Two Walking Robot Simulators

Matthew P. Kelly
Cornell University
mpk72@cornell.edu

July 29, 2014

Abstract

This report is an overview of two simulators: One for a simple bipedal model of walking, and one for the Cornell Ranger [2]. The simple biped model is based on three point masses, connected by force actuators. Ranger is well modeled as a planar biped with four rigid bodies, connected by three torque motors. I developed walking control algorithms for both robots, loosely inspired by the SIMBICON [6] locomotion controller.

1 Introduction:

1.1 Ranger Simulator

The Cornell Ranger is a "four-legged biped", which is designed for high-efficiency walking over smooth level ground. It has four physical legs, but they are mechanically coupled such that they act as one "inner" and one "outer" leg. This was intentionally done so that the robot locomotion is easily modeled using planar (2D) dynamics. The "planar robot" is effectively composed of four rigid bodies: outer foot, outer leg, inner foot, and inner leg. The feet are curved, allowing them to roll along the ground. There are well-defined motor models for each of the joint actuators.

1.2 Dynamics Engine

There are a large number of algorithms for computing the motion of systems of rigid bodies. Previous work on Ranger has been done by symbolically solving the Newton-Euler equations for each possible contact configuration during walking (single stance and double stance), and then using high-order variable-step integration with event detection. This method is accurate, but restricts the available behavior of the robot, and makes it difficult to implement motor models and control functions with discontinuities.

Most modern rigid body simulation packages take a more generalizable approach. They use a low-order, fixed-step integration scheme that checks the configuration of the contacts at each time step. An iterative algorithm is then used to compute the contact forces (or impulses). These methods are described in great detail in review paper by Bender et al. [1]. During double stance, Ranger will have multiple (two) contact forces acting on each of its rigid bodies. The paper by Guendelman et al. [3] provides details as to how to deal with this type of contact.

1.3 Controller Background

My PhD research at Cornell is primarily focused on locomotion control for biped robots. As such, I am interested in making this simulation of Ranger walk. There are a large number of algorithms to do this. The Hybrid Zero Dynamics (HZD) algorithm uses a virtual constraint function in the controller to track a prescribed trajectory [4]. Zero Moment Point (ZMP) controlled robots (such as Honda's Asimo) search for [nearly] statically stably gaits, with reliance on ankle torques. Capture Point controlled robots (Mark Raibert's hopping robots [5], and [rumored] Boston Dynamic's Big Dog and Atlas) view walking as a perturbation of falling, relying on carefully placed foot steps. For this project, I am particularly interested in the SIMBICON controller [6], which was developed for control of animated bipeds.

At the highest level of the SIMBICON controller is a gait selector, which allows the robot to switch between different gaits {walking, standing, running, jumping, ...}. The next level down is a finite state machine, where each state accomplishes a simple action {pick-up foot, swing leg, extend leg, ...}. Within each state, a set of two nested Proportional-Derivative controllers (top level is a local controller, lower level is a global controller) are then used to compute the torque at each joint.

2 Simulation Overview

This report discusses two simulations: the Simple Biped simulation, and the Ranger Simulator. The dynamics and control algorithms are different for these two simulations, but the general architecture of the simulation is shared (home-made impulse-based physics engine, written in Java). The remainder of this section discusses a few neat features of the simulation architecture.

2.1 Camera Tracking

In both simulations, it is necessary for the camera to track the robot as it is walking. If the camera is always looking directly at the robot's center of mass, then the simulation looks jerky and it is hard to see the motion of the robot. This problem is solved by modelling the camera as a dynamical system, which is tracking a reference (robot center of mass). Horizontal motion on level ground is made apparent by adding tick-marks to the ground animation.

2.2 GUI

A graphical user interface (GUI) panel is used in both simulations for user interaction. This panel allows the user to adjust the gravity vector, change the simulation playback speed, and display/hide the contact impulses. The Simple Biped simulation has a variety of sliders to adjust the control parameters for each gait. The Ranger simulator has sliders that adjust the motor controllers, as well as change the motor model on the fly.

