# Final Report for M&AE 491
# Senior Design Project
# under Professor A. Ruina

May 20, 2006

Justin D. Webb
912 Wilson St.
Bohemia, NY 11716
(631) 374-2644

# Abstract

# Introduction

*Figure 1: Biped Walker without any control system*

The Marathon Walking Robot project was devised as to create a stable, efficient, and robust walking biped that could traverse long distances with minimal failure. The robot would be based on passive dynamics principles, whereby a dynamic approach to modeling the robot as an assembly of pendulums of sorts [1] would be preferred over the more commonly applied kinematics based approach that uses servo motors and other actuators to constantly provide control to the system at each moment in time. Though

[1] A Three-Dimensional Passive-Dynamic Walking with Two Legs and Knees
Collins, S.H., Wisse, M. , Ruina, A.

robust and highly effective at performing complex motions, this approach is highly inefficient, raising concerns about practical applications to solving real engineering problems. The passive dynamics approach will open up new possibilities for efficient, life-like robotic motion that could benefit development of prosthetics and other research.

Work done by Haberland and Stiesberg succeeded in transforming the walker from an unreliable machine to a modestly efficient robot that was able to able to break ground and accomplish its purpose: maintaining a stable walking cycle for an extended period of time The control system was completely redesigned, implementing proportional control and a state based control structure that would continuously monitor information coming from the sensors, decide what phase the robot is in, and actuate the appropriate controls for that state. The success they achieved, however, was limited in that the robot's control algorithm was relatively simple, and could only recover from minor perturbations during the cycle.[2] A stable gait was obtained by fine tuning various system parameters, such as the amount of power with which to activate the motors, and relative positions that the proportional control algorithms would seek to place the moving limbs. This method did not account for the variability that would be presence over a long cycle, and measures were taken to make the control algorithms more adaptable.

The overall control of the robot had to be extensively modified before the algorithms could be made more "intelligent". Recommendations published by Haberland were consulted, and the results listed in this paper are the outcome of implementing several of these changes, in addition to others modifications that were performed in order to make the robot achieve a stable, long range walking cycle.

# Methods

## 3.1 Implementation of Proportional-Derivative Control

### 3.1.1 History of Marathon Walker Control Methods

As postulated by Haberland, one of the major causes of failure was the randomness associated with hip and ankle actuation. Hip and ankle motion was the result of a proportional control, whereby the motor robot controller senses the actual positions of these appendages relative to some desired position, and applies a signal to the motor proportional to this error, given in equation (1) and (2). [3]

$$e(t) = r - y(t) \qquad (1)$$

$$u(t) = K_P \cdot e(t) \qquad (2)$$

[2] Haberland

[3] Experimental Systems and Nonlineaer Dynamics 3rd Edition
Psiaki, M.L.

where e(t) is the time varying error signal(a function of r, the target position, and y(t), the time varying actual position), u(t) is the control signal sent to the motor, and $K_P$ is the proportional gain. By adjusting $K_p$, one could increase the response rate of which the motor applies to proper control signal in order to drive the actual position to the desired position. Care should be taken, however, because as this response rate increases with respect to the sampling rate of the system(i.e. the rate at which the controller can gather data from the sensors, make the calculations, and output the control signal), the ideal approximation of the control method as being continuous becomes invalid, resulting in jerky and imprecise control.

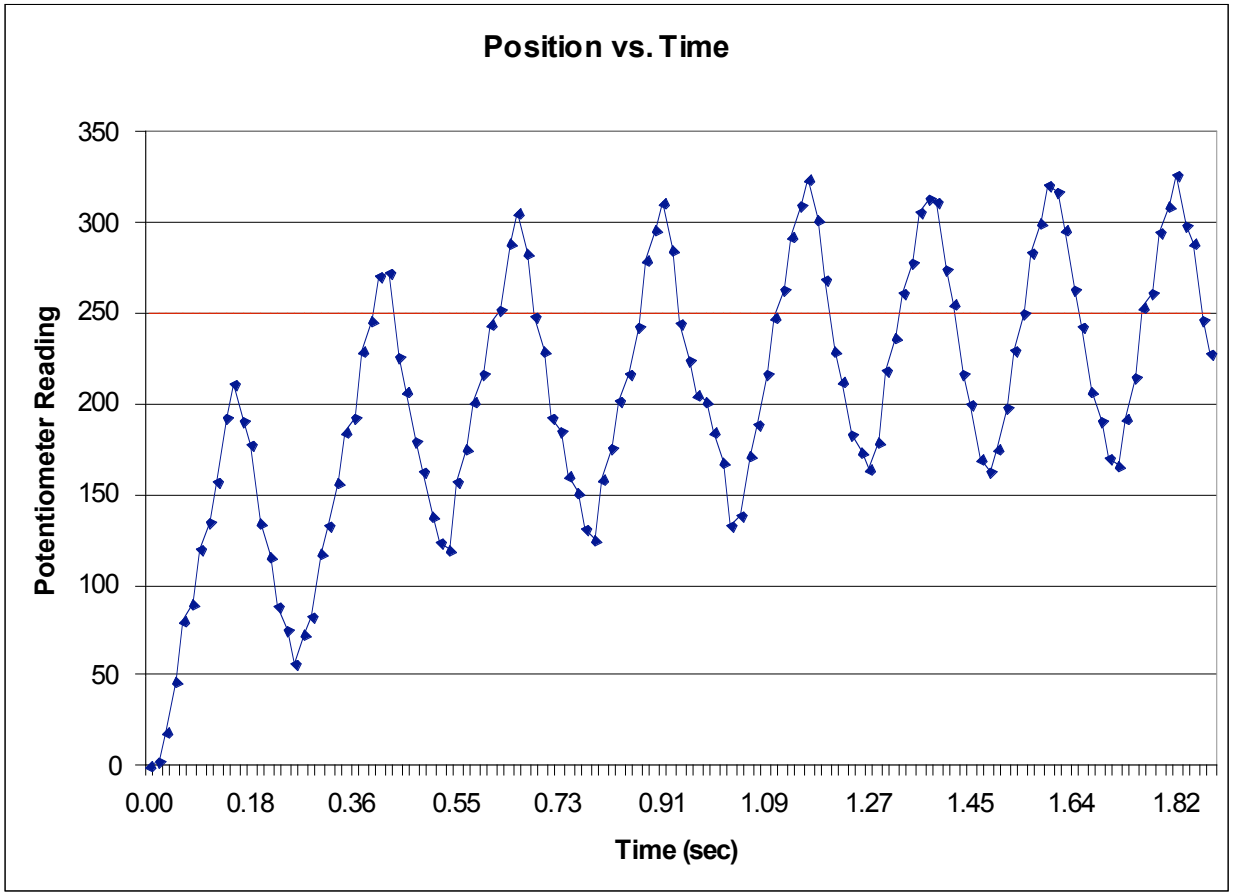



*Figure 2: Potentiometer Output Signal (position) versus Time. Red line indicates target position.*

### 3.1.2 Using Velocity Data to Damp Out Oscillations

This control algorithm, though effective in many applications, is not ideal for generating a stable gait. The problem lies in the inertial properties of the system bodies, especially the hips. In equation (2), that at the limit of the error signal, e(t) approaches zero, the control signal is decidedly zero. This presents a problem in that the motion of the body will overshoot the desired position, without any significant source of damping. The control algorithm will seek to correct this error, and apply a reverse torque to bring the

appendage back to the right position, causing overshoot again. The addition of damping is accomplished by derivative control, given by equations (3) and (4).[4]

$$e(t) = r - y(t) \qquad (3)$$

$$u(t) = K_P \cdot e(t) - K_D \cdot \dot{y}(t) \qquad (4)$$

Equation (3) is identical to equation (1), but notice the addition term in equation (4). Here, the control signal is subtracting out the velocity of the moving body, and scaling it by a derivative gain term, $K_D$. This allows the controller to "predict" when the forced body will be in the desired position, and cause it to slow down as it approaches said position. This induced damping effectively reduces the magnitude and number of oscillations, allowing a target position to be achieved quickly. Additionally, it allows the user to set a high value of $K_P$, which allows faster response and reduces the steady state error.

Haberland's report details the hardware used for determining position, which will be summarized here. The position signal, y(t), which is an angular position in our robot, was found from potentiometer sensors that measure rotation. The robot controller sees an input range of 0 to 5 V, which is interpreted as a 10 bit signal ranging from 0 to 1023. The sensor rotates approximately 320˚, which corresponds to a proportionality constant of approximately three integers per degree of rotation.

