's data
beta0 = [25 -55 120]*pi/180;
beta0 = [beta0 -beta0];
%Half period (step length) used for actuation frequency
Tdrv = [tdes tdes tdes];
phi = [0 0 0];
% phi = [-.5 0 0];
cval = [cval dx dy l0 ldev beta0 Tdrv phi];
%cval7-12 => dx, cval13-18 => dy, cval19-24 => l0
%cval25-30 => ldev, cval31-36 => beta0, cval37-39 => Tdrv
%cval40-42 => phi
ldevX = 1;
betaX = 1;

% Starting guesses
v0 = cval(6);
```

```
% v0 = 0.286674934558385;
delta0 = .15;
fp_guess = [v0 delta0 tdes ldev(1:3) beta0(1:3) ...
  Tdrv(1:3) phi(1:3) ldevX betaX];

%Correct cval based on fp_guess
cval(5) = tdes;
cval(25:30) = ldevX*ldev;
betaC = beta0;
betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
cval(31:36) = betaC;
cval(37:39) = Tdrv;
cval(40:42) = phi;

%Initial conditions
theta0 = 0;
thetad0 = 0;
tsdes = [0 0 0];
x_t = zeros(1,6);
y_t = zeros(1,6);
xd = -v0*sin(delta0);
yd = v0*cos(delta0);
X0 = [0 xd 0 yd theta0 thetad0];


% Pre-solve foot place finder
```

```
footplace_init(tsdes,X0,x_t,y_t)
%Plot initial guess
[x,xd,y,yd,theta,thetad,t,ind,Fx,Fy,M,fpxtot,fpytot,...
  leq,lspr,E] = simgait_multileg(X0,2,1);
calc_C
C

%Call fixed point finder
fp = fsolve(@fp_multileg,fp_guess(act_var),options);
fptemp = fp_guess;
fptemp(act_var) = fp;
fp = fptemp
fp_guess

%Set parameters based on the resulting fixed point
v0 = fp(1);
delta0 = fp(2);
tdes = fp(3);
ldev = [fp(4:6) fp(4:6)];
beta0 = [fp(7:9) -fp(7:9)];
Tdrv = fp(10:12);
phi = fp(13:15);
ldevX = fp(16);
betaX = fp(17);
%Correct cval
```

```
cval(5) = tdes;
cval(25:30) = ldevX*ldev;
betaC = beta0;
betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
cval(31:36) = betaC;
cval(37:39) = Tdrv;
cval(40:42) = phi;

%Set initial conditions
xd = -v0*sin(delta0);
yd = v0*cos(delta0);
X0 = [0 xd 0 yd theta0 thetad0];

%foot place finder
footplace_init(tsdes,X0,x_t,y_t)

[x,xd,y,yd,theta,thetad,t,ind,Fx,Fy,M,fpxtot,fpytot,...
  leq,lspr,E,Ct] = simgait_multileg(X0,2,1);
calc_C
C(act_con)

% dval = [1e-6 1e-5 1e-1 1e-4];
% eval = findeig([v0 delta0 0 0],dval)
```

A.2.2 freq_fitter.m

```
function omega = freq_fitter(v)
%Uses a fourth order polynomial to estimate the desired
%frequency from the desired velocity.  This uses measured
%data of actual cockroach relationships. "Dynamic and
%Static Stability in Hexapedal Runners", Ting, Blickhan
%and Full. Journal of Theoretical Biology, 1994, 197:
%251 269

freqtot2 = [2.9; 4; 4.9; 5; 6.4; 8.8; 9.2; 10.5; 12.4;
  10.6; 11.5; 12.1; 11.6; 13.1; 13.8; 13.7; 13.4; 13.7;
  13.9; 14.4; 13.6; 13.8];
forv2 = [6; 8.5; 12; 12.2; 13; 18.5; 20; 26; 26; 29;
  29.5; 31; 32; 33; 38; 39; 41; 45; 45; 50; 53; 56];
forv2 = forv2/100;

% Create fit of frequency versus velocity and stride
% length versus velocity
% fitfreq = polyfit(forv2,freqtot2,4);
% omega = polyval(fitfreq,v);

fitfreq1 = polyfit(forv2(1:14),freqtot2(1:14),1);
fitfreq2 = polyval(fitfreq1,.35);
if v <= .35
  omega = polyval(fitfreq1,v);
```

```
else
  omega = fitfreq2;
end
```

A.2.3 footplace_init.m

This function sets some initial parameters as global values. Essentially it calculates where the feet must have been placed if you were already running at a fixed point.

```
function [] = footplace_init(tsdes,X0,x_t,y_t)
%Finds the initial foot placement at the beginning of a
%simulation. All values are returned as global variables.
global cval tstartstance FootAct fpx fpy IsPush
tdes = cval(5);
vdes = cval(6);
IsPush6 = [IsPush IsPush];

%Set defaults
FootAct = true(1,6);
tstartstance = zeros(1,6);
%Convert tsdes to tstartstance

%Set initial gusses for tstartstance
%tsdes would allow legs to start at different times, but
%all current codes set tsdes to zeros.
```

```
for n = 1:3
  if tsdes(n) > 0
    tstartstance([n n+3]) ...
      = [tsdes(n)-2*tdes tsdes(n)-tdes];
  else
    tstartstance([n n+3]) = [tsdes(n) tsdes(n)-tdes];
  end
end
%Correct for out of range
while min(tstartstance) <= -2*tdes ...
    || max(tstartstance) > 0
  tstartstance(tstartstance <= -2*tdes) = ...
    tstartstance(tstartstance <= -2*tdes) + 2*tdes;
  tstartstance(tstartstance > 0) = ...
    tstartstance(tstartstance > 0) - 2*tdes;
end
%Find correct foot placement
[fpx,fpy] = footplace(X0);
for n = 1:6
  X0t = X0+[x_t(n) 0 vdes*tstartstance(n)+y_t(n) 0 0 0];
  [fpxt,fpyt] = footplace(X0t);
  fpx(n) = fpxt(n);
  fpy(n) = fpyt(n);
  [~,~,~,lt,drt] = legforces(0,zeros(6,1));
  %Turn off legs that should not be active
  if IsPush6(n)
    if drt(n) > lt(n)+eps
      FootAct(n) = false;
    end
  else
    if drt(n) < lt(n)-eps
      FootAct(n) = false;
    end
  end
end
```

A.3 Fixed Point Finding

A.3.1 fp_multileg.m

This is the main function that fsolve refers to when finding a fixed point. It runs a simulation based on the fixed point guess, then calls calc_C to calculate the cost function.

```
function C = fp_multileg(X)
%Used with fsolve to find a fixed point of a multileg,
%ldev model
global cval tstartstance fpx fpy FootAct fp_guess
global act_var act_con ldev

%Retrieve active variables
```

```
Xtemp = fp_guess;
Xtemp(act_var) = X;

%Set parameters
v0 = Xtemp(1);
delta0 = Xtemp(2);
tdes = Xtemp(3);
ldev = [Xtemp(4:6) Xtemp(4:6)];
beta0 = [Xtemp(7:9) -Xtemp(7:9)];
Tdrv = Xtemp(10:12);
phi = Xtemp(13:15);
ldevX = Xtemp(16);
betaX = Xtemp(17);

%Correct cval
cval(5) = tdes;
cval(25:30) = ldevX*ldev;
betaC = beta0;
betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
cval(31:36) = betaC;
cval(37:39) = Tdrv;
cval(40:42) = phi;

%Set initial conditions
theta0 = 0;
```

```
thetad0 = 0;
tsdes = [0 0 0];
x_t = zeros(1,6);
y_t = zeros(1,6);
xd = -v0*sin(delta0);
yd = v0*cos(delta0);
X0 = [0 xd 0 yd theta0 thetad0];
footplace_init(tsdes,X0,x_t,y_t)

%Solve for final conditions
[x,xd,y,yd,theta,thetad,t,ind]=simgait_multileg(X0,1,0);
calc_C
%Select constraints
C = C(act_con);
```

A.3.2 calc_C.m

This script is run from fp_multileg to calculate the cost function. fsolve minimizes the cost functions to find a fixed point.

```
tdes = cval(5);
vdes = cval(6);

%Constraints for one step of front leg
N = min(ind(ind(:,4)>0,4)); %first RSP placement
vf = sqrt(xd(N)^2+yd(N)^2);
```

```
deltaf = atan2(-xd(N),yd(N)) - theta(N);


%Calculate constraints
C(1) = (vf-v0)/vdes;                    %vf = v0
C(2) = (deltaf + delta0)/(10*pi/180); %deltaf = -delta0
v = y(N)/t(N);
C(3) = (v - vdes)/vdes;                 %vavg = vdes
xdes = 0;
C(4) = (x(N)-xdes)/(tdes*vdes/10);     %x1 = 0
```

A.3.3 find_sigfam.m

This script is used to find a family of fixed points at a given heading angle and leg function combination. It finds fixed points for slopes at increments of 5 degrees.

