Modeling, Simulation, and Analysis of a Bipedal Walker

Simon Ng
Computer Science
Michigan State University
ngsimon@msu.edu

Graduate Mentor: **Haiyang Zheng**
Research Supervisor: **Dr. Jonathan Sprinkle**
Faculty Mentor: **Prof. S. Shankar Sastry**

July 29, 2005

Summer Undergraduate Program in
Engineering at Berkeley (SUPERB) 2005

Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Modeling, Simulation, and Analysis of a Bipedal Walker

Simon Ng

Abstract

The goal of this project is to simulate a bipedal walker that walks down a slight slope without any power. We would like to make a bipedal walker that allows continuous dynamics to be interrupted by discrete time events. We will use existing equations from past papers as well as reduplicate dynamics given by Aaron Ames and Bobby Gregg. We will use a software called HyVisual. An analysis of how this was implemented in HyVisual as well as how the graphics were animated in Ptolemy. The results of using these tools will be a bipedal walker where one can expand the model in the future.

1 INTRODUCTION

Consider these dynamics equations of motion

$$M(\theta)\ddot{\theta} + N(\theta,\dot{\theta})\dot{\theta} + \frac{1}{a}g(\theta) = 0 \tag{1}$$

$$\text{where } \theta = \begin{pmatrix} \theta_{ns} \\ \theta_s \end{pmatrix}$$

$$M(\theta) = \begin{pmatrix} \beta^2 & -(\beta+1)\beta\cos(\theta_{ns} - \theta s) \\ -(\beta+1)\beta\cos(\theta_{ns} - \theta s) & (\beta+1)^2(\mu+1)+1 \end{pmatrix} \tag{2}$$

$$N(\theta,\dot{\theta}) = \begin{pmatrix} 0 & -(\beta+1)\beta\dot{\theta}\sin(\theta_{ns} - \theta s) \\ (\beta+1)\beta\dot{\theta}_{ns}\sin(\theta_{ns} - \theta s) & 0 \end{pmatrix} \tag{3}$$

$$g(\theta) = \begin{pmatrix} g\beta\sin(\theta_{ns}) \\ -((\beta+1)(\mu+1)+1)g\sin(\theta_s) \end{pmatrix} \tag{4}$$

The dynamics of the bipedal walker have been calculated and one is looking for a way to test out the equations. One approach to get a better idea for a simulation to occur is on a computer to get an intuitive understanding of the dynamics and to check if the assumptions are correct. By doing this, one will not need to put together any physical objects to check the dynamics. Under this project, an animation of this bipedal walker will be achieved. Given the mathematics from Aaron Ames and Bobby Gregg[4], the process of making the bipedal walker will be implemented in HyVisual[1]. After the components are implemented in HyVisual, the graphics animation will be done in Ptolemy II[2]. In this project, a explanation of how to make these components were built and work together will be explained in more detail.

2 BACKGROUND INFORMATION

The main tool used in this project is:

2.1 HYVISUAL

HyVisual is the software used to model hybrid systems. HyVisual is a subset of Ptolemy where the graphical animators are given while the mathematical expressions are made in HyVisual. Block diagrams for ordinary differential equations, bubble and arc diagrams for finite state machines, and plotters are used in this project to demonstrate continuous time dynamics with discrete time events. Once this is done, this can be put into the graphical animator in Ptolemy where one can utilize the tools to make a graphical simulation.

