RC_Biped Final Report

Stephen Bagg
M&AE 490 Spring 2007
Lab members: Alex Veach, Denise Wong
Dept: Theoretical and Applied Mechanics
Professor: Andy Ruina
Funding: ELI Undergraduate Research

Abstract:

The RC_Biped project team is investigating the stability of biped robots, and finding useful algorithms to balance bipeds using foot placement balancing. The project involves unique challenges of accurately calculating the robots orientation using accelerometers and gyroscopes, writing controllers to control multiple motors simultaneously, and using the orientation data to balance the robot. Challenges in filtering and analyzing sensor data were also encountered during the course of this project. Although it is not yet completed, significant progress has been made toward the final objective.

Introduction:

The RC_Biped project began in the summer of 2006 and is part of an ongoing investigation in biped locomotion. The objective of the laboratory is to learn about human locomotion by building robots that mimic human walking. This robot is unique in that the chassis and motors are all mass produced parts. All other robots in the laboratory are completely custom designed for the specific project. In using mass produced parts, we were able to dramatically reduce the build time and design time for the project.

The objective of this project is to investigate the stability of biped robots. The initial goal is to create a robot that is able to balance using foot placement balance with small feet. Most modern biped robots use large feet in order to help maintain balance. These robots remained balanced while walking by always keeping the center of mass of the robot over one of the large platform feet. However, humans do not balance in such a way, and so we are trying to develop a robot that balances in a manner similar to humans. Foot placement balancing uses strategically placed steps to maintain a form of dynamic balance. The balance is dynamic because the robot is constantly moving. The design of the robot creates a situation in which it needs constant adjustments in order to stay balanced.

To facilitate our progress towards achieving our final objective, we made a list of a series of goals that we should complete first. These goals were designed to be interesting ways to gain knowledge and increase our abilities to the point that we would be able to tackle our final goal; a robot capable of balancing using foot placement balancing.

The motivation behind studying foot placement balancing and locomotion better understand how humans are able to walk, and how we can design machines to walk with a human gait. Our laboratory does not dream of creating robot slaves to help humans with their chores, but rather wants to apply what we have learned to improve prosthetic devices. If a prosthetic leg was able to walk using similar balance methods as a natural human leg, the user would be able to walk with a much more natural gait and would not be hindered by the artificial limb.

Materials:

The robot chassis was made of aluminum parts purchased from Lynxmotion (see appendix for parts list). The torso is made of lexan plastic (lynxmotion). The legs are made of aluminum brackets with ball bearings and servo motors at the joints. There are currently two types of servo horns connecting the motors to the chassis. One type is made of aluminum, and the other is made of a hard plastic. The plastic horns are used of joints which are not frequently under large amounts of torque, with the metal horns used for high torque joints. The plastic horns are more adjustable, and so are preferable; however they were yielding in the high torque joints and so metal horns were necessary.

The chassis creates a robot with 3-degree of freedom hips, 1-degree of freedom knees, and un-actuated ankles. There are three servo motors controlling the hips. One motor is used to swing the leg forward (x-direction), one swings the legs outward (z-direction), and one is used to twist the leg (y-direction). This is similar to a human hip, which also has 3 degrees of freedom. The knee is also similar to a human knee in that it can only swing forward or backward. The ankles are not controlled by any motors.

There are two types of feet and ankles which can be put on the robot. One type of foot is a large aluminum platform foot that was purchased from lynxmotion. This type of foot is screwed directly to the leg, and so the ankle has no degrees of freedom. The second type of foot is a custom CNC machined aluminum foot. This foot is much smaller and thinner than the platform foot, and also has a hinge joint at the ankle. Since there is a hinge joint without a motor, the foot is free to rotate around that joint. We designed the ankle in this way so that the robot would be required to use foot placement balance to keep from falling over. Whenever the robot has only one foot on the ground,

it will be falling in some direction, and so will need to move the other foot to catch itself. Then if it lifts the first foot, it will again be falling in some direction and will to take a step to catch itself again. Even if the center of mass is perfectly aligned over the ankle of the foot on the ground, it will be in a state of unstable equilibrium, and will shortly begin falling. The only statically stable position is one where the two legs are staggered and slightly spread apart. This forms a triangle with the ground, and the robot will not fall unless perturbed by some outside force. However, when the robot is in this position, it is not able to move without using foot placement balancing; so although it is stable, it is undesirable. There are also two types of servo motors on the robot. The joints which we anticipated to need the most precision and torque use an expensive Hitec HSR-5995TG motor; while the other joints use a Hitec HS-475HB motor. The HSR-5995TG motors are used to actuate the knees and the x-directions of the hip, with the y and z-directions being controlled using the HS-475HB motors.

