ROBOTIC BALANCE THROUGH AUTONOMOUS OSCILLATOR CONTROL AND THE DYNAMIC INCLINOMETER

By

NATHAN JAMES NIPPER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING

UNIVERSITY OF FLORIDA

2001

ACKNOWLEDGMENTS

I would like to thank my wife, Brianna, who experienced many nights alone while I worked in the lab and also endured a pregnancy while I finished my research. This would not have been possible without her unconditional love and support.

Special thanks go to Johnny Godowski. His dreams have inspired this work. He is a visionary and a pioneer in the robotics field. This work would not be possible without his guidance and advice.

I would also like to thank Dr. Eric Schwartz and Dr. Antonio Arroyo for allowing me to pursue my research in the Machine Intelligence Lab. It has truly been a great experience.

Special thanks go to my parents, for always believing in me. I hope that I will be the same inspiration for my children that they have been for me.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering

ROBOTIC BALANCE THROUGH
AUTONOMOUS OSCILLATOR CONTROL
AND THE DYNAMIC INCLINOMETER

By

Nathan James Nipper

August 2001

Chairman: Dr. Antonio Arroyo, Associate Professor
Major Department: Electrical and Computer Engineering

In order to have robotic systems capable of complex tasks in ever-changing and uncertain environments, we must develop systems with the capability to balance in the general dynamic case. The primary purpose of the research is to demonstrate that active balance through torque application is achievable and able to be implemented independent of support base orientation.

This goal can be realized by using an inertial sensor array that can generate accurate position and velocity data independent of support base orientation, in conjunction with a platform that can demonstrate efficient balance control using position and velocity data by varying only stiffness and driving.

CHAPTER 1
INTRODUCTION

Research Goals

The primary purpose of the research is to demonstrate that active balance through torque application is achievable and able to be implemented independent of support base orientation in the general dynamic case. This goal can be realized through the combination of two platforms:

1. An inertial sensor array that can generate accurate position & velocity data independent of support base orientation.
2. A platform that can demonstrate efficient balance control using position and velocity data by varying only stiffness and driving.

The design and control of these devices form the basis of this thesis.

Motivation for Balance

In order to have robust systems capable of complex tasks in ever-changing and uncertain environments, we need to have systems that model nature. The primary reason for modeling nature is that nature is efficient. The balance and coordination achieved by humans and animals has proven to be stable in varying environments and robust in multiple applications.

Another reason for developing human-like robots is that the planet is pre-engineered for human function. Our world consists of uncertain terrains such as stairs and buildings. Robots of the future will have to be able to perform their tasks and move around in these uncertain and varying environments.

Nature has determined that legs are the best method for locomotion. Land-dwelling animals use legs to get from one place to the next.

Another reason for legs is that walking is a very efficient use of energy. The trade-off is that the geometry and dynamic model become very complex. Red Whittaker, of Carnegie Mellon University, says, "Without mobility, a robot is limited to only the space where it can reach. So mobility, as a function, is the crux – the essential capability for complex achievement."

The major disadvantage to creating human-like robots (two legs) is that balance is more difficult. Gill Pratt, the director of the MIT Leg Lab, notes, "With two legs, you have to do dynamic balancing." The cost of greater mobility is decreased stability. In order to receive the benefit of human-like robots, we have to overcome the problem of balance in the general dynamic case.

Balance by Support Shift

There are two types of balance, support shift and torque balance.

Support shift balance is the typical solution to the inverted pendulum (or rotational inverted pendulum) problem. In this case, the support base of the robot can be moved laterally to balance a load. The result is the balance is achieved because the support base remains beneath the load.

Support shift balance can also be observed during walking, the system can be dynamically tuned to achieve a stable oscillation around the support base. The support base is shifted slightly to maintain balance.

The advantage of support shift balance is that it uses low energy to accomplish balance within a specified non-constrained path. When the balance parameters are no longer within the path constraints of support shift balance, torque balance is required.

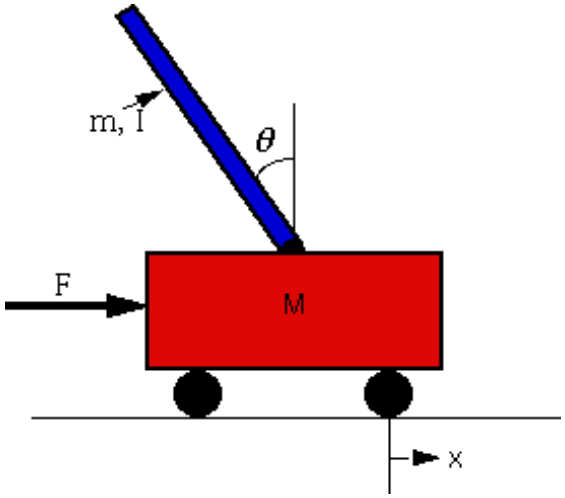


Figure 1-1: Inverted Pendulum balanced by Support Shift

Balance by Torque

Applying torque between respective segments of a system to induce a support reaction that will return the system to the desired position

Torque balance requires much higher energy than support shift balance, but it is necessary in situations where the path or support base is constrained. The following examples describe situations where torque balance is needed:

1. Standing on one leg.
2. Walking on a beam (straight line or a tight-rope).
3. When acted on by an outside force.
4. When a sudden direction change is necessary.
5. Riding a bicycle (support constrained).

The Importance of Balance

In order to accomplish complex tasks, a free system must continually configure itself in the driving field while maintaining balance. The robot becomes the link between the object to be acted on and the support base of the robot.

It is essential to maintain coordinated balance in order to apply the required forces to complete a general task. The most efficient balancing system would contain both support-shift and torque balance components and be capable of operation in a dynamically changing environment.

CHAPTER 2
LITERATURE REVIEW

This chapter summarized many of the current balance technologies being used today.

3D One-Legged Hopper – MIT (1984)

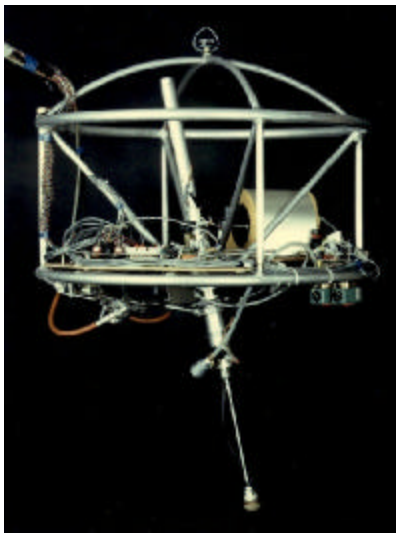


Figure 2-1: 3D One-Legged Hopper

The 3D One-Legged Hopper has a 2-axis hip powered by hydraulics. The leg is powered by compressed air.

The robot is able to remain upright by bouncing on the single leg. The action resembles a pogo stick. The robot must keep hopping in order to maintain balance.

The equations of motion reduce to a ballistics problem when the robot is in the air. This is due to the structure. The frame is very heavy (38 lb.) while the leg is relatively light. So movement of the leg in the air does not significantly affect the inertia

of the system. Using this assumption, the location and orientation of the landing can be calculated. This robot may use an inertial sensor to determine orientation at actuation, and then position the actuator for next landing to follow the desired path.

Spring Flamingo – MIT (1996)

The Spring Flamingo is a planar bipedal walking robot. It must be constrained laterally, but is capable of maintaining balance and taking steps. This robot is still quasi-static, although beginning to use series elastic actuators for minimal force control [1].

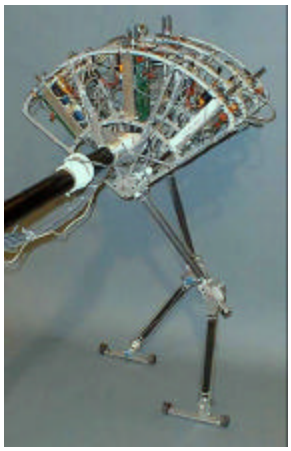


Figure 2-2: Spring Flamingo

The robot uses support-shift balance to remain upright and costs approximately $100,000.

Gyrobot – University of Illinois

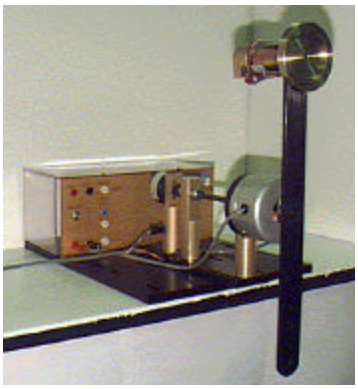


Figure 2-3: Gyrobot

Gyrobot is a single link pendulum with an inertial, motor-driven flywheel located at the free end of the link. The support pivot is constrained and not actuated. The motor controls the acceleration of the flywheel, which passively transfers energy in the flywheel to energy in the link [2].

A shaft encoder at the pivot determines the position and velocity of the pendulum link. A change in velocity (acceleration) of the flywheel causes a change in link angle, which enables the pendulum to swing up from the downward vertical stable equilibrium to the inverted unstable equilibrium and maintain balance at the unstable inverted position.

Gyrobot is an example of achieving balance exclusively by torque application

Balancing Act – Deakin University

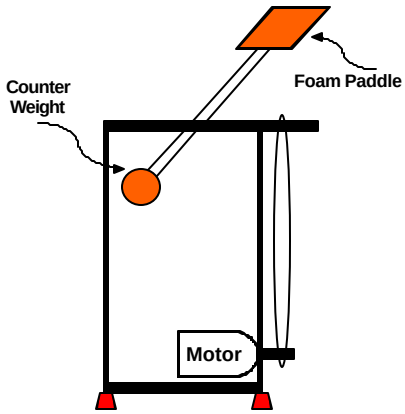


Figure 2-4: Balancing Act

Balancing Act is a two-legged platform that balances through torque application. A single inertial sensor provides position and angular velocity of the robot platform. A foam paddle exerts torque by pushing against the air to maintain balance.

A fuzzy logic controller uses the position and velocity information to drive the motor and keep the robot balanced. The result is a slow rocking motion as the robots oscillates around the upright position.

Since Balancing Act is a heavy, high inertia platform inconsistencies in quasi-static measurement data are masked.

CHAPTER 3
BALANCE IN THE GENERAL DYNAMIC CASE

Sensors for Balance

There are three types of measurement sensors that can be used for balance. These methods are protraction, pendulum sensing, and Dynamic Inclinometry.

Protraction is using a mechanical or optical sensor to determine the angle of the system with respect to the base. The actual sensor could be a potentiometer or an optical encoder. This method can generate accurate measurement information of position and velocity, but the orientation and position of the platform base must be known. If the platform base were tilted, this sensing method would no longer work.

Pendulum sensing could be accomplished by using a single inertial sensor, like an accelerometer. This method can yield effective results in the quasi-static case, but does not work in the general dynamic case.

Dynamic Inclinometry (DI) uses the linear disparity between two inertial sensors, placed a small distance apart, to determine angular velocity. The displacement angle can be computed by the static acceleration plus a rotational acceleration. This is a valid approach for the general dynamic case. Sensing by protraction and pendulum methods breakdown in the general dynamic case.

Balance by Protraction

Protraction sensing is typical for solving the inverted pendulum problem using support shift. The deflection angle can differentiated to provide velocity and acceleration

information. This sensing method is only valid for cases where orientation of base with respect to gravity is known (special lab case).

Balance by Pendulum

In this method, a pendulum, or single inertial sensor, is used to determine the deflection angle. This sensing method is only valid for quasi-static case, when the system is either stationary or moving very slowly.

When system is falling, pendulum measurement becomes zero. Pendulum measurement is even worse when additional forces act in addition to gravity.

To understand this phenomenon, we need a way to describe the total effect of the forces acting on the system. The variable n will be used to describe the relationship between the total acceleration of the system and the acceleration due to gravity.

$$n = \frac{\text{Tangential Acceleration of System } (a_t)}{\text{Tangential Acceleration due to Gravity } (a_{tg})}$$

Figure 3-1: Calculating the Total Effect of Forces on a System

For the purpose of this discussion, the system is an inverted pendulum with a single two-axis accelerometer mounted at the top of the link (Figure 3-2).

Consider the static case (n = 0). In this case, the system is not moving, and so the only force acting on the system is gravity. The position of the system can be determined by the vector sum of the radial and tangential accelerations due to gravity.