3 MODELING

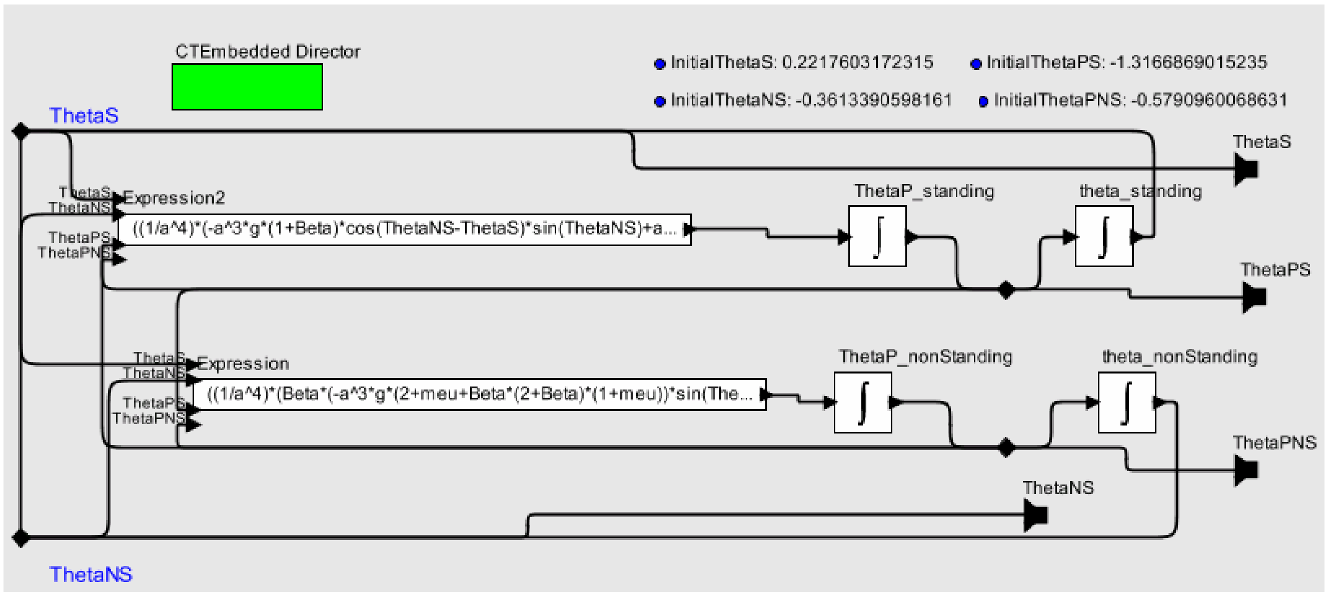

CTEmbedded Director
InitialThetaS: 0.2217603172315
InitialThetaPS: -1.3166869015235
InitialThetaNS: -0.3613390598161
InitialThetaPNS: -0.5790960068631
ThetaS
Expression2
((1/a^4)*(-a^3*g*(1+Beta)*cos(ThetaNS-ThetaS)*sin(ThetaNS)+a...
ThetaP_standing
theta_standing
ThetaPS
Expression
((1/a^4)*(Beta*(-a^3*g*(2+meu+Beta*(2+Beta)*(1+meu))*sin(The...
ThetaP_nonStanding
theta_nonStanding
ThetaPNS
ThetaNS

Figure 1. Continuous Time Dynamics. Top expression represents the dynamics for θ_s and the bottom expression represents the dynamics for θ_{ns}

Modeling the bipedal walker consists of implementing mathematics that allow continuous time dynamic equations to be interrupted by discrete time events at any time. There have been initial variables set based off of a paper by Goswami, Thuilot, and Espiau[3]. To do this, we will use ordinary differential equations to implement the continuous time dynamics and finite state machines to represent discrete time events.

To implement the ordinary differential equations in HyVisual, tools such as an expression operator, relation, integrator, and output port are used. An expression operator is represented by a block, relations are represented by black diamonds, integrators are represented by blocks with integrator signs, and output ports are represented by black polygons. An expression operator is just a way of implementing a mathematical function.

The way that the dynamics were implemented was the following

$$\ddot{\theta_s} = f(\dot{\theta_{ns}}, \dot{\theta_s}, \theta_{ns}, \theta_s) \tag{5}$$

$$\ddot{\theta_{ns}} = g(\dot{\theta_{ns}}, \dot{\theta_s}, \theta_{ns}, \theta_s) \tag{6}$$