The mother board was designed by a former student in the laboratory. Mounted on it are three axes of gyroscopes and accelerometers. The accelerometers detect angular velocities, and the accelerometers detect linear accelerations. Attached to the mother board is the daughter board. The daughter board houses the wireless module as well as the motor and power connections.

The computer code for the robot is written in the language of C. It is currently using a mixture of fixed point and ffloat arithmetic. The ffloat system was written by members of our laboratory for the Cornell Ranger project. It has many of the benefits of floating point numbers, but the algebraic and comparison functions have been written to be much faster. The Ranger team found that when using floating point the code was taking too long to run, and it became necessary for them to write their own version. The drawbacks to the ffloat system is that the numbers only carry four significant figures of resolution, and the syntax is less clear than the conventional floating point syntax. Because of the low resolution, much of the filtering is still done in fixed point. Also, all of the motor commands must be done in fixed point. Conversion functions to and from ffloat are already completed by the Ranger team, so using both systems is relatively simple. Fixed point numbers can only be integers, which means that they can not contain

any decimal places. This adds complications when performing multiply of divide operations, because any fraction between one and negative one will become zero.

With the exception of any initialization functions, the main code for this project runs on a one millisecond interrupt timer. This puts a finite limit on the amount of processing that can be performed in the main code. We have not had any complications with the time limit thus far, however the final objective will take more processing than we are currently using, and may run close to the millisecond barrier. Fixed point operations are completed much faster than ffloat calculations; so doing as many of the computations as possibly in fixed point will reduce the time it takes for the code to run.

Methods:

This section lists and describes the goals that we have created to lead up to the final objective. Since this is an ongoing project, the entire list has not been completed.

Goal 1: Construct robot (completed)

The first step of any project is often some kind of construction or design. For this project we purchased all the necessary parts from a commercial robot manufacturer, so there was no design needed. The circuit board and the code for the circuit board were already supplied from other robotics projects in the laboratory.

Goal 2: Control of a single motor (completed)

Motors have a very important roll in this project. They act as the 'muscles' of the robot, and allow us to move the joints in a controlled manner. This step was mostly programming. Although the code running the circuit board was given to us, it didn't do anything more than manage the board. This task was an introduction to writing simple code to control motors in a predetermined way.

Goal 3: Control multiple motors simultaneously (completed)

Since taking a step involves the movement of more than a single joint in a single direction, a way to control multiple motors was needed. This step also served to further develop our programming skills. The end result was a function (move_motor) which can control any number of motors going to individual destinations at varying speeds. For a more in depth description of the motor function, see discussion or appendix.

Goal 4: Accurate orientation measurements of robot (in progress)

A preliminary version of the necessary code has been developed, but we later found this to be insufficient. More than anything, this task is about integrating the sensors into the code, and filtering the raw data to obtain a useable signal. The original idea was to filter and integrate the data gathered by the gyroscopes. Theoretically this would give a valid measurement of the orientation of the circuit board; however we found this method to have a large amount of drift. This means that over time the value would drift from the correct value, and the robot would no longer have correct data regarding its orientation. This would make it impossible for the robot to be able to balance itself, and so a new method needed to be devised. The new approach uses both accelerometer and gyroscope data to find the correct orientation of the robot. Good progress has been made on this task by lab manager Jason Cortell.