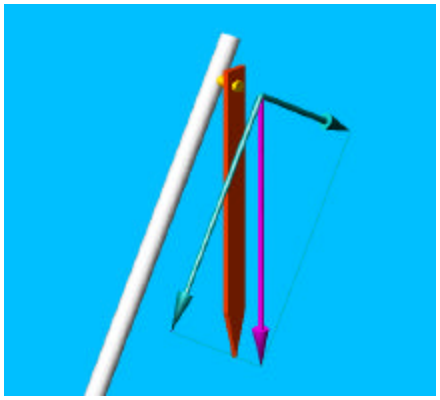


Figure 3-2: Pendulum Measurement when n = 0

The next case to consider is when n = 1. In this scenario, the tangential acceleration of the system is equal to the tangential acceleration due to gravity (Figure 3-3). The result is that the two tangential vectors cancel each other and the direction of the resulting vector is always along the shaft of the system. This is true regardless of the actual direction of gravity with respect to the sensor. This is where the pendulum method breaks down. There is no way to get the actual position of the system because the tangential acceleration at the sensor is zero.

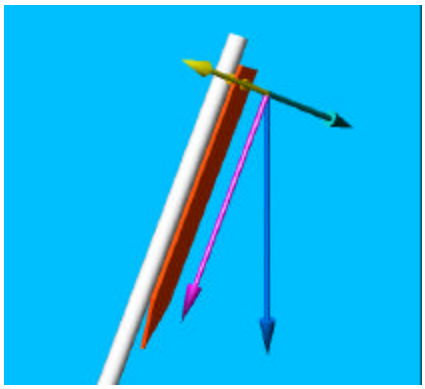


Figure 3-3: Pendulum Measurement when n = 1

The final case is when n is greater than 1. In this case, the tangential acceleration of the system is greater than the tangential acceleration due to gravity. The direction of the resulting vector is actually in the opposite direction from the actual gravity vector. In Figure 3-4, the tangential acceleration of the system is twice the tangential acceleration due to gravity.

Figure 3-4: Pendulum Measurement when n > 1

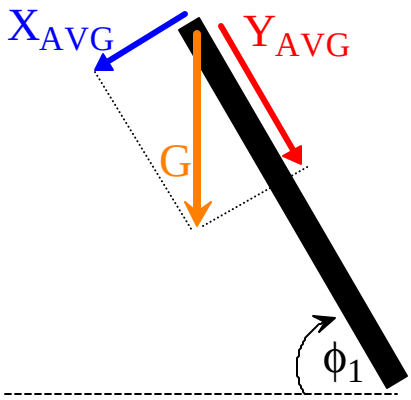
It is clear from this analysis that the pendulum method will not yield useful results in the general dynamic case. This is why the method of Dynamic Inclinometry was developed, to provide useful measurements for balance in the general dynamic case.

Balance by Dynamic Inclinometry

This method uses two dual-axis accelerometers, mounted a small distance apart. The architecture functionally models human balance system, in that it measures the same physical parameters. Linear acceleration can be determined from average readings from linear accelerometer array. Angular acceleration can be determined by disparity in the linear array. The dynamic position is derived from the combination of linear and angular accelerations.

Figure 3-5: Dynamic Inclinometer Test Platform

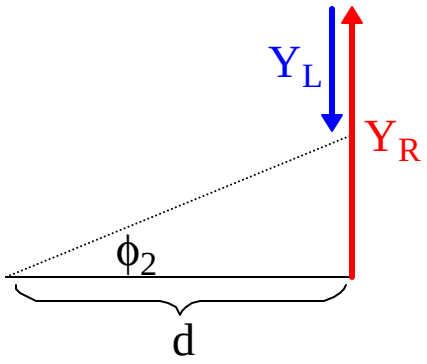
In the static case (n = 0), linear acceleration is measured using the average of both accelerometers (Figure 3-6). Angular acceleration is not present. The static position, ϕ_1, is calculated from these average accelerations.



$$\phi_1 = \text{TAN}^{-1}[X_{AVG} / Y_{AVG}]$$

Figure 3-6: Calculation of ϕ_1

In the dynamic case, The DI uses the linear acceleration disparity to determine angular acceleration. The dynamic position, ϕ_2, is calculated from this disparity.



$$\phi_2 = TAN^{-1}[(Y_R - Y_L) / d]$$

Figure 3-7: Calculation of ϕ_2

The actual position of the system (θ) is a function of both the static position (ϕ_1) and the dynamic position (ϕ_2), with a known or assumed moment of inertia. This method can yield the angle of the system in the general dynamic case

$$\theta = \Phi_1 + (\Phi_2 / I)$$

Figure 3-8: Calculation of Position (θ)

Implementation of Dynamic Inclinometer

There were many issues that had to be solved for the actual implementation of the Dynamic Inclinometer theory. The first hurdle was that software code needed to be very efficient to achieve the goal of a 30Hz-sampling rate. The target processor for this application was the Motorola 68HC11, since the application is primarily for embedded systems. The actual sampling rate on the final product was about 40Hz.

Another problem was that there were sensor errors from multiple sources. These sources included bias, scale factor, non-linearity, asymmetry, noise, drift, and

temperature. Eventually, a good method for calibration was realized to minimize effects from bias and scale factor errors.

The calibration fixture designed in AutoCAD to provide consistent measurements at known positions. Spreadsheet analysis was performed to determine error sources and magnitudes. These values were used in the software to reduce the errors seen by the system.

Figure 3-9: DI Calibration Fixture

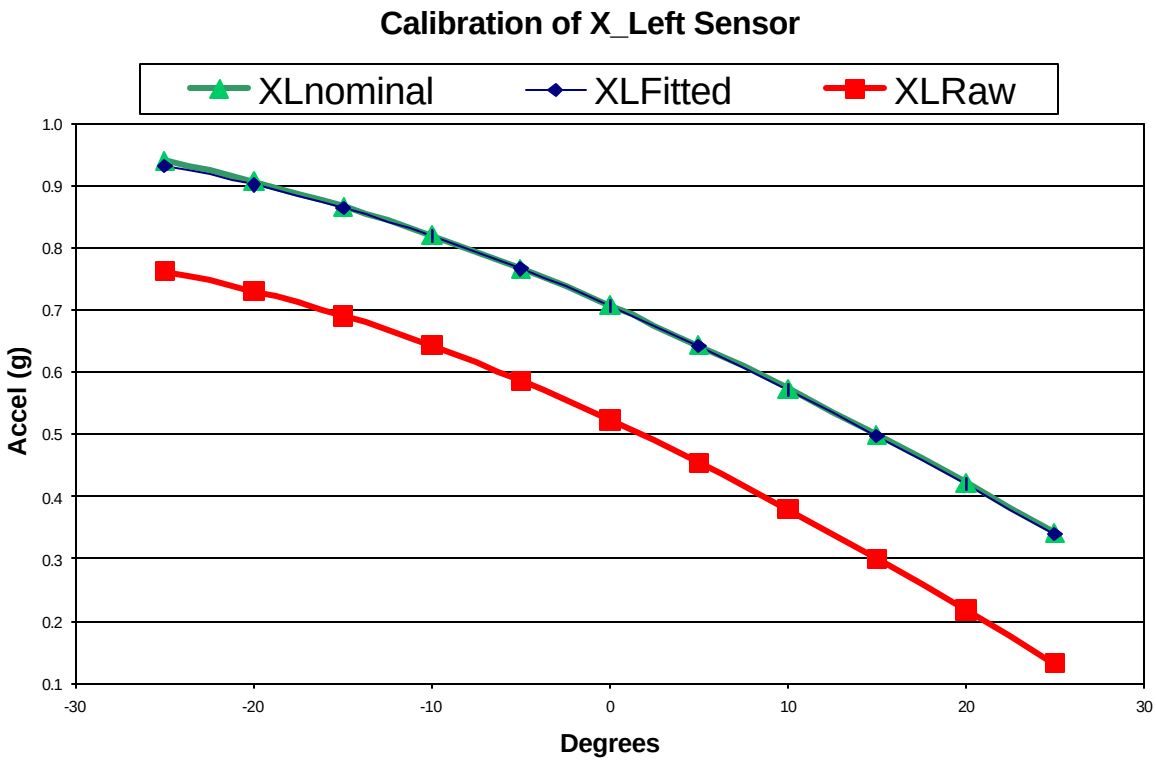


Figure 3-10: DI Calibration Results

Another source of difficulty in the implementation process was a sensor asymmetry that occurred around 1G. A plausible explanation for this occurrence could not be ascertained. The solution to this problem was to rotate the sensors 45 degrees to remove this part of the curve.

This fix made the small angle trigonometric approximation problematic. The solution was to use trig lookup table. The lookup table was at first very computationally intensive. This was reduced significantly with a binary search algorithm.

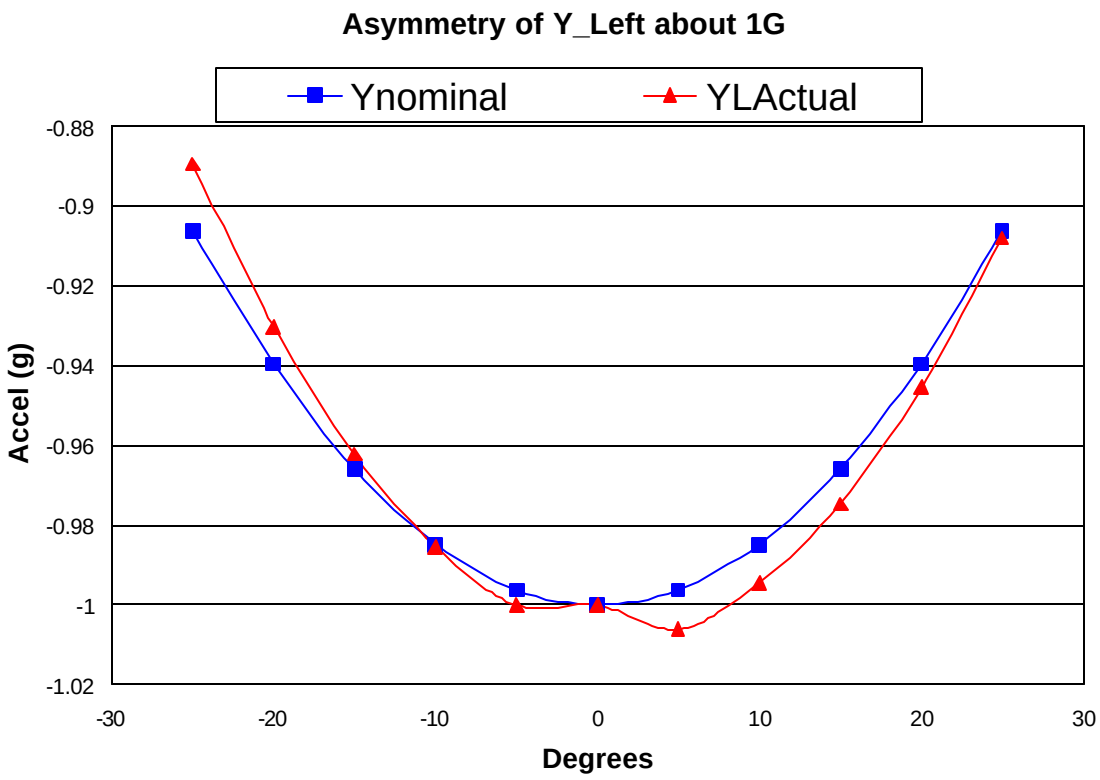


Figure 3-11: Asymmetry about 1G

A low-pass filter was used on the sensor output to limit high-frequency noise. Additional software filtering techniques were also required. Noise is especially problematic in disparity-based angular acceleration calculation. Drift is not a significant source of error. Multiple calibrations show very little change in average sensor data.

Variations due to temperature are ignored. Adjustable parameters facilitate calibration from all error sources.

