Powered “Passive” Dynamic Walking

John Camp
The Sibley School of Mechanical and Aerospace Engineering
Cornell University, Ithaca NY
Masters of Engineering Project Report
August 1997

Advisors: Andy Ruina, Subhas Desa

Abstract

We demonstrate that a simple open-loop actuation/control scheme is all that is required to produce stable, powered, human-like walking motions in a set of roughly human-like legs. This work builds off of the passive dynamic models pioneered by McGeer (1989) and later studied by Garcia (1997). Our model is constrained to move in the sagital plane. It has two identical rigid, straight legs with arbitrary mass distribution, that are hinged to each other at the hip. It has two rigid, round massless feet, one hinged to the base of each leg. The floor is flat, level and rigid. The feet have plastic (no-slip, no-bounce) collisions with the floor at ‘heelstrike’. The energy required to balance that lossed at heelstrike is added through part of the stride by an extension of the stance foot. This extension is powered by a permanent magnet DC motor acting under a constant voltage. When the motor is not activated the feet are held at a prescribed angle by a mechanical stop and the robot moves only under the influence of gravity. This model exhibits stable walking over a large range of control and actuation parameters. It shows behaviors similar to those of the passive walkers previously studied. Stable and unstable gait limit cycles and period doubling, for a variety of structural, physical and control/actuation parameters, have been observed.

1. Introduction

The majority of powered dynamic walking robots built to date require complicated actuation and control schemes. Such schemes take the dynamics of the walkers limbs into account, but only adversarially, as the passive dynamics of the links must be overcome in order to impose the desired walking motion.

It has been shown by McGeer that stable human-like walking motions can be exhibited by uncontrolled sets of rigid links powered only by gravity. McGeer’s work in passive dynamic walking has shown that coordination in human walking does, to some extent, rely on the passive dynamics of our legs. If one’s goal is to study active, human-like, robotic walking it is natural that one would also take advantage of the passive dynamics of the robot’s limbs to reduce the energetic and computational cost of coordinating the walker’s motion. A simple, open loop, powered walking scheme was studied by McGeer (1989, 1990, 1991). McGeer’s powered model achieved stable walking. It had straight legs and replaced the slope required for the passive walkers with impulsive excitations of the stance foot at ‘push off’ and through instantaneous extensions of the walker’s stance leg at points during the swing phase.

2. Our Model

A schematic of our model, and the different phases of a typical stride, can be seen in figure 1. The walker is two dimensional. It has two identical rigid legs connected by a frictionless hinge at the hip. It has circular rigid, massless feet which are pin-jointed to their respective legs and are extended (when in powered mode) by a permanent magnet DC motor acting under a constant voltage. The feet rest against mechanical stops which keep them in a prescribed position through the passive mode. The legs can have an arbitrary mass distribution. The walker moves on a level, rigid floor. When the swing foot strikes the floor at heel-strike the collision is plastic (no-slip, no-bounce).

Following McGeer and Garcia the swing foot is allowed to briefly pass through the floor in order to avoid the inevitable foot scuffing of straight legged walkers. There are walking motions of this model that do not require this concession (as the stance leg can be raised), but it is included for those walking motions that do not begin the extension of the stance foot until after the swing leg has passed the stance leg. All of the motions shown in this paper are of the latter type.

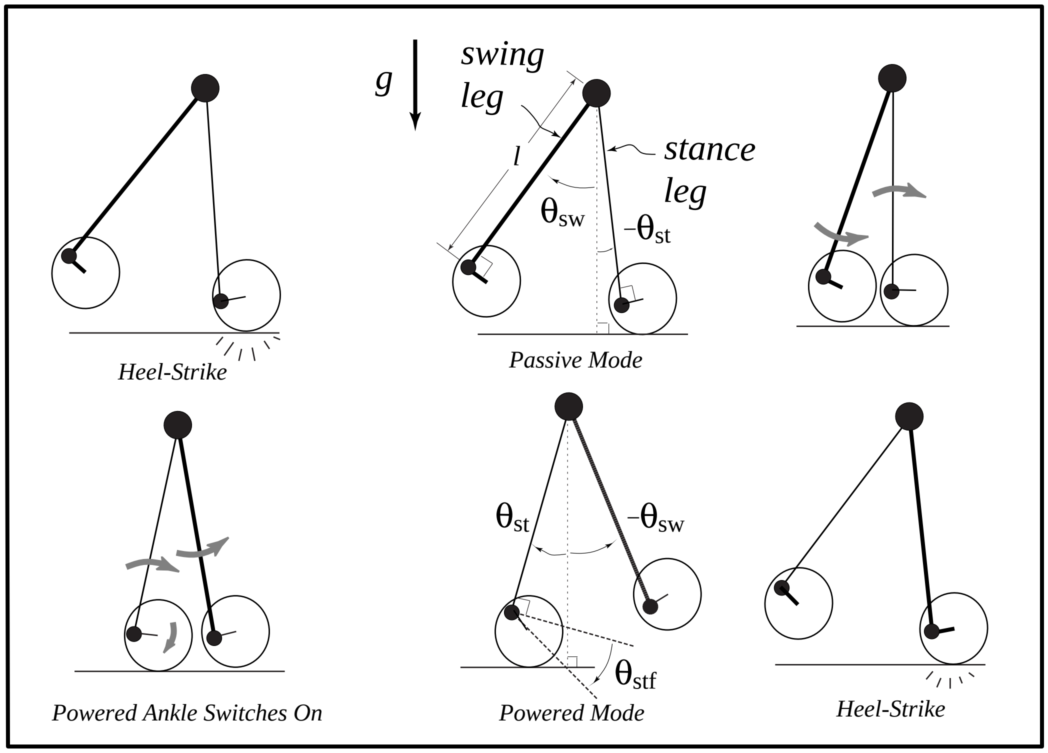

figure 1

3. A Typical Stride

A typical stride, shown in figure 1, has two modes, *passive* and *powered*. As with the two modes of knee-jointed walkers (McGeer, Garcia) each mode has it own set of governing dynamical equations. In the passive mode the walker is identical to the passive straight legged walkers studied by McGeer. In the powered mode the walker has three links, the stance foot, stance leg and the swing leg.

The stride 'begins' after heelstrike in passive mode, with both feet locked at a prescribed angle. The walker continues only under the influence of gravity until the swing leg reaches the critical angle at which a switch is thrown, beginning powered mode (i.e., the motor is switched on). It stays in powered mode until heelstrike is detected. Following McGeer, Coleman and Garcia the state immediately following heel strike is found, given the state immediately before heelstrike, through a set of jump conditions (conservation of angular momentum about the hip and the new contact point, over the collision). At heelstrike the previous stance foot, now the swing foot, is instantaneously retracted to the locked position.

4. Geometry

Our definition of the generalized coordinates of the system follows from Garcia's work. A sketch of the relevant angles and structural dimensions is shown in figure 2. A list, and description, of all system parameters is provided in Appendix A.

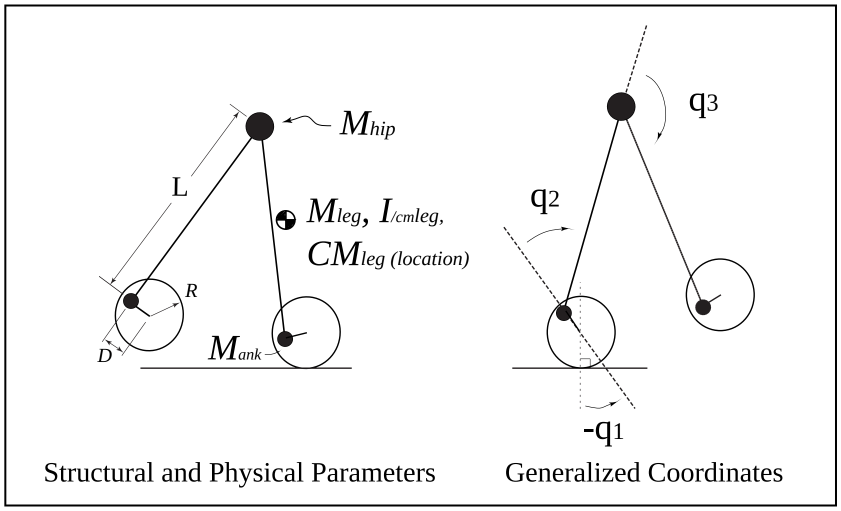
M_{hip}
L
M_{leg}, $I_{/cmleg}$, $CM_{leg\ (location)}$
R
D
M_{ank}
q_3
q_2
$-q_1$
Structural and Physical Parameters
Generalized Coordinates

figure 2

5. Actuation and Control

For our work we wanted the model's control and actuation scheme to be as simple as possible both mechanically and electronically, so as to allow the possibility of future experimentation. In the development of the model a number of different actuation/control schemes were considered. In the end, a permanent magnet DC motor acting under a constant voltage was decided upon. It was chosen as it is inexpensive, open-loop and essentially 'on/off'. The motor torque, at each time step of the numerical integration of the powered mode equations, is found from the steady state torque/speed characteristic of permanent magnet DC motors. We have chosen to neglect any transient motor dynamics. A sketch of the motor torque/speed characteristic can be seen in figure 3. The relation ship is linear, and the two relevant control parameters are the stall torque (the torque the motor applies when the rotor is not moving) and the no-load angular velocity.