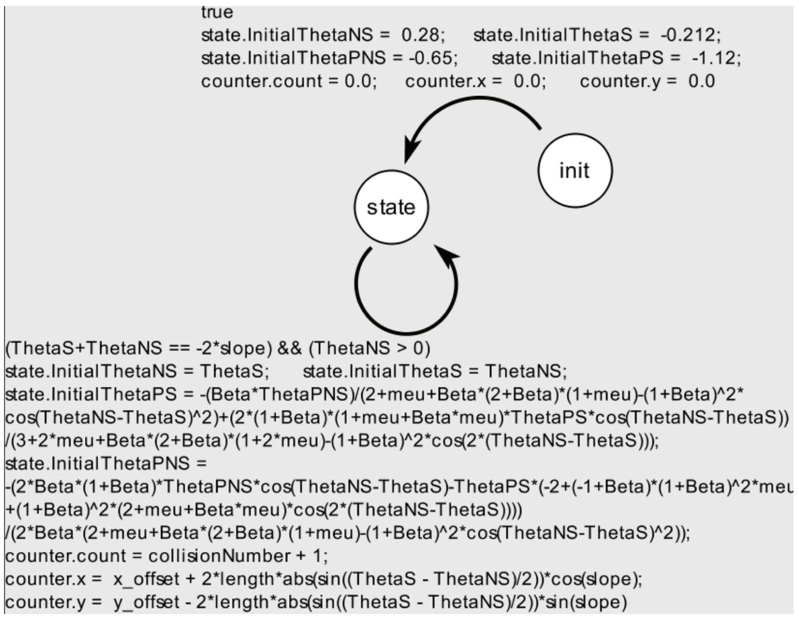
tps= $\theta_{ns} - \theta_s$
q1=$(\beta+1)^2\beta^2\dot{\theta_{ns}}\sin(tps)\cos(tps)\dot{\theta_{ns}}$
q2=$((\beta+1)\beta\dot{\theta_s}\sin(tps))(\beta+1)^2(\mu+1)+1))\dot{\theta_s}$
q3=$((\frac{1}{a}g\beta\sin(\theta_{ns}))(\beta+1)^2(\mu+1)+1))$
q4=$((-(\beta+1)\beta\cos(tps))\frac{1}{a}(-\beta\mu-\beta-\mu-2)g\sin\theta_s$
q5=$((\beta+1)\beta^3\dot{\theta}_{ns}\sin(tps))\dot{\theta}_{ns}$
q6=$(\beta+1)^2\beta^2\dot{\theta_s}\sin(tps)\cos(tps)\dot{\theta_s}$
q7=$\frac{1}{a}g\beta^2-(\beta+1)\sin(\theta_{ns})\cos(\theta_{ns})$
q8=$((-(\beta+1)\beta^2\beta\cos(tps))\frac{1}{a}(-\beta\mu-\beta-\mu-2)g\sin\theta_s$

$$\ddot{\theta_s} = \frac{q1+q2-q3+q4}{(\beta^2((b1)^2(u1)+1))-((b1)^2\beta^2\cos(tps)^2)} \tag{7}$$

$$\ddot{\theta_s} = \frac{q5+q6+q7+q8}{(\beta^2((b1)^2(u1)+1))-((b1)^2\beta^2\cos(tps)^2)} \tag{8}$$

The following is demonstrated in figure 1. There are four input ports and one output port on each expression operator. The input ports denote $\dot{\theta_{ns}}, \dot{\theta_s}, \theta_{ns}, \theta_s$. This is needed because the continuous dynamics need to change over time and a way to do it is to re-input the changing dynamics over time. The output port on each expression go to two different integrators to get an output of $\dot{\theta_s}$and θ_s for the stance leg, and $\dot{\theta_{ns}}$ and θ_{ns} for the non-stance leg. This allows the continuous dynamics to be recalculated while being able to go to the output or back into the expression. Two relations on each expression is used to take the output of each integrator to multiple destinations. For $\dot{\theta_{ns}}$ and $\dot{\theta_s}$, the output will go to the next integrator to make θ_{ns} or θ_s, the output ports so it can go to something outside of the finite state machine, and back into the expression because of the changing dynamics over time. For θ_{ns} and θ_s, the output will go to the output ports for something outside the finite state machine and back into the expression. This is one way to allow the dynamics to be continuous.