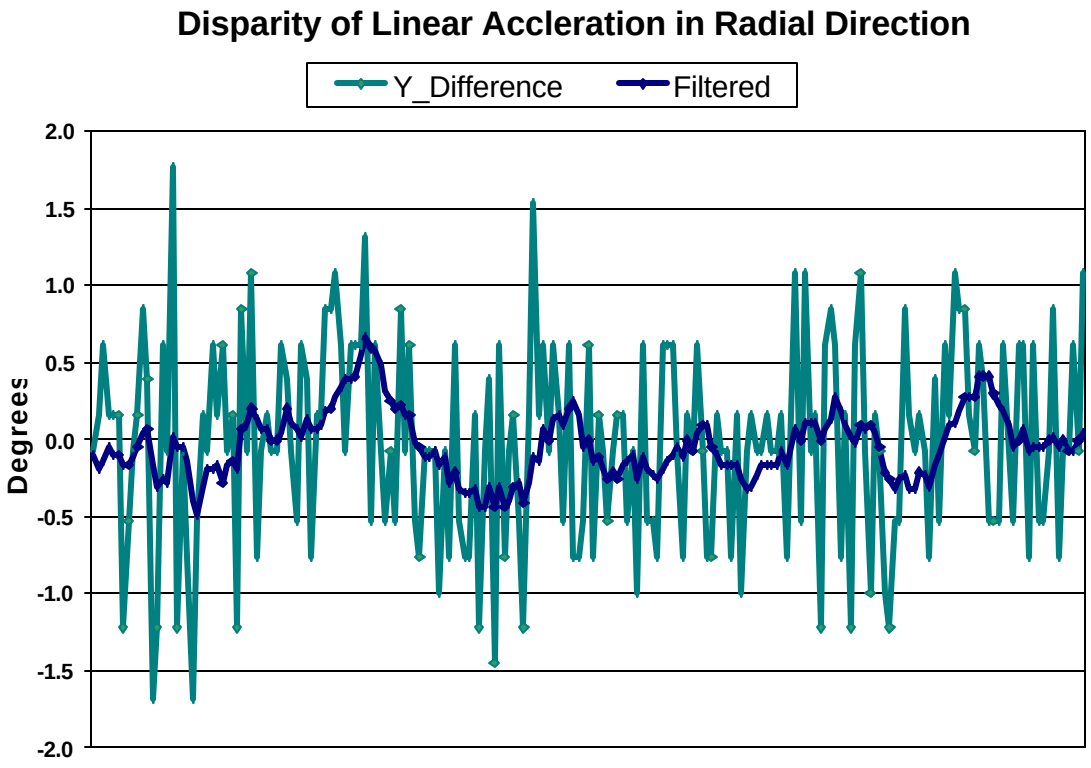


Figure 3-12: Linear Acceleration Disparity Noise

The results of the implementation were very promising. The calculation of linear acceleration (ϕ_1) is very stable and repeatable. The calculation of angular acceleration (ϕ_2) is still noisy, but much improved over previous attempts. This method shows promise with reduction of angular acceleration noise and available added resolution.

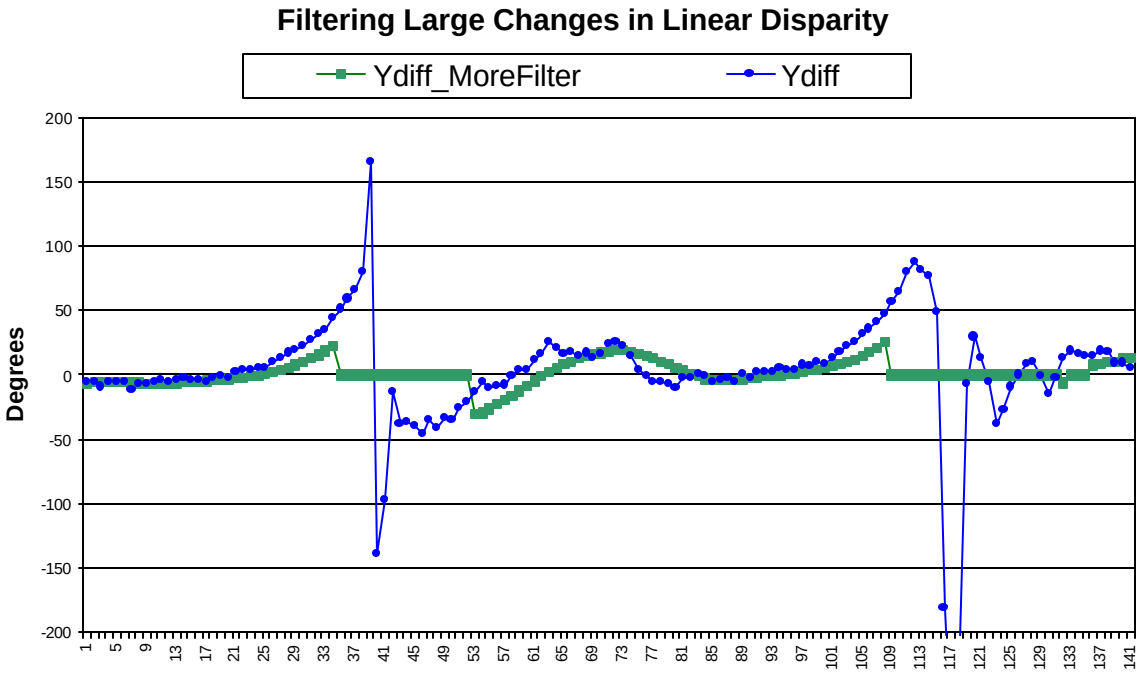


Figure 3-13: Linear Acceleration Disparity Noise

CHAPTER 4
AUTONOMOUS OSCILLATOR CONTROL

The Dynamic Inclinometer demonstrates that active balance using inertial sensors is possible, given the proper actuation system to perm torque balance. SwingBOT is a useful platform for implementing balance through oscillator control using the Dynamic Inclinometer. SwingBOT is a simple harmonic oscillator controller that exerts torque as a function of position and velocity. It is a concept illustration platform to show that torque balance is possible.

The pendulum model, both in its inverted form and in the non-inverted or swing form, is recognized here as a useful general basic model employable by analogy to all dynamic robotic tasks of similar order, including balance, and aspects of locomotion and manipulation.

The implementation embodied here constitutes a simple case, in which the inherent passive stiffness, which is elastic resistance to deflection, of the system is overcome in a limited region by actively exerted, augmentative, deflection-related torque along with velocity-dependent torque to achieve maximum efficiency.

The most efficient method of increasing the frequency of oscillation is by increasing the stiffness. This can be done through active deflection-related torque application to augment the inherent passive stiffness of the system. This is known as stiffening torque. Stiffening torque generally affects stiffness, and therefore frequency of oscillation [3].

The most efficient method of driving oscillations is through actively applied, velocity-related torque. This is known as driving torque. Driving torque changes the energy of the system, and so generally affects the amplitude of oscillation.

Both driving and stiffening torque can be applied as described above, or in the reverse sense. The deflection-related torque may be applied to counteract inherent stiffness, making the system behave as though it were less stiff. Similarly, the driving torque can be applied in a negative sense, to reduce the amplitude, instead of augmenting it.

In the theoretical approach, the system is configured with passive elastic deflection restraint such that the inherent stiffness is consistent with the desired oscillating frequency. No active energy expenditure is generally needed to maintain desired frequency in this approach, except as occasionally required to induce some small and temporary deviation from the inherent frequency.

To drive oscillations, a driving torque proportional to the measured angular swing velocity would be exerted. To drive the frequency of these oscillations, a stiffening torque proportional to the measured angular displacement would be exerted. To achieve the theoretical optimum for efficient oscillator control, both the driving torque and the stiffening torque are applied simultaneously. This type of control is known as the Proportional Derivative Control Law [4].

$$\tau = K_1(\theta - \theta_0) + K_2 * d\theta/dt$$

Figure 4-1: Proportional Derivative Control Law

The general theoretical approach requires the measurement of both position and velocity, and the calculation of respective torques proportional to each. The gain values, K_1 and K_2, are selected to achieve the desired function.

This means that if the swing is displaced a small amount from the vertical, torque will be applied to actively increase this deflection. This active effect compounds, since as the torque is applied the swing is displaced even further, giving rise to even greater applied torque, and so the swing is displaced further still.

Eventually a point is reached where the servo is overpowered by inherent elastic restraint (gravity acting to swing the pendulum back), or the servo runs into its limit of travel, and so can exert no additional torque.

At this point, the swing begins to return, swinging back through the zero point and continuing to the other side, where similar corresponding activity takes place. In this way the swing amplitude is continually increased until a balance is achieved between applied power and inherent elastic restraint. In this case, the inherent elastic restraint is gravity.

By adjusting the gains, or proportionality of both driving and stiffening torques, the amplitude of the resulting oscillations can be adjusted.

The servo cannot respond immediately to a change in signal, as there is some real time delay involved. Also, if the servomotor is turning rapidly, its own inertia must be overcome before it can stop and reverse direction. This same inertia is a factor as the motor accelerates from rest. The servo as presently configured is limited in its angular travel to slightly under ninety degrees in either direction from the center value. The commands to the servo must keep the desired servo position within these travel limits.

In this configuration, the predominance of these effects throughout the swing cycle appears favorable. The delays shift the phase of the applied torque with respect to the swing angle, and thereby may cause the applied action to have more of a driving effect.

This would be consistent with our expectations, since in oscillatory motion, the velocity is phase delayed by ninety degrees. Since driving is theoretically proportional to velocity, any phase shift or delay effect with respect to position contributes more to driving, in either a helpful or adverse sense depending on conditions. It may be possible to alter these delays to further modify the driving effect in useful ways.

At the extremes of swing amplitude, the servo travels from one limit to the other, and so is always configured to apply its full torque to greatest effect. This is another synergistic system quality.

The power of this method is in the ability to induce oscillations in analogous robotic systems. A swinging or waving arm or walking leg, flapping wing, waving antenna or flag, or any similar robotic structure whose chief function is to oscillate can be simply and effectively excited to do so by suitable adaptation of this control method.

Platform Mechanics

The SwingBOT apparatus can be best understood as functionally modeling the familiar example of a child swinging on a swing.

If the swing is at rest and the child in the swing remains motionless, no torque is induced on the system and the swing remains at rest.

Once the child starts to move the legs and upper body, a torque is induced on the system and the swing begins to oscillate. The child can increase the amplitude of these

oscillations by moving the legs and upper body so as to induce a torque on the system that leads the phase of the swing itself by 90°.

Conversely, the child can decrease the amplitude of the swing's oscillations by inducing a torque on the system that lags the phase of the swing by 90°.

Figure 4-2: SwingBOT Platform in AutoCAD

Platform Configuration

The simple non-inverted pendulum consists of the following:

A) A support pivot, (the top of the swing, from which the rest of it swings) being a horizontally oriented rod constrained to allow only rotation about its longitudinal axis,

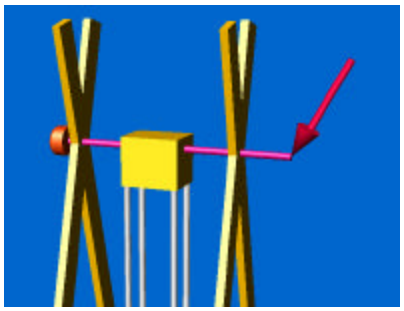


Figure 4-3: Support Pivot (A)

B) A first linear structural element perpendicularly attached to (hanging from) the support pivot and thereby constrained to allow only rotation in a vertical plane about the horizontal pivot axis (swinging), and

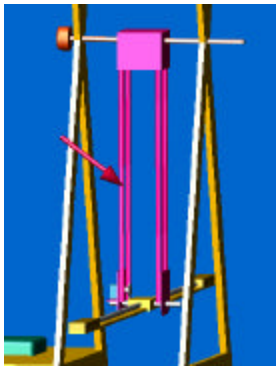


Figure 4-4: Swing Body (B)

C) A second linear structural element (crossbar), being a attached to the first linear structural element by constraint (crossbar pivot) allowing only rotation in a vertical plane (see-sawing) about a crossbar pivot axis oriented parallel to the support pivot axis (top) and displaced some distance (bottom of the swing). The crossbar is actuated by

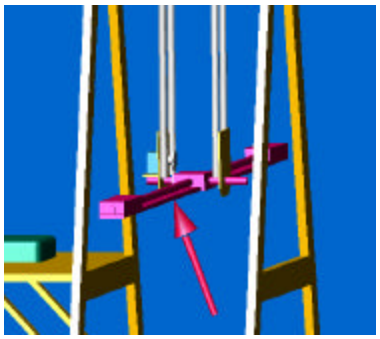


Figure 4-5: Crossbar (C)

D) A servomotor responding to the actions of

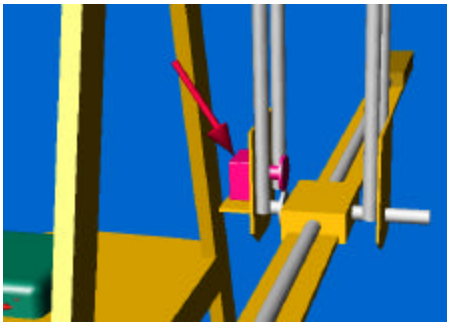


Figure 4-6: Servomotor (D)

E) A controller using inputs from

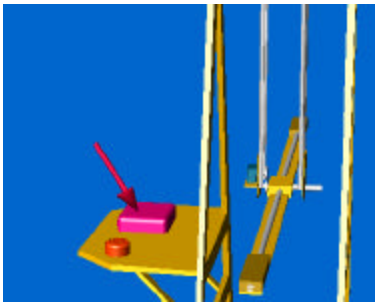


Figure 4-7: Controller (E)

F) An angular position sensor mounted suitably with respect to the support pivot axis and

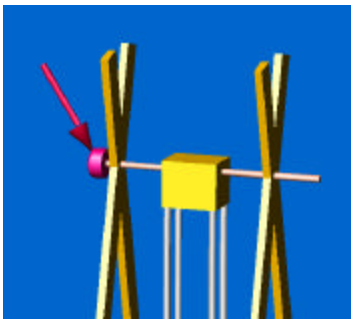


Figure 4-8: Angular Position Sensor (F)

G) Two gain selectors configured for operator input.