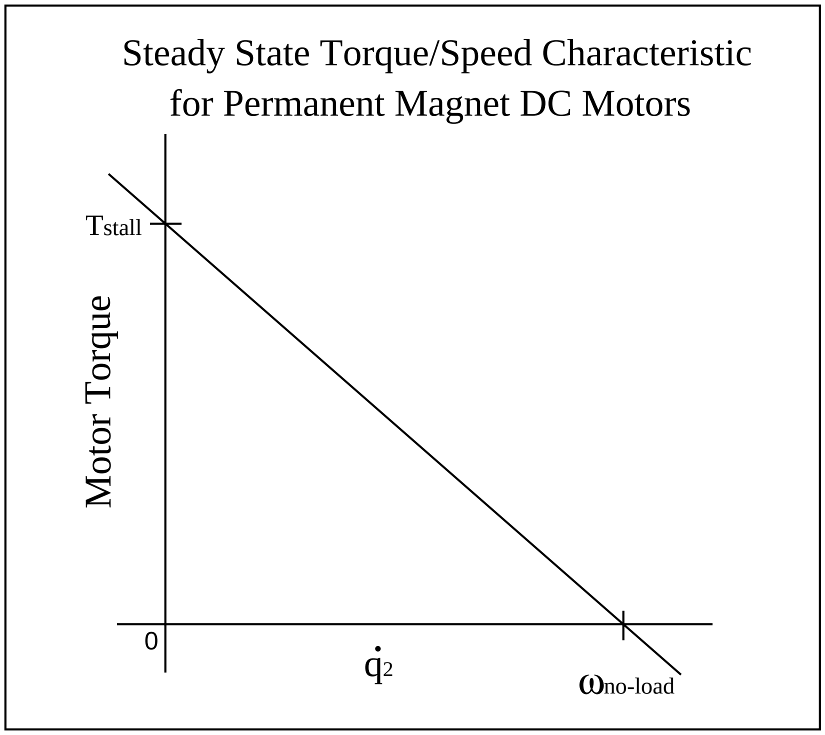

figure 3

6. The Equations of Motion

The governing equations of motion were originally derived ‘by hand’ using Kane’s method, then later using AutoLev. The hand derived passive mode equations were used in a straight legged passive simulation written as a precursor to the powered simulation. For the powered simulation, however, we directly went to the AutoLev equations. AutoLev was also used for the jump conditions as it is capable of deriving expressions for the angular momentum of the system, or any sub-system, about any point (see Appendix B, for all AutoLev code).

7. The Search for Period-One Gait Cycles

As with the work of McGeer, Coleman and Garcia our analyses of the powered walker are based on our treatment of a step as a Poinacre map or, the ‘stride function’ as McGeer called it. Gait limit cycles correspond to fixed points of the map, or in other words the roots of the function:

$$f(\theta) = P(\theta) - \theta$$

Where $P(\theta)$ is the Poincare map and θ is a set of initial conditions. In his analysis of passive walkers, Garcia chose the instant immediately before heelstrike as his Poincare section, as the number of independent state elements required to describe the system was reduced from 4, two angles two angular velocities (6 in the case of the knee-jointed walker) to 3, one angle two angular velocities (2 for the simplest walker). This simplification can be made as only one generalized coordinate is required to describe the walker when both feet are in contact with the floor. The instant immediately before heelstrike in the powered gait cycle, however, differs from that of the passive walkers as the stance foot is in an extended position. This position is solely dictated by the dynamics of the system, so the clean geometry of double stance is disrupted. We have therefore, chosen a different point in the powered gait cycle for our Poincare section. We choose the instant at which the walker enters powered mode for our section, as the swing leg angle is known (it is a control parameter), the stance foot is in the locked position and the angular velocity of the stance leg relative to the stance foot is zero. We are left with 3 independent state elements, one angle (the stance foot angle) and two angular velocities (the stance foot/leg and the swing leg).

A Newton/Raphson search was used to find the roots of f(θ). The stability of the fixed points of the map was then characterized by the eigen values of a numerical estimation of the Jacobian of the map near the fixed point. The Newton/Raphson method used in the search for limit cycles does not converge unconditionally. It was up to us therefore, to select a set of structural, physical and actuation/control parameters with a limit cycle as well as to find a set of initial conditions in the neighborhood of that cycle in order for the search to converge. Without any insight into the passive dynamics of the system this would have been quite time consuming as there are 11 system parameters (see Appendix A for list of physical, structural and control parameters). We have, however, developed an understanding of passive dynamic walking and we allowed this understanding to guide our search for powered limit cycles. The result was quite nice.

We assumed that sets of physical/structural parameters that exhibited robust passive walking would exhibit powered walking, assuming the actuation scheme reasonably mimics the effects of the slope used in the passive walkers. The foot motion that would achieve this was not exactly known at the time, but we hoped that smoothly extending the foot in a reasonably anthropomorphic way would be good enough. It is.

Building off of Garcia's simplified model of a straight legged point foot walker (1997) we set the mass of the legs to zero and gave the powered walker a hip mass to 'foot mass' (the 'foot mass' was located at the ankle) ratio of 1000. To approximate the structural parameters of the passive walker the foot radius was set to zero and the ankle length was made small compared to the length of the leg, as the simplified passive walker walks stably on shallow slopes. The set of initial conditions we used was actually taken (and slightly modified) from a stable limit cycle found, for this set of parameters, by the passive simulation. As soon as a stall torque that did not cause the foot to either roll over or fall back was found (a few tries), the walker exhibited stable walking (max eigen value of 0.65, see fig. 4). The ankle length, actuator parameters, and the swing leg angle which triggers the start of the powered mode were all just ‘eyeballed’ yet the passive dynamics of the system still coordinated the motion.

8. Dynamic Behavior

There has been every indication that the dynamic behavior of the powered walker is as rich as that of its passive cousins. As indicated above the powered walking behavior has paralleled the passive walking in every walking regime studied thus far. For the majority of the analyses performed to date all of the actuation and control parameters were fixed except for the stall torque of the motor. The stall torque was ramped up and down as it is a characteristic forcing somewhat analogous to the slope of the passive walkers. Stable and unstable limit cycles for a variety of structural and physical parameters and period doubling have been found. Stance, swing and foot angles vs. nondimensional time for a typical stable limit cycle can be seen in figure 4.

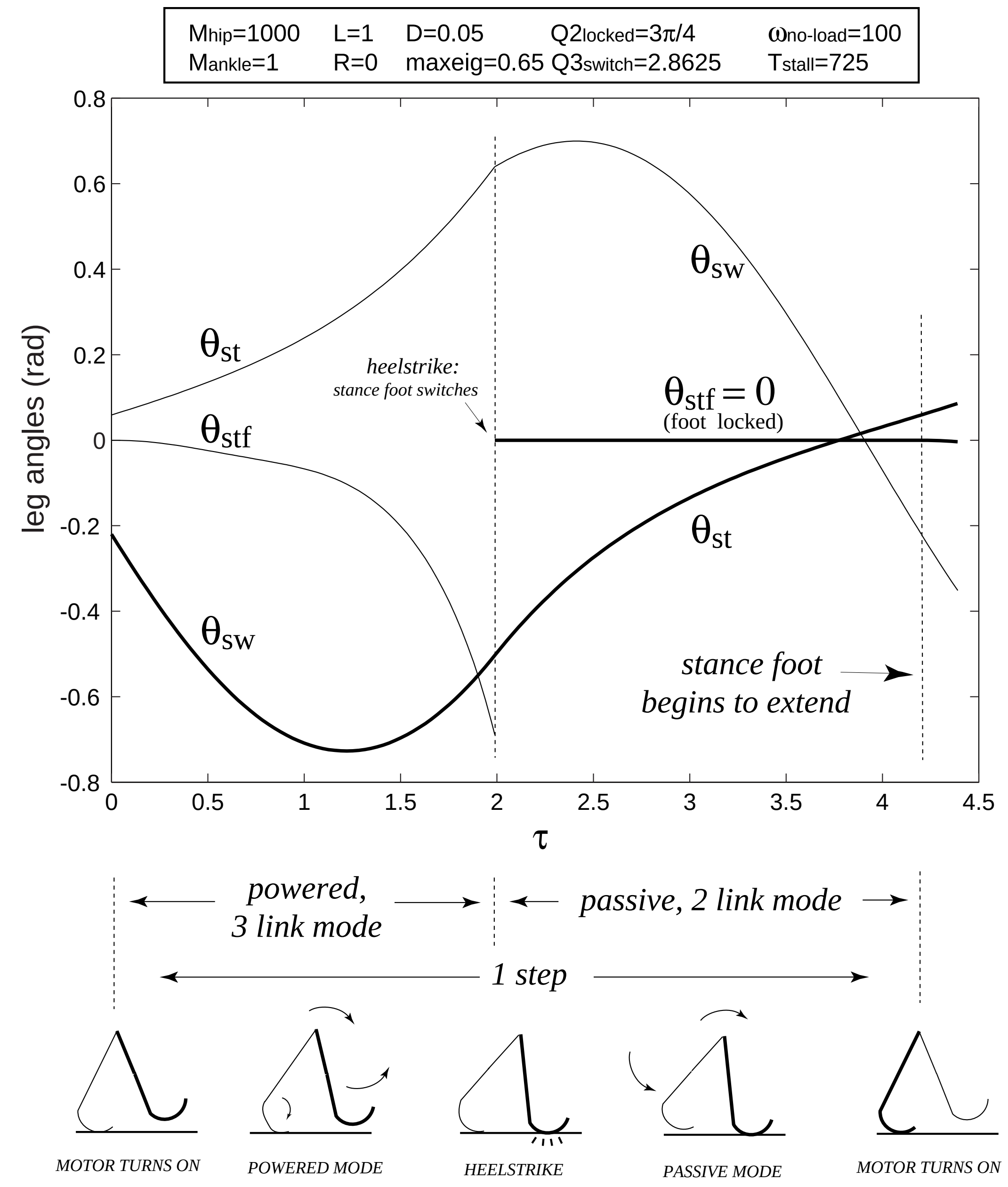

figure 4
A typical stable gait cycle

There have been some rather interesting limit cycles found in which the foot does not extend monotonically. In these walking motions the stance foot rises and falls (sometimes more than once) always ending at heelstrike in some extended position. The motions in which the stance foot behaves in this way may be written off as simply a curiosity, but what they show is that these walkers will walk stably even when the motion of the foot can't be doing a particularly good job at fooling the walker into 'thinking' that it is walking down a slope. It could be, however, that these motions of the foot actually stabilize the walker, but either way it seems that the details of the foot extension are not too important to successful powered walking. An example of such stable limit cycle (max. eigen value: 0.61) can be seen in figure 5.