2.3 Timer

These simulations runs much faster than real time. A simple pause command is not sufficient to ensure consistent playback speed, so a proportional-integral (PI) controller is used to adjust the pause duration on the fly to compensate for the non-real-time nature of the operating system. Playback timing error is typically on the order of milliseconds, in both real-time and slow-motion modes of operation.

3 Dynamics - Simple Biped

The simple biped model is made up of three point masses connected by force and torque actuators. This means that the dynamics for the robot are identical to those of a system of three point masses. Each point mass exerts forces on the others via actuators. The forward simulation was done using symplectic euler integration. I use continuous collision detection between each point mass and the ground. The collision location and time is determined using a linear model for velocity. Once this is done, the time step is split into two parts: before and after collision. The simulation then runs each part of the time step using the correct contact mode. The ground is defined analytically, either as a sum of sine waves or as a constant, making it easy to compute the normal and tangential vectors of the collision impulse at contact.

4 Controller - Simple Biped

The control algorithm for the simple biped is based on hierarchical finite state machines (FSM). The top level FSM allows the user to select between three possible gaits: {STAND, WALK, JUMP}. Each of these gaits then itself runs a FSM. The STAND and JUMP gaits use a SIMBICON controller [6]. The WALK gait uses a heuristic controller, that I developed which is loosely based on SIMBICON.

4.1 Simple Biped - WALK gait - Double Stance

In Double Stance, both feet are on the ground and work together to deliver a desired force vector to the hip mass. This desired force vector is composed of two elements: normal and tangential to the stance (front) leg. The normal component is calculated using a PD controller on the stance leg length, forcing the hip to move like an inverted pendulum. The hip motor is turned off during double stance.

$$f_N = k_p(l_{target} - l_1) - k_d \dot{l}_1 \tag{1}$$

The tangential component is calculated such that it will attempt to control the energy of the pendulum model. Calculated as follows:

$$\dot{\theta}_{target} = \frac{-1}{l_1}\sqrt{v_{target}^2 + 2gl_1\cos(\theta_1)} \tag{2}$$

$$f_T = \frac{-k}{l_1}(\dot{\theta}_{target} - \dot{\theta})_1 \tag{3}$$

Once the desired force vector on the hip is known, a system of linear equations can be generated to compute the force required along each of the legs (taking weight into account). These axial leg forces are then saturated to satisfy actuator limits and then passed to the motors.

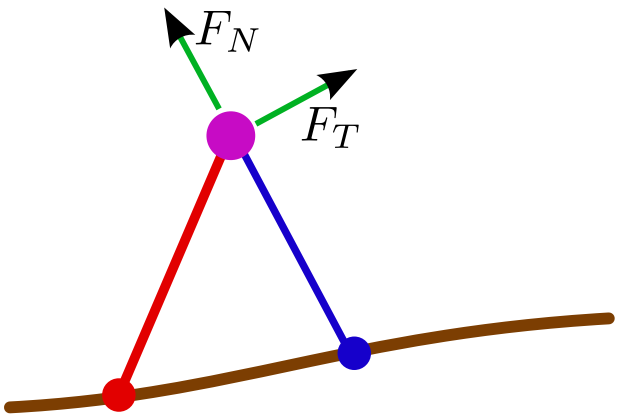

Figure 1: The normal and tangential target forces on the hip during double stance

4.2 Simple Biped - WALK gait - Single Stance

To make things interesting, I've added the constraint that the walking controller cannot use ankle torque. This is like walking with small feet. This restriction makes simple controllers fail, since the system is highly non-linear and under-actuated. The approach that I take is somewhat similar to the hybrid zero dynamics (HZD) controllers developed by Grizzle et al. [4]. Here I use virtual constraints, by way of PD controllers, to keep the system on low-dimensional manifold in the state space. Then a phase variable is used to select a target point along that manifold.

At the core of this algorithm is a Bezier curve, which the swing foot is attempting to track using a phase variable. I use a curve with four control points, at the corners of a rectangle: left edge aligned with the place where the foot left the ground, top is (4/3)*(foot clearance), and the right side is set based on where the foot should strike the ground, assuming that the ground is level.