```
% clear all, close all, %clc
% format long, format compact
global cval tstartstance fpx fpy FootAct fp_guess
global act_var act_con ldev IsPush fp
dval = [1e-6 1e-5 1e-1 1e-4];
%The following variables need to be either set in a batch
%collection script, or manually declared.
% loadfile = ['..\FixedPoints_multileg_varyvdesF\'...
%     'gaitfam_k150_PushA_varyphi_varysig_intvdes'...
%     '_delta20_ldevF016M013B0095_betaF025M055B120']
% sv_for = false; %Travelling forward in sigma
% cont = true;    %Continuing from the middle?
% leg_sw = [false true false];  %Which legs switch fnct
% Pushflg = 'B'   %Which gait type?
% deltaExt = .15; %What heading angle


%Load and plot starting point
% load temp
load(loadfile)
save temp
%Use to keep a portion of a saved file
% Nend = length(exflgtot)
% cvaltot = cvaltot(1:Nend-1,:);
% fptot = fptot(1:Nend-1,:);
% IsPushtot = IsPushtot(1:Nend-1,:);
% X0tot = X0tot(1:Nend-1,:);
% tstarttot = tstarttot(1:Nend-1,:);
% FootActtot = FootActtot(1:Nend-1,:);
% evaltot = evaltot(1:Nend-1,:);
% Ctot = Ctot(1:Nend-1,:);
% exflgtot = exflgtot(1:Nend-1,:);
% Fxmaxtot = Fxmaxtot(1:Nend-1,:);
% Fymaxtot = Fymaxtot(1:Nend-1,:);
% Fxcumtot = Fxcumtot(1:Nend-1,:);
% Fycumtot = Fycumtot(1:Nend-1,:);
% load temp2

%Choose which point to use as a starting guess
% n = 1;
% n = length(exflgtot)
if cont
  nA = 1:length(exflgtot);
  nA = nA(exflgtot == 1);
  if sv_for
    n = nA(end)
  else
    n = nA(1)
  end
else
  if sv_for
    n = 1;
  else
    n = length(exflgtot);
  end
end

cval = cvaltot(n,:);
fp = fptot(n,:);
IsPush = logical(IsPushtot(n,:));
%Switch legs pushing/pulling direction if required
```

```
IsPush(leg_sw) = ~IsPush(leg_sw)
ldev = abs(fp(4:6));
ldev(IsPush) = -ldev(IsPush)
fp(4:6) = ldev;
%Set heading angle
fp(2) = deltaExt; %delta

%Clear previous data (comment out if desired)
cvaltot = [];
fptot = [];
IsPushtot = [];
X0tot = [];
tstarttot = [];
FootActtot = [];
evaltot = [];
Ctot = [];
exflgtot = [];
Fxmaxtot = [];
Fymaxtot = [];
Fxcumtot = [];
Fycumtot = [];
Ctrantot = [];

%Find additional gaits
%These are the options for the fsolve routine
```

```
act_var = [1 13 14 15];
%var order = [v0 delta0 tdes 4-6=>ldev1-3 7-9=>beta1-3
%10-12=>Tdrv1-3 13-15=>phi1-3 ldev% beta%]
act_con = [1 2 3 4];
%con order = [Dv Ddelta vdes xdrift]
TypX = [.25 .5 .05 .01*ones(1,3) pi/2*ones(1,3) ...
  .05*ones(1,3) pi/4*ones(1,3) 1 1]';
options = optimset('MaxIter',100,'MaxFunEvals',1500,...
  'Display','iter','TolX',1e-7,'TolFun',1e-8,...
  'TypicalX',TypX(act_var),'ScaleProblem','Jacobian');
%Identify desired slopes in steps of 5 degrees
if sv_for
  sigind = cval(4)*180/pi:5:90
else
  sigind = cval(4)*180/pi:-5:0
end
if cont
  sigind = sigind(2:end)
end

for sig = sigind
  %Set parameters for new fixed point
  cval(4) = sig*pi/180;
  vdes0 = .35;
  vdes90 = .20;
```

```
vdes = vdes0 + sig/90*(vdes90 - vdes0)
tdes = inv(freq_fitter(vdes))/2
cval(5) = tdes;
fp(3) = tdes;
cval(37:39) = tdes;
fp(10:12) = tdes;
cval(6) = vdes;
fp_guess = fp;
%Call fixed point finder
disp(['Working on sigma = ' num2str(sig)])
[fp,C,exflg] = fsolve(@fp_multileg,fp_guess(act_var)...
  ,options);
fptemp = fp_guess;
fptemp(act_var) = fp;
fp = fptemp
%Set parameters based on the resulting fixed point
v0 = fp(1);
delta0 = fp(2);
tdes = fp(3);
ldev = [fp(4:6) fp(4:6)];
beta0 = [fp(7:9) -fp(7:9)];
Tdrv = fp(10:12);
phi = fp(13:15);
ldevX = fp(16);
betaX = fp(17);
```

```
%Correct cval
cval(5) = tdes;
cval(25:30) = ldevX*ldev;
betaC = beta0;
betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
cval(31:36) = betaC;
cval(37:39) = Tdrv;
cval(40:42) = phi;
%Set initial conditions
xd = -v0*sin(delta0);
yd = v0*cos(delta0);
X0 = [0 xd 0 yd 0 0];
%foot place finder
tsdes = zeros(3,1);
x_t = zeros(6,1);
y_t = zeros(6,1);
footplace_init(tsdes,X0,x_t,y_t)
%Simulate at new fixed point to gather forces, Ctran
[~,~,~,~,~,~,~,ind,Fx,Fy,~,~,~,~,~,~,Ct] ...
  = simgait_multileg(X0,1,0);
Fxmax = max(Fx(:,1:3));
Fxmin = min(Fx(:,1:3));
rec_min = abs(Fxmin) > abs(Fxmax);
Fxmax(rec_min) = Fxmin(rec_min);
Fymax = max(Fy(:,1:3));
```

```
Fymin = min(Fy(:,1:3));
rec_min = abs(Fymin) > abs(Fymax);
Fymax(rec_min) = Fymin(rec_min);
step = max(ind(:,4));
Fxcum = [max(sum(Fx(1:step,:),2)) ...
  min(sum(Fx(1:step,:),2))];
Fycum = [max(sum(Fy(1:step,:),2)) ...
  min(sum(Fy(1:step,:),2))];
e = findeig([v0 delta0 0 0],dval)
step = ind(ind(:,4)>0,4);
Ctran = Ct(step)

if sv_for
  %Store variables - forward travelling
  cvaltot = [cvaltot;cval];
  fptot = [fptot;fp];
  IsPushtot = [IsPushtot;IsPush];
  X0tot = [X0tot;X0];
  tstarttot = [tstarttot;tstartstance];
  FootActtot = [FootActtot;FootAct];
  evaltot = [evaltot;e'];
  Ctot = [Ctot;C];
  exflgtot = [exflgtot;exflg];
  Fxmaxtot = [Fxmaxtot;Fxmax];
  Fymaxtot = [Fymaxtot;Fymax];
  Fxcumtot = [Fxcumtot;Fxcum];
  Fycumtot = [Fycumtot;Fycum];
  Ctrantot = [Ctrantot;Ctran];
else
  %Store variables - backward travelling
  cvaltot = [cval;cvaltot];
  fptot = [fp;fptot];
  IsPushtot = [IsPush;IsPushtot];
  X0tot = [X0;X0tot];
  tstarttot = [tstartstance;tstarttot];
  FootActtot = [FootAct;FootActtot];
  evaltot = [e';evaltot];
  Ctot = [C;Ctot];
  exflgtot = [exflg;exflgtot];
  Fxmaxtot = [Fxmax;Fxmaxtot];
  Fymaxtot = [Fymax;Fymaxtot];
  Fxcumtot = [Fxcum;Fxcumtot];
  Fycumtot = [Fycum;Fycumtot];
  Ctrantot = [Ctran;Ctrantot];
end

%Save results
vdes = num2str(cval(6)*100);
delta = num2str(fp(2)*100);
if length(delta) == 1
```

```
    delta = ['0' delta];
  end
  savefile = ['..\FixedPoints_MultiLeg_varyvdesF\'...
    'gaitfam_k150_Push' Pushflg '_varyphi_varysig_'...
    'intvdes_delta' delta '_ldevF016M013B0095_'...
    'betaF025M055B120']
  save(savefile,'cvaltot','fptot','IsPushtot','X0tot',...
    'tstarttot','FootActtot','evaltot','Ctot',...
    'exflgtot','Fxmaxtot','Fymaxtot','Fxcumtot',...
    'Fycumtot','Ctrantot')
  if exflg ~= 1
    %If the last fixed point was no good
    break
  end
end
%Plot eigenvalues
figure,plot(cvaltot(:,4)*180/pi,sort(abs(evaltot),2))
```

A.4 Fixed Point Analysis

A.4.1 findeig.m

This function is used to calculate the eigenvalues of a given fixed point.

```
function [e] = findeig(init,dval)
%findeig calculates eigenvalues for the six legged model
```

```
%driven by ldev. Leg switch occurs on corresponding zero
%force for that leg only.  Discrete eigenvalues are based
%on simulating for two stances of the front leg.
%Eigenvalues are with respect to states of v, delta,
%theta, and thetadot.

global fpx fpy FootAct tstartstance
%save a copy of initial global variables
fpx0 = fpx;
fpy0 = fpy;
FootAct0 = FootAct;
tstartstance0 = tstartstance;
%Initialize variables
tsdes = zeros(3,1);
x_t = zeros(6,1);
y_t = zeros(6,1);

%Single dval input
if length(dval) == 1
  dval = ones(4,1)*dval;
end

vi = init(1);
deltai = init(2);
thetai = init(3);
```

```
thetadi = init(4);
deltai = -(mod(-(deltai+pi),2*pi)-pi);
% Enforces the range -pi < delta <= pi
thetai = -(mod(-(thetai+pi),2*pi)-pi);

%v variation
disp('Varying v')
[xdi,ydi] = states_xy(vi+dval(1),deltai,thetai);
ival = [0 xdi 0 ydi thetai thetadi];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_vU,deltaf_vU] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_vU = theta(indf2);
thetadf_vU = thetad(indf2);
final_vU = [vf_vU deltaf_vU thetaf_vU thetadf_vU];
%Lower point
[xdi,ydi] = states_xy(vi-dval(1),deltai,thetai);
ival = [0 xdi 0 ydi thetai thetadi];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
```

```
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_vL,deltaf_vL] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_vL = theta(indf2);
thetadf_vL = thetad(indf2);
final_vL = [vf_vL deltaf_vL thetaf_vL thetadf_vL];


%delta variation
disp('Varying delta')
[xdi,ydi] = states_xy(vi,deltai+dval(2),thetai);
ival = [0 xdi 0 ydi thetai thetadi];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_dU,deltaf_dU] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_dU = theta(indf2);
thetadf_dU = thetad(indf2);
final_dU = [vf_dU deltaf_dU thetaf_dU thetadf_dU];
%Lower point
[xdi,ydi] = states_xy(vi,deltai-dval(2),thetai);
ival = [0 xdi 0 ydi thetai thetadi];
```

```
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_dL,deltaf_dL] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_dL = theta(indf2);
thetadf_dL = thetad(indf2);
final_dL = [vf_dL deltaf_dL thetaf_dL thetadf_dL];

%theta variation
disp('Varying theta')
[xdi,ydi] = states_xy(vi,deltai,thetai+dval(3));
ival = [0 xdi 0 ydi thetai+dval(3) thetadi];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_tU,deltaf_tU] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_tU = theta(indf2);
thetadf_tU = thetad(indf2);
final_tU = [vf_tU deltaf_tU thetaf_tU thetadf_tU];
```