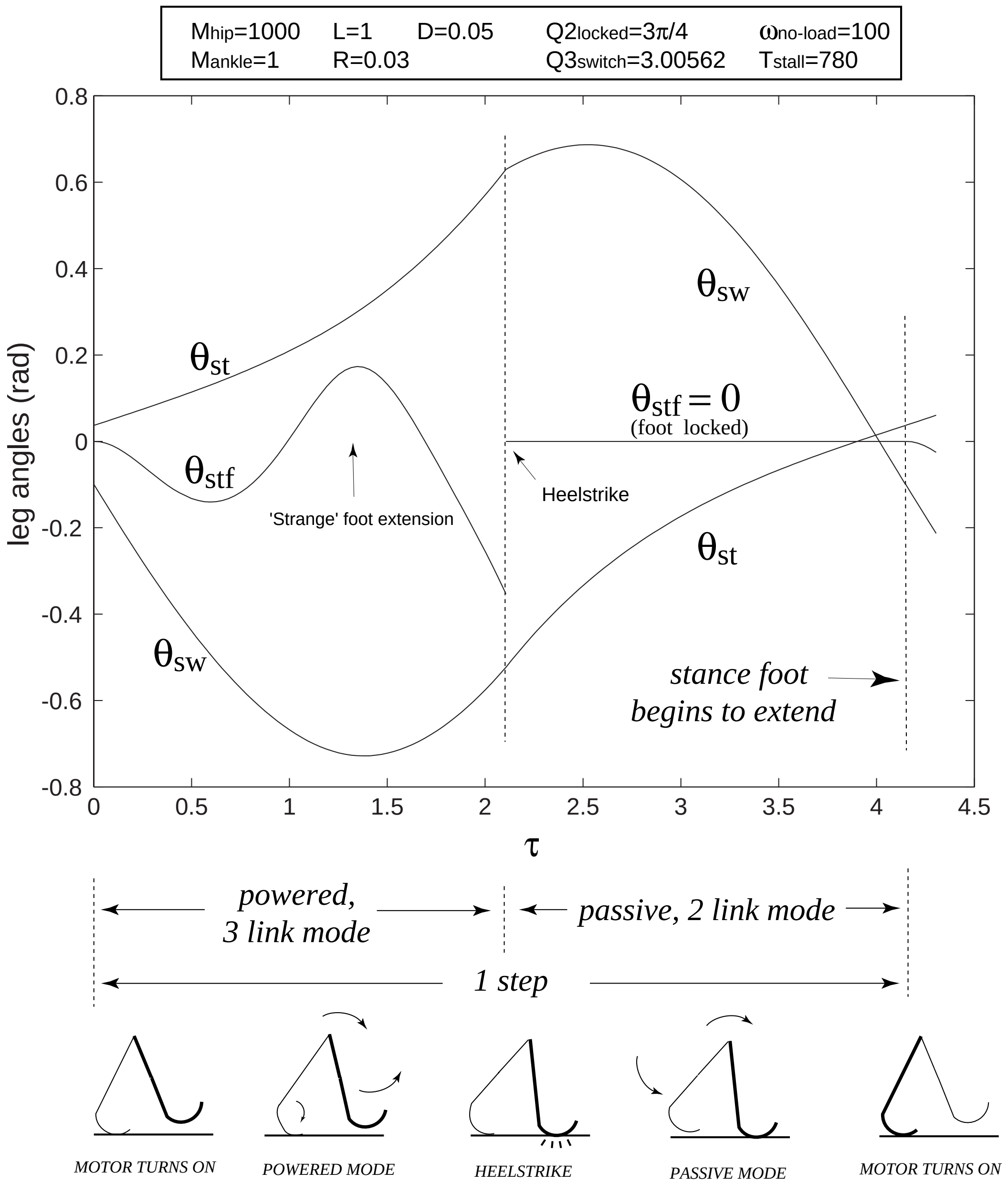

figure 5

8.1. Gait Cycle Stability

Our interest is not only in the existence of gait limit cycles, but also in their stability. Once a limit cycle is found we would like to classify it as either stable or unstable. To do so we numerically estimate the Jacobian of the map in the neighborhood of the fixed point, by perturbing the walker's initial conditions from the fixed point and seeing how those perturbations grow/decay after one step. If all of the eigen values of the Jacobian are within the unit circle the limit cycle is stable.

Below is a plot of the largest eigen value (abs) vs. stall torque for a powered walker. Our interest is in the largest eigen value as the walker is unstable if any of the eigen values of the Jacobian are larger than one in magnitude. A good deal of interesting behavior was found within the range of stall torques seen here. One interesting feature is that the onset of the rising and falling of the stance foot (in this case it actually falls back first) does not bring with it a major change in the stability of the walker.

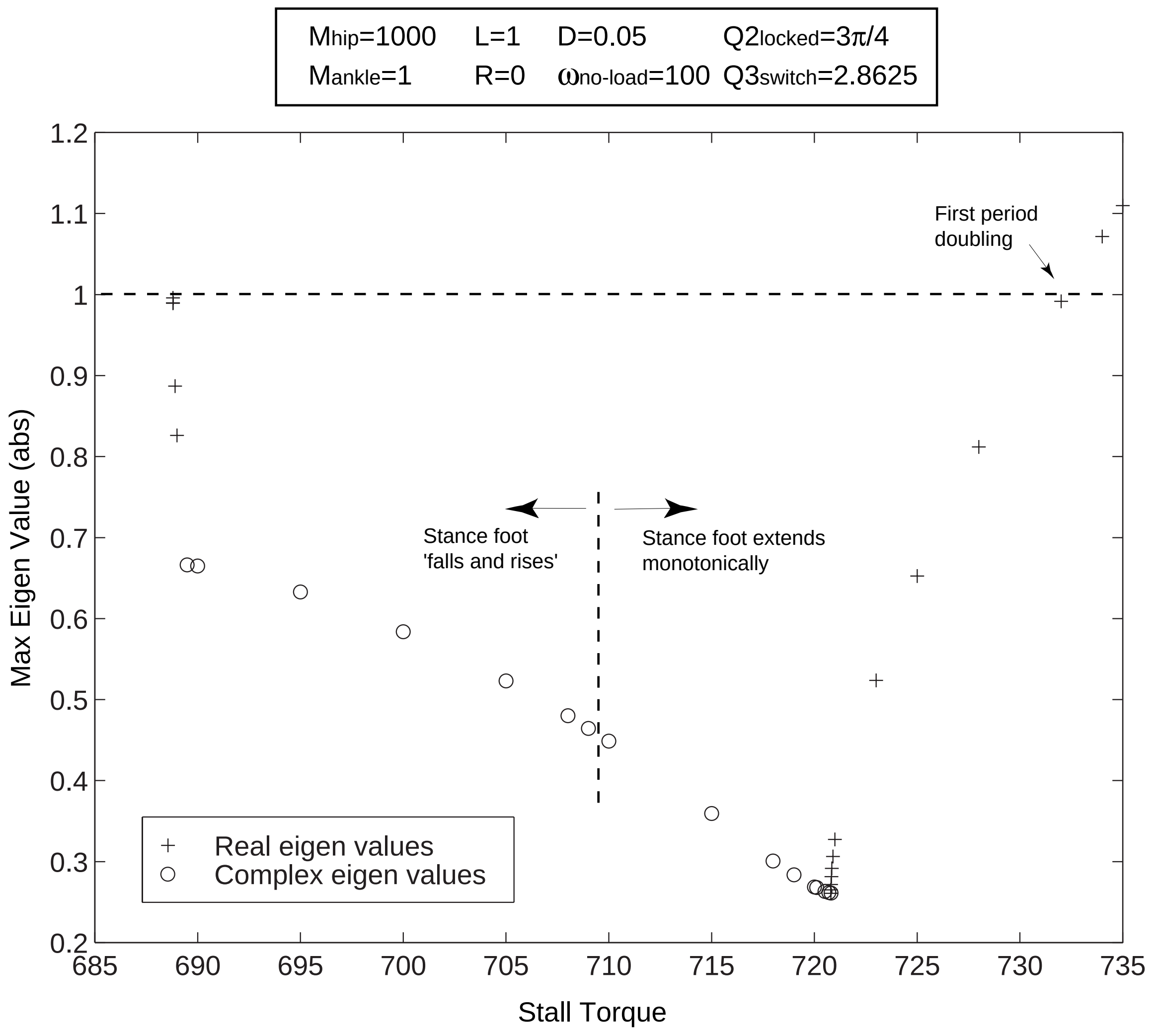

figure 6

9. Future Work

The addition of actuation and control to passive dynamic walkers brings with it issues quite similar to those dealt with in the passive walker research. Many questions arise, such as:

- What role does the actuation and control of these walkers play in the coordination and stability of these walking motions?
- What roles do the actuation/control parameters play in the efficiency of powered walking?

- What is the simplest control scheme that will still walk stably?
- How stable can a controlled dynamic walker become?
- What control scheme/physical parameter combination yields the best performance in terms of stability vs. computational cost?

With this simulation in place at least some of these questions can begin to be answered.

The simulation, as it stands at the time of this writing, can be easily expanded to support a knee-jointed powered model. The way in which we have chosen to power the walker will allow us to make this change without deriving any new equations of motion. This is the case as we are free to begin the extension of the foot after kneestrike has occurred (i.e., the walker would effectively be straight legged during powered mode). One only needs to insert a call to a knee-jointed module (integrator) into the main program. A change that can be made quite easily.