The phase variable is computed as a linear function of center of mass position, projected onto the ground. The phase variable starts at 0 (swing foot just left the ground) and goes to 1 (when the foot is expected to strike the ground). This phase variable then specifies a point on the Bezier curve where the swing foot should ideally be. A virtual swing leg is then constructed, and the real swing leg uses a PD controller on the leg and hip motors to track the virtual leg. This is shown in Figure 2. The stance leg againt attempts to maintain a fixed length, to make the system behave similiarily to an inverted pendulum. All controllers have a nonlinear term that compensates for the static forces on each actuator due to the weight of the hip and feet.

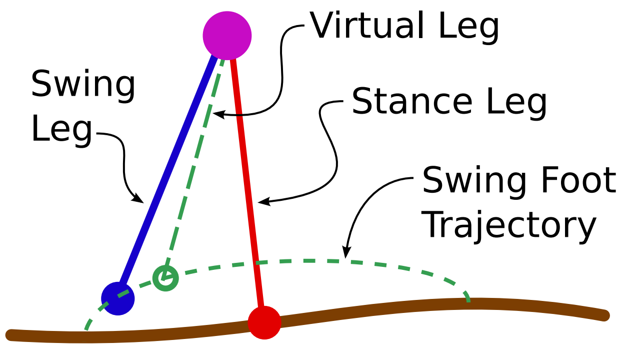

Figure 2: The Bezier curve and virtual leg, used for trajectory tracking in single stance.

This controller is able to create a stable walking gait for the Simple Biped, including the ability to start from a stand-still. It also is able to do this while walking over smooth but hilly ground, with large (12°) slopes and changes in elevation of nearly a third of the nominal leg length.

5 Dynamics - Ranger Model

The dynamics for the Ranger model are more complicated than the simple biped, since Ranger is a closed kinematic chain of rigid bodies. I model the system as four rigid bodies, connected by five joints. Three of the joints connect rigid bodies, and two connect a rigid body to the ground. I express each joint as a constraint for the solver.

The pin joint constraints between bodies are simple, and remain the same throughout the simulation. The body-ground joints are more complicated, as they have two modes of contact: Pinned and Free [1]. For each of these joints, I write out the constraint for both modes, and then let a finite state machine select which constraints are active or not when solving for the contact impulses.

The joint constraints use a position-based scheme, such that the positions are correct at the end of the time step. To achieve this, I include the symplectic euler integration step inside of the constraints.

[1]There should be a third contact mode: Sliding. By neglecting the sliding contact mode, I introduce an error in the contact mode - allowing the ground to pull on the system or allowing the foot to penetrate the ground. I tolerate these errors, since they are corrected by the following time step.

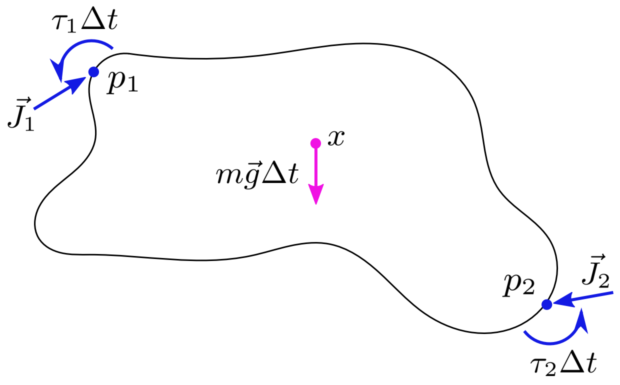

Figure 3: A single rigid body for the ranger model dynamics

This calculation was done symbolically using Matlab, which then automatically wrote the core of the dynamics file in the physics engine.