Figure 4-9: Gain Selects (G)

We refer again to the case of the case of a child swinging on a swing:

The support pivot (A) is the top of the swing-set, from which the rest of the swing is suspended.

B above is the hanging swing structure that hangs the chair from the support.

C above is the child seated in the chair, who exerts torque by holding on to B, while effecting dynamic body and leg motions through motor action (D), to excite the swing system into desired oscillations.

The controller (E) is the brain of the child, which senses parameters related to swing amplitude (F), and chooses appropriate desired gain (G). These values are

processed to calculate and effect the motor action (D), which drives the body and legs to exert torque, thereby driving the swing.

In the SwingBOT, the support pivot (A) is mounted atop a wooden dual A-frame structure resting on the floor for support. The Pivot axis is oriented horizontally at the top of the A-frame structure, extending between the peaks of the two A-frame trusses comprising the support structure.

Figure 4-10: Actual SwingBOT Platform

The pendulum and crossbar swing from this support axis. The controller, with its associated power supply, occupies a shelf partway up the outside of one of the A-frame trusses. Ribbon-wires connect power and signal to the servomotor and two potentiometers. One potentiometer is mounted along the pivot axis atop the A-frames to determine swing position angle, and the other receives manual input to implement desired gain.

Platform Electronics

Processor

The processor is a Motorola 68HC11. It is installed on a MTJPRO11 board from Mekatronix [5]. The 8-bit processor is operated in expanded mode at 2MHz with 32K external SRAM. The internal processor hardware used in this project includes the A/D Converter and Output Compare subsystems.

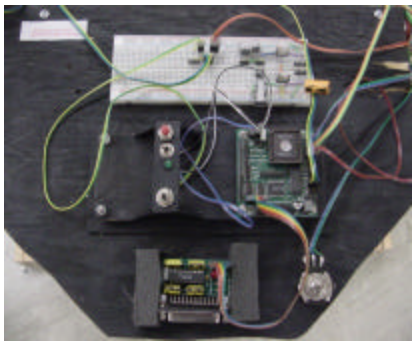


Figure 4-11: SwingBOT Electronics

The HC6811 contains an 8-bit Analog-to-Digital converter. There are eight individual inputs to the system, which takes 16ms per conversion. The Output Compare system allows timer-driven interrupts based on a 16-bit free-running counter. This hardware system allows pulse-width modulated (PWM) waveforms to be generated using interrupts.

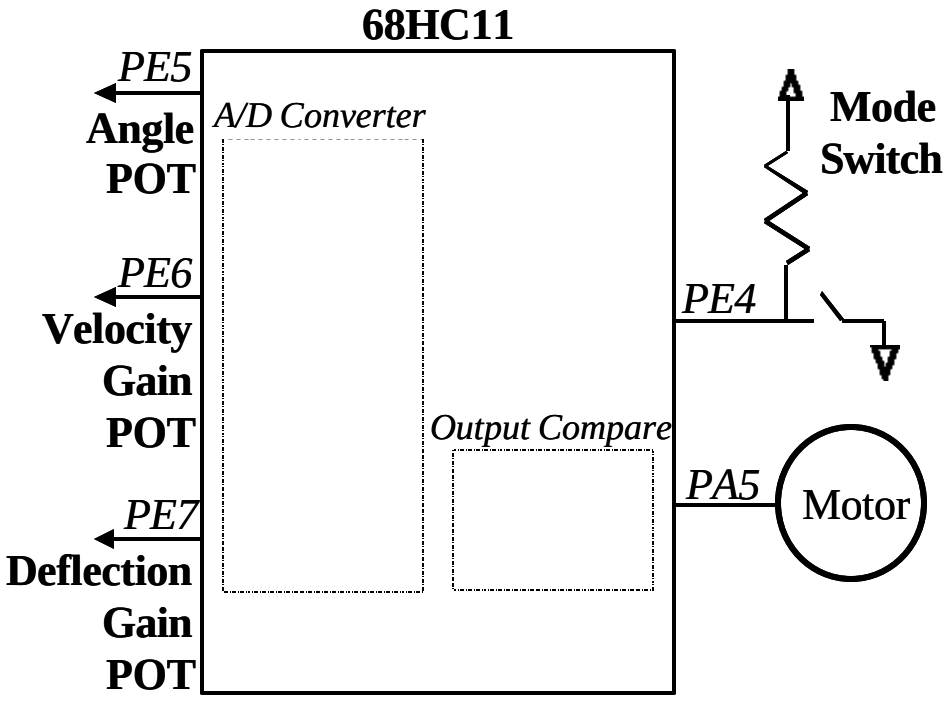


Figure 4-12: SwingBOT Electronics Block Diagram

The software for this project was developed using the ImageCraft C compiler for the 68HC11.

Sensors

The swing uses three sensors during operation.

A 1K potentiometer is used to measure the deflection angle of the swing from the resting position (Angle POT). The outer pins are connected to V_{cc} and GND to create a voltage divider output with the center contact. This voltage is read using the 8-bit A/D converter hardware inside the 68HC11. This results in an angle resolution of approximately 1degree.

Another 1K potentiometer is used to determine the open-loop gain of the motor output (Gain POT). This is a user input that can be varied during operation. A similar voltage divider circuit is read using the A/D converter to determine this desired gain.

A Single-Pole, Single-Throw switch is connected to one of the 68HC11 digital inputs. This is a user input to determine if the crossbar will be driven to increase or decrease the amplitude of the oscillations (Mode Switch).

Actuation

A Cirrus CS-80 servomotor controls the position of the crossbar. The rated stall torque is 343 oz/in. The position of the servo is determined by the positive width of a 50Hz PWM signal. The positive pulse width for the center position is 1550ms. The PWM signal is interrupt driven and uses the Output Compare hardware in the 68HC11. Due to the mechanical structure of the platform, the motor is restricted to 160 degrees of rotation.

Swing Operation

The torque output of the servomotor is directly proportional to the magnitude of the difference between the current and desired positions. The user input at the gain potentiometers (K_1 and K_2) scales this torque value accordingly. The software for this application commands the motor to a higher torque based on the swing velocity and the amount of deflection from the resting position of the swing. The direction that this torque is applied depends on the mode of operation.

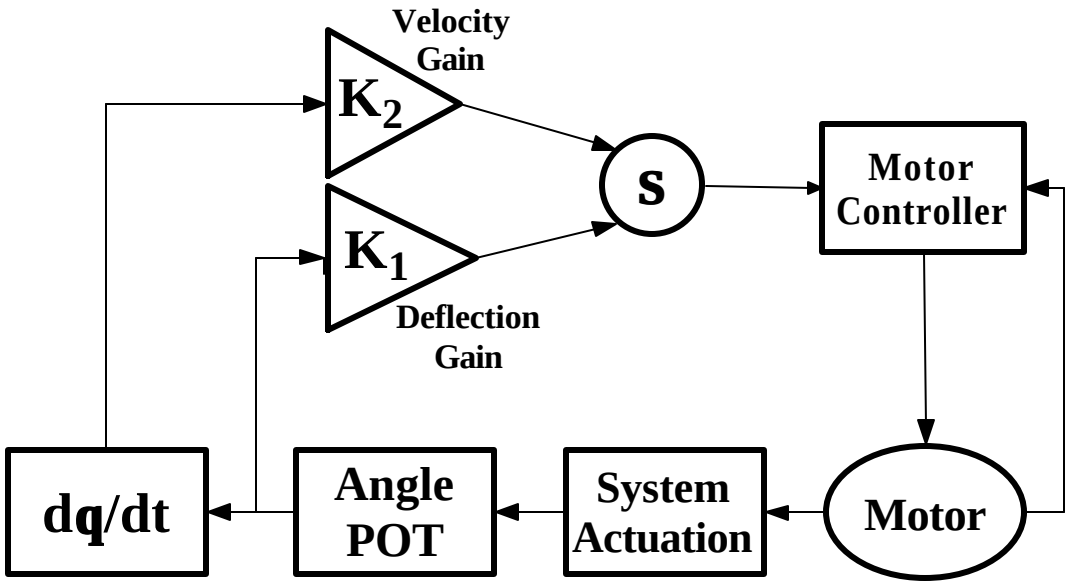


Figure 4-13: SwingBOT Control System

SwingBOT has two modes of operation. In the first mode, the stiffening torque is applied to counteract inherent stiffness, and driving torque is applied to increase the swing velocity. This applied torque, coupled with the driving forces of the system dynamics, increases the amplitude of the swing oscillations. At maximum gain, the swing can achieve amplitudes of about 30 degrees from the resting position.

In the second mode, the torque is applied in the opposite sense. The stiffness is increased and the driving torque decreases the swing velocity. The result is that the oscillation amplitude is reduced. This results in a system that controls the crossbar to maintain the resting position.

CHAPTER 5
CONCLUSION AND FUTURE APPLICATIONS

The conclusion of this research is that active balance through torque application, in the general dynamic case, is achievable using inertial sensors. The Dynamic Inclinometer can generate position & velocity data in the general dynamic case, and the SwingBOT demonstrates oscillator control using position and velocity data by varying only stiffness and driving

There are many possibly applications of this work. The sensing methods of the Dynamic Inclinometer and a torque balance platform could be combined to accomplish autonomous bike riding. Another application is that the SwingBOT oscillator control code can be used to drive oscillators in the general case. These oscillators can be used to drive any elastically linked oscillator.

Another feature of this work that will require more study is the addition of feedback capability to continue operation when moment of inertia changes.

The combination of balance by torque and support shift in the general dynamic, through dynamic inclinometry, is the most promising architecture for future robotic platforms that emulate living systems.