The angular velocity term, $\dot{y}(t)$, was found from a differentiator analog circuit, shown below. The input voltage, $V_{IN}$, is the voltage from the potentiometer. The values of the feedback resistor and the capacitor are

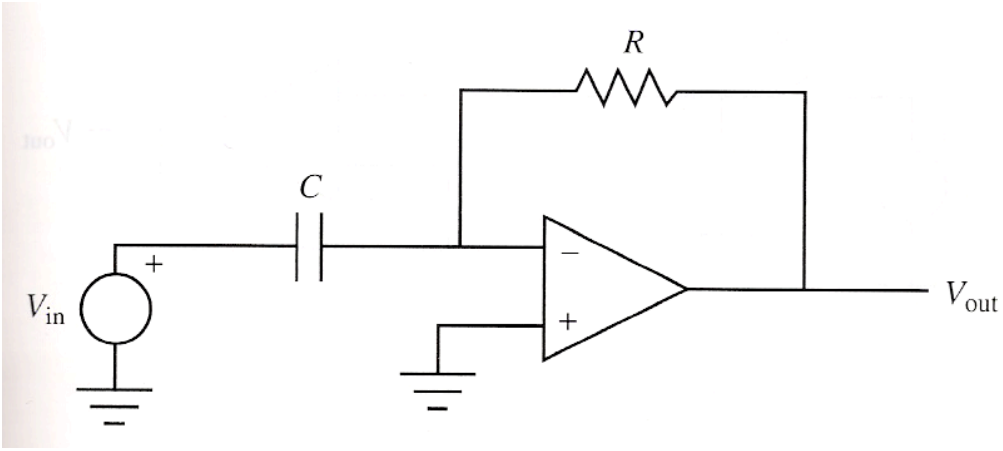



*Figure 3: Differentiator Circuit Diagram*[5]

[4] Experimental Systems and Nonlineaer Dynamics 3rd Edition
Psiaki, M.L.

[5] Introduction to Mechatronics and Measurement Systems 2nd Edition
Alciatore, D.G. , Histand, M.B.

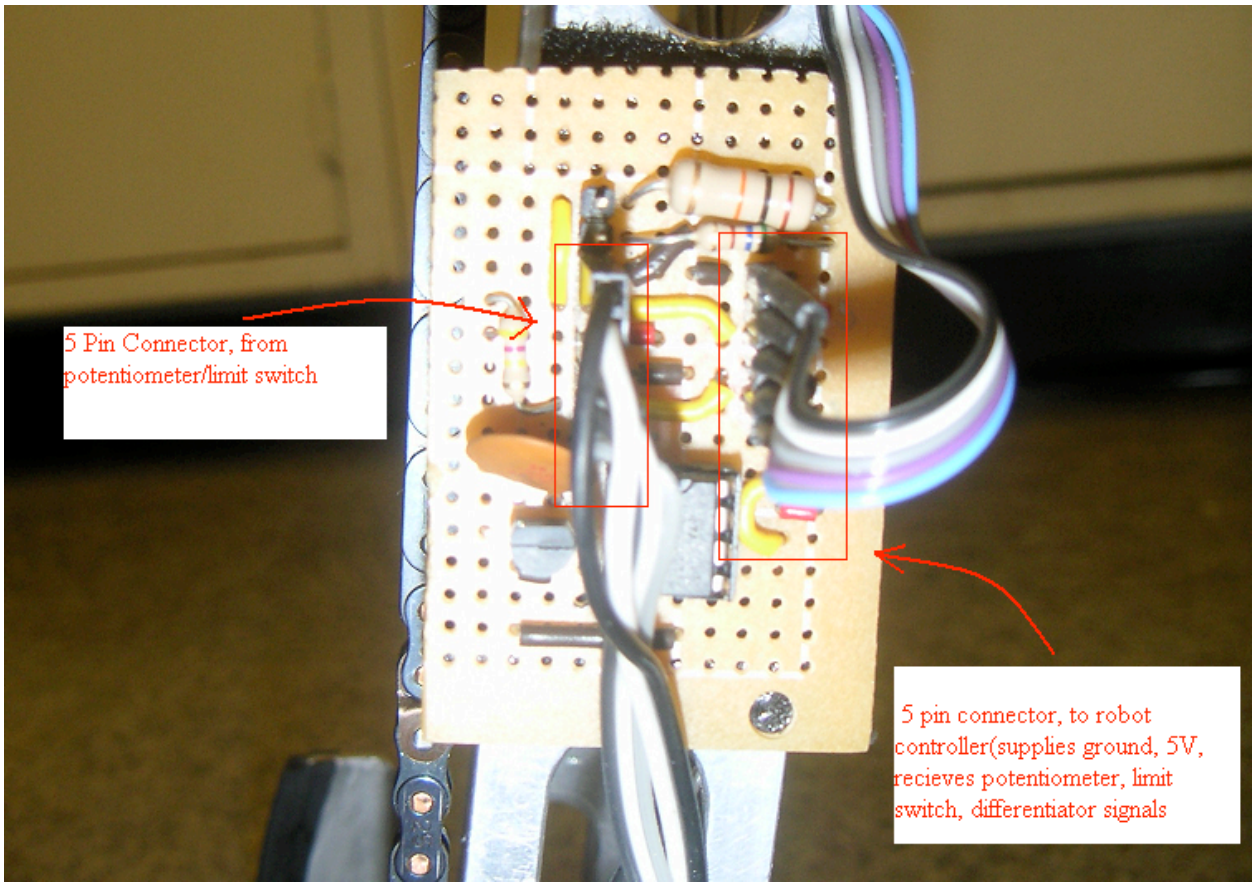



*Figure4: Differentiator Circuit and Sensor Interface Board*

After successful construction of the analog circuit and minor changes in the robot code, the implementation of PD control in ankle actuation was found to be less than desirable. Values of the proportional and derivative gains were adjusted by observing the result of each change, and basing the next iteration of changes on these findings. This work produced only modestly improved results and several causes for the instability were hypothesized.

## 3.2 Improving P-D Control

### 3.2.1 Analyzing Computer Architecture: Loop Time and PWM Frequency

As mentioned above, the effectiveness of PD control theory is heavily influenced by the rate at which the controller can make minute adjustments to the positions of the driven bodies. The documentation published by Innovation First claimed that the user controlled processor iterated as fast as possible, and depended on the nature of the programming. There were concerns that the control loop speed may have been slowed by the amount of programming, and that the PWM frequency was inadequate as well, so efforts were made to ascertain these frequencies. To test these parameters, commands that would cause digital outputs to go high or low were written into the main loop function. These digital outputs were then monitored on an oscilloscope to determine frequency. It was determined that the ran as fast as possible, and was dependent on the level of coding. The PWM frequency was tested by setting it to a specific duty cycle

and monitoring the square wave output. The frequency was found to be 60hz, which verified the published values in the IFI Controller documentation. [6]

### 3.2.2 Mechanical Lag: Effect of Backlash on Sensor Accuracy

After determining that the published values of the PWM frequency and control loop time were accurate and not the immediate cause of the erratic oscillations, the hardware itself was inspected. It was presumed that the loose motion in the gears due to the clearance existing between mechanically contacting parts was the cause of the problematic behavior. This backlash would lead to undesirable foot control because the robot controller, which receives position data from potentiometers mounted to gears, would not be aware of the motion until the teeth on the sensor sprocket made contact with the teeth of the drive chain sprocket. To test this hypothesis, the drive chain was fixed in place and the gears were rotated until the teeth contacted the chain in both directions. The sensor input data was displayed on the computer monitor, showing a range of twenty integers which corresponds to approximately five degrees of motion.

This error is significant because the controller cannot adequate regulate the ankle position and speed without accurate sensor data. The robot will apply a control signal to the motor, which will cause the ankles to accelerate quickly. However, the backlash prevents the controller from seeing a change in position and speed, and will not adjust the signal as it should in order to achieve smooth motion. This becomes especially problematic when the motor is reversing direction, hence the excessive ankle oscillation about the desired point.

### 3.2.3 Direct Coupling of Sensors to Drive Chain

To minimize backlash, the sensor mounting scheme was redesigned to allow direct coupling, thus reducing backlash to the amount in which the teeth can move in the chains, which is negligible. The new designed utilized the existing bracket, drive chain sprocket, and ball bearings. A new shaft was machined that would be connected to the sprocket with a retainer clip. The sensor end has a hole drilled to the diameter of the potentiometer output shaft, with a set screw holding the shaft in place. A spring loaded clip was machined out of delrin that would allow the sensor body to move laterally as the shaft turned, but would prevent rotation of the sensor body. A fixed bracket to hold the body was avoided because the output shaft may not be aligned perfectly with the sprocket: as the shaft rotates, the sensor body would want to shift slightly, potentially damaging the sensor if not allowed to do so.