```
%Lower point
[xdi,ydi] = states_xy(vi,deltai,thetai-dval(3));
ival = [0 xdi 0 ydi thetai-dval(3) thetadi];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_tL,deltaf_tL] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_tL = theta(indf2);
thetadf_tL = thetad(indf2);
final_tL = [vf_tL deltaf_tL thetaf_tL thetadf_tL];

%thetad variation
disp('Varying theta''')
[xdi,ydi] = states_xy(vi,deltai,thetai);
ival = [0 xdi 0 ydi thetai thetadi+dval(4)];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_tdU,deltaf_tdU] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
```

```
thetaf_tdU = theta(indf2);
thetadf_tdU = thetad(indf2);
final_tdU = [vf_tdU deltaf_tdU thetaf_tdU thetadf_tdU];
%Lower point
[xdi,ydi] = states_xy(vi,deltai,thetai);
ival = [0 xdi 0 ydi thetai thetadi-dval(4)];
footplace_init(tsdes,ival,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind] ...
  = simgait_multileg(ival,2,0);
indF = ind(ind(:,1)>0,1);
indf2 = indF(2);
[vf_tdL,deltaf_tdL] ...
  = states_vd(xd(indf2),yd(indf2),theta(indf2));
thetaf_tdL = theta(indf2);
thetadf_tdL = thetad(indf2);
final_tdL = [vf_tdL deltaf_tdL thetaf_tdL thetadf_tdL];


%Calculate Jacobian
jacc = [(final_vU-final_vL)/(2*dval(1));
  (final_dU-final_dL)/(2*dval(2));
  (final_tU-final_tL)/(2*dval(3));
  (final_tdU-final_tdL)/(2*dval(4))];
e = eig(jacc);


%Reset global values
```

```
fpx = fpx0;
fpy = fpy0;
FootAct = FootAct0;
tstartstance = tstartstance0;
end

function [xd,yd] = states_xy(v,delta,theta)
%Translates states from v, delta to xd,yd
%(Only works for LSP delta formulation)

xd = -v*sin(delta+theta);
yd = v*cos(delta+theta);
end

function [v,delta] = states_vd(xd,yd,theta)
%Translates states from xd,yd to v, delta
%(Only works for LSP delta formulation)

v = sqrt(xd^2+yd^2);
delta = atan2(-xd,yd) - theta;
end
```

A.4.2 run_fp.m

This script is used to read and plot a fixed point from a file.

```
clear all, close all, %clc
format long, format compact
global cval tstartstance fpx fpy footact fp ispush

%load in a gaitfamily
% load ..\fixedpoints_mineig\gaitfam_pushB_varysig_...
%   k150_intvdes_delta15_varyldev_betaF025M055B120
load ..\fixedpoints_multileg_varyvdesf\gaitfam_k150...
  _pushB_varyphi_varysig_intvdes_delta05_...
  ldevF016M013B0095_betaF025M055B120

%select fixed point
n = 4;
n = length(exflgtot);
% cvaltot(:,4)'*180/pi
% % exflgtot'
% n = input('n: ')
cval = cvaltot(n,:);
fp = fptot(n,:);
ispush = ispushtot(n,:);
x0 = x0tot(n,:);
tstartstance = tstarttot(n,:);
footact = footacttot(n,:);

%load in a single fixed point
```

```
% load .\vary_ldev_only\fp_pushmb_sig20_vdes3167_...
%   delta15_k100_ldevf030m008b018_betaf025m055b120
% load temp
% ispush = [true false true]

%set parameters and plot the resulting motion
tsdes = zeros(3,1);
x_t = zeros(6,1);
y_t = zeros(6,1);
footplace_init(tsdes,x0,x_t,y_t)
[x,xd,y,yd,theta,thetad,t,ind,fx,fy,m,fpxtot,fpytot,...
  leq,lspr,e,ct] = simgait_multileg(x0,2,1);
v0 = fp(1);
delta0 = fp(2);
vdes = cval(6);
calc_C

%plot leg response with step starts indicated
titlestr = {'simulated lsp legs';...
  'simulated rsp legs';'';'';'';''};
ylabstr = {'front left leg';'front right leg';...
  'mid right leg';'mid left leg';...
  'back left leg';'back right leg'};
indnum = [1;4;2;5;3;6];
figure
```

```
for graphnum = 1:6
  subplot(3,2,graphnum)
  plot(t,lspr(:,indnum(graphnum)),'b-',...
    t,leq(:,indnum(graphnum)),'g--')
  title(titlestr(graphnum))
  ylabel(ylabstr(graphnum))
  xlim([0 4*tdes])
  hold on
  yspace = ylim;
  for n = 1:3
    plot(n*tdes*[1;1],[yspace],'r:')
  end
  text(tdes*[1 2 3],yspace(1)*[1 1 1],...
    {'t_d_e_s','2t_d_e_s','3t_d_e_s'},...
    'horizontalalignment','center',...
    'verticalalignment','top')
  set(gca,'ytick',[],'xtick',[])
end

%display transport cost
indft = ind(ind(:,4) > 0,4);
ct(indft(1))
```

A.4.3 plotgaits.m

This script is used to make contour plots from a set of gait families.

```
clear all, close all
%Plot all gait families important parameters on a contour
%plot

%Define which families to collect
sigA = (0:5:90)'
deltaA = 0:5:45;
sigM = sigA*ones(size(deltaA));
deltaM = ones(size(sigA))*deltaA;
Pushflg = {'A';'FB';'MB';'B'};
Pushsym = {'bs';'gv';'m^';'ro'};
l0 = [.032 .026 .019];
mnldev = .2*l0
mxldev = .9*l0
mg = 0.0025*9.81;

%Set up figure handles
eplt = figure;
Cplt = figure;
robplt = figure;
% for cnt = 1:3
%     ldevplt(cnt) = figure;
% end
```

```
v0plt = figure;
for cnt = 1:3
  phiplt(cnt) = figure;
end
for cnt = 1:4
  Flatplt(cnt) = figure;
end
for cnt = 1:4
  Ffaplt(cnt) = figure;
end
for Pind = 1:4
  figure(eplt)
  eval(['e' char(Pushflg(Pind)) ...
    'plt = subplot(2,2,Pind);'])
  figure(Cplt)
  eval(['C' char(Pushflg(Pind)) ...
    'plt = subplot(2,2,Pind);'])
  figure(v0plt)
  eval(['v0' char(Pushflg(Pind)) ...
    'plt = subplot(2,2,Pind);'])
  figure(robplt)
  eval(['rob' char(Pushflg(Pind)) ...
    'plt = subplot(2,2,Pind);'])
  for cnt = 1:3
    %     figure(ldevplt(cnt))
    %     eval(['ldev' char(Pushflg(Pind)) ...
    %       'plt(cnt) = subplot(2,2,Pind);'])
    figure(phiplt(cnt))
    eval(['phi' char(Pushflg(Pind)) ...
      'plt(cnt) = subplot(2,2,Pind);'])
  end
  for cnt = 1:4
    figure(Flatplt(cnt))
    eval(['Flat' char(Pushflg(Pind)) ...
      'plt(cnt) = subplot(2,2,Pind);'])
    figure(Ffaplt(cnt))
    eval(['Ffa' char(Pushflg(Pind)) ...
      'plt(cnt) = subplot(2,2,Pind);'])
  end
end

%Collect data
for Pind = 1:4
  %Declare empty data sets
  eval(['emax' char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
  eval(['Ctran' char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
  eval(['nsteps' char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
```

```
eval(['v0' char(Pushflg(Pind)) ...
  '_M = NaN(length(sigA),length(deltaA));'])
for cnt = 1:3
  eval(['phi' num2str(cnt) char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
  eval(['ldev' num2str(cnt) char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
  eval(['Flat' num2str(cnt) char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
  eval(['Ffa' num2str(cnt) char(Pushflg(Pind)) ...
    '_M = NaN(length(sigA),length(deltaA));'])
end
eval(['FlatS_' char(Pushflg(Pind)) ...
  '_M = NaN(length(sigA),length(deltaA));'])
eval(['FfaS_' char(Pushflg(Pind)) ...
  '_M = NaN(length(sigA),length(deltaA));'])
%Step through delta
for dind = 1:length(deltaA)
  deltaS = num2str(deltaA(dind));
  if deltaA(dind) < 10
    deltaS = ['0' deltaS];
  end
  loadfile= ['gaitfam_k150_Push' char(Pushflg(Pind))...
    '_varyphi_varysig_intvdes_delta' deltaS ...
    '_ldevF016M013B0095_betaF025M055B120'];
```

```
if exist([loadfile '.mat'],'file')
  %Prevents a load command for a nonexistent file
  loadfile
  load(loadfile)
  %Step through sigma
  for sigind = 1:length(sigA)
    %check for a valid data point that matches the
    %sigma required
    ind = 1:length(exflgtot);
    log_ind = (abs(cvaltot(:,4)*180/pi - ...
      sigA(sigind))<10000*eps);
    val_ind = ind(log_ind);
    %Remove non fp
    if exflgtot(val_ind)~=1
      val_ind = [];
    end
    %remove ldev violations
    for fpnum = 4:6
      if max(abs(fptot(val_ind,fpnum))) ...
          > mxldev(fpnum-3)
        val_ind = [];
      end
      if min(abs(fptot(val_ind,fpnum))) ...
          < mnldev(fpnum-3)
        val_ind = [];
```

```
          end
        end
        %Remove other fixed points known to be bad
%         if Pind == 4 && sigind <= 4 && dind == 8
%           warning('manual remove')
%           val_ind = [];
%         end
        %Collect data for valid points
        if val_ind
          %Find maximum eval
          etemp = abs(evaltot(val_ind,:));
          for cnt = 1:2
            [~,rmv] = min(abs(etemp - 1));
            etemp(rmv) = 0;
          end
          eval(['emax' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = max(etemp);'])
          eval(['Ctran' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = Ctrantot(val_ind);'])
          indrob = (percimpA == .5);
          nsteps = numstepstot(val_ind,indrob);
          if ~recoveredtot(val_ind,indrob)
            nsteps = 100;
          end
          eval(['nsteps' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = nsteps;'])
          eval(['v0' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = fptot(val_ind,1);'])
          for cnt = 1:3
            eval(['phi' num2str(cnt) char(Pushflg(...
              Pind)) '_M(sigind,dind) = ' ...
              'fptot(val_ind,cnt+12)*180/pi;'])
            eval(['ldev' num2str(cnt) char(Pushflg(...
              Pind)) '_M(sigind,dind) = ' ...
              'abs(fptot(val_ind,cnt+3));'])
            eval(['Flat' num2str(cnt) char(Pushflg(...
              Pind)) '_M(sigind,dind) = ' ...
              'Fxmaxtot(val_ind,cnt);'])
            eval(['Ffa' num2str(cnt) char(Pushflg(...
              Pind)) '_M(sigind,dind) = ' ...
              'Fymaxtot(val_ind,cnt);'])
          end
          eval(['FlatS_' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = Fxcumtot(val_ind,1);'])
          eval(['FfaS_' char(Pushflg(Pind)) ...
            '_M(sigind,dind) = Fycumtot(val_ind,1);'])
        end
      end
    else
      disp([loadfile ' does not exist'])
```