10. Conclusion

Whether one's interest is in biomechanics, autonomous robots, or simply system dynamics, the control of a dynamic system such as this one always brings with it trade offs between performance and financial, computational and philosophical cost. What one is willing to pay may differ greatly across these fields, but there is no question of the advantage of exploiting, rather than fighting, the passive dynamics of the system.

McGeer's discovery that unpowered, uncontrolled sets of rigid links can achieve stable human-like walking motions brought into light the extent to which the passive dynamics of the human body is capable of coordinating human locomotion. With his work came the possibility of stable, powered, human-like walking with out closed-loop control. We have found that a simple open loop controller/actuator is all that is needed to power these walkers and that the powered behavior parallels that of the passive walkers, as hoped.

11. Acknowledgments

Many thanks to Andy, Subhas, Anindya Chatterjee, Mike Coleman and Mariano Garcia for their years of advice and patience.

References

[1] M. Coleman, A. Chatterjee, and A. Ruina. Motions of a rimless spoked wheel: A simple 3D system with impacts. *Dynamics and Stability of Systems*, 1997. Accepted for publication.

[2] M. Coleman and A. Ruina. A tinkertoy model that walks. 1997. In preparation.

[3] M. J. Coleman. *A Stability Study of a Three-dimensional Passive-dynamic Model of Human Gait*. PhD thesis, Cornell University, Ithaca, NY, 1997. In preparation.

[4] J. Furusho and A. Sano. Sensor-based control of a nine-link biped. *The International Journal of Robotics Research*, 9(2), April 1990.

[5] M. Garcia, A. Chatterjee, A. Ruina, and M. J. Coleman. The simplest walking model: Stability, complexity, and scaling. Accepted for publication in ASME Journal of Biomechanical Engineering, 1997.

[6] J. A. Golden and Y. F. Zheng. Gait synthesis for the SD-2 biped robot to climb stairs. *International Journal of Robotics and Automation*, 5(4), 1990.

[7] A. Goswami, B. Thuilot, and B. Espiau. Compass-like biped robot, part I: Stability and bifurcation of passive gaits. Rapport de recherche 2996, Unite de recherche INRIA Rhone-Alpes, St. Martin, France, October 1996.

[8] A. A. Grishin, A. M. Formal'sky, A. V. Lensky, and S. V. Zhitomirsky. Dynamic walking of a vehicle with two telescopic legs controlled by two drives. *The International Jorunal of Robotics Research*, 13(2):137 -147, April 1994.

[9] Y. Hurmuzlu. Dynamics of bipedal gait: Part I objective functions and the contact event of planar ffve-link biped. *Journal of Applied Mechanics*, 60:331 -337, June 1993.

[10] Y. Hurmuzlu. Dynamics of bipedal gait: Part II stability analysis of a planar ffve-link biped. *Journal of Applied Mechanics*, 60:337 -343, June 1993.

[11] T. McGeer. Stability and control of two-dimensional biped walking. Technical Report CSS-IS TR 88-01, Simon Fraser University for Systems Science, Burnaby, B.C., Can., 1988.

[12] T. McGeer. Powered flight, child's play, silly wheels, and walking machines. Technical report, Simon Fraser University, Burnaby, British Columbia, Canada, 1989.

[13] T. McGeer. Dynamics and control of bipedal locomotion. *Progress in Robotics and Intelligent Systems*, 1990.

[14] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62 -82, April 1990.

[15] T. McGeer. Passive walking with knees. In *Proceedings 1990 IEEE International Conference on Robotics and Automation*, pages 1640 -1645, Los Alamitos, CA, 1990. IEEE.

[16] T. McGeer. Passive dynamic catalogue. Technical report, Aurora Flight Sciences Corporation, 1991.

[17] T. McGeer and L. H. Palmer. Wobbling, toppling, and forces of contact. *American Journal of Physics*, 57:1089 -1096,December 1989.

[18] H. Miura and I. Shimoyama. Dynamic walk of a biped. *International Journal of Robotics Research*, 3(2):60 -74,1984.

[19] S. Mochon and T. McMahon. Ballistic walking: An improved model. *Mathematical Biosciences*, 52:241 -260,1980.

[20] S. Mochon and T. A. McMahon. Ballistic walking. *Journal of Biomechanics*, 13:49 -57,1980.

[21] C. Shih, W. A. Gruver, and T. Lee. Inverse kinematics and inverse dynamics for control of a biped walking machine. *Journal of Robotic Systems*, 10(4):531 -555,1993.

[22] G. Taga, Y. Yamaguchi, and H. Shimizu. Self-organized control of bipedal locomotion by neural oscillators in an unstable environment. *Biological Cybernetics*, 65(3):147 -159,1991.

[23] M. Vukobratovic, B. Borovac, D. Surla, and D. Stokic. *Scientiffc Fundamentals of Robotics 7 Bip ed Locomotion: Dynamics, Stability, Control, and Application*. Communications and Control Engineering Series. Springer-Verlag, Berlin, 1990.

Appendix A: System Parameters

Physical parameters for the composite leg:

M	total leg mass
$I_{/cm}$	moment of inertia of the composite leg about its center of mass
X,Y_{cm}	X and Y coordinates of the cm, measured in the leg fixed basis

Structural parameters:

L	leg length (the distance from the ankle to the hip)
D	the distance from the ankle to the center of the foot
R	the radius of the foot
$Q2_{lock}$	the prescribed locked foot angle

Actuation/control parameters:

Tstall	the stall torque of the motor
$\omega_{no\text{-}load}$	the unloaded angular velocity of the motor
Q3switch	the swing leg angle at which the walker enters powered mode

Appendix B: AutoLev

Equation Generation Passive Mode:

```
%   Derivation of equations of motion for the two-link mode.

newtonian n
bodies a,b,f
points h,o,c,ank

variables u{2}'
variables q1,q1',q3,q3'
constants L,g,m,R,I,Xcma,Xcmb,Ycm,d,q2
mass a=m,b=m,f=0

inertia a,0,0,I,0,0,0
inertia b,0,0,I,0,0,0
inertia f,0,0,0,0,0,0

q1' = u1
q3' = u2

simprot(n,f,3,q1)
simprot(f,a,3,q2)
simprot(a,b,3,q3)

%   position vectors
p_o_c> = R*n1>
p_c_ank> = d*f1>
p_ank_h> = L*a1>
p_h_Bo> = (Xcmb)*b1> - Ycm*b2>
p_o_Bo> = p_h_Bo> + p_ank_h> + p_o_c> + p_c_ank>
p_ank_Ao> = Xcma*a1> + Ycm*a2>
p_o_Ao> = p_o_c> + p_ank_Ao> + p_c_ank>

%   velocities and angular velocities
v_o_n>=R*q1'*n2>
v_c_n>=dt(p_o_c>,n)  +  v_o_n>
v_Ao_n>=dt(p_o_Ao>,n)  +  v_o_n>
v_Bo_n>=dt(p_o_Bo>,n)  +  v_o_n>

%save Vel_lock.all,45:48

w_f_n>=q1'*n3>
w_a_n>=q1'*n3>
w_b_n>=(q1'+q3')*n3>

%   active forces
force_ao>=-m*g*n1>
force_bo>=-m*g*n1>

zero=fr()+frstar()
kane()

code dynamics() passive_mode.f
```

Equation Generation, Powered Mode:

```
%  Derivation of equation of motion of Powered Mode

newtonian n
bodies a,b,f
points h,o,c,ank

variables u{3}'
variables q1,q1',q2,q2',q3,q3'
constants L,g,m,R,I,Xcma,Xcmb,Ycm,d,mtorq
mass a=m,b=m,f=0

inertia a,0,0,I,0,0,0
inertia b,0,0,I,0,0,0
inertia f,0,0,0,0,0,0

q1' = u1
q2' = u2
q3' = u3

simprot(n,f,3,q1)
simprot(f,a,3,q2)
simprot(a,b,3,q3)

%  position vectors
p_o_c> = R*n1>
p_c_ank> = d*f1>
p_ank_h> = L*a1>
p_h_Bo> = Xcmb*b1> - Ycm*b2>
p_o_Bo> =  p_o_c> + p_c_ank> + p_ank_h> + p_h_Bo>
p_ank_Ao> = Xcma*a1> + Ycm*a2>
p_o_Ao> = p_o_c> + p_ank_Ao> + p_c_ank>

%  velocities and angular velocities
v_o_n>=R*q1'*n2>
v_c_n>=dt(p_o_c>,n)  +  v_o_n>
v_Ao_n>=dt(p_o_Ao>,n)  +  v_o_n>
v_Bo_n>=dt(p_o_Bo>,n)  +  v_o_n>

%save pow_vel.all,47:50

w_f_n>=q1'*n3>
w_a_n>=(q1'+q2')*n3>
w_b_n>=(q1'+q2'+q3')*n3>


force_ao>=-m*g*n1>
force_bo>=-m*g*n1>
torque_f>=mtorq*n3>
torque_a>=-mtorq*n3>

zero=fr()+frstar()
kane()

code() dynamics powered_mode.f
```