[6] http://www.ifirobotics.com/edu-rc.shtml

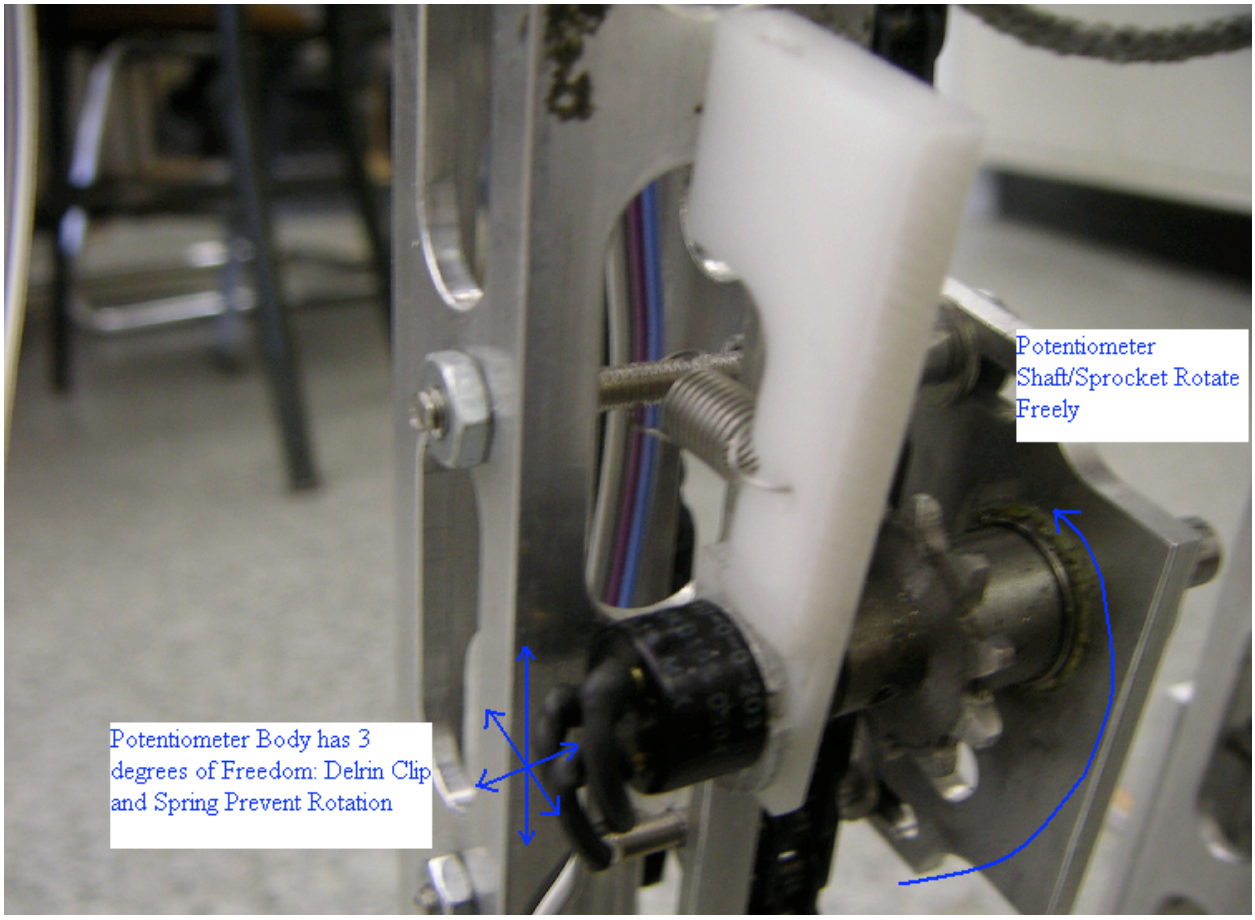
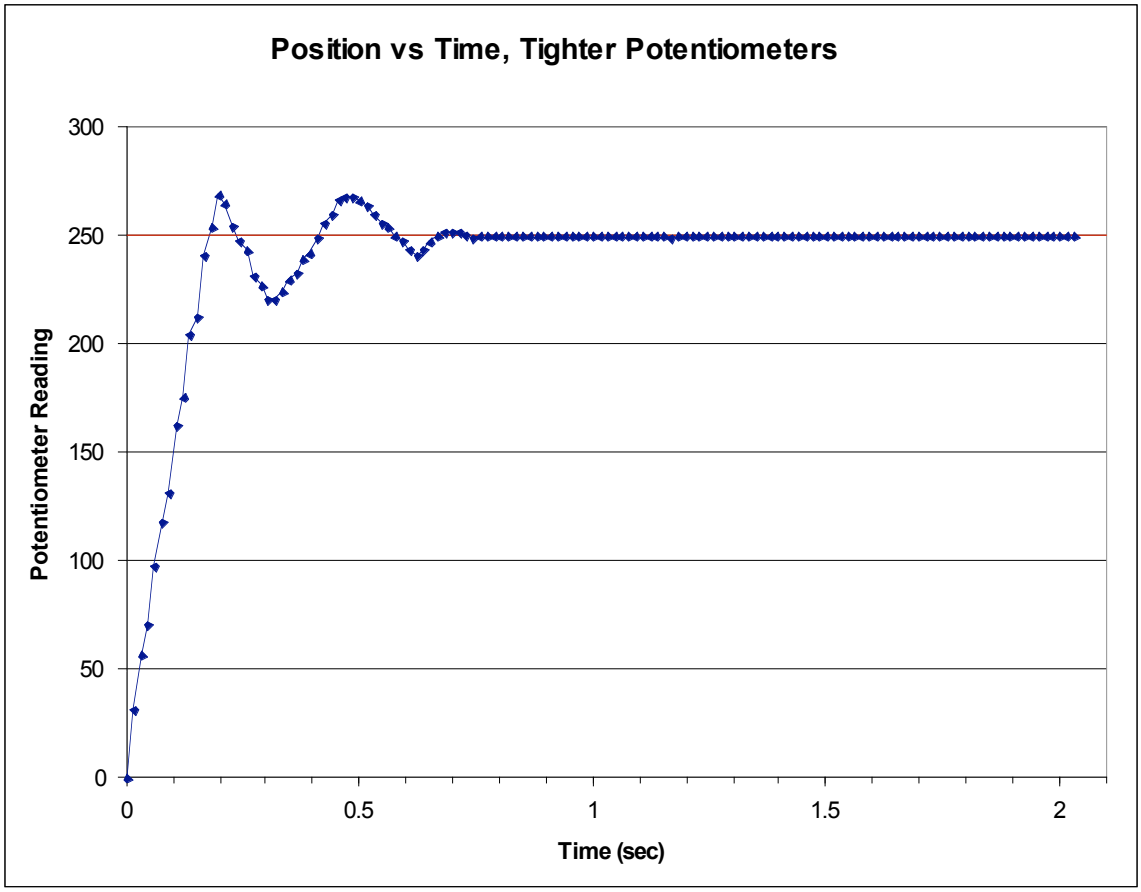



*Figure 5 : New Potentiometer Mounting Configuration*

Direct coupling of the potentiometers to the drive chain improved the feet response drastically. Figures blah and blah shows plots taken made from data points collected by a Labview acquisition program. At this stage, it becomes apparent that more drastic changes would be needed to make the robot walk for an extended period of time. The old drive belts were replaced by stronger, but heavier, steel chains, adding more weight to the outer legs and increasing asymmetry. Hip control was less than satisfactory, and alternate algoritms for controlling its action were devised. After considering these two problems, efforts were made to replace the existing speed controllers with custom H-bridges designed by Ko Ihara.