APPENDIX A
DYNAMIC INCLINOMETER SOFTWARE CODE

```
/*********************************************************************
 * Title        t_bot10l.c
 * Programmer   Nathan J. Nipper
 * Date         July 21, 2001
 * Version      10m
 *
 * Description
 *  This driver file provides the initialization and ISRs necessary
 *  to operate the Input Capture subsystem of the MC68HC11.  The
 *  original intent is to provide a method for reading PWM inputs.
 *  Accelerometers are connected to IC1 - IC4
 *
 *  Version 7: Adding servo code for PA5 (OC3).
 *  Version 7d: Change acceleration calculations to eliminate FDIV
 *              Added PSH/PUL to "asm" wait loops -> able to reset OK
 *  Version 8:Complete overhall of IC1_ISR to minimize sample time.
 *            ISR1 disables interrupt and allows main to continue
 *  Version 9:Make similar version 8 changes to TIC2 - TIC4
 *  Version 10: Add Theta calculations and simplify servo ISR
 *          10f: Add code for analog port & SCDR read
 *          10g: Develop menu system
 *          10h: Correct Accel SF Errors
 *          10i: Add good calibration numbers & fix equations
 *             10j: Add binary search
 *          10k: Add Degree Table
 *                    10l: Able to change InertiaInv on the fly
 *                              Added Software Filtering
 *          10m: Menu system for selecting run-time parameters (I, bias, PHI1on, PHI2on)
 *********************************************************************/


/*************************** Includes *********************************/
#include <stdio.h>
#include <hc11.h>
/**********************************************************************/


/*************************** Defines **********************************/
#define TIC4   *(unsigned short volatile *)(_IO_BASE + 0x1E)  /* Same as TOC5 */
#define DUMMY_ENTRY (void (*)(void))0xFFFE   /* Default for unused interrupts */
extern void _start(void)  ;               /*Initialize RESET vector in crt11.s */
#define PERIOD 40000                       /* 50Hz refresh for each servo */

#define IC1_Mask 0xFB
#define IC2_Mask 0xFD
#define IC3_Mask 0xFE
#define IC4_Mask 0xF7

#define CLEAR_BIT(x,y) x &= ~y;  // Clear all bits in x that are set in y
#define SET_BIT(x,y) x |= y;     // Set all bits in x that are set in y
#define CLEAR_FLAG(x,y) x &= y;  // Clearing Interrupt Flags

#define POT1 analog(7)

#define BIT0 0x01;
#define BIT1 0x02;
#define BIT2 0x04;
#define BIT3 0x08;
#define BIT4 0x10;
```

```
#define BIT5 0x20;
#define BIT6 0x40;
#define BIT7 0x80;

#define TotalInv1 0.00158158     // Sensor Calibration Parameters
#define TotalInv2 0.00158158
#define TotalInv3 0.00261835
#define TotalInv4 0.00261835

#define CalBias1 12.01616
#define CalBias2 -12.85670
#define CalBias3 11.79635
#define CalBias4 -12.48550

#define SF1 0.94948403
#define SF2 0.89607333
#define SF3 0.93150855
#define SF4 0.85716817

#define ZeroPos1 0.64297
#define ZeroPos2 0.70908
#define ZeroPos3 0.34202
#define ZeroPos4 0.70891
/*********************************************************************/

/************************** Prototypes ********************************/
#pragma interrupt_handler TIC1_isr
#pragma interrupt_handler TIC2_isr
#pragma interrupt_handler TIC3_isr
#pragma interrupt_handler TIC4_isr
#pragma interrupt_handler TOC3_isr
#pragma interrupt_handler TOV_isr

void init_SCI(void);
void init_InputCapture(void);
void init_servo(void);
void init_overflow(void);
void servo(int);    /* send desired Pulse Width(0 or 2000 to 4000) */
void init_analog(void);
int analog(int);
int get_char(void);
float binsearch(float);
void put_char(int);
int read_int(void);


/*********************************************************************/


/***************** T-BOT Sensor/Motor Connections *********************/
/*
       Four Acceleration Inputs -- This is the Order of ALL Data Arrays!

       1. X_Left  = Ideal_XL = Sensor_YL  => IC1 (PA2) = X_Left  = Orange (C)
       2. Y_Left  = Ideal_YL = -Sensor_XL => IC2 (PA1) = Y_Left  = Yellow (D)

       3. X_Right = Ideal_XR = Sensor_YR  => IC3 (PA0) = X_Right = Orange (C)
       4. Y_Right = Ideal_YR = -Sensor_XR => IC4 (PA3) = Y_Right = Yellow (D)

       OC3 (PA5) => Servo Motor PWM Drive Signal
*/
/*********************************************************************/


/***************************** Globals ********************************/
unsigned int IC1Count, HighTimeIC1, LowTimeIC1, LastIC1;
unsigned int IC2Count, HighTimeIC2, LowTimeIC2, LastIC2;
unsigned int IC3Count, HighTimeIC3, LowTimeIC3, LastIC3;
unsigned int IC4Count, HighTimeIC4, LowTimeIC4, LastIC4;

int DiffIC1, DiffIC2, DiffIC3, DiffIC4;
```

```
int Test, i, j, k, m;
int servo_pulse_width, Servo_P;

float AccelSum[4], AccelBias[5];
float AccelIC1, AccelIC2, AccelIC3, AccelIC4;
float PHI_1, PHI_2, Theta_P, Motor_P;
float X_Left, X_Right, Y_Left, Y_Right;
float InertiaInv;
float Degs1, Degs2, Degs2Last;
float Degs2Filter, Degs2FilterLast, Degs2FilterDiff;

float XL[300],YL[300],XR[300],YR[300],P1[300],P2[300],TH[300];
float Degs1Array[300], Degs2Array[300], ThetaArray[300];
float AvgArray[10];
float Degs1Bias, Degs2Bias;

unsigned int Test_time, Overflow_count;
unsigned int Timer[200], Overflow[200];
unsigned int ICounter[200], Reg2[200], chan[200], ic_number;
unsigned int AvgPtr, Degs1On, Degs2On;


/**########## Used to Determine motor Update ##########**/


char servo_signal_state, servomask, Mask, Regs[200], test, Menu2;

float InvTan[601] = {
0.267949192, 0.269820708, 0.271693987, 0.273569043, 0.275445891, 0.277324544,
0.279205017, 0.281087323, 0.282971477, 0.284857493, 0.286745386, 0.288635169,
0.290526857, 0.292420464, 0.294316005, 0.296213495, 0.298112948, 0.300014378, 0.3019178,
0.30382323, 0.305730681, 0.30764017, 0.30955171, 0.311465316, 0.313381004, 0.315298789,
0.317218686, 0.319140709, 0.321064876, 0.322991199, 0.324919696, 0.326850382,
0.328783271, 0.33071838, 0.332655724, 0.33459532, 0.336537181, 0.338481326, 0.340427769,
0.342376526, 0.344327613, 0.346281047, 0.348236844, 0.35019502, 0.35215559, 0.354118573,
0.356083983, 0.358051837, 0.360022153, 0.361994946, 0.363970234, 0.365948033, 0.36792836,
0.369911232, 0.371896666, 0.373884679, 0.375875289, 0.377868512, 0.379864365,
0.381862867, 0.383864035, 0.385867886, 0.387874438, 0.389883708, 0.391895715,
0.393910476, 0.395928009, 0.397948332, 0.399971464, 0.401997423, 0.404026226,
0.406057892, 0.408092441, 0.410129889, 0.412170257, 0.414213562, 0.416259824,
0.418309061, 0.420361293, 0.422416538, 0.424474816, 0.426536146, 0.428600547, 0.43066804,
0.432738642, 0.434812375, 0.436889258, 0.43896931, 0.441052552, 0.443139004, 0.445228685,
0.447321617, 0.44941782, 0.451517313, 0.453620118, 0.455726256, 0.457835746, 0.45994861,
0.46206487, 0.464184545, 0.466307658, 0.46843423, 0.470564281, 0.472697834, 0.474834911,
0.476975533, 0.479119721, 0.481267499, 0.483418888, 0.48557391, 0.487732589, 0.489894945,
0.492061002, 0.494230783, 0.496404311, 0.498581608, 0.500762698, 0.502947603,
0.505136348, 0.507328956, 0.509525449, 0.511725853, 0.513930192, 0.516138488,
0.518350766, 0.520567051, 0.522787366, 0.525011737, 0.527240189, 0.529472745,
0.531709432, 0.533950273, 0.536195295, 0.538444523, 0.540697983, 0.5429557, 0.545217699,
0.547484008, 0.549754652, 0.552029658, 0.554309051, 0.55659286, 0.55888111, 0.561173828,
0.563471042, 0.565772778, 0.568079065, 0.57038993, 0.5727054, 0.575025504, 0.577350269,
0.579679725, 0.582013898, 0.584352819, 0.586696515, 0.589045016, 0.591398351, 0.59375655,
0.59611964, 0.598487654, 0.600860619, 0.603238567, 0.605621527, 0.60800953, 0.610402607,
0.612800788, 0.615204105, 0.617612588, 0.620026269, 0.62244518, 0.624869352, 0.627298817,
0.629733609, 0.632173758, 0.634619298, 0.637070261, 0.639526681, 0.64198859, 0.644456023,
0.646929013, 0.649407593, 0.651891799, 0.654381664, 0.656877222, 0.65937851, 0.661885561,
0.664398412, 0.666917096, 0.669441652, 0.671972113, 0.674508517, 0.6770509,
0.679599298, 0.682153749, 0.68471429, 0.687280959, 0.689853792, 0.692432828, 0.695018106,
0.697609663, 0.700207538, 0.702811771, 0.705422401, 0.708039467, 0.710663009,
0.713293068, 0.715929683, 0.718572896, 0.721222746, 0.723879277, 0.726542528,
0.729212542, 0.731889362, 0.734573028, 0.737263585, 0.739961075, 0.742665542,
0.745377028, 0.748095579, 0.750821238, 0.75355405, 0.75629406, 0.759041313, 0.761795855,
0.764557731, 0.767326988, 0.770103672, 0.772887831, 0.775679511, 0.77847876, 0.781285627,
0.784100158, 0.786922404, 0.789752412, 0.792590233, 0.795435917, 0.798289512,
0.801151071, 0.804020643, 0.80689828, 0.809784033, 0.812677955, 0.815580099, 0.818490516,
0.821409261, 0.824336386, 0.827271946, 0.830215995, 0.833168589, 0.836129783,
0.839099631, 0.842078191, 0.845065519, 0.848061673, 0.851066709, 0.854080685,
0.857103661, 0.860135695, 0.863176845, 0.866227173, 0.869286738, 0.872355601,
0.875433823, 0.878521466, 0.881618592, 0.884725265, 0.887841546, 0.8909675, 0.894103191,
0.897248684, 0.900404044, 0.903569337, 0.90674463, 0.909929988, 0.91312548, 0.916331174,
0.919547138, 0.922773441, 0.926010153, 0.929257345, 0.932515086, 0.935783449,
```

```
0.939062506, 0.942352329, 0.945652991, 0.948964567, 0.95228713, 0.955620757, 0.958965522,
0.962321502, 0.965688775, 0.969067417, 0.972457508, 0.975859126, 0.979272351,
0.982697263, 0.986133944, 0.989582475, 0.993042939, 0.99651542, 1, 1.003496765,
1.0070058, 1.010527192, 1.014061027, 1.017607393, 1.021166379, 1.024738073, 1.028322566,
1.031919949, 1.035530314, 1.039153752, 1.042790358, 1.046440225, 1.050103449,
1.053780125, 1.057470351, 1.061174223, 1.06489184, 1.068623303, 1.07236871, 1.076128164,
1.079901766, 1.08368962, 1.08749183, 1.091308501, 1.095139739, 1.098985651, 1.102846344,
1.106721929, 1.110612515, 1.114518213, 1.118439135, 1.122375395, 1.126327107,
1.130294386, 1.134277349, 1.138276113, 1.142290798, 1.146321522, 1.150368407,
1.154431576, 1.15851115, 1.162607256, 1.166720019, 1.170849566, 1.174996025, 1.179159526,
1.183340199, 1.187538177, 1.191753593, 1.195986581, 1.200237278, 1.204505822, 1.20879235,
1.213097004, 1.217419925, 1.221761255, 1.226121139, 1.230499724, 1.234897157,
1.239313585, 1.243749161, 1.248204036, 1.252678364, 1.257172299, 1.261685998,
1.266219621, 1.270773326, 1.275347275, 1.279941632, 1.284556562, 1.289192232, 1.29384881,
1.298526466, 1.303225373, 1.307945704, 1.312687636, 1.317451347, 1.322237014,
1.327044822, 1.331874952, 1.33672759, 1.341602923, 1.346501142, 1.351422438, 1.356367004,
1.361335036, 1.366326733, 1.371342293, 1.37638192, 1.381445819, 1.386534195, 1.391647258,
1.39678522, 1.401948294, 1.407136697, 1.412350648, 1.417590366, 1.422856077, 1.428148007,
1.433466383, 1.438811438, 1.444183406, 1.449582523, 1.455009029, 1.460463166,
1.465945179, 1.471455316, 1.476993828, 1.482560969, 1.488156995, 1.493782166,
1.499436745, 1.505120998, 1.510835194, 1.516579605, 1.522354507, 1.528160179,
1.533996902, 1.539864964, 1.545764652, 1.551696259, 1.557660082, 1.56365642, 1.569685577,
1.57574786, 1.58184358, 1.587973051, 1.594136594, 1.600334529, 1.606567185, 1.612834891,
1.619137983, 1.625476801, 1.631851687, 1.63826299, 1.644711062, 1.651196259, 1.657718944,
1.664279482, 1.670878245, 1.677515606, 1.684191948, 1.690907656, 1.697663119,
1.704458734, 1.711294902, 1.718172028, 1.725090524, 1.732050808, 1.7390533, 1.746098431,
1.753186632, 1.760318346, 1.767494016, 1.774714096, 1.781979042, 1.789289319,
1.796645398, 1.804047755, 1.811496875, 1.818993247, 1.826537369, 1.834129745,
1.841770886, 1.84946131, 1.857201543, 1.864992118, 1.872833576, 1.880726465, 1.888671342,
1.896668769, 1.904719321, 1.912823577, 1.920982127, 1.929195568, 1.937464506,
1.945789558, 1.954171346, 1.962610506, 1.971107679, 1.979663518, 1.988278686,
1.996953856, 2.005689708, 2.014486937, 2.023346245, 2.032268347, 2.041253967,
2.050303842, 2.059418718, 2.068599355, 2.077846524, 2.087161007, 2.096543599,
2.105995109, 2.115516356, 2.125108173, 2.134771408, 2.144506921, 2.154315584,
2.164198288, 2.174155933, 2.184189436, 2.194299731, 2.204487764, 2.214754498,
2.225100911, 2.235528, 2.246036774, 2.256628263, 2.267303512, 2.278063586, 2.288909564,
2.299842547, 2.310863654, 2.321974022, 2.333174808, 2.34446719, 2.355852366, 2.367331554,
2.378905995, 2.39057695, 2.402345703, 2.414213562, 2.426181858, 2.438251943, 2.450425198,
2.462703025, 2.475086853, 2.487578139, 2.500178362, 2.512889034, 2.525711689,
2.538647896,
2.551699247, 2.564867368, 2.578153915, 2.591560574, 2.605089065, 2.618741138, 2.63251858,
2.64642321, 2.660456884, 2.674621494, 2.688918967, 2.703351271, 2.71792041, 2.732628431,
2.747477419, 2.762469503, 2.777606854, 2.792891687, 2.808326261, 2.823912886,
2.839653913, 2.855551747, 2.871608841, 2.887827699, 2.904210878, 2.920760989, 2.9374807,
2.954372735, 2.971439875, 2.988684963, 3.006110903, 3.023720665, 3.041517279,
3.059503847, 3.077683537, 3.096059589, 3.114635316, 3.133414104, 3.152399419,
3.171594802, 3.19100388, 3.210630362, 3.230478041, 3.250550801, 3.270852618, 3.291387561,
3.312159796, 3.333173587, 3.354433305, 3.375943423, 3.397708524, 3.419733305,
3.442022577, 3.464581272, 3.487414444, 3.510527275, 3.533925079, 3.557613303,
3.581597536, 3.605883509, 3.630477104, 3.655384355, 3.680611456, 3.706164764,
3.732050808};

float DegVal[601] = {
                                                  15, 15.1,
15.2, 15.3, 15.4, 15.5, 15.6, 15.7, 15.8, 15.9, 16, 16.1,
16.2, 16.3, 16.4, 16.5, 16.6, 16.7, 16.8, 16.9, 17, 17.1,
17.2, 17.3, 17.4, 17.5, 17.6, 17.7, 17.8, 17.9, 18, 18.1,
18.2, 18.3, 18.4, 18.5, 18.6, 18.7, 18.8, 18.9, 19, 19.1,
19.2, 19.3, 19.4, 19.5, 19.6, 19.7, 19.8, 19.9, 20, 20.1,
20.2, 20.3, 20.4, 20.5, 20.6, 20.7, 20.8, 20.9, 21, 21.1,
21.2, 21.3, 21.4, 21.5, 21.6, 21.7, 21.8, 21.9, 22, 22.1,
22.2, 22.3, 22.4, 22.5, 22.6, 22.7, 22.8, 22.9, 23, 23.1,
23.2, 23.3, 23.4, 23.5, 23.6, 23.7, 23.8, 23.9, 24, 24.1,
24.2, 24.3, 24.4, 24.5, 24.6, 24.7, 24.8, 24.9, 25, 25.1,
25.2, 25.3, 25.4, 25.5, 25.6, 25.7, 25.8, 25.9, 26, 26.1,
26.2, 26.3, 26.4, 26.5, 26.6, 26.7, 26.8, 26.9, 27, 27.1,
27.2, 27.3, 27.4, 27.5, 27.6, 27.7, 27.8, 27.9, 28, 28.1,
28.2, 28.3, 28.4, 28.5, 28.6, 28.7, 28.8, 28.9, 29, 29.1,
29.2, 29.3, 29.4, 29.5, 29.6, 29.7, 29.8, 29.9, 30, 30.1,
30.2, 30.3, 30.4, 30.5, 30.6, 30.7, 30.8, 30.9, 31, 31.1,
31.2, 31.3, 31.4, 31.5, 31.6, 31.7, 31.8, 31.9, 32, 32.1,
```

```
32.2, 32.3, 32.4, 32.5, 32.6, 32.7, 32.8, 32.9, 33, 33.1,
33.2, 33.3, 33.4, 33.5, 33.6, 33.7, 33.8, 33.9, 34, 34.1,
34.2, 34.3, 34.4, 34.5, 34.6, 34.7, 34.8, 34.9, 35, 35.1,
35.2, 35.3, 35.4, 35.5, 35.6, 35.7, 35.8, 35.9, 36, 36.1,
36.2, 36.3, 36.4, 36.5, 36.6, 36.7, 36.8, 36.9, 37, 37.1,
37.2, 37.3, 37.4, 37.5, 37.6, 37.7, 37.8, 37.9, 38, 38.1,
38.2, 38.3, 38.4, 38.5, 38.6, 38.7, 38.8, 38.9, 39, 39.1,
39.2, 39.3, 39.4, 39.5, 39.6, 39.7, 39.8, 39.9, 40, 40.1,
40.2, 40.3, 40.4, 40.5, 40.6, 40.7, 40.8, 40.9, 41, 41.1,
41.2, 41.3, 41.4, 41.5, 41.6, 41.7, 41.8, 41.9, 42, 42.1,
42.2, 42.3, 42.4, 42.5, 42.6, 42.7, 42.8, 42.9, 43, 43.1,
43.2, 43.3, 43.4, 43.5, 43.6, 43.7, 43.8, 43.9, 44, 44.1,
44.2, 44.3, 44.4, 44.5, 44.6, 44.7, 44.8, 44.9, 45, 45.1,
45.2, 45.3, 45.4, 45.5, 45.6, 45.7, 45.8, 45.9, 46, 46.1,
46.2, 46.3, 46.4, 46.5, 46.6, 46.7, 46.8, 46.9, 47, 47.1,
47.2, 47.3, 47.4, 47.5, 47.6, 47.7, 47.8, 47.9, 48, 48.1,
48.2, 48.3, 48.4, 48.5, 48.6, 48.7, 48.8, 48.9, 49, 49.1,
49.2, 49.3, 49.4, 49.5, 49.6, 49.7, 49.8, 49.9, 50, 50.1,
50.2, 50.3, 50.4, 50.5, 50.6, 50.7, 50.8, 50.9, 51, 51.1,
51.2, 51.3, 51.4, 51.5, 51.6, 51.7, 51.8, 51.9, 52, 52.1,
52.2, 52.3, 52.4, 52.5, 52.6, 52.7, 52.8, 52.9, 53, 53.1,
53.2, 53.3, 53.4, 53.5, 53.6, 53.7, 53.8, 53.9, 54, 54.1,
54.2, 54.3, 54.4, 54.5, 54.6, 54.7, 54.8, 54.9, 55, 55.1,
55.2, 55.3, 55.4, 55.5, 55.6, 55.7, 55.8, 55.9, 56, 56.1,
56.2, 56.3, 56.4, 56.5, 56.6, 56.7, 56.8, 56.9, 57, 57.1,
57.2, 57.3, 57.4, 57.5, 57.6, 57.7, 57.8, 57.9, 58, 58.1,
58.2, 58.3, 58.4, 58.5, 58.6, 58.7, 58.8, 58.9, 59, 59.1,
59.2, 59.3, 59.4, 59.5, 59.6, 59.7, 59.8, 59.9, 60, 60.1,
60.2, 60.3, 60.4, 60.5, 60.6, 60.7, 60.8, 60.9, 61, 61.1,
61.2, 61.3, 61.4, 61.5, 61.6, 61.7, 61.8, 61.9, 62, 62.1,
62.2, 62.3, 62.4, 62.5, 62.6, 62.7, 62.8, 62.9, 63, 63.1,
63.2, 63.3, 63.4, 63.5, 63.6, 63.7, 63.8, 63.9, 64, 64.1,
64.2, 64.3, 64.4, 64.5, 64.6, 64.7, 64.8, 64.9, 65, 65.1,
65.2, 65.3, 65.4, 65.5, 65.6, 65.7, 65.8, 65.9, 66, 66.1,
66.2, 66.3, 66.4, 66.5, 66.6, 66.7, 66.8, 66.9, 67, 67.1,
67.2, 67.3, 67.4, 67.5, 67.6, 67.7, 67.8, 67.9, 68, 68.1,
68.2, 68.3, 68.4, 68.5, 68.6, 68.7, 68.8, 68.9, 69, 69.1,
69.2, 69.3, 69.4, 69.5, 69.6, 69.7, 69.8, 69.9, 70, 70.1,
70.2, 70.3, 70.4, 70.5, 70.6, 70.7, 70.8, 70.9, 71, 71.1,
71.2, 71.3, 71.4, 71.5, 71.6, 71.7, 71.8, 71.9, 72, 72.1,
72.2, 72.3, 72.4, 72.5, 72.6, 72.7, 72.8, 72.9, 73, 73.1,
73.2, 73.3, 73.4, 73.5, 73.6, 73.7, 73.8, 73.9, 74, 74.1,
74.2, 74.3, 74.4, 74.5, 74.6, 74.7, 74.8, 74.9, 75
};

/*********************************************************************/


void main(void)
{ /*********************************************************************/
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* Clear screen */
  char place[]= "\x1b[1;1H";          /* Home cursor  */

  init_SCI();
  init_analog();
  init_InputCapture();
  init_overflow();
  init_servo();

  /* Initialize Run-Time Parameters */
  InertiaInv = 700;
  Degs1On = 1;
  Degs2On = 1;
  Degs1Bias = 0;
  Degs2Bias = 0;

while(1)
{
  /* MAIN MENU */
  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/
```

```
  printf("\nRUN-TIME PARAMETERS\n");
  printf("\tInertiaInv..................... %.1f\n",InertiaInv);
  printf("\tPHI_1_On....................... %u\n",Degs1On);
  printf("\tPHI_1 Bias..................... %.1f\n",Degs1Bias);
  printf("\tPHI_2_On....................... %u\n",Degs2On);
  printf("\tPHI_1 Bias..................... %.1f\n",Degs2Bias);

  printf("\n\nMAIN MENU\n"
         "\t1. Run the T_BOT\n"
         "\t2. Examine Sensor Outputs & Calculations\n"
         "\t3. Change Run-Time Parameters\n"
         "\t4. Display Motor Update Table (after running TBOT)\n\n"
         " Enter a Number: ");

  test = getchar();