Jump Conditions:

```
%  Finds the expression for the angular momentum of the system about the new contact point and the
%  angular momentum of the 'back' leg about the hip before heelstrike.

newtonian n
bodies a,b,f
points h,o,c,ank,sank,cp,sf

variables u{3}'
variables q1,q1',q2,q2',q3,q3'
constants L,g,m,R,I,Xcma,Xcmb,Ycm,d,q2loc
mass a=m,b=m,f=0

inertia a,0,0,I,0,0,0
inertia b,0,0,I,0,0,0
inertia f,0,0,0,0,0,0

q1' = u1
q2' = u2
q3' = u3

simprot(n,f,3,q1)
simprot(f,a,3,q2)
simprot(a,b,3,q3)

%  position vectors
p_o_c> = R*n1>
p_c_ank> = d*f1>
p_ank_h> = L*a1>
p_h_Bo> = Xcmb*b1> - Ycm*b2>
p_o_Bo> =  p_o_c> + p_c_ank> + p_ank_h> + p_h_Bo>
p_ank_Ao> = Xcma*a1> + Ycm*a2>
p_h_Ao> = -Xcmb*a1> + Ycm*a2>
p_o_Ao> = p_o_c> + p_ank_Ao> + p_c_ank>

p_sank_h> = L*b1>
p_sank_sf> = cos(q2loc)*D*b1> -sin(q2loc)*D*b2>
p_sf_o> = p_o_c> + p_c_ank> + p_ank_h> + p_h_sank> + p_sank_sf>

 p_cp_o> = -dot( p_sf_o>,n2>)*n2>
p_cp_Ao> =  p_cp_o> + p_o_Ao>
p_cp_Bo> =  p_cp_o> + p_o_Bo>

%  velocities and angular velocities
v_o_n>=R*q1'*n2>
v_c_n>=dt(p_o_c>,n) + v_o_n>
v_Ao_n>=dt(p_o_Ao>,n) + v_o_n>
v_Bo_n>=dt(p_o_Bo>,n) + v_o_n>

w_f_n>=q1'*n3>
w_a_n>=(q1'+q2')*n3>
w_b_n>=(q1'+q2'+q3')*n3>

h_a_h_n> = momentum(angular,h,a)
h_s_cp_n> = momentum(angular,cp,a,b)
h_a_h_n = dot(H_A_h_n>,n3>)
h_s_cp_n = dot(H_s_cp_n>,n3>)
save mom.all,75:78
```

Appendix C: The Simulation, m-files

What follows is a list of all m-files associated with the simulation, a brief explanation of the functions they serve and a hard copy of the actual code.

<u>m-files</u>:

height.m	: returns the height and vert. veloc. of the bottom of the swing foot
init.m	: assigns values to the system parameters/initial conditions
lock_int.m	: the integrator for locked mode, calls *loc_qddt*
loc_qddt.m	: returns the state derivitive for passive mode
mom.m	: returns the desired angular momenta of the sys. before heelstrike
powanim.m	: animates the cartoon created by powered
powered.m	: the main m-file of the simulation that creates a cartoon of the robot
pow_int.m	: the integrator for powered mode, calls *pow_qddt* and *height*
pow-jump.m	: returns the state after heelstrike given the state before heelstrike
pow_qddt.m	: returns the state derivitive for powered mode
pow_stor.m	: stores state vectors for later animation
search.m	: main m-file of limit cycle search
step.m	: returns the state after one step given its initial condidions

<u>height.m</u>

```
function  [h,hd]=height(q);

global D L Q2LOC

qf=q(1);  q1=q(2);  q2=q(3);  qfl=Q2LOC;
qf_dot=q(4);  q1_dot=q(5);  q2_dot=q(6);

cqf=cos(qf);
sqf=sin(qf);

cqf1=cos(qf+q1);
sqf1=sin(qf+q1);

cqf12=cos(qf+q1+q2);
sqf12=sin(qf+q1+q2);

cqf12fl=cos(qf+q1+q2-qfl);
sqf12fl=sin(qf+q1+q2-qfl);


h=D*(cqf+cqf12fl)+L*(cqf1+cqf12);

%total  derivative
hd=(-L*sqf12 - D*sqf12fl)*q2_dot ...
   +(L*(-sqf1 - sqf12) - D*sqf12fl)*q1_dot ...
   +(D*(-sqf-sqf12fl) + L*(-sqf1-sqf12))*qf_dot;
```

<u>init.m</u>

```
function [T0,Q0] = init

% [T0,Q0] = pow_init
%
% Define System Parameters M, Icm, L, Xcm, g, gam
% and Initial Conditions Q0 = [q1 q2 q1dot q2dot]

        global M I L XCMA XCMB YCM G D Q1loc R Tstall
```

```
            global wnoload Q2LOC Q3switch

            global Mfoot Mframe Mhip Xcmfoot Xcmframe Xcmhip
            global Icmfoot Icmframe Icmhip

disp('here we go!')

% Geometry & Environment
            L           = 1;                    % Length of each leg
            R           = 0.0;                  % Radius of foot
            D           = 0.05;                 % Distance from center of foot to ankle
            G     = 10;                         % gravity
            Q2LOC   = 3*pi/4;
            Q3switch=  1.0*2.8625;

%  Motor
            Tstall     =   692;
            wnoload =  100;

% Initial Conditions
            T0          = 0;
            %Q0         = [-pi      0           0           0           0           0];
            %Q0         = [-Q2LOC           Q2LOC   Q3switch  .6*pi/2   0           -1.1];
            %Q0         = [0.1171-Q2LOC Q2LOC   Q3switch  .6*0.4726           0           -0.2559];

     %      Q0          = [-2.13817295623406    Q2LOC    Q3switch   0.43154401278947         0   ...
0.77748810310435];
            Q0          = [-2.29640259398916    Q2LOC    Q3switch  1.0*0.36665928489824          0 ...
   -2.36656356044038];

% Each leg is composed of 3 parts: hip, frame, foot:
% M:      Mass of each part
% Xcm:    Center of mass measured along the leg starting at foot
% Icm:    Moments of inertia for each part about its own center of mass

            Mhip      = 1000;  Mframe    = 0;                          Mfoot    = 1;
            Xcmhip   = L;       Xcmframe = 0.5;                      Xcmfoot = 0;
            Icmhip    = 0;       Icmframe = Mframe*L/12;              Icmfoot  = 0;
                                 Ycmframe = 0;
% Assemble leg
            % Mass of one complete leg
            M           = Mhip + Mframe + Mfoot;
            % Center of mass of each leg
            XCMA    = (Mhip*Xcmhip + Mframe*Xcmframe + Mfoot*Xcmfoot)/ M;
            YCM      = (Mframe*Ycmframe/M);
            % Moment of inertia
            I           = Icmhip + Mhip*((XCMA-Xcmhip)^2 + YCM^2) ...
                          + Icmframe + Mframe*((XCMA-Xcmframe)^2 + YCM^2) ...
                          + Icmfoot + Mfoot*((XCMA-Xcmfoot)^2 + YCM^2);

            XCMB = L - XCMA;
```

---

lock_int.m

```
function [Q,T,collision] = pw_integ(Q0,T0,Steps)

% [Q,T,collision] = locked_integ(Q0,T0,Steps)
%
```

```
% Integrate from initial conditions until the swinging leg
% angle reaches Q3switch

% Index 1: standing leg, index 2: swinging leg

 disp      ('   integrating locked-mode...')

             global Q2LOC Q3switch

% Init IC's, Tolerances
             Y           = [Q0(1:2) Q0(3)+2*pi Q0(4:6)];
             T           = T0;
             Tmax        = 2.5;
             yk          = Q0;                tk        = T0;
             Tol         = 0.09;
             itr         = 1;                 tstep     = 0.01;
             enter       = 0;                 exit      = 0;
             switch      = 0;                 error     = 0;


while (switch==0) & (tk-T0 < Tmax) & (error==0)
             tk = T0 + itr*tstep;
             k1 = tstep * feval('loc_qddt',tk          ,yk        );
             k2  =  tstep  *  feval('loc_qddt',tk+0.5*tstep,yk+0.5*k1);
             k3  =  tstep  *  feval('loc_qddt',tk+0.5*tstep,yk+0.5*k2);
             k4 = tstep * feval('loc_qddt',tk+     tstep,yk+     k3);
             yk = yk + (k1+2*(k2+k3)+k4)/6;

             while yk(3)<0, yk(3)=yk(3)+2*pi; end;

             while yk(3)>=2*pi, yk(3)=yk(3)-2*pi;end;

             T = [T ; tk];
             Y = [Y ; yk];


             % Check if walker fell over
             %if abs(yk(1)) < pi/2
             % error  = 1;
             %end

             % abs(yk(1)+Q2LOC)

             %event detection



             if (yk(6)<0)&(yk(3)+yk(6)*tstep<=Q3switch)
                switch=1;
                yb=yk;
                tb=tk;
                 dt=(Q3switch-yk(3))/yk(6);
                  ynext=[0,0,0,0,0,0];
                  ylast=[1,1,1,1,1,1];
                jj=0;
                 while ynext~=ylast
            jj=jj+1;
                          ylast=ynext;
                k1 = dt * feval('loc_qddt',tb      ,yb      );
                k2 = dt * feval('loc_qddt',tb+0.5*dt,yb+0.5*k1);
                k3 = dt * feval('loc_qddt',tb+0.5*dt,yb+0.5*k2);
                k4 = dt * feval('loc_qddt',tb+   dt,yb+   k3);
                ynext = yb + (k1+2*(k2+k3)+k4)/6;
                dt=dt+(Q3switch-ynext(3))/ynext(6);
          end %while

          T = [T ; tb+dt];
          Y = [Y ; ynext];

        end % if

%%%%end event detection

        itr         = itr + 1;
end             % while ...
Q       = Y;
```

