Balancing Robot

Daniel Bauen

Brent Zeigler

December 3, 2004

Initial Plan

The objective of this project was to design and fabricate a robot capable of sustaining a vertical orientation by balancing on only two wheels. The balancing robot is a highly unstable two wheeled robot. The largest mass, the battery pack, is positioned high above the axle, making the robot an inverted pendulum with a low natural frequency. The robot will naturally tend to tip over, and, the further it tips, the stronger the force causing it to tip over.

The original plan was to use a gyroscope and an accelerometer to measure the rate of angular rotation and the relative source of gravity; however, these devices are too expensive for the project. Therefore an inexpensive potentiometer was used to measure the angle of the robot relative to the floor. In addition, a microcontroller was used to process the data from the sensors, and control the motors accordingly to allow the robot to balance.

Ideally, an Atmel AVR pic microprocessor was to be used to control the robot. However, because it must be programmed in C (and since C was never taught to mechanical engineers), it was unusable in the allotted time period. Instead, a Basic Stamp 2SX was used to control the robot. Even though the basic stamp's processing capabilities are far inferior to the Atmel AVR, it could be programmed in basic. This allowed for adequate processing speed and saved vast amounts of time by not having to learn a new programming language. In addition, the motor control algorithm would implement PID based closed loop feedback control. Data acquisition from the potentiometer would sent back to the basic stamp and used to determine motor input.

Balancing Robot Photo and Components:

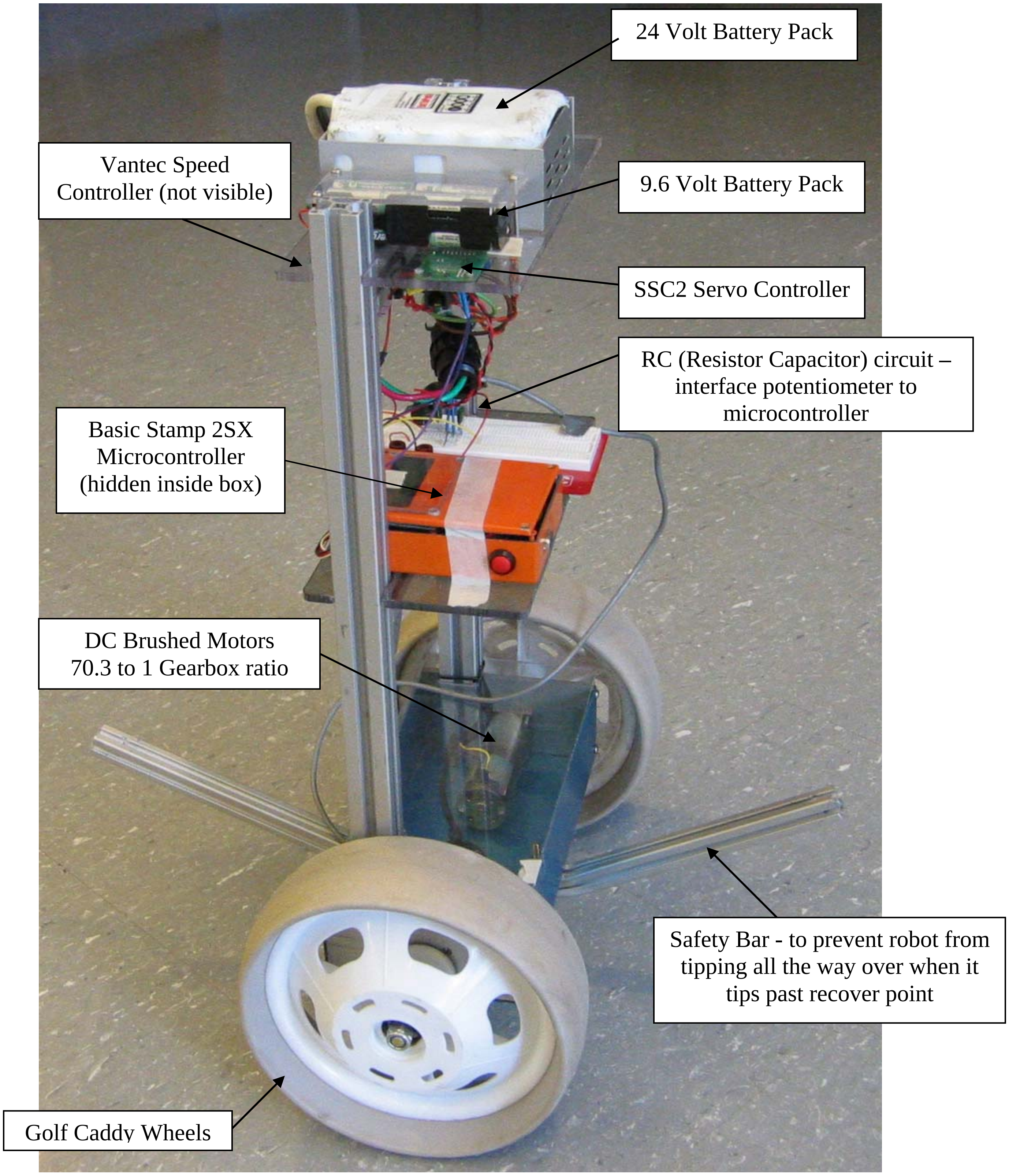
24 Volt Battery Pack
Vantec Speed Controller (not visible)
9.6 Volt Battery Pack
SSC2 Servo Controller
RC (Resistor Capacitor) circuit – interface potentiometer to microcontroller
Basic Stamp 2SX Microcontroller (hidden inside box)
DC Brushed Motors 70.3 to 1 Gearbox ratio
Safety Bar - to prevent robot from tipping all the way over when it tips past recover point
Golf Caddy Wheels