 if(test == '1')
  {
   printf("%s", clear);  /*clear screen*/
   printf("%s", place);  /*home cursor*/
   printf("\n\n\tT-BOT is currently Operating!!\n");
   while(SCDR != 0x1b)
   {
      SET_BIT(TMSK1,0x0F);            /* Only all 4 IC Interrupts.... */
      while ((TMSK1 & 0x0F) != 0);    /* Wait until all IC_Ints are disabled */

          AccelIC1 = CalBias1 - (HighTimeIC1 * TotalInv1);
          X_Left = (AccelIC1 * SF1) + ZeroPos1;

          AccelIC2 = CalBias2 + (HighTimeIC2 * TotalInv2);
          Y_Left = (AccelIC2 * SF2) + ZeroPos2;

          AccelIC3 = CalBias3 - (HighTimeIC3 * TotalInv3);
          X_Right = (AccelIC3 * SF3) + ZeroPos3;

          AccelIC4 = CalBias4 + (HighTimeIC4 * TotalInv4);
          Y_Right = (AccelIC4 * SF4) + ZeroPos4;

      /*
      VARIABLE DEFINITION
          PHI_1 = Direction of Linear Acceleration
          PHI_2 = Angular Acceleration
          Theta = PHI_1 + (PHI_2 * InertiaInv)
      */

      PHI_1 = (X_Left + X_Right) / (Y_Left + Y_Right);  /* This is the average of X & Y
*/


      PHI_2 = (Y_Right - Y_Left) + 0.01568;  /* Bias Error Offset */
      PHI_2 = PHI_2 * InertiaInv;

      /* Software filtering */

          if(AvgPtr > 9) AvgPtr = 0;

      AvgArray[AvgPtr] = PHI_2;
      AvgPtr++;

          Degs2Last = Degs2;              /* Save the Previous Value of Degs2 &
Degs2Filter */
          Degs2FilterLast = Degs2Filter;

          Degs2 = 0;
          for(k=0; k < 10; k++)
          {
               Degs2 = Degs2 + AvgArray[k];
          }
      Degs2Filter = Degs2 * 0.1;             /* Averaging FIlter (divide by 10) */
```

```
      Degs2FilterDiff = Degs2Filter - Degs2FilterLast;

      Degs2 = Degs2Filter;
      if(Degs2 > 30)  Degs2 = 30;      /* High & Low limits for Degs2 */
      if(Degs2 < -30) Degs2 = -30;

      if(Degs2FilterDiff > 4)  Degs2 = 0;
      if(Degs2FilterDiff < -4) Degs2 = 0;

      if(Degs2Last == 0) Degs2 = 0;    /* Filter one last time */

      Degs2 = Degs2 - Degs2Bias;      /* Correct for Bias Error */
         Degs1 = binsearch(PHI_1) - Degs1Bias;  /* Correct for Bias Error */

      if(Degs2 < -2)  Degs2 = Degs2 * 2;     /* Falls slower to right ???? */

      if(Degs2On == 0)  Degs2 = 0;
      if(Degs1On == 0)  Degs1 = 45;

         Servo_P = (Degs1 + Degs2) * -19 + 3755;
      if(Servo_P > 3660)  Servo_P = 3660;
      if(Servo_P < 2140)  Servo_P = 2140;
      servo(Servo_P);

    }  /* END of while(SCDR) loop */
  }  /* END if if() loop */

  if(test == '2')
  {
    printf("%s", clear);  /*clear screen*/
    printf("%s", place);  /*home cursor*/
    printf("\nData Collection Phase\n");

   for(k=0;k<300;k++)
    {
      SET_BIT(TMSK1,0x0F);            /* Only all 4 IC Interrupts.... */
      while ((TMSK1 & 0x0F) != 0);    /* Wait until all IC_Ints are disabled */

         AccelIC1 = CalBias1 - (HighTimeIC1 * TotalInv1);
         X_Left = (AccelIC1 * SF1) + ZeroPos1;

         AccelIC2 = CalBias2 + (HighTimeIC2 * TotalInv2);
         Y_Left = (AccelIC2 * SF2) + ZeroPos2;

         AccelIC3 = CalBias3 - (HighTimeIC3 * TotalInv3);
         X_Right = (AccelIC3 * SF3) + ZeroPos3;

         AccelIC4 = CalBias4 + (HighTimeIC4 * TotalInv4);
         Y_Right = (AccelIC4 * SF4) + ZeroPos4;

         XL[k] = X_Left;
         YL[k] = Y_Left;
         XR[k] = X_Right;
         YR[k] = Y_Right;

      PHI_1 = (X_Left + X_Right) / (Y_Left + Y_Right);

      PHI_2 = (Y_Right - Y_Left) * InertiaInv;

         Degs1Array[k] = binsearch(PHI_1);
      Degs2Array[k] = PHI_2;
      ThetaArray[k] = Degs1Array[k] + Degs2Array[k];

   }  /* END of for() loop */
    printf("\tXL \tYL \tXR \tYR \tPhi_1 \tPhi_2 \tTheta \n");
  for(k=0;k<300;k++)
   {
    printf("\t%.3f \t%.3f \t%.3f \t%.3f \t%.1f \t%.1f \t%.1f
\n",XL[k],YL[k],XR[k],YR[k],Degs1Array[k],
    Degs2Array[k], ThetaArray[k]);
```

```
   }
   printf("Done with Testing!!");
   test = getchar();
  } /* END of if() loop */

  if(test == '3')
  {
       Menu2 = '0';
       while(Menu2 != '6')
       {
      printf("%s", clear);  /*clear screen*/
      printf("%s", place);  /*home cursor*/

      printf("\nPARAMETER SELECTION MENU\n");
      printf("\t1. InertiaInv..................... %.1f\n",InertiaInv);
      printf("\t2. PHI_1_On....................... %u\n",Degs1On);
      printf("\t3. PHI_1 Bias..................... %.1f\n",Degs1Bias);
      printf("\t4. PHI_2_On....................... %u\n",Degs2On);
      printf("\t5. PHI_1 Bias..................... %.1f\n",Degs2Bias);
      printf("\t6. Return to Main Menu\n\n");
      printf(" Enter a Number: ");
      Menu2 = getchar();

      if(Menu2 == '1')
      {
        printf("\n\nEnter new value for InertiaInv: ");
        InertiaInv = read_int();
      }

      if(Menu2 == '2')
      {
        printf("\n\nEnter new value for PHI_1_On: ");
        Degs1On = read_int();
      }

      if(Menu2 == '3')
      {
        printf("\n\nEnter new value for PHI_1 Bias (Enter 10x Desired Angle): ");
        Degs1Bias = read_int();
        Degs1Bias = Degs1Bias * 0.1; /* Unable to enter floating point directly */
      }

      if(Menu2 == '4')
      {
        printf("\n\nEnter new value for PHI_2_On: ");
        Degs2On = read_int();
      }

      if(Menu2 == '5')
      {
        printf("\n\nEnter new value for PHI_2 Bias (Enter 10x Desired Angle): ");
        Degs2Bias = read_int();
        Degs2Bias = Degs2Bias * 0.1; /* Unable to enter floating point directly */
      }
    } /* End of Menu2 while loop */
  }

  if(test == '4')
  {
      printf("%s", clear);  /*clear screen*/
      printf("%s", place);  /*home cursor*/
  }

 }  /* END of while(1) loop */

} /*********************************************************************/


void init_SCI(void)   /* Set SCI for 9600, 8-n-1, TE, RE */
{ /*********************************************************************/
  CLEAR_BIT(SPCR,0x20);
```

```
  SCCR1 = 0X00;
  BAUD = 0xb0;  /* 0xbO is 9600 0x35 is 300 baud */
  SCCR2 = 0x0C;
} /*********************************************************************/


int get_char(void)  /* Waits for character at serial port & return ASCII value */
{ /*********************************************************************/

  while ((SCSR & 0x20) == 0);   /* Wait for character */
  return(SCDR & 0x7F);          /* Return 7-bit ASCII by stripping bit 8*/
} /*********************************************************************/


void init_InputCapture(void)   /* TIC1 - TIC4 Initialization */
{ /*********************************************************************/
 INTR_OFF();

  SET_BIT(PACTL, 0x04);    /* Set I4/O5 bit to enable IC4 */
  TCTL2 = 0xAA;            /* Set all 4 IC Chhannels to Falling edge trigger */

                           /* Enable & Disable IC1-IC4 interrupts in Main */
  CLEAR_FLAG(TFLG1,0x0F);  /* Clear IC1-IC4 interrupt flags */

  IC1Count = 0;
  IC2Count = 0;
  IC3Count = 0;
  IC4Count = 0;

  INTR_ON();
} /*********************************************************************/


void init_servo(void)  /* Initialize variables for servo on PA5 (OC3) */
{ /*************************************1111*********************************/
  INTR_OFF();

  servo_pulse_width = 0;      /* Motor starts turned off */
  TCTL1 = 0x40;               /* Clear OC3 on first interrupt */

  INTR_ON();
} /*********************************************************************/


void init_overflow(void)  /* Initialize variables Timer Overflow */
{ /*********************************************************************/
  INTR_OFF();

  Overflow_count = 0;
  CLEAR_FLAG(TFLG2,0x80);  /* Clear Timer Overflow interrupt flags */
  SET_BIT(TMSK2,0x80);     /* Enable TOV interrupt */

  INTR_ON();
} /*********************************************************************/


void init_analog(void)  /* Power up A/D */
{ /*********************************************************************/
  SET_BIT(OPTION,0x80);
} /*********************************************************************/


int analog(int port)   /* Returns the analog value read from A/D PORTE */
{ /*********************************************************************/
  ADCTL=port;          /* Address the selected channel */
  while((ADCTL & 0x80) == 0); /* Wait for A/D to finish */
  return(ADR1);               /* Return analog value */
} /*********************************************************************/


void servo(int newposition)
```

```
 /* Servo on PA5 <- position defined by newposition     *
  * This subroutine takes the new position in degrees   *
  * and calculates the pulse HighTime in E-Clocks.  PWM *
  * signal is generated by ISR.                         */
{ /*******************************************************************/

  if (newposition != 0)
  {
    servo_pulse_width = newposition;
    SET_BIT(TMSK1,0x20);          /* Enable OC3 interrupt */
  }

  else CLEAR_BIT(TMSK1,0x20);     /* Disable OC3 interrupt */

} /*******************************************************************/


void TIC1_isr()         /* Increment IC1Count */
{ /*******************************************************************/
  CLEAR_FLAG(TFLG1,0x04);       /* Clear TIC1 interrupt Flag*/
  ic_number = 1;

   if (IC1Count == 0)
   {
   }

  if (IC1Count == 1)
  {
    LastIC1 = TIC1;
    SET_BIT(TCTL2,BIT5);        /* Set next interrupt for Falling Edge */
    CLEAR_BIT(TCTL2,BIT4);
  }

  if (IC1Count == 2)
  {
    HighTimeIC1 = TIC1 - LastIC1;
    LastIC1 = TIC1;
    CLEAR_BIT(TCTL2,BIT5);
    SET_BIT(TCTL2,BIT4);        /* Set next interrupt for Rising Edge */
    TMSK1 &= IC1_Mask;          /* Disable Interrupt */
  }

  IC1Count++;
  if (IC1Count > 2) IC1Count = 0;
} /*******************************************************************/


void TIC2_isr()            /* Increment IC3Count */
{ /*******************************************************************/
  CLEAR_FLAG(TFLG1,0x02);  /* Clear TIC2 interrupt Flag*/
  ic_number = 2;

  if (IC2Count == 0)
  {
  }


  if (IC2Count == 1)
  {
    LastIC2 = TIC2;
    SET_BIT(TCTL2,BIT3);        /* Set next interrupt for Falling Edge */
    CLEAR_BIT(TCTL2,BIT2);
  }

  if (IC2Count == 2)
  {
    HighTimeIC2 = TIC2 - LastIC2;
    LastIC2 = TIC2;
    CLEAR_BIT(TCTL2,BIT3);
    SET_BIT(TCTL2,BIT2);        /* Set next interrupt for Rising Edge */
    TMSK1 &= IC2_Mask;          /* Disable Interrupt */
```

```
  }

  IC2Count++;
  if (IC2Count > 2) IC2Count = 0;
} /*********************************************************************/


void TIC3_isr()            /* Increment IC3Count */
{ /*********************************************************************/
  CLEAR_FLAG(TFLG1,0x01);  /* Clear TIC3 interrupt Flag*/
  ic_number = 3;


  if (IC3Count == 0)
  {
  }

  if (IC3Count == 1)
  {
    LastIC3 = TIC3;
    SET_BIT(TCTL2,BIT1);        /* Set next interrupt for Falling Edge */
    CLEAR_BIT(TCTL2,BIT0);
  }

  if (IC3Count == 2)
  {
    HighTimeIC3 = TIC3 - LastIC3;
    LastIC3 = TIC3;
    CLEAR_BIT(TCTL2,BIT1);
    SET_BIT(TCTL2,BIT0);        /* Set next interrupt for Rising Edge */
    TMSK1 &= IC3_Mask;          /* Disable Interrupt */
  }

  IC3Count++;
  if (IC3Count > 2) IC3Count = 0;
} /*********************************************************************/


void TIC4_isr()            /* Increment IC4Count */
{ /*********************************************************************/
  CLEAR_FLAG(TFLG1,0x08);  /* Clear TIC4 interrupt Flag*/
  ic_number = 4;

  if (IC4Count == 0)
  {
  }

  if (IC4Count == 1)
  {
    LastIC4 = TIC4;
    SET_BIT(TCTL2,BIT7);        /* Set next interrupt for Falling Edge */
    CLEAR_BIT(TCTL2,BIT6);
  }

  if (IC4Count == 2)
  {
    HighTimeIC4 = TIC4 - LastIC4;
    LastIC4 = TIC4;
    CLEAR_BIT(TCTL2,BIT7);
    SET_BIT(TCTL2,BIT6);        /* Set next interrupt for Rising Edge */
    TMSK1 &= IC4_Mask;          /* Disable Interrupt */
  }

  IC4Count++;
  if (IC4Count > 2) IC4Count = 0;
} /*********************************************************************/


void TOV_isr()           /* Increment Overflow_count */
{ /*********************************************************************/
  CLEAR_FLAG(TFLG2,0x80);  /* Clear TOF interrupt Flag*/
```

```
Overflow_count += 1;
} /*********************************************************************/


void TOC3_isr()  /* Interrupt handler for PA5 (OC3) servo signals */
{ /*********************************************************************/

  if ((TCTL1 & 0x30) == 0x30)  /* Was Last Output High? */
  {
    TOC3 = TOC3 + servo_pulse_width;  /* Set HIGH Time */
    TCTL1 = 0x20;                     /* Set next Output to Low */
  }
  else        /* Last Output was Low */
  {
   TOC3 = TOC3 + (PERIOD - servo_pulse_width);    /* Set LOW Time */
    TCTL1 = 0x30;                     /* Set next Output to High */
  }

  CLEAR_FLAG(TFLG1,0x20);          /* Clear OC3 flag */

} /*********************************************************************/


/*  Binary Search Algorithm */
/*  Returns Theta_1 degree value */

float binsearch(float target)
{
  int  middle;
  int  low = 0;
  int  hi  = 600;

  while (low < hi)
  {
    middle = (low + hi)/2;
    if (target < InvTan[middle])
       hi = middle-1;
    else // Assert: target >= InvTan[middle]
         if (target > InvTan[middle])
       low = middle+1;
  }

  return DegVal[middle];
}


int read_int(void)
/*********************************************************************
 * Function description:                                             *
 *   Receives a sequence of numerical characters from the serial port *
 *   and converts to a signed 16-bit binary number. The number is     *
 *   read modulo 2**15.       The number is echoed as it is typed. A CR and *
 *
 * Returns:  16-bit signed integer                                   *
 *                                                                   *
 * Inputs                                                            *
 *   Parameters: None                                                *
 *   Globals:    None                                                *
 *   Registers:  SCSR, SCDR                                          *
 * Outputs                                                           *
 *  Parameters: None                                                 *
 *  Globals:    None                                                 *
 *  Registers:  None                                                 *
 * Functions called: get_char(), put_char                            *
 * Notes: None                                                       *
 *********************************************************************/
{
 int digit, number, sign;

 sign = 0;
 digit = get_char();
```

```
 put_char(digit);    /* Echo all entered digits */
 if( digit == '-') {sign = 1; digit = get_char();put_char(digit);}
 if( digit == '+') {sign = 0; digit = get_char();put_char(digit);}

 number = 0;

 while ((digit >= 0x30) && (digit <= 0x39))
   {
 /*Convert digit from ASCII to binary equivalent*/
    digit = digit-0x30;

 /*Multiply previous number by 10*/
    number = (number<<3) + (number<<1) + digit;
    digit = get_char();
    put_char(digit);

   } /*end of while*/

 put_char(13); put_char(10);/*carriage-return, line-feed*/

 if(sign==1) return(-number); else return(number);

}

/**********************End read_int *********************************/

void put_char(int outchar)
/*********************************************************************
 * Function description:                                             *
 *   Writes an ASCII character to the serial port.                   *
 *                                                                   *
 * Examples:                                                         *
 *   put_char(65);                                                   *
 *   put_char('A');                                                  *
 *                                                                   *
 * Returns: None                                                     *
 *                                                                   *
 * Inputs                                                            *
 *   Parameters: outchar                                             *
 *   Globals:    None                                                *
 *   Registers:  SCSR                                                *
 * Outputs                                                           *
 *   Parameters: None                                                *
 *   Globals:    None                                                *
 *   Registers:  SCDR                                                *
 * Functions called: None                                            *
 * Notes: None                                                       *
 *********************************************************************/
{
  int test = 0;
/*Wait for transmitter ready*/
  while (test == 0)
  {
    test = SCSR & 0x80;
  }
/*Transmit character*/
  SCDR = outchar;
}
/************************ End put_char ****************************/


/******************* Initialize interrupt vectors *********************/
#pragma abs_address:0xffd6
void (*interrupt_vectors[])(void) =
       {
       DUMMY_ENTRY,   /* SCI */
       DUMMY_ENTRY,   /* SPI */
       DUMMY_ENTRY,   /* PAIE */
       DUMMY_ENTRY,   /* PAO */
       TOV_isr,       /* TOF */
       TIC4_isr,      /* TOC5 */
```

```
        DUMMY_ENTRY,   /* TOC4 */
        TOC3_isr,      /* TOC3 */
        DUMMY_ENTRY,   /* TOC2 */
        DUMMY_ENTRY,   /* TOC1 */
        TIC3_isr,      /* TIC3 */
        TIC2_isr,      /* TIC2 */
        TIC1_isr,      /* TIC1 */
        DUMMY_ENTRY,   /* RTI */
        DUMMY_ENTRY,   /* IRQ */
        DUMMY_ENTRY,   /* XIRQ */
        DUMMY_ENTRY,   /* SWI */
        DUMMY_ENTRY,   /* ILLOP */
        DUMMY_ENTRY,   /* COP */
        DUMMY_ENTRY,   /* CLM */
        _start                /* RESET */
        };

#pragma end_abs_address
```