loc_qddt.m

```
function [Qddot,Vcm1,Vcm2,MM] = loc_qddt(t,Q)

% [Qddot,Vcm1,Vcm2,MM] = loc_qddt(t,Q)
%
%  This function returns the state derivitive of the powered walker
%  while in passive mode:    Qddot      = [q1dot 0 q3dot udot]
%  Vcm1,Vcm2 = Velocities of cemters of mass of each leg
%  MM = Generalized Mass Matrix

        global M I L XCMA XCMB YCM G D R

Q1      = Q(1);         Q2      = Q(2);         Q3      = Q(3);
U1      = Q(4);         U2      = Q(6);

% Precalculate sin & cos functions that are used very often
%q1q2   = q1+q2; q1gam  = q1+gam;         q1q2gam = q1+q2+gam;
%sq1    = sin(q1);      sq2     = sin(q2);
%cq1    = cos(q1);      cq2     = cos(q2);
%sq1gam         = sin(q1gam);
%sq1q2gam = sin(q1q2gam);


% RHS
rh(1,1) = M*(L*XCMB*sin(Q3)*U1^2+D*XCMB*sin(Q2+Q3)*U1^2+ ...
        L*YCM*cos(Q3)*(U1+U2)^2+D*YCM*cos(Q2+Q3)*(U1+U2)^2+ ...
        R*YCM*cos(Q2+Q1+Q3)*(U1+U2)^2-G*(D*sin(Q1)+XCMA*sin(Q2+Q1)+ ...
        YCM*cos(Q2+Q1))-G*(D*sin(Q1)+L*sin(Q2+Q1)+XCMB*sin(Q2+Q1+Q3)- ...
        YCM*cos(Q2+Q1+Q3))-D*R*sin(Q1)*U1^2-L*YCM*cos(Q3)*U1^2- ...
        D*YCM*cos(Q2+Q3)*U1^2-L*R*sin(Q2+Q1)*U1^2- ...
        L*XCMB*sin(Q3)*(U1+U2)^2-D*XCMB*sin(Q2+Q3)*(U1+U2)^2- ...
        R*XCMB*sin(Q2+Q1+Q3)*(U1+U2)^2-R*(D*sin(Q1)+XCMA*sin(Q2+Q1)+ ...
        YCM*cos(Q2+Q1))*U1^2);

rh(2,1) = -M*(G*(XCMB*sin(Q2+Q1+Q3)-YCM*cos(Q2+Q1+Q3))- ...
        (L*XCMB*sin(Q3)+D*XCMB*sin(Q2+Q3)-L*YCM*cos(Q3)- ...
        D*YCM*cos(Q2+Q3))*U1^2);
```

```
% Mass Matrix
MM=zeros(2,2);
MM(1,1) =  -2*I - M*(D^2+R^2+XCMA^2+YCM^2+2*D*XCMA*cos(Q2)+ ...
            2*D*R*cos(Q1)+2*R*XCMA*cos(Q2+Q1)-2*D*YCM*sin(Q2)-  ...
            2*R*YCM*sin(Q2+Q1)) - M*(D^2+L^2+R^2+XCMB^2+YCM^2+ ...
            2*D*L*cos(Q2)+2*D*R*cos(Q1)+2*L*XCMB*cos(Q3)+2*L*YCM*sin(Q3)+  ...
            2*D*XCMB*cos(Q2+Q3)+2*D*YCM*sin(Q2+Q3)+2*L*R*cos(Q2+Q1)+ ...
            2*R*XCMB*cos(Q2+Q1+Q3)+2*R*YCM*sin(Q2+Q1+Q3));

MM(1,2) =  -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
            D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q2+Q1+Q3)+ ...
            R*YCM*sin(Q2+Q1+Q3));

MM(2,1) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
            D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q2+Q1+Q3)+ ...
            R*YCM*sin(Q2+Q1+Q3));

MM(2,2) =  -I - M*(XCMB^2+YCM^2);

% Compute Qddot vector
            Udot       = (MM\(rh))';
            Qddot      = [U1 0 U2 Udot(1) 0 Udot(2)];
```

---

mom.m

```
function [hminus,MM]  = mom(Q)

global M I L XCMA XCMB YCM G D R Q2LOC

Q1          = Q(1);                 Q2          = Q(2);    Q3          =  Q(3);
U1          = Q(4);                 U2          = Q(5);                U3          =  Q(6);

hhip= (I-M*(XCMA*XCMB-YCM^2))*(U1+U2) - D*M*(XCMB*cos(Q2)+ ...
            YCM*sin(Q2))*U1 - M*R*(XCMB*cos(Q1+Q2)+YCM*sin(Q1+Q2))*U1;


hcp =   D*M*(2*D+L*cos(Q2)+XCMA*cos(Q2)+2*R*cos(Q1)+XCMB*cos(Q2+Q3)+ ...
        YCM*sin(Q2+Q3)-YCM*sin(Q2)-2*sin(Q1)*(D*sin(Q1)+L*sin(Q1+Q2)- ...
            D*sin(Q2LOC)*cos(Q1+Q2+Q3)-(L-D*cos(Q2LOC))*sin(Q1+Q2+Q3)))*U1 + ...
            (I-M*(XCMB*sin(Q1+Q2+Q3)*(D*sin(Q1)+L*sin(Q1+Q2)-  ...
            D*sin(Q2LOC)*cos(Q1+Q2+Q3)-(L-D*cos(Q2LOC))*sin(Q1+Q2+Q3))- ...
            XCMB^2-YCM^2-L*XCMB*cos(Q3)-L*YCM*sin(Q3)-D*XCMB*cos(Q2+Q3)- ...
            D*YCM*sin(Q2+Q3)-R*XCMB*cos(Q1+Q2+Q3)-R*YCM*sin(Q1+Q2+Q3)- ...
        YCM*cos(Q1+Q2+Q3)*(D*sin(Q1)+L*sin(Q1+Q2)-D*sin(Q2LOC)*cos(Q1+Q2+Q3)- ...
            (L-D*cos(Q2LOC))*sin(Q1+Q2+Q3))))*(U1+U2+U3) + (I+L*M*(L+D*cos(Q2)+ ...
            XCMB*cos(Q3)+YCM*sin(Q3)+R*cos(Q1+Q2)-sin(Q1+Q2)*(D*sin(Q1)+L*sin(Q1+Q2)-  ...
            D*sin(Q2LOC)*cos(Q1+Q2+Q3)-(L-D*cos(Q2LOC))*sin(Q1+Q2+Q3)))+ ...
            M*(XCMA^2+YCM^2+D*XCMA*cos(Q2)+R*XCMA*cos(Q1+Q2)-D*YCM*sin(Q2)- ...
            R*YCM*sin(Q1+Q2)-XCMA*sin(Q1+Q2)*(D*sin(Q1)+L*sin(Q1+Q2)-  ...
            D*sin(Q2LOC)*cos(Q1+Q2+Q3)-(L-D*cos(Q2LOC))*sin(Q1+Q2+Q3))- ...
            YCM*cos(Q1+Q2)*(D*sin(Q1)+L*sin(Q1+Q2)-D*sin(Q2LOC)*cos(Q1+Q2+Q3)-  ...
            (L-D*cos(Q2LOC))*sin(Q1+Q2+Q3))))*(U1+U2) - M*R*(YCM*sin(Q1+Q2)- ...
            2*R-2*D*cos(Q1)-L*cos(Q1+Q2)-XCMA*cos(Q1+Q2)-XCMB*cos(Q1+Q2+Q3)- ...
            YCM*sin(Q1+Q2+Q3))*U1;

hminus = [ hcp ; hhip ];
```

```
MM=zeros(2,2);

MM(1,1) =  -2*I - M*(D^2+R^2+XCMA^2+YCM^2+2*D*XCMA*cos(Q2)+ ...
            2*D*R*cos(Q1)+2*R*XCMA*cos(Q2+Q1)-2*D*YCM*sin(Q2)-  ...
            2*R*YCM*sin(Q2+Q1)) - M*(D^2+L^2+R^2+XCMB^2+YCM^2+ ...
            2*D*L*cos(Q2)+2*D*R*cos(Q1)+2*L*XCMB*cos(Q3)+2*L*YCM*sin(Q3)+  ...
            2*D*XCMB*cos(Q2+Q3)+2*D*YCM*sin(Q2+Q3)+2*L*R*cos(Q2+Q1)+  ...
            2*R*XCMB*cos(Q2+Q1+Q3)+2*R*YCM*sin(Q2+Q1+Q3));

MM(1,2) =  -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
            D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q2+Q1+Q3)+ ...
            R*YCM*sin(Q2+Q1+Q3));

MM(2,1) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
            D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q2+Q1+Q3)+ ...
            R*YCM*sin(Q2+Q1+Q3));

MM(2,2) =  -I - M*(XCMB^2+YCM^2);
```

powanim.m

```
%  this m-file animates the state response calculated by
%  the powered simulation.

  global Q2LOC D R

clear X
close  all

temp  =  linspace(0,2*pi,50);
cirx  =  R*cos(temp);
ciry  =  R*sin(temp);


X=Qhist;
[n,m]=size(X);

q1 = X(:,1);
q2 = X(:,2);
q3 = X(:,3);

sq1 = sin(q1); cq1 = cos(q1);
sq2 = sin(q2); cq2 = cos(q2);
sq12=sin(q1+q2); cq12 = cos(q1+q2);
sq123=sin(q1+q2+q3); cq123 = cos(q1+q2+q3);
sq1234=sin(q1+q2+q3-Q2LOC);  cq1234 = cos(q1+q2+q3-Q2LOC);

xf = -R*q1;
yf  =  R*ones(n,1);
xa = xf - D*sq1;
ya = yf + D*cq1;
xh = xa - L*sq12;
yh = ya + L*cq12;
x2 = xh - L*sq123;
y2 = yh + L*cq123;
xsf = x2 - D*sq1234;
```

```
ysf = y2 + D*cq1234;

figure(3)
axis('square')
axis('off')
axis([-1.5,1.5,-1.5,1.5]);
hold

side=1.5;
a=[-side -side -side side side;side -side side side -side];
b=[0 -side side side -side; 0 side side -side -side];
for i=1:n
  cla

   line(a,b,'color','w')

   plot(cirx+xf(i),ciry+yf(i),'c');
   plot(cirx+xsf(i),ciry+ysf(i),'c');

  x=[xf(i) xa(i) xh(i) x2(i); xa(i) xh(i) x2(i) xsf(i)];
  y=[yf(i) ya(i) yh(i) y2(i); ya(i) yh(i) y2(i) ysf(i)];

  stride=xsf(i);


%   x=[0 -L*sin(X(i,1));-L*sin(X(i,1)) -L*sin(X(i,1))-L*sin(X(i,1)+X(i,2))];
%   y=[0  L*cos(X(i,1)); L*cos(X(i,1))  L*cos(X(i,1))+L*cos(X(i,1)+X(i,2))];
  line(x,y)
 drawnow
end
```