Goal 5: One dimensional "inverted pendulum" balance (completed)

This task involves moving the motors in order to keep the torso of the robot stable. In order to do this, we must have accurate data regarding its orientation, and be able to move the motors accordingly. In this task, a person physically holds the legs in the air, and physically tilts the legs in order to change the orientation of the robot. The robot responds by moving its hips in order to maintain a constant orientation of its torso. Both legs move in unison during this experiment. The controller I designed to accomplish this task is a Proportional controller. That means that the correction that the robot makes is related to how far away it is away from its desired position. This method was successful in keeping the robot in what it thought was a level orientation, however because of the drift in the orientation data, the robot appeared to gradually be leaning over. When the new method for calculating the orientation is finished, the robot will be able to stay at a constant correct position.

Goal 6: One dimensional single step balance (completed)

This task is also a test of integrating the gyro data into the code and responding accordingly with the motors. In this task the robot is restricted to one dimensional motion, and the robot will take a single step in response to which way the robot is falling. We physically limit the robot so that it is only able to fall forward or backward, and the robot takes a single step with its right leg in order to put it in a state of equilibrium. After taking the single step, the robot is in the stable staggered leg position, and must be

restarted before the test is run again. Along with testing the gyro data, this goal helps to find efficient step trajectories. This goal was mainly worked on by Alex Veach.

Goal 7: Three dimensional "inverted pendulum" balance (in progress)

This is similar to goal 5, but the robot tries to keep its orientation in all three dimensions instead of a single dimension. This requires several transformations rotational transformations for the motors, and also requires simultaneous motion of three motors. In this task, the left leg always remains straight, while the user holds onto and rotates the right leg. I have completed the code for this goal, and preliminary tests show that the code was working, however the motors for the y and z-axis proved to be lacking the necessary torque and precision for this goal. The robot was able to keep itself upright in the x-direction, but was not able to compensate errors in the other directions. The less expensive motors tended to jitter when placed under large loads. To accomplish this goal I believe we need to replace the HS-478HB motors with higher quality HSR-5995TG servos. However, this may not be necessary to meet the final objectives, since the motors would be under a much smaller load during the foot placement balancing task.

Goal 8: Two dimensional single step balance (future task)

This involves taking two dimensional steps instead of steps in a single dimension. Where as before the steps were limited to forward or backward; these steps are in a two dimensional plane (the ground). When completing goal 6, we broke up the fall directions into two cases. In this task the two cases must be expanded to four: back-left, back-right, front-left, front-right. The robot will then take a single step in the direction of falling to try and place the robot in a stable position. This position is one of static balance, meaning that the robot is stable, but is not moving. The purpose of this task is to develop a way of calculating the falling direction, and building the program necessary to complete the final objective. Both this task and the final task require the smaller, less stable feet. The robot is able to stand erect on the platform feet without ever becoming unbalanced, so the robot would never reach a state in which it needed to take a step.

Goal 9: Foot placement balancing (future task)

This is the final objective for this project. This task expands on goal 8 in that now in addition to the direction of falling being calculated, an appropriate step length must also be calculated. Then after that step has taken place, the first leg must take another step. In

goal 8 the robot only takes a single step and then stops, but the foot placement balancing task requires constant motion to keep the robot in a form of dynamic balance rather than a static balance. When this task is complete, the robot should constantly be stepping.

Side Project: Trajectory demo (completed)

Mid way during the semester, it came to our attention that a group of elementary school students would be touring our lab as part of the Expand Your Horizons program. As a laboratory we decided that each project team should come up with a fun interactive demo to stimulate the children's interest in robotics. For the RC_Biped we came up with a trajectory demo which demonstrated how many bipeds are controlled. The students would enter a series of positions into four arrays, and the robot would sequentially move through the positions. Each array represented the positions of a single motor, and the program would wait until all the motors had finished moving before moving to the next set of positions. I gave the students control over the x-direction and knee motors for each leg. Since then I have expanded it to also include the z-direction motors. Besides being a fun way to control the robot, we were able to get the robot to walk forward using this demo. However, we were only able to get it to walk with the large platform feet. This type of trajectory control is similar to how most commercial bipeds are controlled, and so no new knowledge has been gained from this.