To demonstrate the discrete time events, a finite state machine was used. A finite state machine is a modeling technique which alllows for states, transitions, and actions[5]. A state represents results that have been completed. Transitions represent a change in state and this is indicated by a guard condition in equation 7. Actions are ongoing activity that the finite state machine is producing at a certain time. In this model, there are two different states, a initial state and a reset state.



```
true
state.InitialThetaNS =  0.28;     state.InitialThetaS =  -0.212;
state.InitialThetaPNS = -0.65;      state.InitialThetaPS =  -1.12;
counter.count = 0.0;     counter.x =  0.0;      counter.y =  0.0
```

```
(ThetaS+ThetaNS == -2*slope) && (ThetaNS > 0)
state.InitialThetaNS = ThetaS;      state.InitialThetaS = ThetaNS;
state.InitialThetaPS = -(Beta*ThetaPNS)/(2+meu+Beta*(2+Beta)*(1+meu)-(1+Beta)^2*
cos(ThetaNS-ThetaS)^2)+(2*(1+Beta)*(1+meu+Beta*meu)*ThetaPS*cos(ThetaNS-ThetaS))
/(3+2*meu+Beta*(2+Beta)*(1+2*meu)-(1+Beta)^2*cos(2*(ThetaNS-ThetaS)));
state.InitialThetaPNS =
-(2*Beta*(1+Beta)*ThetaPNS*cos(ThetaNS-ThetaS)-ThetaPS*(-2+(-1+Beta)*(1+Beta)^2*meu
+(1+Beta)^2*(2+meu+Beta*meu)*cos(2*(ThetaNS-ThetaS))))
/(2*Beta*(2+meu+Beta*(2+Beta)*(1+meu)-(1+Beta)^2*cos(ThetaNS-ThetaS)^2));
counter.count = collisionNumber + 1;
counter.x =  x_offset + 2*length*abs(sin((ThetaS - ThetaNS)/2))*cos(slope);
counter.y =  y_offset - 2*length*abs(sin((ThetaS - ThetaNS)/2))*sin(slope)
```

Figure 2. Finite State Machine. Has two states: Initial state and Reset State

3.1 Initial State

The initial state is the way to start the finite state machine. There is a guard set to true so when the model starts, it will always have an initial output. In this model, there are 4 states, state.InitialThetaS, state.InitialThetaNS, state.InitialThetaPS, and state.InitialThetaPNS. The state.InitialThetaS and state.InitialThetaNS are used for the angular position of the stance leg and non-stance leg. The state.InitialThetaPS and state.InitialThetaPNS are used for the angular velocity of the stance leg and non-stance leg. By making these initial states, one can set the legs accordingly so that there is enough initial momentum for a stable limit cycle. The counter.x and counter.y is based off the Cartesian coordinate system from where the leg starts. The initial start is (0, 0). The transition from the initial state is to go into the reset state where the mathematics is done.

3.2 Reset State

The reset state is needed for the discrete time events to occur. The reset state can jump around to different functions, new states, at any given time when the guard condition is true. Inside the reset state are where the continuous time dynamics are. Outside of the state, there is another guard condition. The guard condition is as follows:

$$\theta_{ns} + \theta_s = -2\alpha \text{ and } \theta_{ns} > 0$$

This guard condition comes from the Gowsami paper where we used some of his mathematical techniques to solve the problem. The guard condition is a transition equation where when this is true, the legs will switch states where the non-stance leg becomes the stance leg and the stance leg becomes the non-stance leg. This guard condition will allow the transition to another totally different state where the angular velocities have changed because of the impact to the ground, so they can not be simply switched. Because of this, the angular velocities have to be recomputed. The offsets for making the switch of the legs have to be recomputed every time the guard is true. This was done through basic geometry to reposition the center mass and stance leg. The Cartesian coordinates for where the walker is now have to be recorded. To do this, a counter has been made inside the state. The counter is supposed to record the change of position of where the walker is at in the simulation. When the guard condition is true, the Cartesian coordinates are computed in the reset map. After these are computed, the counter records them and increments the counter for collisions up one. This will allow the reset map to restart using current conditions and settings on the new Cartesian coordinates thus allowing the same dynamics to go over and over every time a collision occurs.

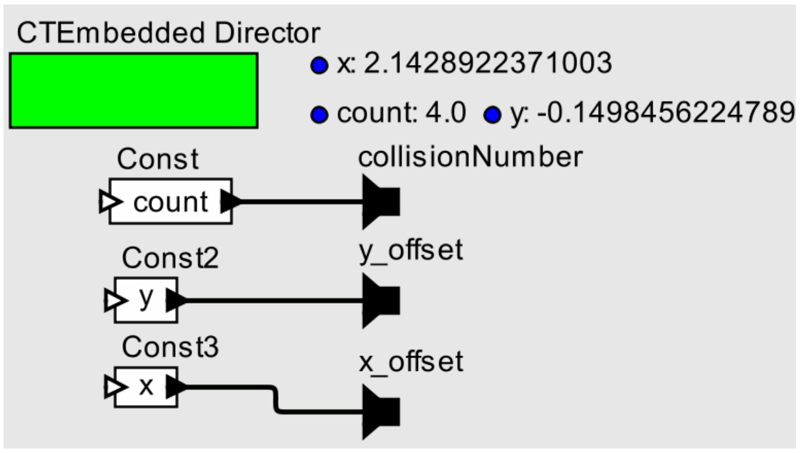


Figure 3. Counter for Collisions and Offsets

In figure four, This is a hierarchial view that has the dynamics on one side and the animation on the other. The x and y displacements for the center mass are computed there with expression operators. The periodic samplers allow the continous dynamics to become discrete so the animation actors can translate the outputs correctly. This is where the initial setting from the Goswami paper are so we can test out some conditions to make the best possible model.