Figure 3 shows a free body diagram for a single rigid body, used as the template for generating joint constraints. The following equations show the impulse dynamics calculations to compute the change in rotational and translational velocity. Note that the dynamics of this model are in 2D.

$$\Upsilon = \hat{\boldsymbol{k}} \cdot \left((p_1^- - x^-) \times \vec{J}_1 + (p_2^- - x^-) \times \vec{J}_2 \right) / I \tag{4}$$

$$\dot{\theta}^+ = \dot{\theta}^- + \tau_2 \Delta t + \tau_1 \Delta t + \Upsilon \tag{5}$$

$$\dot{x}^+ = \dot{x}^- + \left(\vec{J}_1 + \vec{J}_2 + m\vec{g}\Delta t \right) / m \tag{6}$$

Once the new velocities are known, we use them to compute the new position (x) and orientation (θ) of the rigid body and the joint contact points (p_i).

$$\theta^+ = \theta^- + \dot{\theta}^+ \Delta t \tag{7}$$

$$x^+ = x^- + \dot{x}^+ \Delta t \tag{8}$$

$$p_1^+ = p_1^- + \Delta t \left(\dot{x} + (\dot{\theta}^+ \hat{\boldsymbol{k}}) \times (p_1^- - x^-) \right) \tag{9}$$

$$p_2^+ = p_2^- + \Delta t \left(\dot{x} + (\dot{\theta}^+ \hat{\boldsymbol{k}}) \times (p_2^- - x^-) \right) \tag{10}$$

Now we have a set of equations, for each rigid body, that give the state at the next time step as a function of the state at the previous time step and the contact impulses (which are as of yet unknown). In order to find the contact impulses, and thus the state of the system at the next time step, we need to impose constraints on each joint. For this model, there are two types of joint contacts: *Pin* and *Free*.

Suppose p_A is a joint location on body A, and p_B is a joint location on body B. To connect these two bodies with a pin joint, then the following constraint equation is imposed:

$$0 = p_A^+ - p_B^+ \tag{11}$$

For a free joint A, such as the contact point on the bottom of the foot when the foot is swinging, the joint forces are constrained to be zero.

$$0 = \vec{J}_A \tag{12}$$

The bottom of each foot of the robot is actually a section of a circle. To allow for smooth rolling, at the start of each time step the contact point for the foot is computed as the point on the foot that is closest to the ground. Since the foot is a circle, this calculation is done from the analytic definition of a circle, rather than selecting a vertex of the convex polygon. If the foot is orientated such that the circular sole of the foot is not on the ground, then the contact solver will return the heel, toe, or back of the foot as appropriate.

Since the system is a kinematic chain (or loop), the contact forces for all joints must be solved simultaneously. This is done by way of a matrix inverse [1], since the system is small (only ten unknowns). The contact configuration is passed to the dynamics algorithm, which then computes the joint forces based on these contacts.

At each time step the contact solver runs the dynamics algorithm with the current contact configuration. It then selects one of the contacts and checks if the contact is feasibly at the end of the time step:

```
switch (contactMode){
  case FREE:
    if (contactHeight < groundHeight)){
      contactMode = ContactMode.PINNED;
    }
  case PINNED:
    if (contactImpulse.y < 0) {
      contactMode = ContactMode.FREE;
    }
}
```

If the contact was not feasible, then the contact mode for this contact is switched, and the dynamics solver re-runs the time step. At this point, the solution is

[1]The numerical solution for the contact impulses ($x = A\backslash b$) was computed using the *jblas - Linear Algebra for Java* package.
`http://mikiobraun.github.io/jblas/`

accepted without checking feasibility. At the next time step it will check the other foot.

Interestingly, if you try to modify the contact mode of both feet in a single time step, then the dynamics solver goes unstable whenever a contact is changed. This happens even if you make only one change at a time and iterate within a given time step. One possible explanation for this behavior is that the position-based constraints take two time steps to fix the relative velocity at the joint when a contact mode is changed. By alternating contact solves between feet, it gives each contact two time steps to stabilize. A simple way to test this would be to examine the simple case of a rod dropped onto the ground with both ends striking simultaneously. I ran out of time to do this experiment.