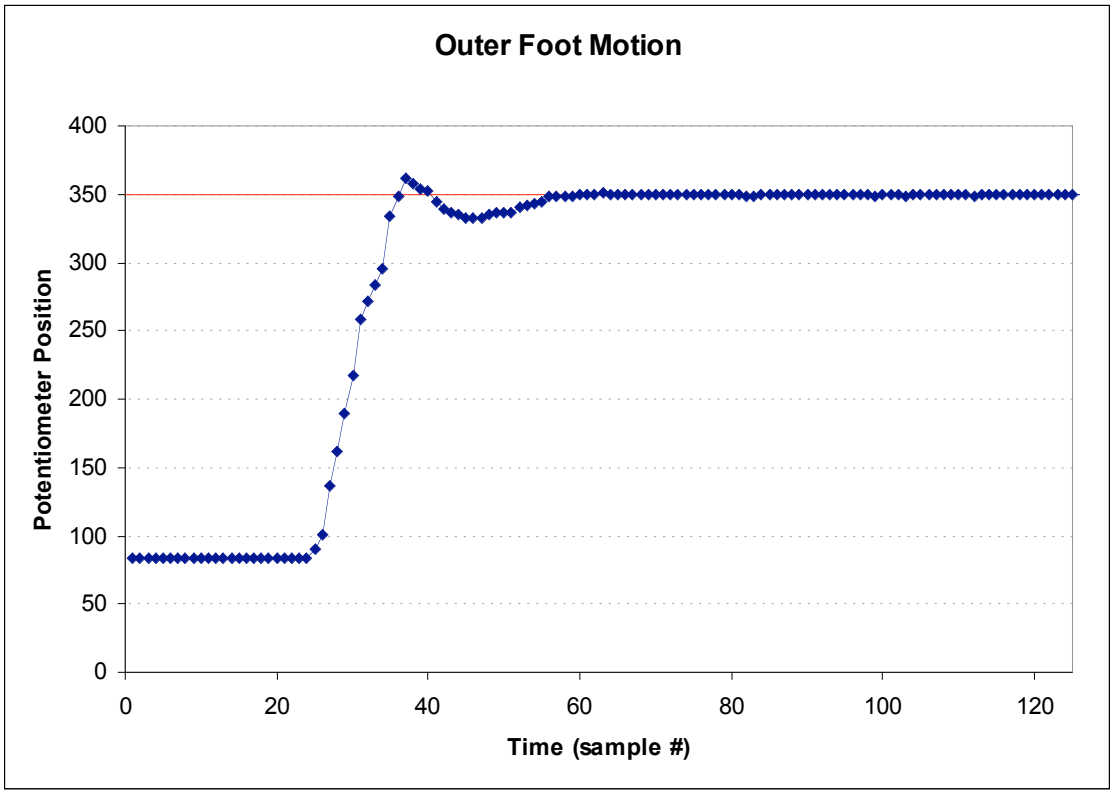




*Figure 6: Improved Foot Motion with new Sensor Mounting Configuration*

## 3.3 Upgrading to Customized H-bridges

### 3.3.1 Custom H-Bridges versus IFI Victor Speed Controllers

Replacing the Victor controllers with the H-bridges would reduce weight asymmetry in the outer legs while also improving motor control. It was hypothesized that these two results would allow better control of the hip swing, which at this point was the main source of failure. Chart blah compares the two motor control technologies, and the necessary changes that would be made in order to make the H-bridges compatible with the IFI controller.

| | **Victor's** | **H-Bridge** | **Solution** |
|---|---|---|---|
| **PWM range** | IFI control sends out PWM signal in the range of 10-20%, with 15% being neutral[7] | Uses a full 0 – 100% range | Use alternate method for controlling PWM: PIC Microcontroller has these capabilities |
| **PWM frequency** | 60 Hz, updated as fast as control loop. [8] | 20 khz | Same as above |
| **Outputs** | 1 PWM output per controller | 1 PWM output, plus two digital outputs to control direction. | Use 6 additional digital out lines from the IFI controller, 2 per motor. |

*Figure 7: Comparison of Victor Controllers to Custom H-bridges*

The benefits of switching to the H-bridges are apparent from this chart. The 20khz duty cycle will allow finer control of the motors, because the signal can be more accurately approximated as being continuous. Consider the extreme case of a duty cycle frequency of 5 hz: the motors are being turned on and off fives time per second, making precision control impossible. At higher frequencies, the inductance of the motor works better to smooth out the motion.

Significant changes to the robot code were necessary in order to make this switch. As stated in the chart, the standard PWM signal produced by IFI's internal components operates in the range of 10% to 20%. However, an alternate method for modulating a square wave pulse existed, using the functions of the PIC microcontroller embedded within IFI's controller.

### 3.3.2 Editing Registers to use Capture/Compare/PWM (CCP) Modules

The IFI controller is programmed to allow a user to use the PWM outputs easily, but without the ability to modify the signal directly. As mentioned above, the new H-bridges require modifications to the

[7] Haberland
[8] http://www.ifirobotics.com/edu-rc.shtml

PWM signal, which cannot be done to the current PWM method. The solution was to use the PIC microcontroller's PWM outputs directly.

By default, Innovation First has set up the pins on the PIC microcontrollers for various purposes. In ifi_aliases.h, , pins are assigned macros to make coding simpler for the user. For instance, the following commands makes the CCP registers easier to access.

```
#define IO_CCP2        TRISEbits.TRISE7
#define IO_CCP3        TRISGbits.TRISG0
#define IO_CCP4        TRISGbits.TRISG3
#define IO_CCP5        TRISGbits.TRISG4
```

In order to gain an understanding of how to work directly with the CCP PWM modules, I consulted a website run by Kevin Watson, who has worked extensively with IFI controllers.[9]

```
void Initialize_LED_PWM(void)
{
        // initialize PWM output 1
        TRISCbits.TRISC1 = 0; // make sure CCP2/PWM1 is setup as an output [110]
        CCPR2L = 0;                             // set initial CCP2/PWM 1 duty cycle [154]   <---CHANGED IN OUR
                                                                                                CODE ON THE FLY
        CCP2CON = 0x0C;                         // put CCP module 2 into PWM mode [149]

        // initialize PWM output 2
        TRISGbits.TRISG0 = 0; // make sure CCP3/PWM2 is setup as an output [121]
        CCPR3L = 0;                             // set initial CCP3/PWM 2 duty cycle [154]
        CCP3CON = 0x0C;                         // put CCP module 3 into PWM mode [149]

        // initialize timer 2
        TMR2 = 0;                               // 8-bit timer 2 register (this is readable and writable)
                                                        //
        PR2     = 499;                          // timer 2 period register - timer 2 increments to this
                                                / value then resets to zero on the next clock and starts
                                                // all over again. Set to (10000000/desired frequency) –
```

This portion of the PWM code initializes the registers that correspond to the PWM output, and allows changes to be made to frequency and Duty Cycle. The generic statement TRISxbits.TRISxy allows the user to change the yth bit of the xth port, which can correspond to various functions. These functions are listed in the PIC18F8250 documentation, in the section titled “I/O Ports”. Below is an excerpt from the PIC manual:

[9] http://kevin.org/frc/2005/edu_led_pwm.zip

TABLE 10-13: PORTG FUNCTIONS

| Name | Bit# | Buffer Type | Function |
|---|---|---|---|
| RG0/CCP3 | bit 0 | ST | Input/output port pin or Capture3 input/Compare3 output/PWM3 output. |
| RG1/TX2/CK2 | bit 1 | ST | Input/output port pin, addressable USART2 asynchronous transmit or addressable USART2 synchronous clock. |
| RG2/RX2/DT2 | bit 2 | ST | Input/output port pin, addressable USART2 asynchronous receive or addressable USART2 synchronous data. |
| RG3/CCP4 | bit 3 | ST | Input/output port pin or Capture4 input/Compare4 output/PWM4 output. |
| RG4/CCP5 | bit 4 | ST | Input/output port pin or Capture5 input/Compare5 output/PWM5 output. |

Legend: ST = Schmitt Trigger input

*Figure 8: Excerpt from PIC documentation* [10]

Controlling the PWM signal is quite simple once the proper ports are initialized. For example, the variable "CCPR2L" controls the duty cycle. If this variable is assigned a value in the main control loop, the PWM signal updates. This variable assumes values from 0 – 256, which allows twice the resolution of the Victor controllers because this range is for one direction only; the Victor's 8 bit range was divided evenly between forward and reverse modes.

In order to use the CCP modules for our robot, I had to modify the code provided by Kevin's website because it only allowed for control of two PWM outputs, whereas our robot required at least three. As seen in the PortG function table, PortG TRS bit 3 corresponds to PWM4, so the following code was added to the function:

```
TRISGbits.TRISG3=0;
CCPR4L=0;
CCP4CON=0x0C;
```

Again, this sets the CCP pin as an output (output when TRISGbit set low), and sets an initial value for the duty cycle. However, this did not produce the desired effect. Irregular signal patterns were seen, suggesting that this choice was not valid. It was possible that IFI's components were trying to use the same module for another purpose. This was checked by inspect the file, ifi_picdefs.h, to make sure that the port was not set up to be used for another purpose.

After reading through the PIC manual to determine which ports were available, and comparing them to IFI's list of specified macros, the bits corresponding to PWM modules in Ports E and G were used. This selection proved successful, and three working PWM signals were observed. The analog signal from one of the potentiometers was used to verify that the square wave signals were adjustable. The signal modulated flawlessly, concluding this phase of the H-bridge implementation.