3.3 Trajectory of Center Mass

This is used to show how the center mass moves with respect to the slope in the x-y plane. The straight line in the graph represents the original starting point of the center mass while going down the slope. The mass dips below the trajection because it is not in a stable limit cycle at first but will eventually fall right on the trajectory line. Originally, the mass dips because of where the starting position of the legs is located at. The mass moves in a periodic motion where the discontinuous spots are the dynamics of where each leg is switched. The center mass trajectory increases when the leg is off the ramp and will peak out at some point. The center mass will than decrease and drop until it hits the transition guard.

3.4 Timed Theta Collision

This graph is used to show how the legs are switched and when a collision occurs. The collision line is showing when a collision will occur and that is when the guard condition is true. The θ_s and θ_{ns} are used to

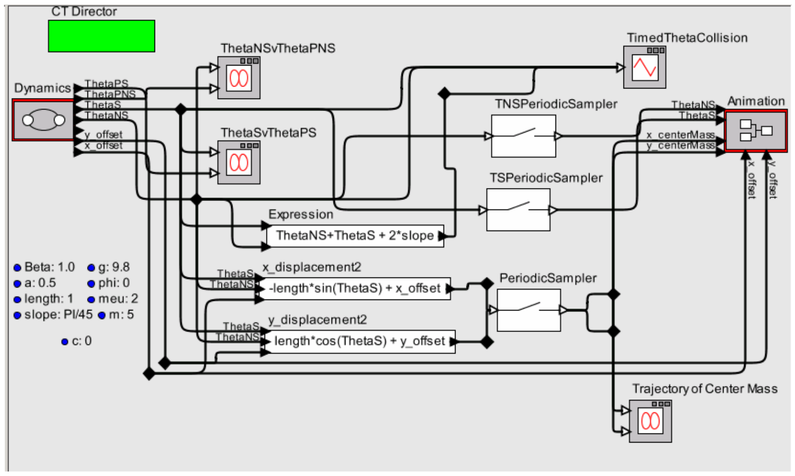


Figure 4. Top View with the dynamics on the left, plots in the middle, and animation on the right

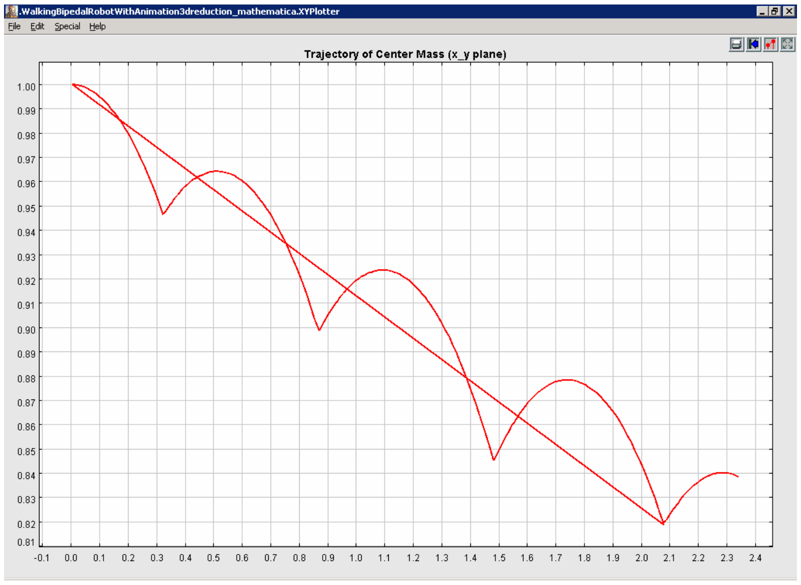


Figure 5. Trajectory of the center mass in the x-y plane

see if the the cycle is periodic over the given time. Over the given time, both legs move in the same motion shown by the periodic lines for each leg.