The feet in this model are very light (0.02 kg) when compared to the legs (4.96kg). When the feet are on the ground, it is necessary to use large torques to move the robot around. Once the foot is in the air, the same large torques will cause angular accelerations that are too large for the integration method. This generally doesn't happen, but occasionally the controller will give some weird motor torque and I wanted to prevent the simulation from exploding. To solve this problem, I saturate the motor torques in the dynamics algorithm if they are likely to cause dynamics that are faster than the current integration method's time step will allow.

6 Controller - Ranger Model

The high level controller is a simple finite state machine (FSM) that has three modes: {WALK, STAND, GUI}. The switching between these modes is controller by the commands from the user, either from the graphical user interface (GUI) or the keyboard. Each of these modes sends a different set of commands to each joint controller, as detailed below.

There are three motors in the Ranger Model: one at the hip and one at each ankle. Each of these motors has a set of simple PD controllers, which are each used to perform some action.

6.1 Ranger Controller: GUI mode

The GUI mode of the controller allows the user to manually set the target joint angles for each joint, as well as the gains for a simple PD joint controller. This is primarily used for debugging.

```
hipMotor.setProgram("GUI");
ankleOneMotor.setProgram("GUI");
ankleTwoMotor.setProgram("GUI");
```

6.2 Ranger Controller: STAND mode

The STAND mode of the controller is the default control mode. It just uses a simple PD control on each ankle to keep the absolute orientation of the foot as level as possible.

```
hipMotor.setProgram("OFF");
ankleOneMotor.setProgram("HOLD");
ankleTwoMotor.setProgram("HOLD");
```

6.3 Ranger Controller: WALK mode

The WALK mode of the controller is used to make the robot walk. The robot is right-left symmetric, so it needs to mirror the controller each step. Rather than use a FSM to do this switching, a program runs on every time step to pick which leg should be the stance leg and which should be the swing leg. There is a contact estimator inside of this program that uses a second-order Butterworth filter to prevent the contacts from rapidly switching when the foot bounces on the ground. This contact estimator is also used to determine which part of the gait the robot is in (Single-Stance or Double-Stance) [2].

```
switch(contactMode) {

case SINGLE:
  if (swingFootAng > criticalFootAng){
    swingAnkMotor.setProgram("PUSH");
  } else {
    swingAnkMotor.setProgram("FLIP");
  }
  stanceAnkMotor.setProgram("HOLD");
  hipMotor.setProgram("SWING");

case DOUBLE:
  if (swingLegAng > criticalLegAng){
    swingAnkMotor.setProgram("FLIP");
  } else {
    swingAnkMotor.setProgram("HOLD");
  }
  stanceAnkMotor.setProgram("HOLD");
  hipMotor.setProgram("OFF");
```

[2]The robot should not reach the flight phase of motion, but if it does then it just uses a PD controller to hold the current joint angles until contact is made with the ground.

}

6.4 Motor Model:

Several years ago, a motor model was created for the motors on Ranger. In this simulation, I use this motor model for calculating torque (for dynamics) and power (for cost of transport). The following parameters were experimentally determined and published in the appendix to the main design paper on the Cornell Ranger [2].

- I_{max} - maximum allowed current
- G - transmission gear ratio
- K - Motor constant
- C_d - viscous friction coefficient
- C - constant friction term
- C_τ - load dependent friction
- R - motor resistance
- V_0 - internal voltage drop
- I_0 - internal current drop

For a given joint angular rate (ω) and applied current(I), the joint torque (τ) and power (P) can be computed as follows:

$$d = \boldsymbol{sign}(w) \tag{13}$$
$$\tau = GKI(1 + dC_\tau) + \omega G C_d + dC \tag{14}$$
$$V = I\left(R + \frac{V_0}{\sqrt{I^2 + I_0^2}} + GK\omega\right) \tag{15}$$
$$P = IV \tag{16}$$

For walking robots it is useful to have a non-dimensional number for measuring efficiency. One such quantity is the *cost of transport* (CoT), the instantaneous cost of transport is given in equation 17. This value is typically averaged over many steps to compare different robots or gaits.