[10] PIC

### 3.3.3 Integrating the Robot Code with the CCP/PWM Function

In order to integrate the new PWM method with the working robot code, the SendPWM() function was rewritten. In user_routines.h, macros for the CCP registers and the digital outputs that are needed for directional control were defined.

```
#define PWM_In                  CCPR2L
#define PWM_Out                 CCPR3L
#define PWM_Hip                 CCPR4L
#define HipForward              rc_dig_out04
#define HipReverse              rc_dig_out05
#define OuterForward            rc_dig_out06
#define OuterReverse            rc_dig_out07
#define InnerForward            rc_dig_out08
#define InnerReverse            rc_dig_out09
```

The following is a portion of the SendPWM function for only the case of inner ankle actuation:

```
SendPWM(char motor, int pwm)
{
      unsigned int pwm_abs;
      switch(motor)
      {
            case IN:
            {

                 InnerForward=(pwm>0); //set digital outs to proper values for
                                       forward, reverse.(pwm>0)=1 if that statement
                                       is true
                  InnerReverse=(pwm<0);
                  if (pwm>1023 &&pwm<-1023 )     //if pwm exceeds bounds, set to
                                                      highest value
                  {
                        PWM_In=1023;
                  }
                  else
                  {
                        pwm_abs=(pwm>=0)*pwm-(pwm<0)*pwm;
                        pwm_abs>>=2;
                        PWM_In=(unsigned int)pwm_abs;
                  }     */
```

Direction control is first set by the InnerForward or InnerReverse statements. For example, the statement,"pwm>0" is equivalent to the integer '1' if that expression is true, or '0' if false. This is convenient since '1' or '0' are the only values needed for direction control. Once direction is set, the PWM value that will be set to the proper output is refined. Since the digital outputs handle direction, the PWM signal must be positive, so the absolute value of the PWM signal is calculated and sent to the appropriate registers.

### 3.3.4 Debugging Software

At this point, I tested the output from the PWM modules and discovered that there were some problems. I noticed erratic squarewave behavior on the oscilloscope and began looking for causes. My first

hypothesis was that two parts of the code were trying to write to one register simultaneously, which may have occurred due the manner in which I attempted to combine CCP PWM code with an existing project. Originally, I copied the code directly from the CCP PWM project into an existing project that included all the walking algorithms. Rather than search through hundreds of lines of code for an error that I may not even be able to find, I performed the following steps to guarantee that I am starting from a clean slate :

1. Created a new project from LED_pwm compressed folder available on the web
2. Edited registers according to method I found last week
3. Began copying in the various functions and initialization protocols from working versions of the Marathon robot code one step at a time.
4. compiled code periodically and tested output on the oscilloscope

After a few trials of this kind, I created a working version of the code that contained all previous functions in conjunction with the new PWM routine. I used the analog signal from the potentiometers to adjust the PWM duty cycle, so I could verify in the oscilloscope that my code functioned properly.

## 3.4 Designing the New Interface Board

The successful integration of the H-bridges with the IFI controller required a new scheme for mounting the bridges. It became immediately clear that the best solution would to design a new interface board that would integrate the new h-bridges, along with other components. The board would also avoid the shortcomings of the old interface board. The soldered DIP header connections often broke when trying to remove the cables, and there were a few extraneous ports.

### 3.4.1 Interface Design Considerations

The major consideration in designing the new board was modular design. Future versions of the robot may require additional sensors, so an efficient way of accommodating these needs would be required. Additionally, the method for wiring these connections would have to foster the ability to make adjustments over the lifetime of the robot. A sleek, robust method for incorporating the H-bridges into this board was also a priority. After considering the above specifications, the following board design decisions were made.

***10 pin Crimp Connectors.*** Experience has shown that stripping the ends of ribbon cable and soldering to the DIP connections often results in failure, as evident by numerous repairs needed to the potentiometers and the interface board. On the contrary, ribbon cable is designed for use with crimp connectors, which allow strong and reliable connections to be made without any soldering. The female end connector comes in normal and 90° configurations, making it a versatile method for wiring external hardware to the board.

***Wire Wrapping.*** Soldering is a good method for making sturdy connections, but is not easily changed, especially when the wiring becomes complicated. Wire wraps provide the user with the ability to remove

wire quickly, and make changes with ease. This was very important to achieving the goal of a modular interface board design.

***H-bridge frame mounted directly above interface board.*** The size of the three h-bridges combined allowed them to be built into the new board, thus simplifying the wiring of the robot as a whole, and reducing weight asymmetry.

### 3.4.2 Interface Board Components and Circuitry

The new board would contain many of the same features as the old version, such as the power regulator for stepping down the battery voltage to 6 V for the IFI controller, and the screw terminal for directing the high current lines. There are appreciable differences as well. The fuse was replaced with a circuit breaker that was mounted on the support board on the robot chassis. There would be three auxiliary ports that future engineers of the project could use for their own purposes. The new design is illustrated below.

- **PWM Output Connectors (3).** The three PWM connectors are placed along the right side. Their spacing was determined to coincide with the spacing of the h-bridge boards' pin spacing. The short distance between this board and the rails that the bridges will be mounted on make it necessary that the cable coming from h-bridges be aligned with the bridges pins and the board sockets, since the cables cannot bend much in that distance.
- **Potentiometer Input Connectors (3).** The top three sockets correspond to each of the potentiometer inputs. Their placement was somewhat arbitrary, barring that the cables from the potentiometers are not long enough to reach the sockets.
- **40 pin IDE connector.** This connector is located at the left as in the last board, so that the ribbon cable does not have wrap around the robot and interfere with its functions in any way. The wiring for the connector is very similar as well; pins 1-8 correspond to PWM pins( which were not used on the previous model), pins 9-24 map to the sixteen digital in/out/analog in pins on the robot controller, 25-29 are grounded, 30-34 are +5V, and 35-40 are the digital interrupts.
- **Auxiliary connectors (3).** In the future, accelerometers, gyroscopes, and other sensors may be added to provide more accurate state information. Rather than incorporate the wiring for these components into the unused pins on the aforementioned connectors, these sensors would have their own connectors to avoid confusing wiring diagrams and mapping problems. For instance, several gyros (with one output each) could be wired on one cable, sharing the +5V and ground pins. Two accelerometers could be wired on their own cable; +5V, ground, and x-y-z axis signals for each sensor. Additionally, the solenoid outputs from the controller could be wired to their own cable. These outputs may be used to control the ratchets (on/off signals are needed for the solenoid that

engages or disengages the ratchet), or to provide directional information to the motor controllers. This would free up more digital outputs on the controller to be used for other purposes.