```
    end
  end
end

%Plot data
for Pind = 1:4
  %Stability
  eval(['contourf(e' char(Pushflg(Pind)) ...
    'plt,sigM,deltaM/100,emax' char(Pushflg(Pind)) ...
    '_M,''LineStyle'',''none'',''LevelStep'',.02)'])
  eval(['colorbar(''peer'',e' ...
    char(Pushflg(Pind)) 'plt)'])
  eval(['title(e' char(Pushflg(Pind)) 'plt,''Push' ...
    char(Pushflg(Pind)) ' Gaits'')'])
  eval(['xlabel(e' char(Pushflg(Pind)) ...
    'plt,''Incline Slope: \sigma (\circ)'')'])
  eval(['ylabel(e' char(Pushflg(Pind)) ...
    'plt,''Heading: \delta_0 (rad)'')'])
  eval(['caxis(e' char(Pushflg(Pind)) ...
    'plt,[0 .7])'])
  set(eplt,'name','Stability','numbertitle','off')
  %Transport Cost
  eval(['contourf(C' char(Pushflg(Pind)) ...
    'plt,sigM,deltaM/100,Ctran' char(Pushflg(Pind)) ...
    '_M,''LineStyle'',''none'',''LevelStep'',.1)'])
```

```
eval(['colorbar(''peer'',C' ...
  char(Pushflg(Pind)) 'plt)'])
eval(['title(C' char(Pushflg(Pind)) 'plt,''Push' ...
  char(Pushflg(Pind)) ' Gaits'')'])
eval(['xlabel(C' char(Pushflg(Pind)) ...
  'plt,''Incline Slope: \sigma (\circ)'')'])
eval(['ylabel(C' char(Pushflg(Pind)) ...
  'plt,''Heading: \delta_0 (rad)'')'])
eval(['caxis(C' char(Pushflg(Pind)) ...
  'plt,[0 2])'])
set(Cplt,'name','Transport Cost','numbertitle','off')
%Initial Velocity
eval(['contourf(v0' char(Pushflg(Pind)) ...
  'plt,sigM,deltaM/100,v0' char(Pushflg(Pind)) ...
  '_M,''LineStyle'',''none'',''LevelStep'',.02)'])
eval(['colorbar(''peer'',v0' ...
  char(Pushflg(Pind)) 'plt)'])
eval(['title(v0' char(Pushflg(Pind)) 'plt,''Push' ...
  char(Pushflg(Pind)) ' Gaits'')'])
eval(['xlabel(v0' char(Pushflg(Pind)) ...
  'plt,''Incline Slope: \sigma (\circ)'')'])
eval(['ylabel(v0' char(Pushflg(Pind)) ...
  'plt,''Heading: \delta_0 (rad)'')'])
eval(['caxis(v0' char(Pushflg(Pind)) 'plt,[0 .4])'])
set(v0plt,'name','Initial Velocity',...
    'numbertitle','off')
  for cnt = 1:3
    %phi
    eval(['contourf(phi' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),sigM,deltaM/100,phi' ...
      num2str(cnt) char(Pushflg(Pind)) ...
      '_M,''LineStyle'',''none'',''LevelStep'',5)'])
    eval(['colorbar(''peer'',phi' char(Pushflg(Pind)) ...
      'plt(' num2str(cnt) '))'])
    eval(['title(phi' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Push' char(Pushflg(Pind)) ...
      ' Gaits'')'])
    eval(['xlabel(phi' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Incline Slope:' ...
      '\sigma (\circ)'')'])
    eval(['ylabel(phi' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Heading: \delta_0 (rad)'')'])
    eval(['caxis(phi' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),[-60 60])'])
    eval(['set(phiplt(' num2str(cnt) ...
      '),''name'',''phi' num2str(cnt) ...
      ''',''numbertitle'',''off'')'])
%     %ldev
%     eval(['contourf(ldev' char(Pushflg(Pind)) 'plt('...
%       num2str(cnt) '),sigM,deltaM/100,ldev' ...
%       num2str(cnt) char(Pushflg(Pind)) ...
%       '_M,''LineStyle'',''none'',''LevelStep'',' ...
%       num2str((mxldev(cnt) - mnldev(cnt))/20) ')'])
%     eval(['colorbar(''peer'',ldev' char(Pushflg( ...
%       Pind)) 'plt(' num2str(cnt) '))'])
%     eval(['title(ldev' char(Pushflg(Pind)) 'plt(' ...
%       num2str(cnt) '),''Push' char(Pushflg(Pind)) ...
%       ' Gaits'')'])
%     eval(['xlabel(ldev' char(Pushflg(Pind)) 'plt(' ...
%       num2str(cnt) '),''Incline Slope:' ...
%       '\sigma (\circ)'')'])
%     eval(['ylabel(ldev' char(Pushflg(Pind)) 'plt(' ...
%       num2str(cnt) '),''Heading: \delta_0 (rad)'')'])
%     eval(['caxis(ldev' char(Pushflg(Pind)) 'plt(' ...
%       num2str(cnt) '),[' num2str(mnldev(cnt)) ...
%       ' ' num2str(mxldev(cnt)) '])'])
%     eval(['set(ldevplt(' num2str(cnt) ...
%       '),''name'',''ldev' num2str(cnt) ...
%       ''',''numbertitle'',''off'')'])
    %Flati
    eval(['contourf(Flat' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),sigM,deltaM/100,Flat' ...
      num2str(cnt) char(Pushflg(Pind)) ...
      '_M/mg,''LineStyle'',''none'',''LevelStep'',.1)'])
    eval(['colorbar(''peer'',Flat' char(Pushflg(Pind))...
      'plt(' num2str(cnt) '))'])
    eval(['title(Flat' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Norm. Lateral Force F_' ...
      num2str(cnt) ' of a Push' char(Pushflg(Pind)) ...
      ' Gait Family - Opt. for Stab.'')'])
    eval(['xlabel(Flat' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Incline Slope:' ...
      ' \sigma (\circ)'')'])
    eval(['ylabel(Flat' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Heading Angle:' ...
      ' \delta_0 (rad)'')'])
    eval(['caxis(Flat' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),[-1 1])'])
    %Ffai
    eval(['contourf(Ffa' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),sigM,deltaM/100,Ffa' ...
      num2str(cnt) char(Pushflg(Pind)) ...
      '_M/mg,''LineStyle'',''none'',''LevelStep'',.1)'])
    eval(['colorbar(''peer'',Ffa' char(Pushflg(Pind)) ...
      'plt(' num2str(cnt) '))'])
    eval(['title(Ffa' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Norm. For/Aft Force F_' ...
      num2str(cnt) ' of a Push' char(Pushflg(Pind)) ...
      ' Gait Family - Opt. for Stab.'')'])
    eval(['xlabel(Ffa' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Incline Slope:' ...
      ' \sigma (\circ)'')'])
    eval(['ylabel(Ffa' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),''Heading Angle:' ...
      ' \delta_0 (rad)'')'])
    eval(['caxis(Ffa' char(Pushflg(Pind)) 'plt(' ...
      num2str(cnt) '),[-1 1.5])'])
  end
  %Sum(Flat)
  eval(['contourf(Flat' char(Pushflg(Pind)) 'plt(4)' ...
    ',sigM,deltaM/100,FlatS_' char(Pushflg(Pind))...
    '_M/mg,''LineStyle'',''none'',''LevelStep'',.1)'])
  eval(['colorbar(''peer'',Flat' ...
    char(Pushflg(Pind)) 'plt(4))'])
  eval(['title(Flat' char(Pushflg(Pind)) ...
    'plt(4),''Norm. Lateral Force \SigmaF of a Push' ...
    char(Pushflg(Pind)) ...
    ' Gait Family - Opt. for Stab.'')'])
  eval(['xlabel(Flat' char(Pushflg(Pind)) ...
    'plt(4),''Incline Slope: \sigma (\circ)'')'])
  eval(['ylabel(Flat' char(Pushflg(Pind)) ...
    'plt(4),''Heading Angle: \delta_0 (rad)'')'])
  eval(['caxis(Flat' char(Pushflg(Pind)) ...
    'plt(4),[0 1.5])'])
  %Sum(Ffa)
```

```
  eval(['contourf(Ffa' char(Pushflg(Pind)) ...
    'plt(4),sigM,deltaM/100,FfaS_' char(Pushflg(Pind))...
    '_M/mg,''LineStyle'',''none'',''LevelStep'',.1)'])
  eval(['colorbar(''peer'',Ffa' ...
    char(Pushflg(Pind)) 'plt(4))'])
  eval(['title(Ffa' char(Pushflg(Pind)) ...
    'plt(4),''Norm. For/Aft Force \SigmaF of a Push' ...
    char(Pushflg(Pind)) ...
    ' Gait Family - Opt. for Stab.'')'])
  eval(['xlabel(Ffa' char(Pushflg(Pind)) ...
    'plt(4),''Incline Slope: \sigma (\circ)'')'])
  eval(['ylabel(Ffa' char(Pushflg(Pind)) ...
    'plt(4),''Heading Angle: \delta_0 (rad)'')'])
  eval(['caxis(Ffa' char(Pushflg(Pind)) 'plt(4),[0 2])'])
end

%xo plot
figure, hold on
axis([-2.5 92.5 -.025 .475])
title('Gait Existence - Opt. for Stab.')
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading Angle: \delta_0 (rad)')
MrkSz = [9 5 5 7];
for Pind = 1:4
  eval(['val_ind = ~isnan(emax' ...
    char(Pushflg(Pind)) '_M);'])
  eval(['legxo(Pind) = plot(sigM(val_ind),' ...
    'deltaM(val_ind)/100,''' char(Pushsym(Pind)) ...
    ''',''MarkerSize'',' num2str(MrkSz(Pind)) ')'])
end
for n = 1:4
  labxo(n) = {['Push ' char(Pushflg(n))]};
end
legend(legxo,char(labxo),'Location','SouthOutside', ...
  'Orientation','Horizontal');