Figure 1: Diagramed Photo, Picture taken while robot was actively balancing.

Equipment:

Body: The structure of the balancing robot was created from disassembled bomb disposal robots, and other scrap parts available from Robojackets robotics club.

Powertrain: A single drive motor powered each wheel of the robot. These 24 volt brushed motors have a 70.3 to 1 planetary gear box to reduce the output shaft speed. The wheels are coupled directly to the output shafts of the gear box. The speed controller used to control the motors is a heavy duty remote control speed controller, Vantec , popular for use in personal robotics applications. The speed controller requires a standard RC signal (1.5 to 2 ms pulses).

Microcontroller: A Parallax brand Basic Stamp 2SX was used to control the robot. This is the same microcontroller used in the controller boxes for ME2110. The Basic Stamp is a very simple microcontroller, and is programmed using Basic. This results in it being simple to program, but a less powerful microcontroller due to it having to interpret Basic on the fly.

Speed Controller Communication: The Basic Stamp microcontroller sends the motor speed commands in a serial format to the SSC2. The SSC2 then converts these signals into standard remote control signals (1.5 to 2.0 ms pulses) that can be interpreted by the Vantec speed controller.

Power: Two battery packs are used to power the robot. One large 24 volt pack is connected to the speed controller, and in turn powers the motors. The other pack is 9.6 volts, and is used to power the Basic Stamp microcontroller and the SSC2. This is done because the Basic Stamp and SSC2 cannot accept voltages larger than 10 volts.

Position Sensor: The angular position of the robot relative to gravity is measured by a potentiometer with a feeler attached to its dial that senses the position relative to the floor. Therefore the robot must currently be on a level surface to balance properly. The robot has been successfully run on surfaces as rough as sidewalks.

System Diagram:

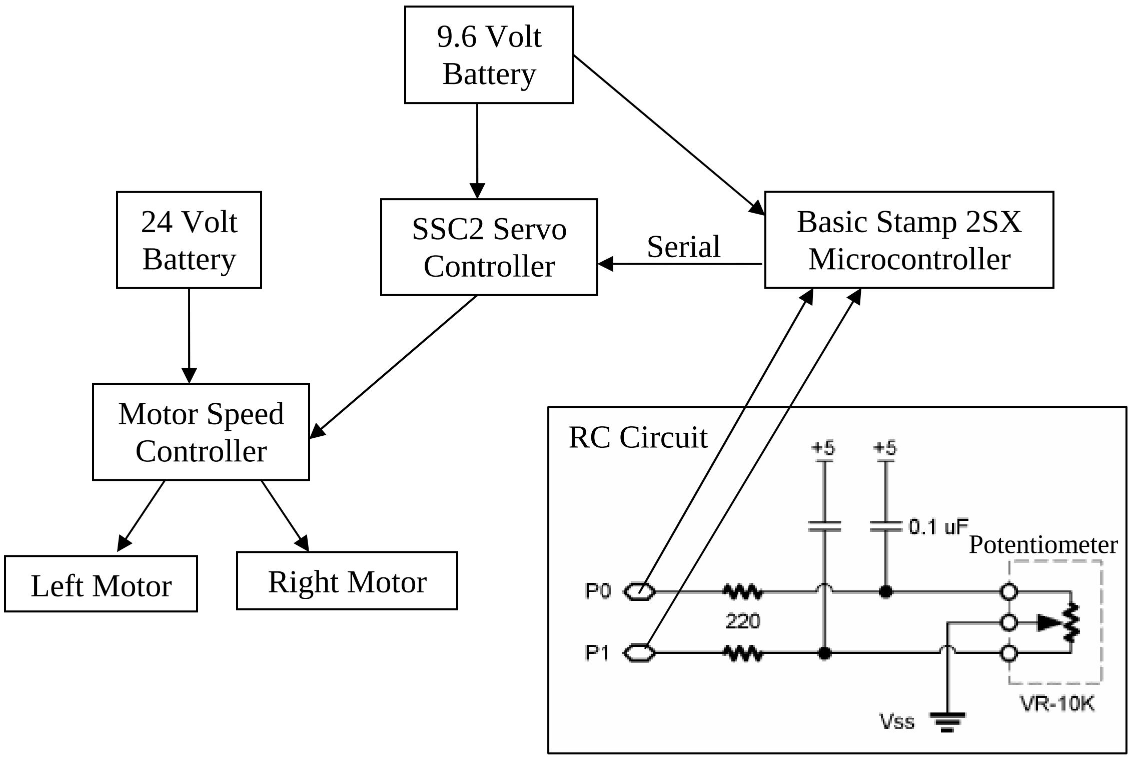

Figure 2. System Diagram

Kinematics:

The balancing robot relied on signal feedback from the potentiometer. For our case, a value of 0 indicated that the robot was completely vertical. From here, three different values had to be determined, Proportional, integral and derivative count values.

First, proportional was examined. This is the value of the current potentiometer output, the error term θ in Figure 3, was multiplied by the P-Gain, converted into motor counts and stored. This gave a motor speed proportional to the error term.

Next, integral term was determined via the summation of error term (ie, the summation of all previous potentiometer readings) and multiplied by the I-Gain. The integral term lets the robot know that it has been at an incorrect angle for too long a period of time, and must correct itself before reaching maximum motor velocity. This was converted into motor counts and stored.

The derivative gain was calculated by taking the difference of the current potentiometer reading from the previous potentiometer reading,

$$(\text{Current error term}) - (\text{Previous error term}) = d\theta \tag{1}$$

This is the angular rate, the rate at which the robot is falling over or recovering. Again, this was multiplied by D-Gain, converted to motor counts, and stored.

Finally, all gain values that had been converted into motor counts were summed, and outputted to the motors. This process ran in a continual loop, read potentiometer value, calculate PID, output motor speed, repeat.

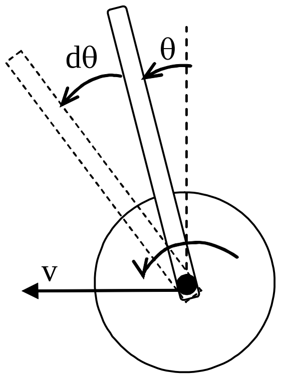

Figure 3. Simple System Dynamics

The proper motor gains and system dynamics can be determined through theory. However for the length of this project, it was satisfactory to use trail and error to find the correct gains.

The first iteration of the balancing code only used the proportional term, and the robot was unable to balance correctly. Either the gain had to be set too high such that the robot oscillated wildly, or the gain was too low such that the robot could not recover. There was no point at which the proportional controller properly balanced the robot.

The second iteration of the balancing code used all three terms, PID. With the proper gains, the robot could balance for a maximum of 30 seconds. It appeared to have problems in that it would overshoot its original place on the floor that it was trying to maintain. The overshoot was unstable, and the robot would eventually fall over.

The third iteration of the balancing code used only the P and I terms, as determined appropriated by Dr. Sadegh. The wheels were also replaced with larger wheels so that the robot had a faster top speed. The D term was determined to be inherent in the inverse pendulum dynamics for balancing, and therefore unnecessary. However when the robot is moving at a certain angle, then the D term is desirable. This was noticeable with the PI control only when the robot was pushed hard in an attempt to imbalance it. It did not sense the large angular rate, and therefore would not move the motors quickly enough to recover. The PI controller worked very well for balancing the robot in place. It was able to balance indefinitely, or until too much random noise was generated by the potentiometer and RC circuit.