Results and Discussion:

This project is an ongoing design project, and not a typical science experiment. Our results are measured by our accomplishments and the progress we have made toward our objectives. This project had two main objectives; the first was creating a robot which uses foot placement to dynamically balance itself, and the second was to see if meaningful information can be gathered from a low cost, commercially produced robot. While we have not reached our first objective, I believe we have shown that mass produced robots can be useful in situations where the design team is small. Because of the time it takes to design and manufacture a custom robot, a commercially produced robot was the only feasible option for this project. Project teams which have built their own chassis usually consist of many members and are extended for longer periods of time. Our accomplishments thus far have paved the way for the final objective, and have shown that it is feasible without manufacturing a custom robot.

The main purpose of this project is to construct useful algorithms for balance and motor control, which can then later be used on a more advanced biped robot. Although our foot placement algorithm is not complete, we have outlined the process, and made good progress towards completion. Our motor control function is complete, and I am confident will be more than adequate for the balance requirements.

Motor Controller:

The motor controller was theorized and coded entirely by myself. The motor control function has three inputs. Two of the inputs are entered directly when the function is called. These are the motor number, and the desired position of the motor respectively. The motor number is simply the name of the motor, and the position is entered as a degree quantity measure off of its neutral position. The neutral position for the motor has been calibrated so that when all motors are neutral, the robot is standing erect. A positive angle refers to a clockwise rotation of the motor, and a negative angle moves the motor counter clockwise. The third input is the steady state speed of the motor. This is an integer value between one and six, and corresponds to the number of positions that the motor will move per millisecond. A position is roughly one sixteenth of a degree. This number is entered into an array labeled 'speed', and value should be entered in the position that corresponds to that motor.

Sample command:

(in this example m_rhipx is a macro for the x motor on the right leg, and corresponds to a value of 1)

```
speed[m_rhipx] = 3;
move_motor(m_rhipx,-60);
```

This set of commands moves the motor to a position of -60 degrees from its neutral position, and at a steady state speed of three positions per millisecond.

The motor controller breaks the motion of a motor into four distinct states. The first state, called the '0' state, occurs when the motor is already at the desired position. No motion occurs in the 0 state. The first state of motion is the '1' state. This is happens when the motor is not at the desired position, and is not yet moving at the steady state speed. The '1' state is a state of constant acceleration, and the motor stays in this state until it is rotating at the steady state speed. When it is traveling at the steady state speed,

it moves into the '2' state. This state continues the motor with a constant speed until it nears the desired position. When it nears the steady state position, the controller moves to a '3' state, and begins a state of constant deceleration. The position at which it switches to this state is a function of the steady state speed, and is specified in an array 'start_deccel', and is found at the beginning of the motor controller code. After the motor has reached its desired position, it switches back to the 0 state, and no more motion takes place.

Balance Algorithm:

The planning for this task was worked primarily by Alex Veach and me. The highest level of the program consists of two states, a 'falling' state, and a 'waiting' state. The waiting state occurs immediately after the robot has completed a step, and before it begins to take another step. The falling state occurs when the robot has registered that it needs to take a step, and is in the process of doing so. The falling state begins by reading sensor data regarding its orientation and angular velocities. From this data it calculates the direction and length of an appropriate step. The controller I have designed for calculating the step length is a proportional derivative controller. This means that the step length is a function of the current angle of the robot, and how fast it is falling. By knowing the angles of the robot measured from vertical, the robot can calculate which way it is falling, and which foot it should step with. There are also sensors located on the feet themselves which tell the robot if the foot is on the ground. By knowing the current positions of the feet, and where they need to go, it can take the appropriate action. After the step is completed, and robot goes back to the waiting state and begins the process over again. This code has not been completed or tested, and so is subject to change. At the moment it is our thought process regarding what we are confident will work.