---

powered.m

```
% Simulation for powered, straight-legged walker
% Includes:
%          - integration of equations of motion
%          - collision check when swinging foot hits ground
%          - jump conditions at foot collison
%          - storage of state history

% Initialize physical parameters & IC's
         clear      all
         global     M I L XCMA XCMB YCM G D R
         % Parameters for storage subroutine
         global     Thist Qhist
         global     Steps

         [T0,Q0]  = init;

% Store IC's                    % Format
         Thist     = [];        % [T]
         Qhist     = [];        % [q1 q2 q3 q1dot q2dot q3dot]

% Define various variables
         Steps     = 0;
         StepMax = 10;          % Maximum number of steps to walk
         stepdisp= 5;           % Display every ... Steps
```

```
%  Walking begins with while loop

while Steps < StepMax
        clear Qstep Tstep Qloc Tloc Qpow Tpow

        Steps              = Steps + 1;
        qtest(Steps)       = Q0(1);

        % WALK ONE STEP
        if (rem(Steps,stepdisp)==0) | (Steps==1)
          disp     (['Integrating  Step  '  num2str(Steps)...
                       ' ;  To go : ' num2str(StepMax-Steps)])
        end

        % Integrate power_mode until collision
        [Qpow,Tpow,collision]      =  pow_int(Q0,T0,Steps);

        [n,m] = size(Qpow);
        Qminus = Qpow(n,:);      %  The state of the walker at
                                                  %  heel-strike
        Tminus = Tpow(n);

        if collision == 0
          disp     ('No Collision detected !')
          pow_stor(Tpow,Qpow)
          break
        end

        % Collision: Jump Conditions over collision
        [Qplus,Stride]      = pow_jump(Qminus);

        % Set IC's for integration of next step
        Q0        = Qplus;
        T0        = Tminus;

        % Integrate locked_mode until power is "turned on"
        [Qloc,Tloc]         =  lock_int(Q0,T0,Steps);

        [n,m] = size(Qloc);
        Q0 = Qloc(n,:);          %  Sets initial conditions for power-mode.
        T0 = Tloc(n);


        Qstep = [Qpow ; Qloc ];
        Tstep = [Tpow ; Tloc ];



        % Append state vectors of this step to state history

        pow_stor(Tstep,Qstep)
        %strideStore(Steps,1)         = Stride;

end                  % while Steps<=StepMax


return
```

pow_int.m

```
function [Q,T,collision] = pow_int(Q0,T0,Steps)

% [Q,T,collision] = pow_int(Q0,T0,Steps)
%
% Integrate from initial conditions until heelstrike

        global L

% Index 1: standing leg, index 2: swinging leg

 disp     ('   integrating power-mode...')

% Init IC's, Tolerances
        Y          = Q0;              T          = T0;
        Tmax       = 1;
        yk         = Q0;              tk         = T0;
        xTol       = -0.01*L;
        itr        = 1;               tstep      = 0.01;
        enter      = 0;               exit       = 0;
        collision  = 0;               error      = 0;


while (collision==0) & (tk-T0 < Tmax) & (error==0)
        tk = T0 + itr*tstep;
        k1 = tstep * feval('pow_qddt',tk        ,yk        );
        k2 = tstep * feval('pow_qddt',tk+0.5*tstep,yk+0.5*k1);
        k3 = tstep * feval('pow_qddt',tk+0.5*tstep,yk+0.5*k2);
        k4 = tstep * feval('pow_qddt',tk+    tstep,yk+    k3);
        yk = yk + (k1+2*(k2+k3)+k4)/6;
        T = [T ; tk];
        Y = [Y ; yk];


        % Check if walker fell over
        %if abs(yk(1)) < pi/2
        % error  = 1;
        %end


        % collision check
        [h,dh]=height(yk);
        if h+dh*tstep<=0
          collision=1;
         yb=yk;
         tb=tk;
         dt=-h/dh;
           ynext=[0,0,0,0,0,0];
           ylast=[1,1,1,1,1,1];
         jj=0;
          while ynext~=ylast
        jj=jj+1;
                ylast=ynext;
                k1 = dt * feval('pow_qddt',tb       ,yb       );
                k2 = dt * feval('pow_qddt',tb+0.5*dt,yb+0.5*k1);
                k3 = dt * feval('pow_qddt',tb+0.5*dt,yb+0.5*k2);
                k4 = dt * feval('pow_qddt',tb+    dt,yb+    k3);
                ynext = yb + (k1+2*(k2+k3)+k4)/6;
                [h,dh]=height(ynext);
                dt=dt-h/dh;
          end %while

          T = [T ; tb+dt];
          Y = [Y ; ynext];

        end  %if

        %%%%%%%%%%%

        itr        = itr + 1;
end                % while ...
Q          = Y;
```

pow_jump.m

```
function [qplus,dcp] = pow_jump(qminus);

%uses mom.m
global Q2LOC D L

[Hminus,MM]=mom(qminus);

qplus(1)=qminus(1)+qminus(2)+qminus(3)-Q2LOC-pi;
qplus(2)=Q2LOC;
qplus(3)=-qminus(3);

qdotplus=-MM\Hminus;

qplus=[qplus';qdotplus(1);0;qdotplus(2)]';

s1=sin(qminus(1));
s12=sin(qminus(1)+qminus(2));
s123=sin(qminus(1)+qminus(2)+qminus(3));
s1234=sin(qminus(1)+qminus(2)+qminus(3)-Q2LOC);

dcp=D*(s1+s1234)+L*(s12+s123);
```

pow_qddt.m

```
function [Qddot,MM] = pow_qddt(t,Q)

% [Qddot,Vcm1,Vcm2,MM] = pw_qddot(t,Q)
%
%  This function returns the state derivitive of the powered walker
%  while in powered mode:   Qddot      = [q1dot q2dot q3dot udot]

        global M I L XCMA XCMB YCM G D R Tstall wnoload

Q1         = Q(1);             Q2         = Q(2);   Q3         =  Q(3);
U1         = Q(4);             U2         = Q(5);              U3         =  Q(6);

MTORQ  =  -(Tstall/wnoload)*U2 + Tstall;
```

```
% Precalculate sin & cos functions that are used very often
%q1q2    = q1+q2; %q1gam  = q1+gam;            %q1q2gam = q1+q2+gam;
%sq1     = sin(q1);             %sq2     = sin(q2);
%cq1     = cos(q1);             %cq2     = cos(q2);
%sq1gam             = sin(q1gam);
%sq1q2gam = sin(q1q2gam);



MM(1,1) = -2*I - M*(D^2+R^2+XCMA^2+YCM^2+2*D*R*cos(Q1)+ ...
          2*D*XCMA*cos(Q2)+2*R*XCMA*cos(Q1+Q2)-2*D*YCM*sin(Q2)- ...
          2*R*YCM*sin(Q1+Q2)) - M*(D^2+L^2+R^2+XCMB^2+ ...
          YCM^2+2*D*L*cos(Q2)+2*D*R*cos(Q1)+2*L*XCMB*cos(Q3)+ ...
          2*L*YCM*sin(Q3)+2*D*XCMB*cos(Q2+Q3)+2*D*YCM*sin(Q2+Q3)+ ...
          2*L*R*cos(Q1+Q2)+2*R*XCMB*cos(Q1+Q2+Q3)+2*R*YCM*sin(Q1+Q2+Q3));

MM(1,2) = -2*I - M*(XCMA^2+YCM^2+D*XCMA*cos(Q2)+ ...
          R*XCMA*cos(Q1+Q2)-D*YCM*sin(Q2)-R*YCM*sin(Q1+Q2)) - ...
          M*(L^2+XCMB^2+YCM^2+D*L*cos(Q2)+2*L*XCMB*cos(Q3)+ ...
          2*L*YCM*sin(Q3)+D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+ ...
          L*R*cos(Q1+Q2)+R*XCMB*cos(Q1+Q2+Q3)+R*YCM*sin(Q1+Q2+Q3));


MM(1,3) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
          D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q1+Q2+Q3)+ ...
          R*YCM*sin(Q1+Q2+Q3));

MM(2,1) = -2*I - M*(XCMA^2+YCM^2+D*XCMA*cos(Q2)+ ...
          R*XCMA*cos(Q1+Q2)-D*YCM*sin(Q2)-R*YCM*sin(Q1+Q2)) - ...
          M*(L^2+XCMB^2+YCM^2+D*L*cos(Q2)+2*L*XCMB*cos(Q3)+ ...
          2*L*YCM*sin(Q3)+D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+ ...
          L*R*cos(Q1+Q2)+R*XCMB*cos(Q1+Q2+Q3)+R*YCM*sin(Q1+Q2+Q3));

MM(2,2) = -2*I - M*(XCMA^2+YCM^2) - ...
          M*(L^2+XCMB^2+YCM^2+2*L*XCMB*cos(Q3)+2*L*YCM*sin(Q3));

MM(2,3) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3));

MM(3,1) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3)+ ...
          D*XCMB*cos(Q2+Q3)+D*YCM*sin(Q2+Q3)+R*XCMB*cos(Q1+Q2+Q3)+ ...
          R*YCM*sin(Q1+Q2+Q3));

MM(3,2) = -I - M*(XCMB^2+YCM^2+L*XCMB*cos(Q3)+L*YCM*sin(Q3));

MM(3,3) = -I - M*(XCMB^2+YCM^2);


RHS(1,1) = -M*(G*(D*sin(Q1)+XCMA*sin(Q1+Q2)+YCM*cos(Q1+Q2))+ ...
          G*(D*sin(Q1)+L*sin(Q1+Q2)+XCMB*sin(Q1+Q2+Q3)- ...
          YCM*cos(Q1+Q2+Q3))+2*D*R*sin(Q1)*U1^2+ ...
          D*YCM*cos(Q2+Q3)*U1^2+D*L*sin(Q2)*(U1+U2)^2+ ...
          D*XCMA*sin(Q2)*(U1+U2)^2+D*YCM*cos(Q2)*(U1+U2)^2+ ...
          L*YCM*cos(Q3)*(U1+U2)^2+L*R*sin(Q1+Q2)*(U1+U2)^2+ ...
          L*XCMB*sin(Q3)*(U1+U2+U3)^2+R*XCMA*sin(Q1+Q2)*(U1+U2)^2+ ...
          R*YCM*cos(Q1+Q2)*(U1+U2)^2+D*XCMB*sin(Q2+Q3)*(U1+U2+U3)^2+ ...
          R*XCMB*sin(Q1+Q2+Q3)*(U1+U2+U3)^2-D*L*sin(Q2)*U1^2- ...
          D*XCMA*sin(Q2)*U1^2-D*YCM*cos(Q2)*U1^2-D*XCMB*sin(Q2+Q3)*U1^2- ...
          L*XCMB*sin(Q3)*(U1+U2)^2-L*YCM*cos(Q3)*(U1+U2+U3)^2- ...
```

```
            D*YCM*cos(Q2+Q3)*(U1+U2+U3)^2-R*YCM*cos(Q1+Q2+Q3)*(U1+U2+U3)^2);

RHS(2,1) = MTORQ - G*M*(XCMA*sin(Q1+Q2)+YCM*cos(Q1+Q2)) - ...
            G*M*(L*sin(Q1+Q2)+XCMB*sin(Q1+Q2+Q3)-YCM*cos(Q1+Q2+Q3)) - ...
            M*(D*YCM*cos(Q2+Q3)*U1^2+L*YCM*cos(Q3)*(U1+U2)^2+ ...
            L*XCMB*sin(Q3)*(U1+U2+U3)^2-D*L*sin(Q2)*U1^2- ...
            D*XCMB*sin(Q2+Q3)*U1^2-L*XCMB*sin(Q3)*(U1+U2)^2- ...
            L*YCM*cos(Q3)*(U1+U2+U3)^2-D*(XCMA*sin(Q2)+YCM*cos(Q2))*U1^2);

RHS(3,1) = M*(D*XCMB*sin(Q2+Q3)*U1^2+L*XCMB*sin(Q3)*(U1+U2)^2- ...
            G*(XCMB*sin(Q1+Q2+Q3)-YCM*cos(Q1+Q2+Q3))- ...
            D*YCM*cos(Q2+Q3)*U1^2-L*YCM*cos(Q3)*(U1+U2)^2);


% Compute Qddot vector
            udot        = (MM\RHS)';
            Qddot       = [U1 U2 U3 udot];
```