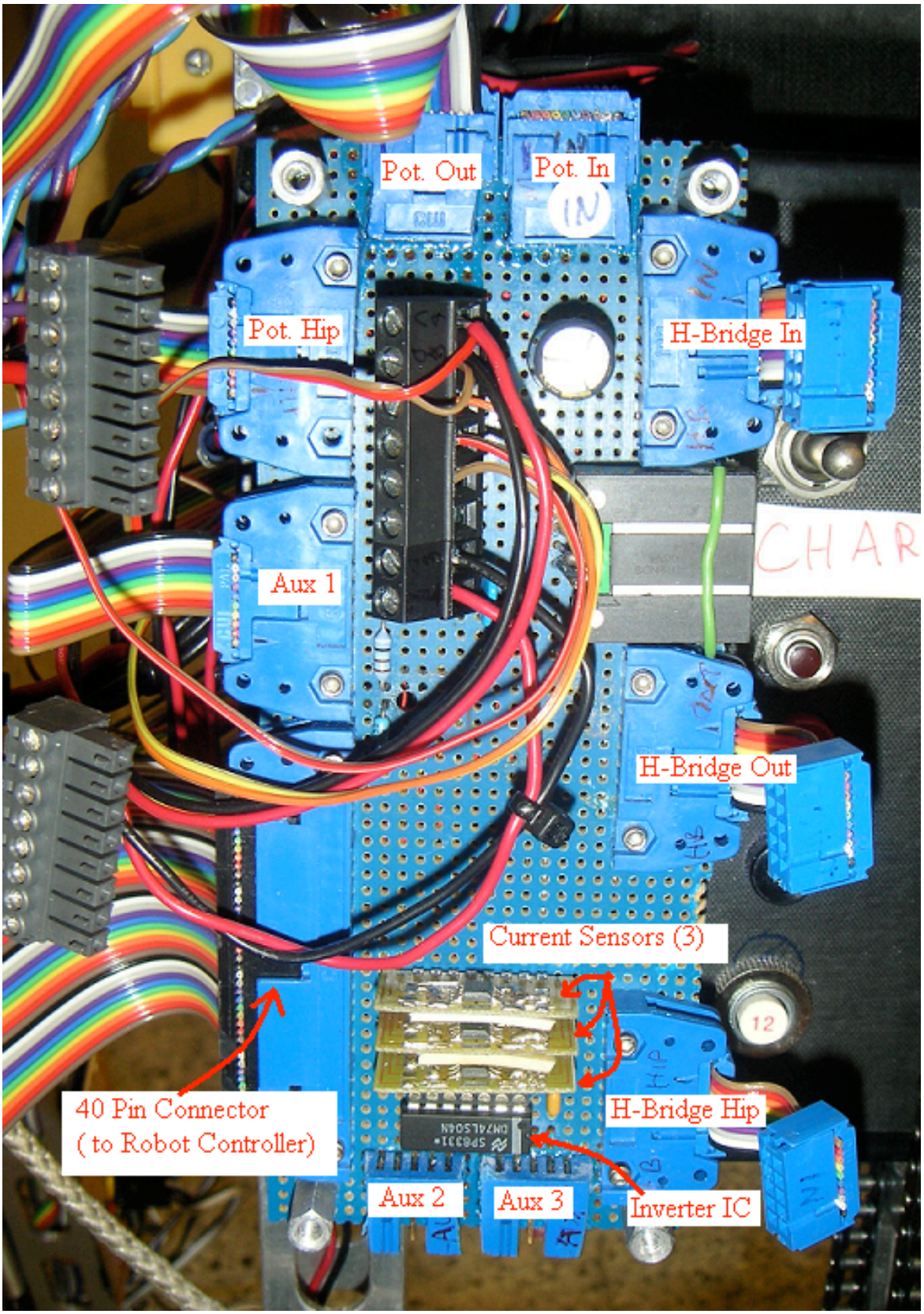



*Figure 9: Photograph of new Interface Board w/o H-bridges*

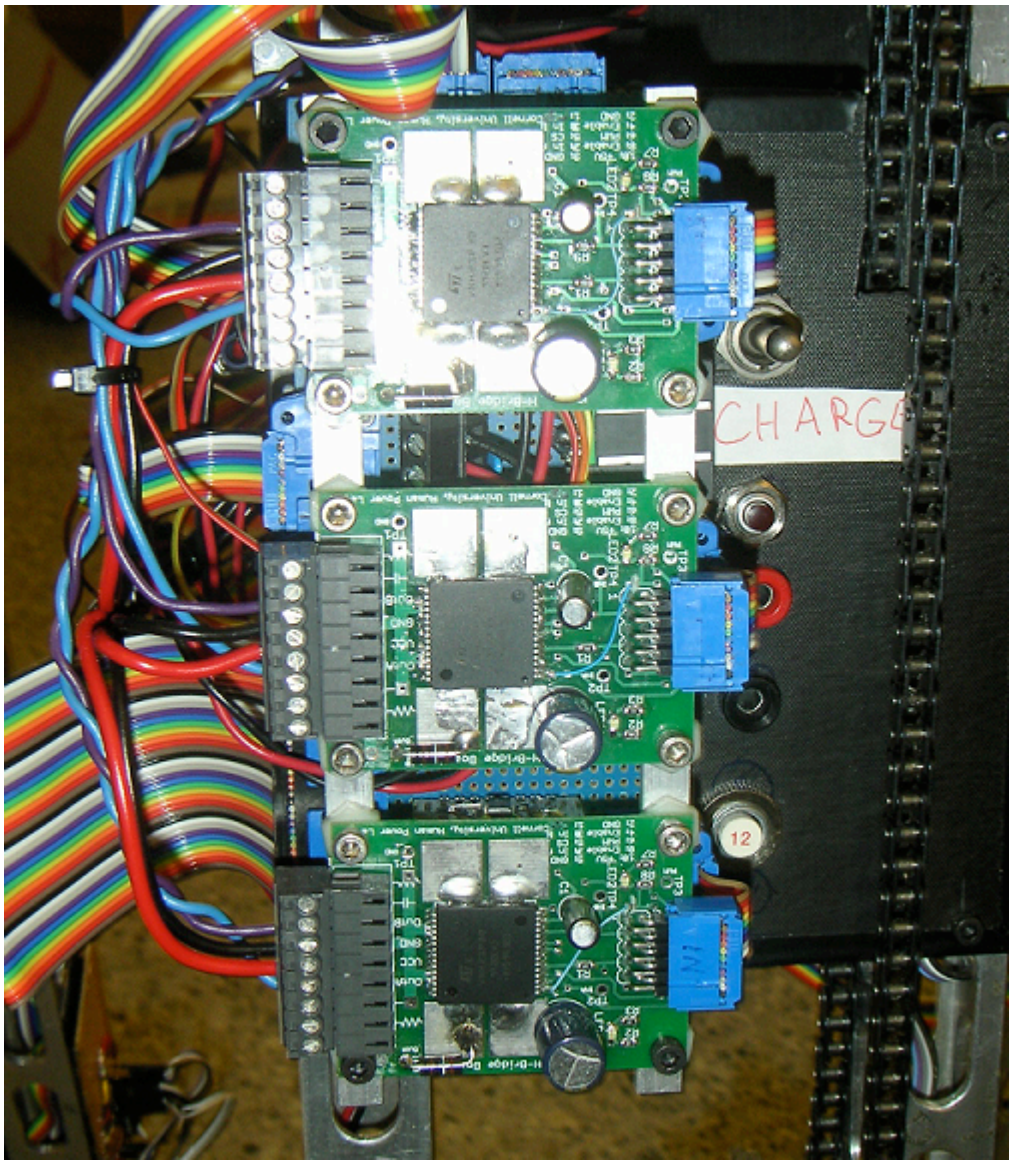



*Figure 10: Interface Board with H-bridges*

### 3.4.2 Bracket Design

The interface board was designed such that the width(shorter dimension) was slight larger than width of h-bridge boards. The h-bridges boards measured 1.5 inches from the center of the predrilled lines(on input/out sides) and 2 inches hole-to-hole on the non-input/output sides. The measurements of 0.7” and 0.59” allow the centers of the h-bridge input pins to line up with the 10 pin connectors directly beneath them. The top and bottom holes are not threaded so that long, 6/32 X 5/8”socket cap screws can go through the rails and thread into standoffs that separate the board from the rails. Half inch standoffs are placed above the rails, fixed to the h-bridge boards.3/4” standoffs are used between the wiring board and the robot chassis. All standoffs are threaded 6/32.