Inverted Pendulum Balance, 1 Dimension:

For the single dimension inverted pendulum balance, I chose to use the x-direction. This corresponds to the forward and backwards direction for the robot. The controller I created for this is a form of Proportional controller. The program begins by taking in the sensor data and finding the current position of the motor. It then compares this value to the desired position to come up with an error value. For this task, the desired value is vertical. After the error term is calculated, it is multiplied by a gain

value. This is necessary to keep the controller stable. The values measured from the gyroscopes have different units than the positions of a motor, and the gain value compensates for this difference. It also keeps the system stable. If the gain were too high, the motors would over compensate for the motions, and drive the system unstable. The gain value currently being used is 0.0125, and works well for the single dimensional case. This new term corresponds to the number of positions that the motors will move to make up to compensate for the system not being vertical. The motor is then told to move to this new position, and this is saved to be updated when the function runs again in another millisecond. Mathematically, the calculations are done as follows

$$Ex = (\theta_D - \theta)$$
$$Ux = K_{px} * Ex$$
$$hipx = hiplast + Ux$$

Ex is the error term, Ux is the error term multiplied by the gain, hipx is the new position of the motor, and hiplast is the previous position of the motor. This algorithm was successful in keeping the robot at the desired position, however because of drifting in the perceived desired position; the robot appeared to be slowly tipping over. This will be fixed once a better system is developed to determine the robot's orientation.

Inverted Pendulum Balance, 3 Dimension:

This task is more complex because the robot must now stay upright in all three dimensions. Along with that, the x and y motors are mounted beneath the z motor on the leg, so if the z motor is not at its neutral position, then movements by the x and y motors will not correspond with rotations about the x and y axes relative to the circuit board and sensors. The coordinate transfer matrix looks like the following:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} mx \\ my \\ mz \end{bmatrix}$$

For the above equation, θ is measured as the negative of the z motor angle. X, Y, and Z are the axes mounted on the circuit board; and mx, my, and mz are the axis in which the motors act. From the above transformation, we find that the axes in which the motors act are:

$$x = \cos(\theta)mx + \sin(\theta)my$$
$$y = -\sin(\theta)mx + \cos(\theta)my$$
$$z = -mz$$

My design for a controller was similar my design for the single dimension balance. I calculated error terms in the same manner, however when assigning values to the motors, I had to account for the transformation change. The motor assignments looked like the following:

$$hipx = hiplast + Ux * \cos(\theta) - Uy * \sin(\theta)$$
$$hipy = hipylast + Ux * \sin(\theta) + Uy * \cos(\theta)$$
$$hipz = hipzlast - Uz$$

The signs of these quantities will change depending on which leg is moving. The motors of the two legs are mounted in such a way that the motors will need to twist in opposite directions.

***Conclusion*:**

Starting from nothing more than parts and a circuit board code, we have created a robot which is capable of complex tasks and computations. The robot is able to take in information about its current state and act to perform desired tasks. Using the pendulum code it is able to successfully keep a constant orientation while being twisted by a user. With the trajectory program the robot is able to perform complex motions using any combination of motors. Although the robot is not yet able to balance using foot placement balancing, most of the necessary work leading up that step are complete.

We have also shown that when time constraints make machining a custom robot chassis an unrealistic option, mass produced robot kits are a viable alternative. Although they may not perform quite to the level of a fully custom design, algorithms can still be tested on mass produced robots, and meaningful knowledge can still be obtained.

Future members of this project team should begin by completing my work on the three dimensional pendulum balance. As was previously mentioned, this may involve purchasing more of the high precision motors. If this is not feasible, I would recommend starting with the two dimensional, single step balance. Much of the work has already been completed for both of these places, and completing them will be a good introduction to the robot code, and will help them familiarize them with the robot's operation and limitations.

Appendix:

Lynxmotion Parts:

-C brackets (used for leg between hip and knee)
http://www.lynxmotion.com/Product.aspx?productID=287&CategoryID=87
part number: ASB-03

-C brackets with ball bearings (used for knee)
http://www.lynxmotion.com/Product.aspx?productID=400&CategoryID=87
part number: ASB-09

-Servo Brackets (used to hold servo motors)
http://www.lynxmotion.com/Product.aspx?productID=286&CategoryID=87
part number: ASB-04

- Lexan Biped Torso
http://www.lynxmotion.com/Product.aspx?productID=437&CategoryID=88
part number: BT-01

-Aluminum Feet
http://www.lynxmotion.com/Product.aspx?productID=430&CategoryID=87
part number: ARF-01