Challenges/ Solutions:

There were several challenges involved in this project. The first is that the Robix kits designated for the projects were not used. Instead a complete robot was built from scratch. The circuits had to be properly connected together, and some custom circuitry work was required. Also, there was no budget available for the robot, so it was constructed from material that was already available.

Although programming a basic stamp is not complicated, programming it for a complex task was very challenging. For starters, the logical test of -1 < 0 would return false. This is because logic in basic is done in binary and doesn't account for the fact that the first bit represents positive or negative. In addition, basic cannot accurately divide by negative numbers for the same reason it can't test to see if a number is less than zero. Also, basic doesn't allow for floating point numbers, so all values were rounded into integer numbers. So, to combat these problems, much of the programming for the robot had to be done in binary. However, once these challenges were overcome, finding good P,I, and D gains was not a problem.

The Basic Stamp 2SX does not have an analog to digital converter. Therefore, the potentiometer cannot simply be connected to the microcontroller. To read the values from the potentiometer, an RC circuit has to be created. The RC circuit consists of the potentiometer as the resistor, and a capacitor. The basic stamp has a function that sends a value of high to the capacitor, charging it, and starts a timer. The capacitor then discharges itself through the resistor and when the pin on the microcontroller goes low, the microcontroller stops the timer and measures how much time has passed. The time value is dependent on the resistance of the potentiometer, and therefore the position of the feeler attached to the potentiometer dial.

Due to the RC circuit being used, there was a lot of random noise, and the precision of the sensor was very low. Using the debug feature available, the values interpreted by the basic stamp from the potentiometer were visible. These varied enough that the robot could be leaned several degrees to either side before there was a noticeable change in the angular position. This is visible when the robot is actively balancing. It tends to balance itself, and then will lean a little until it realizes that it is off balance before trying to correct itself to balance again. This is why it is constantly in motion to stay balanced.

The first revision of the robot had 6 inch diameter wheels. The motors have a lot of torque, but only rotate at approximately 100 rpm. The robot had trouble reacting quickly enough due to the slow motor speed. Since there were no other readily available motors, a pair of larger wheels, 12 inch diameter, were mounted. These made a significant difference. The motor could more easily catch up when the robot was falling, and they didn’t have to turn as quickly when balanced.

Achievements/Results:

After extensive time invested into this project, stability from the two wheel vehicle was achieved. The robot (if properly calibrated) can stand on its own indefinitely with no external input. Figure 1 shows the robot balancing by itself. However, sometimes random noise is generated which causes the robot to become unstable and topple. This is primarily due to the low resolution of the sensor, which could not be upgraded due to cost.

Learning Experiences:

This project provided several learning experiences. First, it turned out to be fairly easy to maintain good stability once everything was tuned properly. This leads us to believe that with higher resolution sensors, the proper processor, and with the right motors, PID technology can be adapted to various technologies with very high success. Any device that uses purely mechanical mass-spring-damper technology (such as a suspension system on a car) should be tested to see if it can be replaced with a similar electronic control. Although the cost associated with such an upgrade may be more expensive (and still rely on mechanical safeties), its performance would be far superior.

References:

Sadegh, Nader. “Final Computer Assignment” ME3015A. 2003

Grasser, Felix. “Joe: A mobile, Inverted Pendulum”. Laboratory of Industrial Electronics, Swiss Federal Institute of Technology Lausanne. 2003

Bickle, Rick. “DC Motor Control Systems for Robot Applications”. http://www.dprg.org/tutorials/2003-10a/motorcontrol.pdf . 2003