3.5 θ_s vs $\dot{\theta}_s$

This graph is showing a stable limit cycle between θ_s vs $\dot{\theta}_s$ over a given amount of time. The periodic movement is through the graph having the same general motion over a given amount of time.

3.6 θ_{ns} vs $\dot{\theta}_{ns}$

This graph is also showing a stable limit cycle but between θ_{ns} vs $\dot{\theta}_{ns}$ over a given amount of time. The periodic movement is through the graph having the same general motion over a given amount of time.

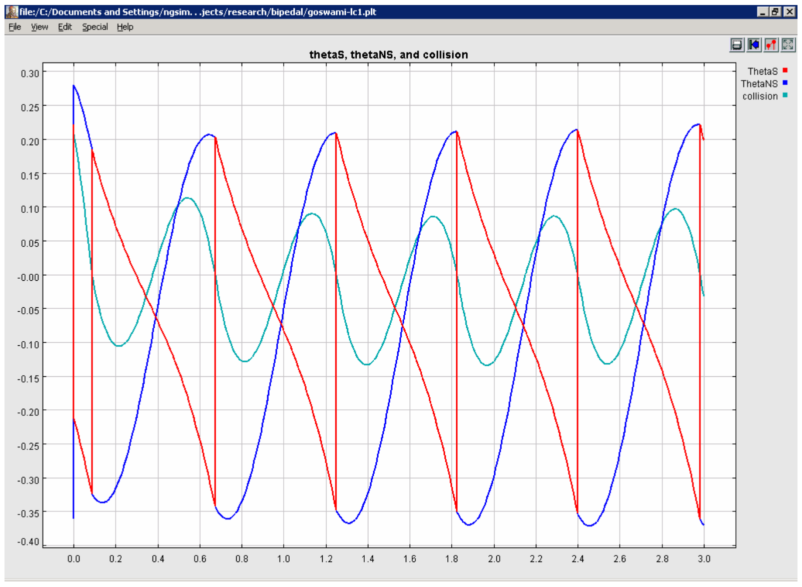


Figure 6. Collision between $theta_s$ and $theta_{ns}$

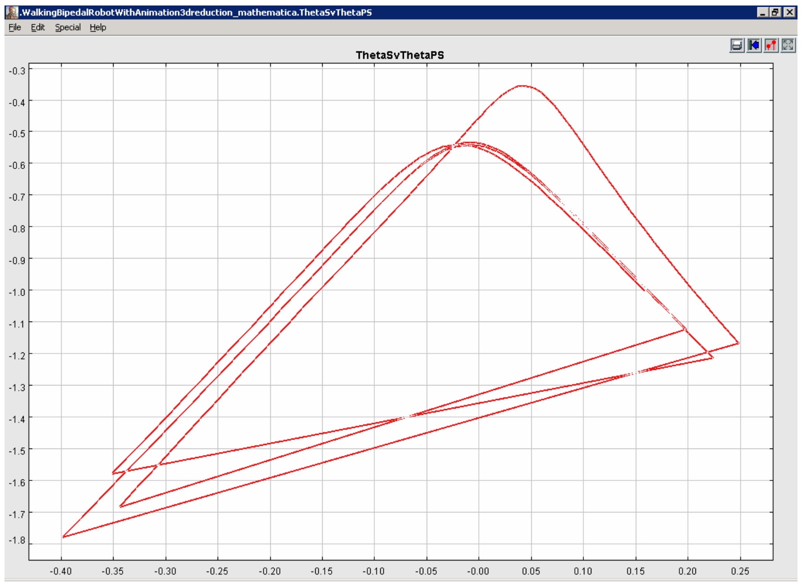


Figure 7. Stable Limit Cycle between θ_s and $\dot{\theta}_s$

4 Graphical Animation

The graphics domain in Ptolemy renders objects and movements based on given inputs and initial settings in the actors. The graphics of the bipedal walker are in two dimensions and fixed in the x-y plane. There are four visualization objects needed for the simulation: swing leg, stance leg, center mass, and a ramp. Some relations are used to stream multiple inputs to some different actors. The Scale actor is used to adjust how the simulation is shown onto the screen. The ViewScreen3D is used render the simulation onto the screen.