-Metal Servo Horn
http://www.lynxmotion.com/Product.aspx?productID=505&CategoryID=75
part number: HMSH-02

-Hitec HS-475HB servo
http://www.lynxmotion.com/Product.aspx?productID=77&CategoryID=38
part number: S475HB

-Hitec HRS-5995TG
http://www2.towerhobbies.com/cgi-bin/wti0001p?&I=LXHZS1&P=ML
(no longer offered at lynxmotion)
part number: LM2865

Motor Controller:

```
void move_motor(short int mnum, short ang)
{short int new_pos,f_pos,dif,absdif,pos;
short int start_deccel[6]={1,4,12,20,26,32};

 /*vals has the information for the motor in it
 mnum - motor number (rk 1, )
 //start_deccel - dif at which to begin decelerating
 pos - motor position
 speed[8] - motor ss speed
 lmov[8] - last motor movement
 mstate[8] - phase indicator (0 is beginning, 1 accel, 2 const vel, 3 decel)
 way to give a motor command
 speed[m_rknee]=1;
 move_motor(m_rknee,45);
 */
        pos= Parameter[29+mnum];
        f_pos = angle2pos(ang);
        dif = f_pos - pos;
        absdif = ABS(dif);

//original values were [1 3 6 10 15 21]
        if (dif == 0)
        {mstate[mnum]=0;}

        switch(mstate[mnum])
        {
        case 0:
        //calculate number of steps
        //start motion new_pos=pos (+-) 1;
            if (dif <0)
            {new_pos = pos - 1;
            lmov[mnum]=1;
            mstate[mnum]=1;}
            else
            {
             if (dif > 0)
                  {new_pos = pos + 1;
                  lmov[mnum]=1;
                  mstate[mnum]=1;}
                  else
                  {new_pos = pos;
                  mstate[mnum]=0;}
            }
        break;
```

```
        case 1:
        //accel
        lmov[mnum]=lmov[mnum]+1;
        if (lmov[mnum]<speed[mnum])
        {
                if(dif<0)
                {new_pos = pos - lmov[mnum];}
                else
                {new_pos = pos + lmov[mnum];}
        }
        else
        {mstate[mnum]=2;
         new_pos=pos;}

        break;

        case 2:
        //const vel: new pos = pos+speed;
        lmov[mnum]=speed[mnum];
        if (absdif <= start_deccel[speed[mnum-1]])
                {mstate[mnum]=3;}

        if(dif<0)
                {new_pos = pos - speed[mnum];}
        else
                {new_pos = pos + speed[mnum];}


        break;

        case 3:
        //deccel
        lmov[mnum]=lmov[mnum]-1;
        if (lmov[mnum]==0)
        lmov[mnum]=1;
        if(dif<0)
                {new_pos = pos - lmov[mnum];}
        else
                {new_pos = pos + lmov[mnum];}
        if (absdif<2)
        mstate[mnum]=0;
        break;
        }
        Parameter[29+mnum]=new_pos;
setPWMval();
}
```

Pendulum Balance:

```
void err_biped(void)
{ffloat d2rad,cz,sz,Epx,Epz,Epy;
  THETAx=S16int2FFloat(Variable[3]);
  THETAy=S16int2FFloat(Variable[4]);
  THETAz=S16int2FFloat(Variable[5]);
  d2rad=IEEE2FFloat(0.01745);
  cz=FFcos(FFmult(d2rad,pos2angle(RHipz))); //cos(thetaz)
  sz=FFsin(FFmult(d2rad,pos2angle(RHipz)));  //sin(thetaz)
  Epx=FFsub(THETAdx,THETAx);
  Epy=FFsub(THETAdy,THETAy);
  Epz=FFsub(THETAdz,THETAz);
  Uxx=FFmult(FFmult(Kpx,Epx),cz);
  Uxy=FFmult(FFmult(Kpx,Epx),sz);
  Uz=FFmult(Kpz,Epz);
  Uyx=FFmult(FFmult(Kpy,Epy),sz);
  Uyy=FFmult(FFmult(Kpy,Epy),cz);

}
```