Appendix A: Program Code

```
' {$STAMP BS2sx}
' -----------------------------------------------------------------------------
' Balancing Robot Program
' -----------------------------------------------------------------------------
' Baud rate Basic Stamp 2SX
'Baud CON $40F0

' I/O Definitions
' -----------------------------------------------------------------------------
PotCW CON 8  ' clockwise pot input
PotCCW CON 9 ' counter-clockwise pot input
' -----------------------------------------------------------------------------

' Constants
' -----------------------------------------------------------------------------
'Scale RCTIME to 0 - 250
Scale CON $002C ' BS2sx
' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------
'Setting up initial variables

rcLf VAR Word ' rc reading - left
'diff VAR Word ' difference between readings
lastPos VAR Word 'Previous servo position
zeroPos VAR Word 'zero position
lsPos VAR Word 'Left motor output
rsPos VAR Word 'right motor output
potPos VAR Word
iState VAR Word
pTerm VAR Word
iTerm VAR Word
dTerm VAR Word
errorTerm VAR Word
mGain VAR Word
'change VAR Word

' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------
'setting values for integral state, last position, and change in position (PID)
iState=0
lastPos=0
'change = 0
```

```
'Initialize Potentiometer to Set Zero Value, this sets the Balance Point
HIGH PotCW ' discharge caps
HIGH PotCCW
PAUSE 1
RCTIME PotCW, 1, rcRt ' read clockwise
RCTIME PotCCW, 1, rcLf' read counter-clockwise
rcRT=1350-rcRT
zeroPos=((rcRT+rcLF)/2)*20  'Potentiometer value 12700 - 14540  Center - 13620


'Main Program Loop, where motor control calculations are performed
Main:

HIGH PotCW         ' discharge caps
HIGH PotCCW      ' discharge caps
PAUSE 1
RCTIME PotCW, 1, rcRt           ' read clockwise
RCTIME PotCCW, 1, rcLf         ' read counter-clockwise
rcRT=1350-rcRT
potPos=((rcRT+rcLF)/2)*20        ' Potentiometer value 12700 - 14540  Center - 13620

'goes to get position, integral and derivative gains
GOSUB Position
GOSUB Integral
GOSUB Derivative

'incrementing integral gain
'sPos = (potPos) - (lastPos-potPos)+iTerm
'sPos=802 - (sPos/20)     '802 is an adjustment value to center at 120 (upright)

'setting value for motor gain
mGain = (pTerm + Iterm)

'checking to see if motor gain is negative
IF mGain.BIT15 THEN motorNeg
'if not negative, then adjust positivly
IF NOT mGain.BIT15 THEN motorAdjustPos


Motors:
mGain = mGain + 130
GOTO drivemotors
GOTO Main

motorNeg:
'inverting mgain to be positive for math calculations
```

```
mGain = -1 * mGain
'adjusting as a negative value (needed for logical operations)
GOTO motorAdjustNeg


motorAdjustNeg:
'setting mgain then reverting back to a negative number
mGain = mGain/55
'capping motor gain
IF mGain > 120 THEN capMotorNeg
mGain = -1*mGain
GOTO Motors

motorAdjustPos:
'setting mgain
mGain = mGain/55
'capping motor gain
IF mGain > 120 THEN capMotorPos
GOTO Motors

capMotorNeg:
'capping motor gain on negative side
mGain = -120
GOTO Motors

capMotorPos:
'capping motor gain on positive side
mGain = 120
GOTO Motors


"""Functions Used to Find PGain, ErrorTerm, etc""""""

Position:  'Position term loop
'setting error term
errorTerm = potPos - zeroPos        '13620 is 'centerline'
'checking if error term is negative
IF errorTerm.BIT15 THEN errorTermNeg
'checking if error term is positive
IF NOT errorTerm.BIT15 THEN errorTermPos
RETURN

errorTermNeg:
'working with a negative error term
errorTerm= -1*errorTerm
'setting pterm
```

```
pTerm = 10*errorTerm/7
pTerm = -1 * pTerm
errorterm = -1*errorTerm
RETURN

errorTermPos:
'setting pterm
pTerm= 10*errorTerm/7
RETURN


""""Functions Used to Find iGain, iState, etc""""""

Integral:  'Integral term loop
'incrementing istate (ie, adding in the error term)
iState=iState+errorTerm
'checking if istate is negative
IF iState.BIT15 THEN iStateNeg
'if not negative, run normal code
IF NOT iState.BIT15 THEN iStatePos
RETURN

iStateNeg:
'invert value for logical operation
iState = -1*iState
'adjust the iterm gain
iTerm= iState/8
're-invert
iState = -1*iState
'invert iTerm
iTerm = -1*iTerm
RETURN

iStatePos:
'normal code, works with positive values
iTerm=1*iState/8
RETURN


""""Functions Used to Find dGain, etc""""""

Derivative: 'Derivative term loop
'finding derivative value
'diff= (lastPos-potPos)
'saving last position as current position to be used in next loop
lastPos=potPos
```

```
'finding the d-term
dTerm = (errorTerm - (lastPos-potPos))
'check if negative
IF dTerm.BIT15 THEN dTermNeg
'if not negative, run normal code
IF NOT dTerm.BIT15 THEN dTermPos
RETURN

dTermNeg:
'invert for logical operation
dTerm= -1*dTerm
add in d-gain value (in this case, left as one)
dTerm = dTerm
're-invert
dTerm = -1*dTerm
RETURN

dTermPos:
'add in d-gain value (in this case, left as one)
dTerm=dTerm
RETURN

drivemotors:

motorout:

mgain = 254 - mgain

lsPos=mgain
rsPos=mgain
SEROUT 10,$40F0,[255,0,rsPos]
SEROUT 10,$40F0,[255,1,lsPos]
'PULSOUT Servo, (750 + sPos) ' move the servo
PAUSE 20

'used to debug (comment out when running to reduce processor time)
'DEBUG HOME, "Position: ", SDEC mgain, "    ", SDEC lastPos, "    ", SDEC potPos, "
", SDEC pTerm, "     ", SDEC iTerm, "   ", SDEC dTerm, "     ", SDEC iState, "        "
GOTO Main
```