$$CoT = \frac{MotorPower}{Weight \cdot Speed} \tag{17}$$

6.5 Motor Controllers:

Each motor controller has a simple PD controller on the command current, which has the following form:

$$I_{cmd} = k_p * (\theta_0 - \theta) + k_d * (\dot{\theta}_0 - \dot{\theta}) \tag{18}$$

This simple PD controller was used inside of all motor programs on the hip motor, and for all ankle motor programs when the foot was on the ground. When the foot was in the air, a special controller was used, since the foot mass and inertia are negligible when compared to linkage inertia when it is on the ground. A model-based controller was used instead, which made the joint act as if it was a virtual spring-mass-damper system, with a known damping ratio and natural frequency. Note the non-linear term that is used to cancel the gravity torque acting on the foot.

$$\begin{aligned} k_p &= \omega_0^2 I_{foot} \\ k_d &= 2\xi\omega_0^2 I_{foot} \\ \tau_{gravity} &= (\vec{r}_{com} - \vec{r}_{joint}) \times (\vec{g} m_{foot}) \\ \tau_{cmd} &= k_p * (\theta_0 - \theta) + k_d * (\dot{\theta}_0 - \dot{\theta}) + \tau_{gravity} \\ I_{cmd} &= \tau_{cmd} / (GK) \end{aligned}$$

6.6 Controller Results:

The walking controller is nearly stable. If you were to compute the step map for the system, you would find that there was at least one fast stable eigenvalue and at least one slow but unstable eigenvalue. This produces a behavior where the system rapidly approaches a periodic trajectory, but then slowly diverges. This behavior is observed on the controller described above. It is almost certainly possible to adjust the many parameters of this controller to produce a mathematically stable gait, although it might require numerical optimization, rather than trial and error guessing. A better solution would be to modify the controller so that it was able to adjust it's own parameters based on the state of the system. One example would be increasing the force of the push-off during double stance. Another would be to change the step length to modulate the energy lost on the next heel-strike.

7 Conclusion

For my project I created two simulations of bipedal walking: one for a simple biped model and another for the Cornell Ranger. The models use a shared simulation architecture, but have distinct dynamics and control algorithms.

The dynamics for the simple biped model were easy to implement, along with a sophisticated collision handling algorithm. On the other hand, the dynamics for Ranger were quite difficult. The difficulty was not in the dynamics themselves but rather in finding a stable scheme for detecting and handling collisions between the feet and the ground.

Both simulations were able to generate walking behavior, although the simple biped controller is much more stable than the Ranger controller. The simple biped controller is loosly based on SIMBICON, although there are several features of my own design. The Ranger controller is somewhat based on the existing controller on the real robot, with several small modifications.

I've uploaded a few animations produced by the simulators to the internet:

- **Video 1 -** (`http://youtu.be/zvAufKB83g4`) - Video of the Ranger Model walking over level ground.
- **Video 2 -** (`http://youtu.be/t1N4WNsyY8M`) - Video of the Simple Biped walking over hilly terrain.
- **Video 3 -** (`http://youtu.be/-ddQZMGZX5g`) - Video of the Simple Biped transitioning between walking and jumping gaits.

References

[1] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. *Computer Graphics Forum*, 33(1):246–270, February 2014.

[2] Jason Cortell, Bram Hendriksen, Chandana Paul, Andy Ruina, Pranav A Bhounsule, and J. G. Daniel Carson. A robot that walked 65 km on a single charge : energy-effective and reliable locomotion using trajectory optimization and stabilization from reflexes . *International Journal of Robotics Research*, pages 1–39, 2012.

[3] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. *ACM SIGGRAPH 2003 Papers on - SIGGRAPH '03*, page 871, 2003.

[4] Akbari Hamed and J. W. Grizzle. Event-based Stabilization of Periodic Orbits for Underactuated 3D Bipedal Robots with Left-Right Symmetry. pages 1–16, 2013.

[5] M. H. Raibert, H. B. Brown, and M. Chepponis. Experiments in Balance with a 3D One-Legged Hopping Machine. *The International Journal of Robotics Research*, 3(2):75–92, June 1984.

[6] Kangkang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON : Simple Biped Locomotion Control. In *SIGGRAPH*, 2007.