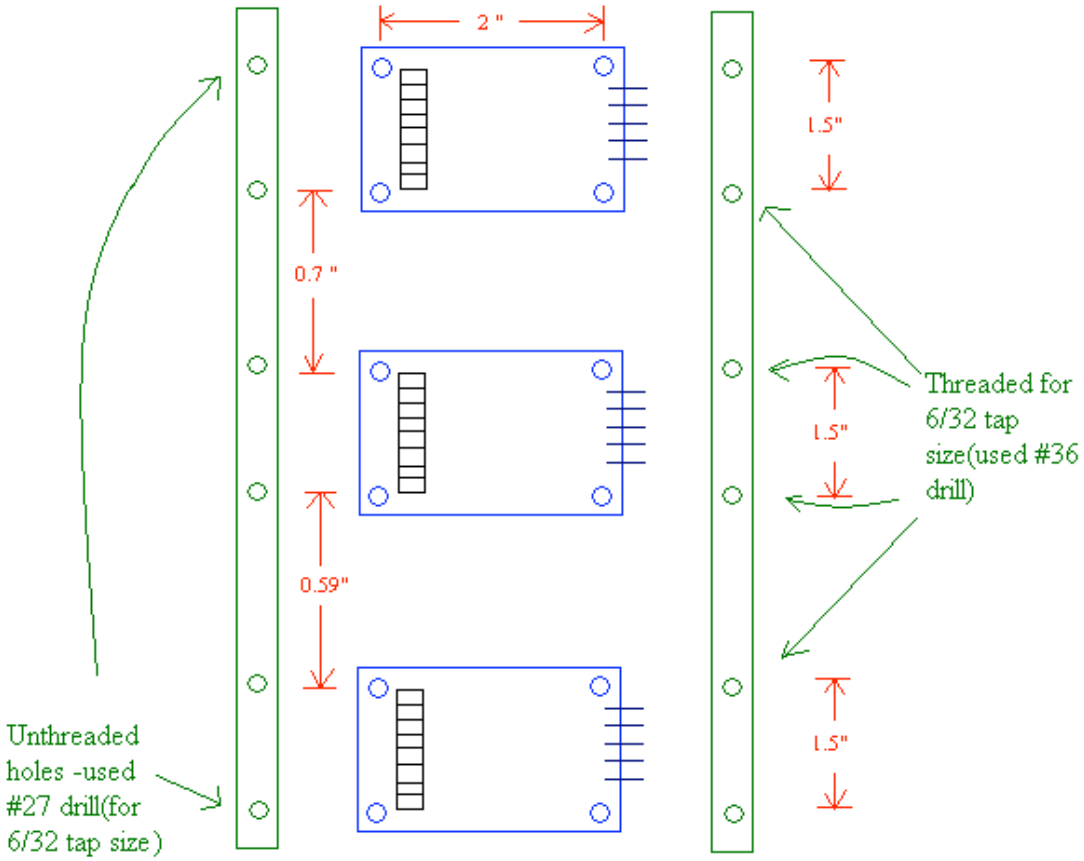



*Figure 11: Drawing of H-bridge bracket with dimensions*

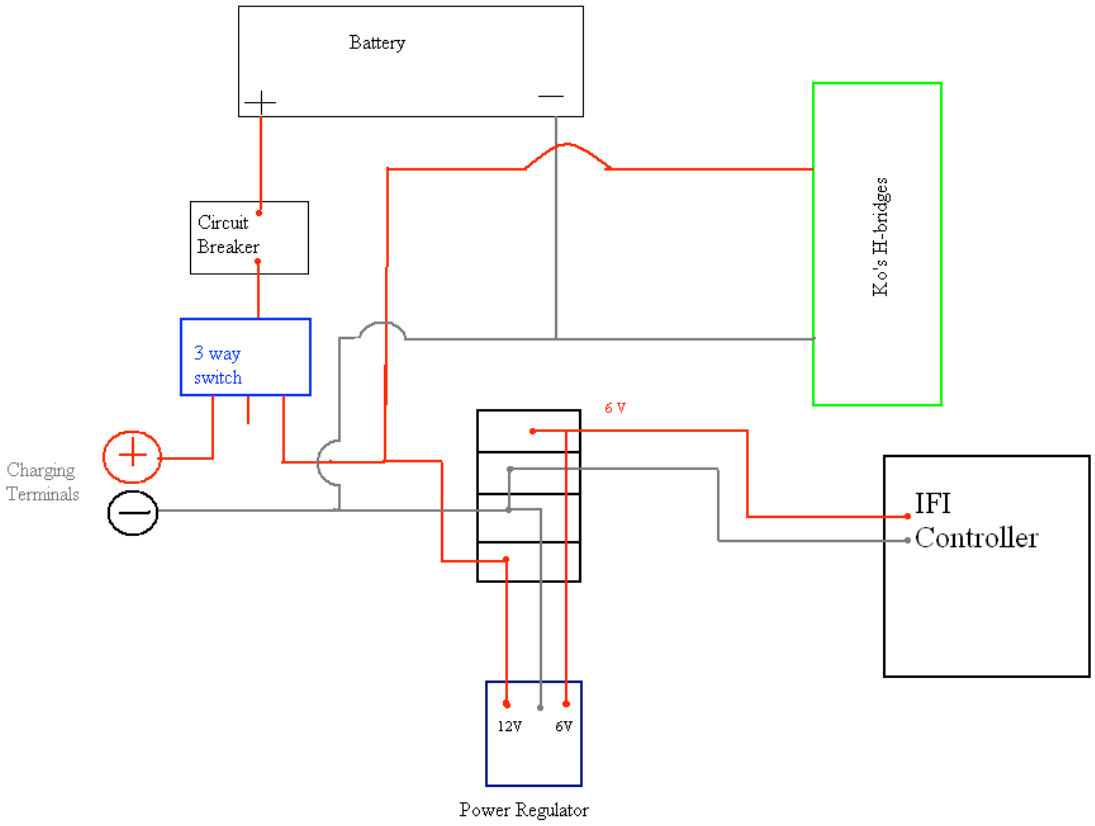



*Figure 12: Power Supply Wiring Diagram*

## 3.5 Improvements to Hip Control

### 3.5.1 Non-linear Hip Control

One of the major sources of failure was the inadequacy of microprocessor to control the hip motion. The high mass and inertia properties of the outer leg made PD control very difficult, mainly because any overshoot meant that gravity would pull the leg further down then expected. It was recommended that non-linear control be attempted, such as dividing the hip-swing into several phases and trying different commands. The first that was attempted was the following algorithm:

- Constant signal to bring rear leg forward, up to certain angle
- No power to leg: leg swings through( like a pendulum) to a forward position
- Constant power to slow down leg over a certain length of time
- High gain, damped linear control to lock leg in position

There were many problems evident with this algorithm. First, the outer legs were too massive for them to act like a simple pendulum during the second phase. The constant reverse power was inadequate to slow down the legs sufficiently, causing wobbling in the fourth phase.

### 3.5.2 Scaling Hip-swing Power based on Angular Velocity

An alternate algorithm was to apply some amount of power during the second phase(more for the outer legs), and apply a reverse signal during phase three to slow it down to zero angular velocity. Then, the PD loop would take over. This algorithm required iterating over the relative positions separating the phases, as well as how much power to send during the constant power phases. The result was reduced bouncing in the final phase, but there was still a significant amount of instability. The algorithm was refined so that the amount of power given to slow down the leg would be proportional the square of the speed. Additionally, the target position of the fourth phase was changed to coincide with the stopped position reached in phase three. This was done because the difference between the stopped and final positions was often large enough that significant force would have to be exerted to bring the leg to the final position, again causing overshoot due to the high inertia. This method is illustrated in the figure blah.

*Figure : 4 Phase Hip-Swing Algorithm*

The method for adjusting the power of the reverse torque phase was done by comparing the velocities at the onset of the reverse torque, in addition to final resting position, to values found with an ideal model that was generated in the lab. These velocities and stop positions were recorded for many trials while the robot was suspended above the ground.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **IN** | **Speed** | 51 | 46 | 48 | 50 | 55 | 51 | 47 |
| | **Position** | 475 | 475 | 476 | 480 | 476 | 477 | 479 |
| **OUT** | **Speed** | -32 | -43 | -43 | -45 | -39 | -41 | |
| | **Position** | 325 | 327 | 324 | 324 | 327 | 326 | |

*Figure 14: Speed and Positions for various trials with robot suspended*

From the above chart, one can see both precision and accuracy in the final position readings, as the target positions for HipSwing ( ) are 480 and 320 for the inner and outer legs respectively, yielding errors of only a degree. These values were achieved using PWM signals that were a certain fraction of the maximum available signal( a higher fraction for the outer legs due to larger mass). In order to make Hipswing( ) adaptable to the varying dynamics of walking motion, the function was programmed to scale the amount of power the motor receives during the reverse torque phase to be proportional to the error between the actual velocity at the beginning of this torque to the ideal velocity, recorded during testing. The equation is as follows:

$$pwm = pwm_{ideal}\left(\frac{V_{actual}}{V_{ideal}}\right)$$

Thus, for +10% error (i.e. the actual speed is greater than the ideal speed by 10%), the output PWM will be 10% larger. The necessity for the initial $pwm_{ideal}$ arose because the mass properties of the leg were not immediately known, so a value for this signal was experimentally determined.

First principles suggest that the energy needed to compensate for the higher velocity might be larger. From an energy standpoint, the torque is applied to do work to slow down the leg swing, yielding the following equations:

$$W = T\theta$$

$$W = \Delta KE = \frac{1}{2} I\varpi^2 \qquad \rightarrow \; T \propto \varpi^2$$

Since the target position and point at which this torque is first applied are fixed, the range over which the torque acts, θ, is constant in this equation. The difficulty with this algorithm is that controlling the PWM signal does not directly correlate with controlling torque. A current control method, which would adjust the PWM signal incrementally until the desired torque is achieved, will be very helpful for this method to function properly. Current control would also have the added benefit of providing the exact torque necessary to move the inertial masses with a desired angular velocity. This will of course involve obtaining mass and inertia properties of the legs and ankles, which may or may not be easily accomplished.

### 3.5.3 Re-enforcing the Hip Motor under High Torques

The method of applying a strong, reverse torque to slow down the outer legs put a great deal of strain on the system. The hip motor bracket would be displaced over time because it relied on friction between the bracket and the shim to remain in place. This shift loosened the hip chain significantly, which made controlling the hip even more difficult. I designed a new mounting bracket that would be supported on the top of the robot chassis. Shims were placed between the top of the bracket and the robot chassis to make slight adjustments to the tension. Under high torque, the drive chain cannot pull the bracket any farther down since the chassis acts as a physical stop for the bracket.

The bracket was also designed to be more easily removed and reattached than its predecessor. The four holes on the right side near the potentiometer mount allow an Allen wrench to be inserted and screw in the hex screws. This requires the user to position the sprocket such that the spokes are not obstructing the four motor mounting screws. An added benefit of this design is that the potentiometer is centered with the output shaft of the motor.