Appendix B: Robot Revision Pictures

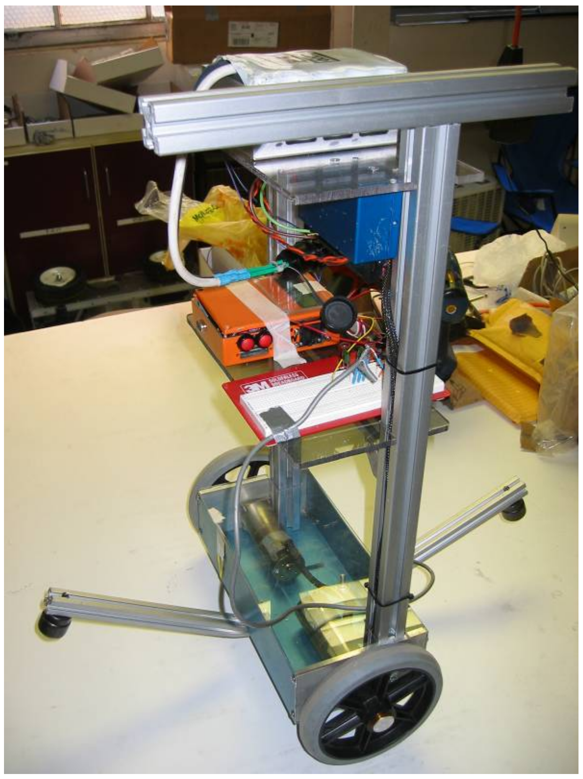
Figure 4: Revision 1

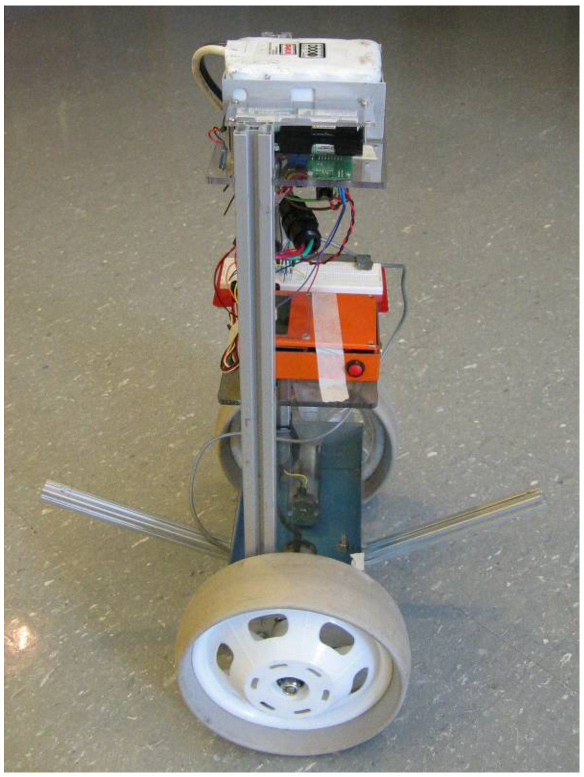
Figure 5: Revision 2