APPENDIX B
SWINGBOT SOFTWARE CODE

```
/*********************************************************************
 * Title        swing5.c                                             *
 * Programmer   Nathan J. Nipper                                      *
 * Date         February 13, 2001                                        *
 * Version      05                                                    *
 *                                                                    *
 Description
    Drive swinging pedulum with servo at OC3(PA5) using
    angle-detecting POT at PE5.  Switch hooked to PE4 determines
    if swing is trying to increase or decrease swing amplitude.
    The magnitude of the driving amplitude is set by the hand POT(PE7)
    Version 03: Clean UP CRT display
    Version 04: Fix parameters for E-Fair Fixture
    Version 05: Code for E-Fair (no keyboard input)
 *********************************************************************/


/*************************** Includes *********************************/
#include <stdio.h>
#include <hc11.h>
#include <serialtp.h>
/*********************************************************************/


/*************************** Defines **********************************/
#define DUMMY_ENTRY (void (*)(void))0xFFFE   /* Default for unused interrupts */
extern void _start(void)  ;               /*Initialize RESET vector in crt11.s */
#define PERIOD 40000                       /* 50Hz refresh for each servo */

#define CLEAR_BIT(x,y) x &= ~y;  // Clear all bits in x that are set in y
#define SET_BIT(x,y) x |= y;     // Set all bits in x that are set in y
#define CLEAR_FLAG(x,y) x &= y;  // Clearing Interrupt Flags

#define POT1 analog(7)
#define POT2 analog(5)

#define BIT0 0x01;
#define BIT1 0x02;
#define BIT2 0x04;
#define BIT3 0x08;
#define BIT4 0x10;
#define BIT5 0x20;
#define BIT6 0x40;
#define BIT7 0x80;
/*********************************************************************/

/************************** Prototypes ********************************/
#pragma interrupt_handler TOC3_isr

void init_SCI(void);
void init_servo(void);
void servo(int);    /* send desired Pulse Width(0 or 2000 to 4000) */
void init_analog(void);
int analog(int);
/*********************************************************************/


/*************************** Globals **********************************/
int i, j, k, Tweak, Max;
```

```
int servo_pulse_width, servo_pos, servo_add;
int SwPos[200];
/*********************************************************************/


void main(void)
{ /*****************************************************************/
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* Clear screen */
  char place[]= "\x1b[1;1H";          /* Home cursor  */

  init_SCI();
  init_analog();
  init_servo();

  servo_pos = 3400;
  servo(servo_pos);  /* Center Servo */

  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/
  printf(" WECLOME TO THE ROBOT SWING!!");
  printf("\x1b[3;1H    Hand Pot (POT1): %d",POT1);       /* Line 3 */
  printf("\x1b[5;1H   Angle Pot (POT2): %d",POT2);       /* Line 5 */
  printf("\x1b[7;1H           ServoPos: %d",servo_pos);  /* Line 7 */

  /* New Code for E-Fair */
  while((PORTE & 0x10) != 0x10);
  while((PORTE & 0x10) != 0x00);  /* Toggle switch to Start Operation */
  while((PORTE & 0x10) != 0x10);

  /*
    printf("\x1b[10;1H Press 'Enter' to Start Swinging...");
    k = read_int();
  */

  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/
  printf(" WECLOME TO THE ROBOT SWING!!");
  printf("\x1b[10;1H      I AM SWINGING NOW            ");

  Tweak = 4;

while(1)
 {
  servo_add = (POT2 - 105);

/*  if (servo_add > -4)
  {
       if (servo_add < 4) servo_add = servo_add * Tweak;
  } */

  if ((PORTE & 0x10) == 0x10) servo_pos = servo_pos - servo_add;
  else servo_pos = servo_pos + servo_add;     /* Test Switch @ PE4 */

  if (servo_pos > 4400) servo_pos = 4400;
  if (servo_pos < 2400) servo_pos = 2400;
  servo(servo_pos);

  Max = 125 + POT1 * 5;              /* Wait Loop to control Amplitude */
  for (j = 0; j < Max; j++);

 }  /* END of while(1) loop */

} /*****************************************************************/


void init_SCI(void)   /* Set SCI for 9600, 8-n-1, TE, RE */
{ /*****************************************************************/
  CLEAR_BIT(SPCR,0x20);
  SCCR1 = 0X00;
  BAUD = 0xb0;  /* 0xbO is 9600 0x35 is 300 baud */
  SCCR2 = 0x0C;
```

```
} /*********************************************************************/


void init_servo(void)  /* Initialize variables for servo on PA5 (OC3) */
{ /*************************************1111*********************************/
  INTR_OFF();

  servo_pulse_width = 0;      /* Motor starts turned off */
  TCTL1 = 0x40;               /* Clear OC3 on first interrupt */

  INTR_ON();
} /*********************************************************************/


void init_analog(void)  /* Power up A/D */
{ /*********************************************************************/
  SET_BIT(OPTION,0x80);
} /*********************************************************************/


int analog(int port)   /* Returns the analog value read from A/D PORTE */
{ /*********************************************************************/
  ADCTL=port;          /* Address the selected channel */
  while((ADCTL & 0x80) == 0); /* Wait for A/D to finish */
  return(ADR1);               /* Return analog value */
} /*********************************************************************/


void servo(int newposition)
/* Servo on PA5 <- position defined by newposition     *
 * This subroutine takes the new position in degrees   *
 * and calculates the pulse HighTime in E-Clocks.  PWM *
 * signal is generated by ISR.                         */
{ /*********************************************************************/

  if (newposition != 0)
  {
    servo_pulse_width = newposition;
    SET_BIT(TMSK1,0x20);          /* Enable OC3 interrupt */
  }
  else CLEAR_BIT(TMSK1,0x20);     /* Disable OC3 interrupt */

} /*********************************************************************/


void TOC3_isr()  /* Interrupt handler for PA5 (OC3) servo signals */
{ /*********************************************************************/

  if ((TCTL1 & 0x30) == 0x30)  /* Was Last Output High? */
  {
    TOC3 = TOC3 + servo_pulse_width;  /* Set HIGH Time */
    TCTL1 = 0x20;                     /* Set next Output to Low */
  }
  else        /* Last Output was Low */
  {
    TOC3 = TOC3 + (PERIOD - servo_pulse_width);    /* Set LOW Time */
    TCTL1 = 0x30;                     /* Set next Output to High */
  }

  CLEAR_FLAG(TFLG1,0x20);          /* Clear OC3 flag */

} /*********************************************************************/


/******************** Initialize interrupt vectors **********************/
#pragma abs_address:0xffd6
void (*interrupt_vectors[])(void) =
       {
       DUMMY_ENTRY,   /* SCI */
       DUMMY_ENTRY,   /* SPI */
       DUMMY_ENTRY,   /* PAIE */
```

```
        DUMMY_ENTRY,   /* PAO */
        DUMMY_ENTRY,   /* TOF */
        DUMMY_ENTRY,   /* TOC5 */
        DUMMY_ENTRY,   /* TOC4 */
        TOC3_isr,      /* TOC3 */
        DUMMY_ENTRY,   /* TOC2 */
        DUMMY_ENTRY,   /* TOC1 */
        DUMMY_ENTRY,   /* TIC3 */
        DUMMY_ENTRY,   /* TIC2 */
        DUMMY_ENTRY,   /* TIC1 */
        DUMMY_ENTRY,   /* RTI */
        DUMMY_ENTRY,   /* IRQ */
        DUMMY_ENTRY,   /* XIRQ */
        DUMMY_ENTRY,   /* SWI */
        DUMMY_ENTRY,   /* ILLOP */
        DUMMY_ENTRY,   /* COP */
        DUMMY_ENTRY,   /* CLM */
        _start                /* RESET */
        };

#pragma end_abs_address
```