Pushsym = {'ks';'kv';'k^';'ko'};
%Find optimized eval
emaxtot = ones(size(emaxA_M));
emaxind = zeros(size(emaxA_M));
Ctrantot = NaN(size(emaxA_M));
v0tot = NaN(size(v0A_M));
phi1tot = NaN(size(phi1A_M));
phi2tot = phi1tot;
phi3tot = phi1tot;
nstepstot = NaN(size(nstepsA_M));
Flat1tot = NaN(size(Flat1A_M));
Flat2tot = Flat1tot;
Flat3tot = Flat1tot;
FlatS_tot = Flat1tot;
```

```
Ffa1tot = Flat1tot;
Ffa2tot = Flat1tot;
Ffa3tot = Flat1tot;
FfaS_tot = Flat1tot;
for Pind = 1:4
  eval(['rep_ind = emax' char(Pushflg(Pind)) ...
    '_M < emaxtot;'])
  eval(['emaxtot(rep_ind) = emax' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  emaxind(rep_ind) = Pind;
  eval(['Ctrantot(rep_ind) = Ctran' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['v0tot(rep_ind) = v0' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['phi1tot(rep_ind) = phi1' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['phi2tot(rep_ind) = phi2' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['phi3tot(rep_ind) = phi3' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['nstepstot(rep_ind) = nsteps' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  for lnum = 1:3
    eval(['Flat' num2str(lnum) 'tot(rep_ind) = Flat' ...
      num2str(lnum) char(Pushflg(Pind)) '_M(rep_ind);'])
    eval(['Ffa' num2str(lnum) 'tot(rep_ind) = Ffa' ...
      num2str(lnum) char(Pushflg(Pind)) '_M(rep_ind);'])
  end
  eval(['FlatS_tot(rep_ind) = FlatS_' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
  eval(['FfaS_tot(rep_ind) = FfaS_' ...
    char(Pushflg(Pind)) '_M(rep_ind);'])
end
emaxtot(~emaxind) = NaN;
%or
% %Find optimized Ctran
% Ctrantot = 10*ones(size(CtranA_M));
% Ctranind = zeros(size(CtranA_M));
% emaxtot = NaN(size(emaxA_M));
% for Pind = 1:4
%    eval(['rep_ind = Ctran' char(Pushflg(Pind)) ...
%      '_M < Ctrantot;'])
%    eval(['Ctrantot(rep_ind) = Ctran' ...
%      char(Pushflg(Pind)) '_M(rep_ind);'])
%    Ctranind(rep_ind) = Pind;
%    eval(['emaxtot(rep_ind) = emax' ...
%      char(Pushflg(Pind)) '_M(rep_ind);'])
% end
% Ctrantot(~Ctranind) = NaN;
% Also need to replace emaxind with Ctranind everywhere
```

```
%Plot emax
figure
contourf(sigM,deltaM/100,emaxtot,...
  'LineStyle','none','LevelStep',.02)
colorbar
caxis([0 .7])
title('Maximum Eigenvalue of Fixed Points')
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading Angle: \delta_0 (rad)')
axis([-2.5 92.5 -.025 .475])
hold on
for Pind = 1:4
  eval(['val_ind = ~isnan(emax' ...
    char(Pushflg(Pind)) '_M);'])
  lege(Pind) = plot(sigM(val_ind),...
    deltaM(val_ind)/100,char(Pushsym(Pind)));
end
labe = {'Push All','Push F,B','Push M,B','Push B'}
legend(lege,char(labe),'Location','SouthOutside', ...
  'Orientation','Horizontal');
%Plot Ctran
figure
contourf(sigM,deltaM/100,Ctrantot,...
  'LineStyle','none','LevelStep',.1)
colorbar
```

```
caxis([0 2])
title(['Transport Cost of a Gait Family' ...
  ' With Fixed l_{dev}'])
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading Angle: \delta_0 (rad)')
axis([-2.5 92.5 -.025 .475])
hold on
for sigind = 1:length(sigA)
  for dind = 1:length(deltaA)
    if emaxind(sigind,dind)
      plot(sigM(sigind,dind),deltaM(sigind,dind)/100,...
        char(Pushsym(emaxind(sigind,dind))))
    end
  end
end
%Plot robustness
figure
contourf(sigM,deltaM/100,nstepstot, ...
  'LineStyle','none','LevelStep',.5)
colorbar
caxis([0 50])
title('Steps to Recover from a Lateral Impulse')
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading Angle: \delta_0 (rad)')
axis([-2.5 92.5 -.025 .475])
```

```
hold on
for Pind = 1:4
  eval(['val_ind = ~isnan(emax' ...
    char(Pushflg(Pind)) '_M);'])
  legr(Pind) = plot(sigM(val_ind), ...
    deltaM(val_ind)/100,char(Pushsym(Pind)))
end
labr = {'Push All','Push F,B','Push M,B','Push B'}
legend(legr,char(labr),'Location','SouthOutside', ...
  'Orientation','Horizontal');
%Plot v0
figure, subplot(2,2,1)
contourf(sigM,deltaM/100,v0tot, ...
  'LineStyle','none','LevelStep',.02)
colorbar
caxis([0 .4])
title('Initial Velocity (m/s)')
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading: \delta_0 (rad)')
axis([-2.5 92.5 -.025 .475])
hold on
for legnum = 1:3
  subplot(2,2,legnum+1)
  eval(['contourf(sigM,deltaM/100,phi' num2str(legnum)...
    'tot,''LineStyle'',''none'',''LevelStep'',5);'])
  colorbar
  caxis([-75 75])
  title(['Phase Angle (\circ) Leg: ' num2str(legnum)])
  xlabel('Incline Slope: \sigma (\circ)')
  ylabel('Heading: \delta_0 (rad)')
  axis([-2.5 92.5 -.025 .475])
  hold on
end
%Plot Flat
mg = .0025*9.81
figure
for lnum = 1:3
  subplot(2,2,lnum)
  eval(['contourf(sigM,deltaM/100,Flat' num2str(lnum) ...
    'tot/mg,''LineStyle'',''none'',''LevelStep'',.05);'])
  colorbar
  caxis([-1 1])
  title(['Normalized Lateral Force: ' num2str(lnum)])
  xlabel('Incline Slope: \sigma (\circ)')
  ylabel('Heading: \delta_0 (rad)')
  axis([-2.5 92.5 -.025 .475])
end
subplot(2,2,4)
contourf(sigM,deltaM/100,FlatS_tot/mg, ...
  'LineStyle','none','LevelStep',.05);
```

```
colorbar
caxis([-.5 2])
title('Normalized Lateral Force: Sum')
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading: \delta_0 (rad)')
%Plot Ffa
mg = .0025*9.81
figure
for lnum = 1:3
  subplot(2,2,lnum)
  eval(['contourf(sigM,deltaM/100,Ffa' num2str(lnum) ...
    'tot/mg,''LineStyle'',''none'',''LevelStep'',.05);'])
  colorbar
  caxis([-1 1])
  title(['Normalized For/Aft Force: ' num2str(lnum)])
  xlabel('Incline Slope: \sigma (\circ)')
  ylabel('Heading: \delta_0 (rad)')
  axis([-2.5 92.5 -.025 .475])
end
subplot(2,2,4)
contourf(sigM,deltaM/100,FfaS_tot/mg, ...
  'LineStyle','none','LevelStep',.05);
colorbar
caxis([-.5 2])
title('Normalized For/Aft Force: Sum')
```

```
xlabel('Incline Slope: \sigma (\circ)')
ylabel('Heading: \delta_0 (rad)')
```

A.5 Robustness Checking

A.5.1 simgait_multileg_latimp.m

This function is similar to the simulator (simgait_multileg), however a lateral impulse is added and a recovery checker.

```
function [xtot,xdtot,ytot,ydtot,thetatot,thetadtot,...
  timetot,indbrktot,Fxtot,Fytot,Mtot,fpxtot,fpytot,...
  leqtot,lsprtot,tot_ener,Ctrantot] = ...
  simgait_multileg_latimp(X0,numperiod,plotflag)
% This function simulates the number of gaits specified
% in numperiod based on the initial conditions X0. States
% are x, xdot, y, ydot, theta, and thetadot. Plots are
% included if plotflag = 1.

global cval tstartstance fpx fpy FootAct
%The following values are input as global variables but
%local copies need to be stored since their values get
%overwritten
tstartstance0 = tstartstance;
fpx0 = fpx;
fpy0 = fpy;
```

```
FootAct0 = FootAct;
options = odeset('Events',@stancebreak, ...
  'InitialStep',.000001);
%Read in global cval
kval = cval(1);
m = cval(2);
Ival = cval(3);
% sigma = cval(4);
tdes = cval(5);
% vdes = cval(6);
% Omega = pi/tdes;
% g = 9.81;

% Initialize storage variables here
xtot = [];    %States
xdtot = [];
ytot = [];
ydtot = [];
thetatot = [];
thetadtot = [];
timetot = [];
Fxtot = [];   %Forces and Moments
Fytot = [];
Mtot = [];
leqtot = [];  %Spring length parameters
```

```
lsprtot = [];
Ctran0 = 0;   %Transport Cost calculator
Ctrantot = [];
fpxtot = fpx; %Foot placements
fpytot = fpy;
indbrktot = zeros(1,6); %Indices of step breaks
%Set stride counters (-1 for not just placed)
stride = -ones(1,6);
end_ft = 1; %Which foot counter is used for termination
stride(tstartstance == tstartstance(end_ft)) = 0;
tsimstart = tstartstance(end_ft);
indbrktot(tstartstance == tsimstart) = 1;

%Declare flags for impulse recovery checks
firstrecovery = false;      %Was the fixed point
  % recovered during the last stance?
terminate = false;          %Should the simulation
  % loop be terminated?
recovered = false;          %Has the fixed point
  % recovered permanently
difftrack = NaN(1,4);
X0init = X0;