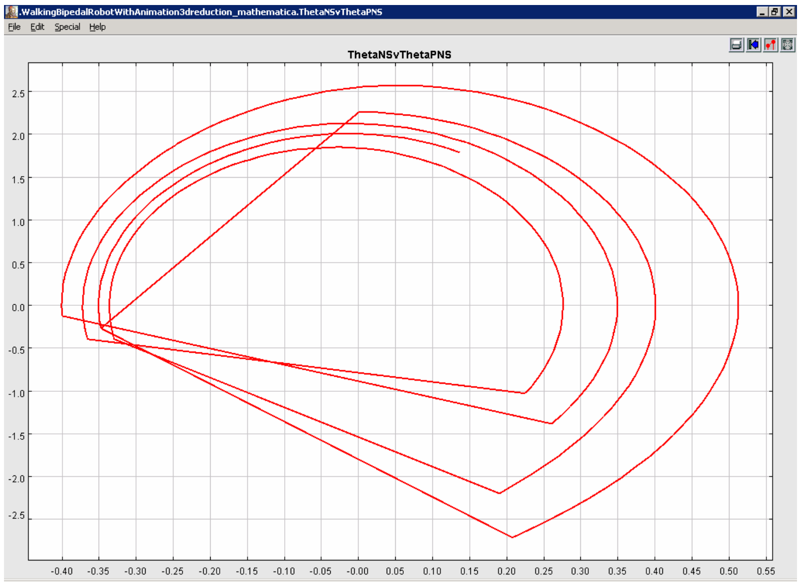


Figure 8. Stable Limit Cycle between θ_{ns} vs $\dot{\theta}_{ns}$

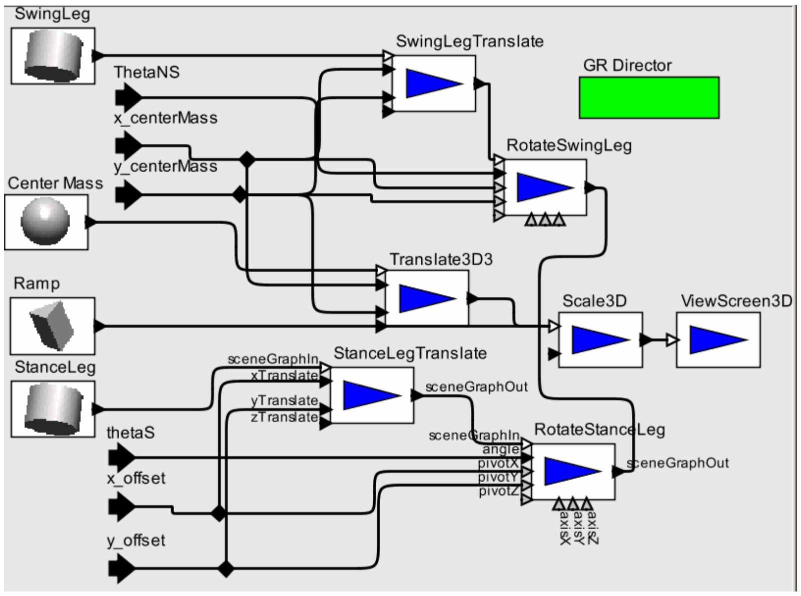


Figure 9. Animation Actor

4.1 Sphere

The sphere actor is used to show the center mass located at the top of the walker. The color is yellow and the radius is 0.03.

4.1.1 Translate

To translate the sphere, the xTranslate and yTranslate will receive its position from the $x_center Mass$ and $y_center Mass$ ports. Since the center mass does not move in an axis direction, it will just move according to the translate actor and not need the rotate actor.

4.2 PolyCylinder

In the PolyCylinder actor, a number of coordinates can be inputted to make the PolyCylinder. These coordinates are given in the x-y plane. In this model, there are 3 different sets of Cartesian coordinates to denote a ramp. After these are made, the thickness was added onto this to make a width of the ramp. The width runs along the z-axis since the coordinates are given in the x-y plane.

4.3 Swing Leg

There are three parts to the swing, non-stance, leg: a cylinder, translate actor, and rotate actor.

4.3.1 Cylinder

The cylinder is used for making the leg visual in the simulation. The leg was made by renaming and making a new radius, length, and color. All of this can be defined in the figure by using the configure area of the translate actor. The radius is 0.01, height is 1.0, and the color is red.