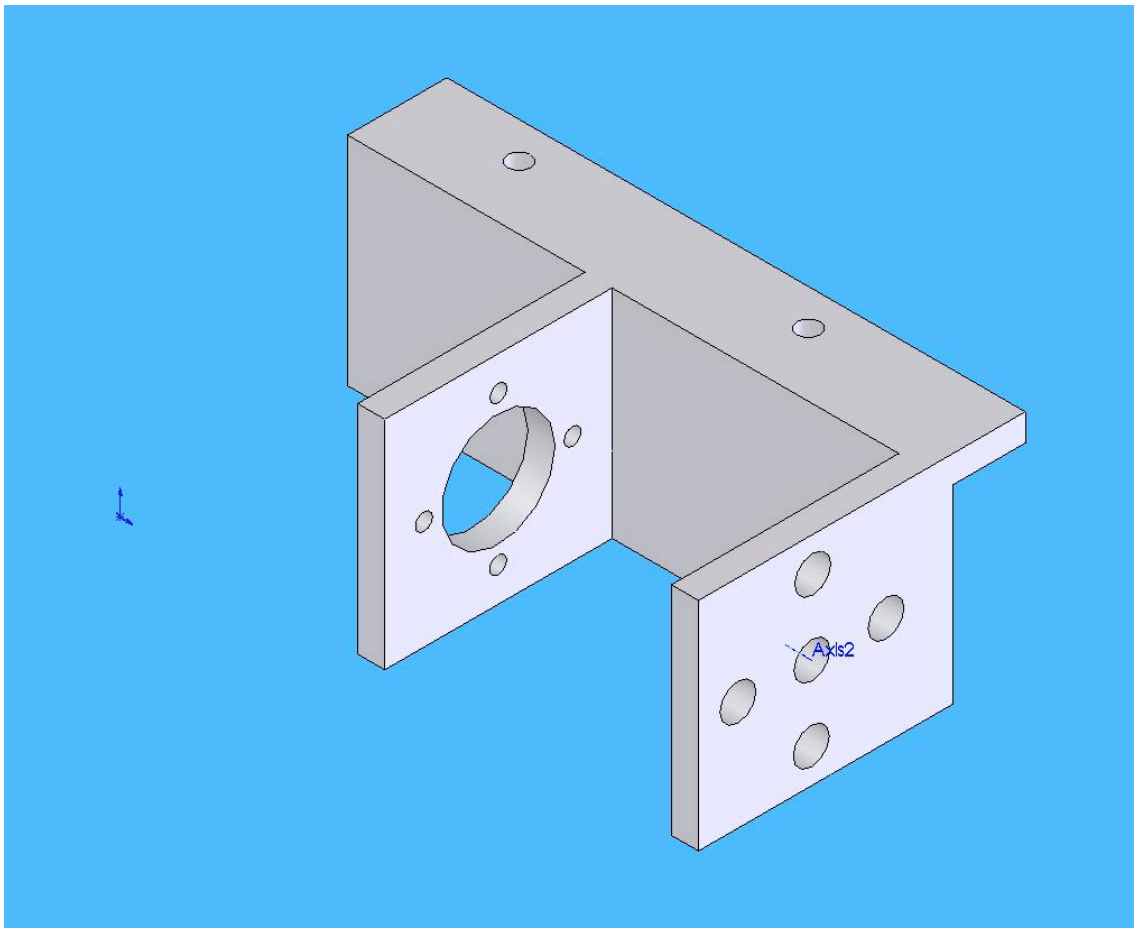



*Figure 15: New Hip Motor Mount*

## 3.6 Upgrading to the KoBrain Microcontroller

In order to realize the goal of a truly robust walker, it was decided that the IFI controller be replaced with the much faster, more versatile chipset engineered by ECE Graduate Student Ko Ihara. The new board, which contains many more inputs/outputs, has built in gyroscopes and accelerometers for measuring orientation and motion, and is far more understood from a programming standpoint, would allow the project to be taken to a whole new level.

The increased speed along will enhance many aspects of the control. As mentioned earlier, control theory is based on the idealization of a continuous loop time with infinite sampling frequency. A faster loop makes this approximation more accurate, thus yielding more precise control. The much faster processor enables the use of the built-in motor optical encoders, which the previous controller was unable to use due to the immense data flow and the inability to process it all. Using the encoders will forego the often temperamental method of using potentiometers geared to the drive chains, thus eliminating all backlash associated problems.

The benefits from increased speed, combined with the built in MEMS will provide the robot with an enhanced awareness of its current state, altogether improving its tendency to make the proper corrections to its motion.

### 3.6.1 Overview of Input/Output/PWM functions on KoBrain Microcontroller

The KoBrain has been programmed to perform the basic tasks that the IFI controller did. Jason Cortell has written code that is attached as an appendix to this paper. Though the details of how these actions are performed are beyond the scope of this report, the following commands were crucial to modifying the robot code to work with the new microcontroller.

```
#define InnerFootDown                   *GPIOB_DR|=0b0000000000000001;
#define InnerFootUp                     *GPIOB_DR&=0b1111111111111110;
```

These commands demonstrate the ability to set a digital output high or low. The second part of this statement is a binary number, where each bit represents a part of the register. If the desire is to set one of the bits high, but not change the others, that bit is OR'ed with 1, guaranteeing that it is high regardless of what the bit used to be. OR'ing the other bits with zero guarantees that those bits will not change regardless of what they used to be. These two facts are easily demonstrated by a simple OR truth table.

The second statement is used to set a bit low, so the AND statement is used. Here, if you and B with 0, it does not matter what the value of B was previously. Likewise, if you and AND the other bits with 1, they will remain the same regardless of their initial value. These two facts can also be illustrated in a truth table.

The following lines of code are important for understanding how various inputs are read.

```
#define LimitHeelIn                     (*GPIOA_DR & 0b0000000000000001)
#define LimitHeelOut                    (*GPIOA_DR & 0b0000000000000010)

#define CurrentHip                      AD7490_Values[6]
#define CurrentOut                      AD7490_Values[7]
#define CurrentIn                       AD7490_Values[8]
#define V_battery                       AD7490_Values[9]

#define EncoderIn                       (*TMRD2_CNTR)
#define PWM_In                          (*PWMA_PWMVAL0)
#define PWM_Out                         (*PWMA_PWMVAL1)
#define PWM_Hip                         (*PWMA_PWMVAL2)
```

Analog inputs are read by reading from a certain section of a data array, which is updated in other parts of the code that are more abstracted from the standard user. The data from the optical encoders are written to a timer counter, which the user can reference via a macro. Digital inputs are operated in the same way. PWM

signals can be modify by writing to the corresponding register with a value ranging from 0 to 1275, and running the function update_PWMA ( ) to change the signal.

Modifying the existing code to work with the new processor was fairly straightforward. The macros for various inputs/outputs were changed to reference the new ports and bits, and functions were modified to work with the new macros. The finalized code is attached as an appendix.

# Results

| Wire Name | From | To(on 40 pin connector) |
|---|---|---|
| Inner Pot. Black (Ground) | In10 | A25 |
| Inner Pot. White (+5V) | In9 | A30 |
| Inner Pot. Grey (Signal) | In7 | A9 |
| Inner Diff. (Signal) | In6 | A12 |
| Inner Limit Black | In8 | A35 |
| Inner Limit White (Ground) | In10 | A25 |
| Out Pot. Black (Ground) | Out10 | A26 |
| Out Pot. White (+5V) | Out9 | A31 |
| Out Pot. Grey (Signal) | Out7 | A10 |
| Out Diff. (Signal) | Out6 | A13 |
| Out Limit Black | Out8 | A36 |
| Out Limit White (Ground) | Out10 | A26 |
| Hip Pot. Black (Ground) | Hip10 | A27 |
| Hip Pot. White (+5V) | Hip9 | A32 |
| Hip Pot. Grey (Signal) | Hip8 | A11 |
| Hip Diff. (Signal) | Hip7 | A14 |
| Inner H-Bridge PWM | HBIN 6 | A1 |
| Inner H-Bridge Ground | HBIN 10 | A27 |
| Inner H-Bridge In A ( direction) | HBIN3 | SOL1 |
| Inner H-Bridge In B ( direction ) | HBIN7 | INV1(Y) |
| Outer H-Bridge PWM | HBOUT 6 | A2 |
| Outer H-Bridge Ground | HBOUT10 | A28 |
| Outer H-Bridge In A ( direction) | HBOUT3 | SOL2 |
| Outer H-Bridge In B ( direction ) | HBOUT7 | INV2(Y) |
| Hip H-Bridge PWM | HBHIP6 | A3 |
| Hip H-Bridge Ground | HBHIP10 | A29 |
| Hip H-Bridge In A ( direction) | HBHIP3 | SOL3 |
| Hip H-Bridge In B ( direction ) | HBHIP7 | INV3(Y) |

| | | |
|---|---|---|
| Inverter Input 1 | INV1(A) | SOL1 |
| Inverter Output 1 | INV1(Y) | HBIN7 |
| Inverter Input 2 | INV2(A) | SOL2 |
| Inverter Output 2 | INV2(H) | HBOUT7 |
| Inverter Input 3 | INV3(A) | SOL3 |
| Inverter Output 3 | INV3(H) | HBHIP7 |