% Simulate stance phases
while stride(end_ft) < numperiod
```

```
tspan = [tsimstart tsimstart+tdes];
[T,X,te,~,ie] = ...
  ode45(@stancedynamics,tspan,[X0 Ctran0],options);
Ctran = X(:,7);
X = X(:,1:6);
% Store relevant data
xtot = [xtot;X(:,1)];
xdtot = [xdtot;X(:,2)];
ytot = [ytot;X(:,3)];
ydtot = [ydtot;X(:,4)];
thetatot = [thetatot;X(:,5)];
thetadtot = [thetadtot;X(:,6)];
timetot = [timetot;T];
Ctrantot = [Ctrantot;Ctran];
%Store index of events
indbrk = zeros(1,6);
for n = 1:length(ie)
  if te(n) == T(length(T))
    if ie(n) <= 6
      %Store a negative value for liftoff stance-breaks
      indbrk(ie(n)) = -length(timetot);
    else
      %and a positive for foot down stance breaks
      indbrk(ie(n)-6) = length(timetot);
    end
  end
end
tstarttol = 1e-10;
indbrk((tstartstance + 2*tdes >= T(length(T)))&...
  (tstartstance + 2*tdes < T(length(T))+tstarttol)) ...
  = length(timetot);
indbrktot = [indbrktot;indbrk];
%Calculate parameters that need stored at every time
for n = 1:length(T)
  [Fx,Fy,M,leq,lspr] = legforces(T(n),X(n,:));
  leq(~FootAct) = 0;
  lspr(~FootAct) = 0;
  Fxtot = [Fxtot;Fx];
  Fytot = [Fytot;Fy];
  Mtot = [Mtot;M];
  leqtot = [leqtot;leq];
  lsprtot = [lsprtot;lspr];
end
%Set up for next stride
ind = length(T);
X0 = X(ind,:);
tstartstance(indbrk > 0) = T(ind);
FootAct(indbrk > 0) = true;
FootAct(indbrk < 0) = false;
[fpxtemp,fpytemp] = footplace(X0);
```

```
fpx(indbrk > 0) = fpxtemp(indbrk > 0);
fpy(indbrk > 0) = fpytemp(indbrk > 0);
tsimstart = T(ind);
%Store new foot placement points
fpxtot = [fpxtot;fpx];
fpytot = [fpytot;fpy];
stride(indbrk > 0) = stride(indbrk > 0) + 1;
%Check for termination on lateral impulse
%if the lead foot or its counterpart was just placed
if indbrk(end_ft) > 0 || indbrk(opp_ft) > 0
  % Check to see how close we are to the fixed point.
  % For d = 0, we only have v and delta, but I will use
  % xd, yd instead (should contain same info).
  scaleval = [-cval(6)*tan(.15) cval(6)];
  if indbrk(end_ft) > 0
    diffinfp = (X0init([2 4]) - X(size(X,1),[2 4])) ...
      ./scaleval;
  else
    %For opposite foot, a fixed point occurs when xd is
    %equal but opposite
    diffinfp = [X0init(2)+X(size(X,1),2) ...
      X0init(4)-X(size(X,1),4)]./scaleval;
  end
  maxdiffinfp = max(abs(diffinfp));
  %Track last four to see if it's going to the wrong
```

```
%fixed point
difftrack(1:3) = difftrack(2:4);
difftrack(4) = maxdiffinfp;
% Check to see if this is within 1% of the fixed
% point value. Want to make sure, however, that it
% isn’t just a random event. Want to only quit when
% we’re sure that no subsequent steps go outside of
% the region. Assume that if the following stance
% phase is within the desired threshold, then it will
% stay there
if firstrecovery
  % If we are in this part, then this will be at
  % least the second time that the value is within 1%
  if maxdiffinfp < maxdiffinfpold ...
      || maxdiffinfp < .001
    %This is a true recovery if the diff has de-
    %creased or if the diff while increasing has
    %stayed below 1/10 the tolerance
    recovered = true;
    terminate = true;
    disp(’The previous and current differences are:’)
    disp([maxdiffinfpold maxdiffinfp])
  else
    if maxdiffinfp >= .01
      %Outside of recovered area
```

```
        firstrecovery = false;
      else
        %Remains inside recovered area but growing
        maxdiffinfpold = maxdiffinfp;
      end
    end;
  else
    if maxdiffinfp < .01
      %Less than 1% difference from fixed point
      %                stride
      firstrecovery = true;
      maxdiffinfpold = maxdiffinfp;
    else
      %stop if we are at an incorrect fixed point
      atfp1 = abs(difftrack(3) - difftrack(1)) < .002;
      atfp2 = abs(difftrack(4) - difftrack(2)) < .002;
      wrongfp = difftrack(4) > 1;
      if atfp1 && atfp2 && wrongfp
        disp(['Returned to wrong fixed point:' ...
          ' Tracking diff:'])
        disp(num2str(difftrack))
        recovered = false;
        terminate = true;
      end
    end
    end
  end
  if terminate
    break
  end
end
numsteps = stride(end_ft)+stride(opp_ft);
%Reset initial values
FootAct = FootAct0;
fpx = fpx0;
fpy = fpy0;
tstartstance = tstartstance0;

%Calculate energy for plotting
kin_ener = m*(xdtot.^2 + ydtot.^2)/2+Ival*thetadtot.^2/2;
pot_ener = sum(kval*(lsprtot - leqtot).^2/2,2);
tot_ener = kin_ener+pot_ener;
% Plot data if required
if plotflag
  figure, hold on
  N = size(fpxtot,1);
  for n = 1:N
    %Foot placement points
    plot(fpxtot(n,:),fpytot(n,:),'x',...
      'MarkerEdgeColor',[(N-n)/(N-1) 0 (n-1)/(N-1)])
```

```
end
%Center of mass trajectory
plot(xtot,ytot,'k-')
axis image,axis equal
markstyle = ['b+';'gx';'r*';'c+';'yx';'m*'];
%Stance break markers
for n = 1:6
  indt = indbrktot(indbrktot(:,n)>0,n);
  plot(xtot(indt),ytot(indt),markstyle(n,:),...
    'LineWidth',1,'MarkerSize',8)
end
figure
%Important states
subplot(2,2,1),plot(timetot,xtot), hold on
title('x')
subplot(2,2,2),plot(timetot,ytot), hold on
title('y')
subplot(2,2,3),plot(timetot,thetatot*180/pi), hold on
title('\theta')
ylabel('\circ')
for n = 1:6
  indt = indbrktot(indbrktot(:,n)>0,n);
  subplot(2,2,1),plot(timetot(indt),xtot(indt),...
    markstyle(n,:),'LineWidth',2,'MarkerSize',8)
  subplot(2,2,2),plot(timetot(indt),ytot(indt),...
    markstyle(n,:),'LineWidth',2,'MarkerSize',8)
  subplot(2,2,3),plot(timetot(indt),thetatot(indt)*...
    180/pi,markstyle(n,:),'LineWidth',2,'MarkerSize',8)
end
figure
%Forces and Moments
COL_ORD=[0 0 1;0 1 0;1 0 0;0 1 1;1 1 0;1 0 1;.6 .6 .6];
subplot(2,2,1), hold on, title('Fx')
plot(timetot,sum(Fxtot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
subplot(2,2,2), hold on, title('Fy')
plot(timetot,sum(Fytot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
subplot(2,2,3), hold on, title('M')
plot(timetot,sum(Mtot,2),'Color',COL_ORD(7,:),...
  'LineStyle','-.')
subplot(2,2,4), hold on
for n = 1:6
  indt = indbrktot(indbrktot(:,n)>0,n);
  subplot(2,2,1), plot(timetot,Fxtot(:,n),...
    'Color',COL_ORD(n,:))
  plot(timetot(indt),Fxtot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,2), plot(timetot,Fytot(:,n),...
    'Color',COL_ORD(n,:))
  plot(timetot(indt),Fytot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,3), plot(timetot,Mtot(:,n),...
    'Color',COL_ORD(n,:));
  plot(timetot(indt),Mtot(indt,n),markstyle(n,:),...
    'LineWidth',2,'MarkerSize',8)
  subplot(2,2,4), plot(0,0,'Color',COL_ORD(n,:))
end
plot(0,0,'Color',COL_ORD(7,:),'LineStyle','-.')
set(gca,'Visible','off')
legend('1','2','3','4','5','6','sum','Location','West')
figure
%Leg lengths
subplot(3,2,1)
plot(timetot,lsprtot(:,1),timetot,leqtot(:,1),':')
title(['Actual and Equilibrium Spring Lengths'...
  ' of a Simulated Cockroach'])
ylabel('Front lng. (m)')
subplot(3,2,3)
plot(timetot,lsprtot(:,2),timetot,leqtot(:,2),':')
ylabel('Mid lng. (m)')
subplot(3,2,5)
plot(timetot,lsprtot(:,3),timetot,leqtot(:,3),':')
xlabel('Time (s)')
ylabel('Back lng. (m)')
```

```
  subplot(3,2,2)
  plot(timetot,lsprtot(:,4),timetot,leqtot(:,4),':')
  title(['Actual and Equilibrium Spring Lengths'...
    ' of a Simulated Cockroach'])
  ylabel('Front lng. (m)')
  subplot(3,2,4)
  plot(timetot,lsprtot(:,5),timetot,leqtot(:,5),':')
  ylabel('Mid lng. (m)')
  subplot(3,2,6)
  plot(timetot,lsprtot(:,6),timetot,leqtot(:,6),':')
  xlabel('Time (s)')
  ylabel('Back lng. (m)')
  figure
  %Energy
  plot(timetot,tot_ener,timetot,kin_ener,'r:',...
    timetot,pot_ener,'b-.')
  legend('Total Energy','Kinetic Energy',...
    'Potential Energy','Location','NorthWest')
  title('Total Energy of a Simulated Cockroach')
  xlabel('Time (s)')
  ylabel('Energy (J)')
end
end

%%% end main function
%%% stancedynamics function

function Xd = stancedynamics(t,X)

global cval percimp

%Call cval
% kval = cval(1);
m = cval(2);
Ival = cval(3);
sigma = cval(4);
tdes = cval(5);
vdes = cval(6);
% Omega = pi/tdes;
g = 9.81;

% read current state
% x = X(1);
xd = X(2);
% y = X(3);
yd = X(4);
% theta = X(5);
thetad = X(6);
```