---

pow_stor.m

```
function pow_stor(TStep,QStep)

% pw_stor(TStep,QStep)
%
% Stores the state history in the matrices Thist and Qhist

            global    Thist Qhist

            Qhist = [Qhist;QStep];
            Thist = [Thist;TStep];
```

---

search.m

```
%  This m-file finds and classifies the stability
%  of limit cycles of the powered walker.
%  John Camp, July 1997

            clear       all
            global      M I L XCMA XCMB YCM G D R

%  Physical Parameters
            [to,yo]   = init;

            tol       =  0.0000000001;
            pert      = sqrt(tol);            %  good for forward diff.
            n         = 0;
            iter      = 100;
            fixedpt   = 'false';

while (strcmp(fixedpt,'true')==0) & (n < iter)

  y = step(yo);

  gold(1) = y(1)-yo(1);
  gold(2) = y(4)-yo(4);
  gold(3) = y(6)-yo(6);

  %  Perturb I.C.'s, get col. #1 of jacobian of G
  yg = yo;
  yg(1) = yo(1)+pert;

  y = step(yg);

  gnew(1) = y(1)-yg(1);
  gnew(2) = y(4)-yg(4);
  gnew(3) = y(6)-yg(6);
  dg(:,1) = (gnew-gold)';

  %  Perturb I.C.'s, get col. #2 of jacobian of G
  yg = yo;
  yg(4) = yo(4)+pert;

  y = step(yg);

  gnew(1) = y(1)-yg(1);
  gnew(2) = y(4)-yg(4);
  gnew(3) = y(6)-yg(6);
  dg(:,2) = (gnew-gold)';

  %  Perturb I.C.'s, get col. #3 of jacobian of G
  yg = yo;
  yg(6) = yo(6)+pert;

  y = step(yg);

  gnew(1) = y(1)-yg(1);
  gnew(2) = y(4)-yg(4);
  gnew(3) = y(6)-yg(6);
  dg(:,3) = (gnew-gold)';

  n
   disp('thinking...')
  dgdy = (1/pert).*dg;

  dely = dgdy\(-gold)';          %   modification to I.C.'s

  yo(1) = yo(1) + dely(1);
  yo(2) = yo(2);
  yo(3) = yo(3);
  yo(4) = yo(4) + dely(2);
  yo(5) = 0;
  yo(6) = yo(6) + dely(3);

  yo

  if abs(dely) < tol/10;
    fixedpt = 'true';
  end

  n = n+1;

end            %  while


eig(dgdy+eye(3))               %  calculates the eigen values of the jacobian near the fixed point
```

step.m

```
function [ynew] = step(Q0)

  %  This function returns the state of the walker
  %  after one step.

            clear Qloc Qpow Qstep Tstep
            global R D L Q2LOC Qstep Tstep Q3switch Tstall wnoload torque



            T0=0;
            Steps=10;                     % not needed for 'step'

  % Integrate power_mode until collision
            [Qpow,Tpow,collision]      =  pow_int(Q0,T0,Steps);

            [pn,m] = size(Qpow);
            Qminus = Qpow(pn,:);      %  The state of the walker at heel-strike
            Tminus = Tpow(pn);

            if collision == 0
              disp     ('No Collision detected !')
              break
            end

  % Collision: Jump Conditions over collision
            [Qplus,Stride]       = pow_jump(Qminus);

% Integrate locked_mode until power is "turned on"
            [Qloc,Tloc]          =  lock_int(Qplus,Tminus,Steps);

            [n,m] = size(Qloc);
            ynew = Qloc(n,:);          %  Sets initial conditions for next step.

Qstep=[Qpow;Qloc];
Tstep=[Tpow;Tloc];

%%%%%%%%%%%%%%%%%%%%%%%%%  animate this step %%%%%%%%%%%%%%%%%%%%%%%%%%%

anima=1;
if anima==0

clear X
close all

temp  =  linspace(0,2*pi,50);
cirx  =  R*cos(temp);
ciry  =  R*sin(temp);


for  jj=1:2

if jj == 1
  X=Qpow;
else
```

```
  X=Qloc;
    disp('switched***********************************')
end

%  X=Qstep;

[n,m]=size(X);

q1 = X(:,1);
q2 = X(:,2);
q3 = X(:,3);

sq1 = sin(q1); cq1 = cos(q1);
sq2 = sin(q2); cq2 = cos(q2);
sq12=sin(q1+q2); cq12 = cos(q1+q2);
sq123=sin(q1+q2+q3); cq123 = cos(q1+q2+q3);
sq1234=sin(q1+q2+q3-Q2LOC);  cq1234 = cos(q1+q2+q3-Q2LOC);

xf = -R*q1;
yf =  R*ones(n,1);
xa = xf - D*sq1;
ya = yf + D*cq1;
xh = xa - L*sq12;
yh = ya + L*cq12;
x2 = xh - L*sq123;
y2 = yh + L*cq123;
xsf = x2 - D*sq1234;
ysf = y2 + D*cq1234;

figure(3)
axis('square')
axis('off')
axis([-4,4,-4,4]);
hold

side=4;
a=[-side -side -side side side;side -side side side -side];
b=[0 -side side side -side; 0 side side -side -side];
for i=1:n
  cla

   line(a,b,'color','w')

   plot(cirx+xf(i),ciry+yf(i),'c');
   plot(cirx+xsf(i),ciry+ysf(i),'c');

  x=[xf(i) xa(i) xh(i) x2(i); xa(i) xh(i) x2(i) xsf(i)];
  y=[yf(i) ya(i) yh(i) y2(i); ya(i) yh(i) y2(i) ysf(i)];


%   x=[0  -L*sin(X(i,1));-L*sin(X(i,1))  -L*sin(X(i,1))-L*sin(X(i,1)+X(i,2))];
%   y=[0   L*cos(X(i,1)); L*cos(X(i,1))   L*cos(X(i,1))+L*cos(X(i,1)+X(i,2))];
   line(x,y)
  drawnow
end

end
end
```

```
clf

Tstep=Tstep*sqrt(9.8);
Tpow=Tpow*sqrt(9.8);
Tloc=Tloc*sqrt(9.8);

[j,m]=size(Tstep)
%plot(Tstep,Qstep(:,1)+Qstep(:,2),[Tstep;Tstep(j)],[Qstep(:,2);Q2LOC]-
Q2LOC,Tstep,Qstep(:,1)+Qstep(:,2)+Qstep(:,3)-pi)
plot(Tpow,Qpow(:,1)+Qpow(:,2),[Tstep;Tstep(j)],[Qstep(:,2);Q2LOC]-
Q2LOC,Tpow,Qpow(:,1)+Qpow(:,2)+Qpow(:,3)-pi)
hold on
plot(Tloc,Qloc(:,1)+Qloc(:,2),Tloc,Qloc(:,1)+Qloc(:,2)+Qloc(:,3)-pi)


%plot(Tstep,ones(j)*(Qstep(1,1)+Qstep(1,2)+Qstep(1,3)-pi)'.')
Tstep=Tstep+Tstep(j);
%plot(Tstep,Qstep(:,1)+Qstep(:,2),[Tstep;Tstep(j)],[Qstep(:,2);Q2LOC]-
Q2LOC,Tstep,Qstep(:,1)+Qstep(:,2)+Qstep(:,3)-pi)
```