4.3.2 Translate

The translate actor is used to position the leg. There is one input and output port, SceneGraphIn and SceneGraphOut, that is used to connect to the other graphics actors. The position of the leg before changing anything with the actor is that half of the leg is above the x-axis, half is below, and it is parallel to the y-axis. To change this, the actor needs to be configured to where the y position goes up 0.5. After the change, the whole leg is above the x-axis and parallel to the y-axis. The leg also has to move according to the mathematics done in the dynamics. To do this, there are some input ports for the Cartesian coordinates to be plugged in and show the amount of translation over a given time.

4.3.3 Rotate

Rotating the leg is needed to show the movement of the leg. Similar to the translate there is an input and output port to connect the graphics actors. When configuring the rotate actor, the pivot location and in which direction the leg moves is the first task. The pivot location is located on the bottom of the leg after the translation has been done. To switch the location and make it at the top, configuring the initial angle to Pi will flip the leg to the bottom so the pivot position to be the top of the leg. To pick an axis of how the leg rotates, the actor has an axis direction section where it can be picked. To rotate on the x-y plane, the z axis of rotation will be picked. For the input ports, angle, pivotX, and pivotY are needed. The angle port is used to show how the leg rotates counterclockwise based on the dynamics. This is done by feeding in ThetaNS into the angle port. The pivotX and pivotY is used to show where the position of the leg will pivot over time.

4.4 Stance Leg

Like the Swing Leg here are three parts to the non-swing, stance, leg: a cylinder, translate actor, and rotate actor.

4.4.1 Cylinder

The cylinder to make the stance leg is the same as above for the swing leg except the color is blue.

4.4.2 Translate

The translate for this leg is also the same as above for the swing leg. The only difference is the xTranslate and yTranslate have different ports coming in. This is different because the pivot position and where the leg is located are not the same for the stance leg.

4.4.3 Rotate

The rotate actor is pretty similar to the swing leg. The axis direction is the same but the major difference is that the pivot position has to be on the bottom and the leg to rotate clockwise. Because of this, the initial angle will be zero to leave the pivot at the bottom. The angle, pivotX, and pivotY are used in the same context but receive different values because of how this leg works.

5 CONCLUSION

With our model, we were able to simulate and animate the two-dimensional bipedal walker. Thanks to Aaron and Bobby, we were able to help them visualize stable limit cycles so this bipedal walker could walk forever if we wanted the simulation. By making this model, we were able to get an intuitive understanding of the dynamics behind this bipedal walker.

6 FUTURE WORK

Due to the time constraint, we were unable to simulate and animate a three-dimensional bipedal walker that could walk in a stable limit cycle. We were able to input the new three-dimensional mathematics into our existing model but we did not have enough time to finish the animation. We would like to finish this at some point if given the time as well as expand on the future horizons that this could bring.

7 Acknowledgements

I would like to thank the National Science Foundation and UC Berkeley for putting on SUPERB-IT. I would like to thank the CHESS project for believing in my abilities to participate in your research group. Thanks to Dr. Jonathan Sprinkle, Rebecca Brown, Shelia Humphreys, Beatriz Flores-Lopez, and the rest of the SUPERB group for all the help and fun times. Last but not least, I would like to thank Haiyang Zheng, Aaron Ames, and Bobby Gregg. Without them, this project would have not been possible.

References

[1] C. Brooks, A. Cataldo, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng. Hyvisual: A hybrid system visual modeler. Technical Memorandum UCB/ERL M05/24, University of California, Berkeley, Berkeley, CA 94720, July 2005.

[2] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in java. Technical Memorandum UCB/ERL M05/23, University of California, Berkeley, July 2005. http://ptolemy.eecs.berkeley.edu/publications/papers/04/ptIIDesignIntro/.

[3] A. Goswami, B. Thuilot, and B. Espiau. Compass-like biped robot: Part 1: Stability and bifurification of passive gaits. Technical report, INRIA Rhone Alpes, France, Oct. 1996.

[4] R. Gregg and A. Ames. Hybrid reduction of a bipedal walker from three dimensions to two dimensions. Tecnical report, University of California Berkeley, Berkeley, CA 94720, June 2005. SUPERB-IT.

[5] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. Invited paper in proceedings of hybrid systems: Computation and control (hscc) lncs tbd, Zurich, Switzerland, Mar. 2005.