```
% Adding in a lateral perturbation only for the first
% stance phase
Fpert = 0;
stancefraction = .125;
tpeak = .002985;
tend = .004;
pertmag = percimp*(m*vdes)*(2/tend);
% Assume that the perturbation occurs at approximately
% the same place in a stance phase.  As such, assume that
% it happens at a fraction of tdes?  In the Kuk. and
% Holmes simulations, it seems that the impulse occurs
% about 1/8 of the duration of the stance phase into the
% stance. Try that?
if (t >= stancefraction*tdes && ...
    t <= (stancefraction*tdes + tpeak))
  % Force is on the first part of the force triangle.
  % Want peak to happen at .002985 s.
  Fpert = pertmag/(tpeak)*(t - stancefraction*tdes);
end
if (t > (stancefraction*tdes + tpeak) && ...
    t <= (stancefraction*tdes + tend))
  % Force is on the second part of the force triangle
  Fpert = pertmag/(tpeak-tend)*...
    (t - stancefraction*tdes - tpeak) + pertmag;
end;
```

```
% solve equations of motion
[Fx,Fy,M] = legforces(t,X);
xdd = (sum(Fx)+Fpert)/m;
ydd = (sum(Fy)-m*g*sin(sigma))/m;
thetadd = sum(M)/Ival;
Xd = [xd;xdd;yd;ydd;thetad;thetadd];

%Calculate transport cost
Xd(7) = sum(sqrt(Fx.^2+Fy.^2))*...
  abs((x*xd+y*yd)/sqrt(x^2+y^2))/(m*g*tdes*vdes);
if isnan(Xd(7)) %remove a discontinuity
  Xd(7) = 0;
end
end

%%% end stancedynamics function

%%% stancebreak function

function [delta, isterminal, direction] ...
  = stancebreak(t,X)
% Determines when a front leg force returns to zero

global cval FootAct tstartstance
```

```
tdes = cval(5);

%Calculate foot ending stance break
[~,~,~,l,dr] = legforces(t,X);
deltal = dr - l;
deltal(~FootAct) = 1;
%Calculate foot beginning stance break
deltat = tstartstance + 2*tdes - t;

%Set return values
delta = [deltal';deltat'];
isterminal = [FootAct';true(6,1)];
%Setting any to zero would ignore the corresponding leg
direction = [zeros(6,1);-ones(6,1)];
minstep = tdes/50;
for n = 1:6
  if abs(t-tstartstance(n)) < abs(minstep)
    isterminal(n) = 0;
  end
end
end

%%% end stancebreak function
```

A.5.2 robustness_family.m

This script takes an existing gait family stored in a *.mat file and appends information regarding how long each takes to recover from various lateral impulses.

```
%Gathers robustness data for a family of fixed points
% clear all, close all
global cval tstartstance fpx fpy FootAct fp IsPush
global percimp
maxstrides = 50;
percimpA = -.8:.1:.8; %Which impulses to simulate

%Need to define a loadfile before running the script
if exist([loadfile '.mat'],'file')
  %but only if they actually exist
  load(loadfile);
end

%Now gather robustness data
numstepstot = zeros(length(exflgtot),length(percimpA));
recoveredtot = false(size(numstepstot));
slope = cvaltot(:,4)*180/pi;
tsdes = zeros(3,1);
x_t = zeros(6,1);
y_t = zeros(6,1);
for n = 1:size(numstepstot,1)
  % Read in each fixed point
```

```
cval = cvaltot(n,:);
fp = fptot(n,:);
IsPush = IsPushtot(n,:);
X0 = X0tot(n,:);
tstartstance = tstarttot(n,:);
FootAct = FootActtot(n,:);
footplace_init(tsdes,X0,x_t,y_t)
if exflgtot(n) ˜= 1
  %Skip the robustness check for nonvalid fixed points
  %This could also be done for unstable fixed points
  numstepstot(n,:) = 100
  recoveredtot(n,:) = false
  disp('Point no good, continuing on ...')
  continue
end
%Calculate steps to recovery
for impind = 1:length(percimpA)
  percimp = percimpA(impind);
  disp(['Working on sigma = ' num2str(slope(n)) ...
    '; Impulse = ' num2str(percimp*100) ...
    '%; delta = ' dS '; Push' Push]);
  [˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,˜,numsteps,...
    recovered]=simgait_multileg_latimp(X0,maxstrides,0)
  numstepstot(n,impind) = numsteps;
  recoveredtot(n,impind) = recovered;
  end
  numstepstot
  recoveredtot
end

%Append data onto existing gait family
save(loadfile,'numstepstot','recoveredtot',...
  'percimpA','-append')
```

A.6 Optimization

A.6.1 find_sigfam_varyldev_?opt.m

Two variations of the find_sigfam script were created to find optimal fixed points. For ? = C, the transport cost was optimized. For ? = e, the stability was optimized. Shown here is a combined version with Copt currently active. By commenting and uncommenting the appropriate places either version can be created.

```
% clear all, close all, %clc
% format long, format compact
global cval tstartstance fpx fpy FootAct fp_guess
global act_var act_con ldev IsPush fp minCostfn fp_init
dval = [1e-6 1e-5 1e-1 1e-4];

%Load and plot starting point
```

```
% load temp2
%Declare variables below outside of script, or activate
% loadfile = ['..\FixedPoints_mineig\gaitfam_PushA_' ...
%   'varysig_k150_intvdes_delta15_varyldev_' ...
%   'betaF025M055B120']
% deltaExt = .15;
% sv_for = true;
% Pushflg = 'A'
load(loadfile)
save temp2

%%%%
%Ctran points used eig points as a starting value.
%The code in this section is for *Copt only
induse = 1:length(exflgtot); %which points to gather
%remove non-fp
induse = induse(exflgtot==1)
%remove ldev violations
l0 = cvaltot(1,19:21);
mxldev = l0*.9;
mnldev = l0*.2;
for n = 4:6
  if max(abs(fptot(induse,n))) > mxldev(n-3)
    disp(['Removing for max ldev' ...
      num2str(n-3) ' constraint'])
    induse = induse(abs(fptot(induse,n)) ...
      <= mxldev(n-3))
  end
  if min(abs(fptot(induse,n))) < mnldev(n-3)
    disp(['Removing for min ldev' ...
      num2str(n-3) ' constraint'])
    induse = induse(abs(fptot(induse,n)) ...
      >= mnldev(n-3))
  end
end
fptotOld = fptot(induse,:);
sigind = round(cvaltot(induse,4)*180/pi)'
%%%%

% n = 1;
% n = length(exflgtot);
% exflgtot
% cvaltot(:,4)'*180/pi
% n = input('n = : ')
if sv_for
  n = min(induse);
else
  n = max(induse);
end
```

```
cval = cvaltot(n,:);
fp = fptot(n,:);
IsPush = logical(IsPushtot(n,:));
%Set heading angle
% fp(2) = deltaExt;

%Clear previous data (comment out if desired)
cvaltot = [];
fptot = [];
IsPushtot = [];
X0tot = [];
tstarttot = [];
FootActtot = [];
evaltot = [];
Ctot = [];
exflgtot = [];
Fxmaxtot = [];
Fymaxtot = [];
Fxcumtot = [];
Fycumtot = [];
Ctrantot = [];

%Find additional gaits
%These are the options for the fsolve routine
act_var = [1 13 14 15];
```

```
%var order = [v0 delta0 tdes 4-6=>ldev1-3 7-9=>beta1-3
%10-12=>Tdrv1-3 13-15=>phi1-3 ldev% beta%]
act_con = [1 2 3 4];
%con order = [Dv Ddelta vdes xdrift]
TypX = [.25 .5 .05 .01*ones(1,3) pi/2*ones(1,3) ...
  .05*ones(1,3) pi/4*ones(1,3) 1 1]';
options = optimset('MaxIter',100,'MaxFunEvals',1500,...
  'Display','iter','TolX',1e-7,'TolFun',1e-8,...
  'TypicalX',TypX(act_var),'ScaleProblem','Jacobian');
%Identify desired slopes in steps of 5 degrees
%%%%
%*Copt
if ~sv_for
  sigind = sigind(end:-1:1)
  fptotOld = fptotOld(end:-1:1,:);
end
%%%%

%%%%
%*eopt
% if sv_for
%   sigind = cval(4)*180/pi:5:90
% else
%   sigind = cval(4)*180/pi:-5:0
% end
```

```
%%%%

for sig = sigind
  %%%%
  %*Copt only
  fp = fptotOld((sigind == sig),:)
  %%%%
  %Set parameters for new fixed point
  cval(4) = sig*pi/180;
  vdes0 = .35;
  vdes90 = .20;
  vdes = vdes0 + sig/90*(vdes90 - vdes0)
  tdes = inv(freq_fitter(vdes))/2
  cval(5) = tdes;
  fp(3) = tdes;
  cval(37:39) = tdes;
  fp(10:12) = tdes;
  cval(6) = vdes;
  fp_guess = fp;
  disp(['Working on sigma = ' num2str(sig)])

  %Minimize transport cost
  minCostfn = 10;
  fp_init = fp;
  %%%%
  %Choose one
  var_final = fminsearch(@Optimizer_C,fp_init(4:6))
%   var_final = fminsearch(@Optimizer_e,fp_init(4:6))
  %%%%
  fp_guess = fp_init;
  fp_guess(4:6) = var_final

  %Call fixed point finder
  [fp,C,exflg] = fsolve(@fp_multileg,fp_guess(act_var)...
    ,options);
  fptemp = fp_guess;
  fptemp(act_var) = fp;
  fp = fptemp
  %Set parameters based on the resulting fixed point
  v0 = fp(1);
  delta0 = fp(2);
  tdes = fp(3);
  ldev = [fp(4:6) fp(4:6)];
  beta0 = [fp(7:9) -fp(7:9)];
  Tdrv = fp(10:12);
  phi = fp(13:15);
  ldevX = fp(16);
  betaX = fp(17);
  %Correct cval
  cval(5) = tdes;
  cval(25:30) = ldevX*ldev;
  betaC = beta0;
  betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
  cval(31:36) = betaC;
  cval(37:39) = Tdrv;
  cval(40:42) = phi;
  %Set initial conditions
  xd = -v0*sin(delta0);
  yd = v0*cos(delta0);
  X0 = [0 xd 0 yd 0 0];
  %foot place finder
  tsdes = zeros(3,1);
  x_t = zeros(6,1);
  y_t = zeros(6,1);
  footplace_init(tsdes,X0,x_t,y_t)
  %Simulate at new fixed point to gather forces, Ctran
  [~,~,~,~,~,~,~,ind,Fx,Fy,~,~,~,~,~,~,Ct] ...
    = simgait_multileg(X0,1,0);
  Fxmax = max(Fx(:,1:3));
  Fxmin = min(Fx(:,1:3));
  rec_min = abs(Fxmin) > abs(Fxmax);
  Fxmax(rec_min) = Fxmin(rec_min);
  Fymax = max(Fy(:,1:3));
  Fymin = min(Fy(:,1:3));
  rec_min = abs(Fymin) > abs(Fymax);
```