LIST OF REFERENCES

[1] Pratt, Williamson, Dillworth, Pratt, Ulland, and Wright, *Stiffness Isn't Everything,* Proceedings of ISER, 1995. http://alpha-bits.ai.mit.edu/projects/leglab/, accessed on 4/13/2001.

[2] A. J. Lee, *Mechanical Design and Internet-Based Control of the Gyrobot*, Master's Thesis, University of Illinois at Urbana-Champaign, 2000.

[3] N. Nipper and J. Godowski, *Robotic Swing Drive as Exploit of Stiffness Control Implementation*, 2001 Florida Conference for Recent Advances in Robotics. http://mil.ufl.edu/~swing/, accessed on 5/25/2001.

[4] Serway, Raymond A. *Physics for Scientists and Engineers.* Saunders College Publishing, Orlando, Florida, 1990.

[5] Doty, Keith. *Talrik Junior Professional Edition Assembly Manua*l. Mekatronix 1999. http://www.mekatronix.com, accessed on 1/11/2001.

BIOGRAPHICAL SKETCH

Nathan Nipper graduated from the University of Florida in May 1997 with a bachelor's degree in electrical engineering.

After undergraduate school, Nathan accepted a full-time position as a Test Engineer for Honeywell Space Systems in Clearwater, FL. His major responsibility was to run the troubleshooting lab for all of the subassembly circuit boards for two flight units that provided guidance and navigation for Titan and Atlas rocket launches. He also acted as a customer interface to provide technical support launch constraints.

After almost three years at Honeywell, Nathan returned to UF for full-time graduate school to finish a master's degree in electrical engineering and pursue a career in digital hardware design. His graduate emphasis and research is in machine intelligence and embedded micro-controller applications in robotics.