```
Fymax(rec_min) = Fymin(rec_min);
step = max(ind(:,4));
Fxcum = [max(sum(Fx(1:step,:),2)) ...
  min(sum(Fx(1:step,:),2))];
Fycum = [max(sum(Fy(1:step,:),2)) ...
  min(sum(Fy(1:step,:),2))];
e = findeig([v0 delta0 0 0],dval)
step = ind(ind(:,4)>0,4);
Ctran = Ct(step)
if sv_for
  %Store variables - forward travelling
  cvaltot = [cvaltot;cval];
  fptot = [fptot;fp];
  IsPushtot = [IsPushtot;IsPush];
  X0tot = [X0tot;X0];
  tstarttot = [tstarttot;tstartstance];
  FootActtot = [FootActtot;FootAct];
  evaltot = [evaltot;e'];
  Ctot = [Ctot;Ct];
  exflgtot = [exflgtot;exflg];
  Fxmaxtot = [Fxmaxtot;Fxmax];
  Fymaxtot = [Fymaxtot;Fymax];
  Fxcumtot = [Fxcumtot;Fxcum];
  Fycumtot = [Fycumtot;Fycum];
  Ctrantot = [Ctrantot;Ctran];
```

```
else
  %Store variables - backward travelling
  cvaltot = [cval;cvaltot];
  fptot = [fp;fptot];
  IsPushtot = [IsPush;IsPushtot];
  X0tot = [X0;X0tot];
  tstarttot = [tstartstance;tstarttot];
  FootActtot = [FootAct;FootActtot];
  evaltot = [e';evaltot];
  Ctot = [Ct;Ctot];
  exflgtot = [exflg;exflgtot];
  Fxmaxtot = [Fxmax;Fxmaxtot];
  Fymaxtot = [Fymax;Fymaxtot];
  Fxcumtot = [Fxcum;Fxcumtot];
  Fycumtot = [Fycum;Fycumtot];
  Ctrantot = [Ctran;Ctrantot];
end

%Save results
vdes = num2str(cval(6)*100);
delta = num2str(abs(fp(2)*100));
if length(delta) == 1
  delta = ['0' delta];
end
if fp(2) < 0
```

```
    delta = ['_' delta];
  end
  %%%%
  %Choose one
  savefile = ['..\FixedPoints_minCtran\gaitfam_Push' ...
    Pushflg '_varysig_k150_intvdes_delta' delta ...
    '_varyldev_betaF025M055B120']
%   savefile = ['..\FixedPoints_mineig\gaitfam_Push' ...
%     Pushflg '_varysig_k150_intvdes_delta' delta ...
%     '_varyldev_betaF025M055B120']
  %%%%
  save(savefile,'cvaltot','fptot','IsPushtot','X0tot',...
    'tstarttot','FootActtot','evaltot','Ctot',...
    'exflgtot','Fxmaxtot','Fymaxtot','Fxcumtot',...
    'Fycumtot','Ctrantot')
  if exflg ~= 1
    %If the last fixed point was no good
    break
  end
  ldevtest = abs(fp(4:6))
  if max(ldevtest > mxldev) || max(ldevtest < mnldev)
    break
  end
end
plot(cvaltot(:,4)*180/pi,sort(abs(evaltot),2))
```

A.6.2 Optimizer_?.m

These functions are called by the corresponding find_sigfam_varyldev_?opt. As before, the ? represents C for transport cost and e for stability. Shown here is a combined version with Copt currently active. By commenting and uncommenting the appropriate places either version can be created.

```
function Costfn = normFerr(var_vals)
%Calculate a cost function based on either maximum
%eigenvalue or transport cost. Penalties are included for
%parameters that do not produce good fixed points.

global fp_init fp_guess act_var act_con minCostfn cval
fp_init([4 5 6]) = var_vals(1:3);
fp_guess = fp_init;

%Set options
act_var = [1 13 14 15];      %v0 and phi
act_con = 1:4;               %all
TypX = [.25 pi/4*ones(1,3)]';
options = optimset('MaxIter',20,'MaxFunEvals',100, ...
  'Display','iter','TolX',1e-7,'TolFun',1e-8,...
  'TypicalX',TypX,'ScaleProblem','Jacobian');

%Call fixed point solver
[fp,C,exflg] = fsolve(@fp_multileg,...
  fp_init(act_var),options);
```

```
disp('Working on higher delta')
%Check to make sure the fixed point is not at a boundary
if fp_guess(2) > .05
  fp_guess(2) = fp_guess(2)*1.05;
else
  fp_guess(2) = fp_guess(2) + .0025;
end
[fp2,C2,exflg2] ...
  = fsolve(@fp_multileg,fp_init(act_var),options);
fp_guess(2) = fp_init(2);

exflg
fptemp = fp_init;
fptemp(act_var) = fp;
fp = fptemp;

%Set parameters based on the resulting fixed point
v0 = fp(1);
delta0 = fp(2);
tdes = fp(3);
ldev = [fp(4:6) fp(4:6)];
beta0 = [fp(7:9) -fp(7:9)];
Tdrv = fp(10:12);
phi = fp(13:15);
ldevX = fp(16);
```

```
betaX = fp(17);
%Correct cval
cval(5) = tdes;
cval(25:30) = ldevX*ldev;
betaC = beta0;
betaC([1 2 4 5]) = betaX*beta0([1 2 4 5]);
cval(31:36) = betaC;
cval(37:39) = Tdrv;
cval(40:42) = phi;

%Set initial conditions
xd = -v0*sin(delta0);
yd = v0*cos(delta0);
theta0 = 0;
thetad0 = 0;
X0 = [0 xd 0 yd theta0 thetad0];

%foot place finder
x_t = zeros(1,6);
y_t = zeros(1,6);
tsdes = [0 0 0];
footplace_init(tsdes,X0,x_t,y_t)

[x,xd,y,yd,theta,thetad,t,ind,Fx,Fy,M,fpxtot,fpytot,...
  leq,lspr,E,Ct] = simgait_multileg(X0,1,0);
```

```
Costfn = 0;

%display current places
% disp(['beta: ' num2str(fp(7:9)*180/pi)])
% disp(['k: ' num2str(cval(1))])
% disp(['vdes: ' num2str(cval(6))])

dval = [1e-6 1e-5 1e-1 1e-4];
eval = findeig([v0 delta0 0 0],dval)

%Calculate Ctran
indstep = max(ind(:,4));
Ctran = Ct(indstep)

%Calculate e
for n = 1:2
  [~,delind] = min(abs(abs(eval)-1));
  eval(delind) = 0;
end
%%%%
%Ctran Option
Costfn = Ctran
%eig Option
% Costfn = max(abs(eval))
%%%%
```

```
%Correction for non fixed point
if exflg~=1
  ferr = sum(C.^2)
  Corfact = max(10+log10(ferr),0)*2 + 2
  Costfn = Costfn + Corfact
end
%Correction for adjacent non fixed point
if exflg2~=1
  ferr2 = sum(C2.^2)
  Corfact2 = max(10+log10(ferr2),0)*.5+.5
  %     Costadj = 1.05^Corfact2 - 1
  Costfn = Costfn + Corfact2
end

%Correction for phi too close to transition
phi = fp(13:15)
mxphi = pi/3;
mnphi = -pi/2;
for legnum = 1:3
  %phi too high
  if phi(legnum) > mxphi
    Costfn = Costfn + ...
      (phi(legnum) - mxphi)*.5/(pi/2-mxphi)
  end
  %phi too low
  if phi(legnum) < mnphi
    Costfn = Costfn + ...
      (-phi(legnum) - -mnphi)*.5/(pi/2-mxphi) + .5
  end
end

%Correction for leg missing pick up time tangentially
mnang = 10;
for legnum = 1:3
  tempind = ind(logical(ind(:,legnum)),legnum);
  footend = abs(tempind(2));
  spr_head = atan((lspr(footend,legnum) - lspr( ...
    footend-1,legnum))/(t(footend)-t(footend-1)))*180/pi;
  eq_head = atan((leq(footend,legnum) - leq( ...
    footend-1,legnum))/(t(footend)-t(footend-1)))*180/pi;
  cross_head = abs(spr_head - eq_head);
  disp(['Leg #' num2str(legnum) ...
    ' crossing heading angle: ' num2str(cross_head)])
  % Correction for leg botching set down tangentially
  if cross_head < mnang
    Costfn = Costfn + (mnang-cross_head)*.5/mnang
  end
  spri_head = atan((lspr(2,legnum) - lspr(1,legnum)) ...
    /(t(2)-t(1)))*180/pi;
  eqi_head = atan((leq(2,legnum) - leq(1,legnum)) ...
    /(t(2)-t(1)))*180/pi;
  init_head = abs(spri_head - eqi_head);
  disp(['Leg #' num2str(legnum) ...
    ' initial heading angle: ' num2str(init_head)])
  if init_head < mnang
    Costfn = Costfn + (mnang-init_head)*.5/mnang
  end
end

%Correction for ldev outside of bounds
l0 = cval(19:21)
ldev = abs(fp(4:6));
disp(['ldev: ' num2str(ldev,3)])
mnldev = .2*l0;
mxldev = .9*l0;
for legnum = 1:3
  if ldev(legnum) < mnldev(legnum)
    disp(['Leg #' num2str(legnum) ' l_{dev} too low'])
    Costfn = Costfn + (mnldev(legnum) - ...
      ldev(legnum))*1/mnldev(legnum) + 30;
  elseif ldev(legnum) > mxldev(legnum)
    disp(['Leg #' num2str(legnum) ' l_{dev} too high'])
    Costfn = Costfn + (ldev(legnum) - ...
      mxldev(legnum))*1/(l0(legnum)-mxldev(legnum)) + 30;
  end
end


Costfn
minCostfn
% pause


if Costfn < minCostfn
  fp_init = fp;  %Start at lost solved point to save time
  minCostfn = Costfn;
  save temp
end
```