TA: Louis Brandy
William Dubel
Max Koessick
Instructor: A. A Arroyo

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Written Report

SEAL

Greg Beckham

Table of Contents:

OPENING

Abstract:

This report describes a robot that will balance an inverted pendulum by moving driving forward and backwards. The robot will determine a center position of the pendulum and then attempt to keep the pendulum at the center position. The name SEAL (Simple Efficient Autonomous Levitator) refers to what the robot is doing as well as to the sea loving mammal that can be taught to balance a ball on its nose.

Executive Summary:

As stated before, this robot is designed to balance an inverted pendulum. It uses a potentiometer to read the position of the pendulum and combines this with previous measured positions to generate the velocity of the moving pendulum. This is then used in a PD control algorithm.

SEAL uses two gear head motors capable of both high rpm and a large amount of torque. These motors are required because SEAL must be able to change directions quickly and reach a speed high enough to get under the moving pendulum.

SEAL's pendulum consists of a half in wooden dowel held in place by two go-kart wheel bearings. Attached to that is a 3 foot threaded steel rod with a PVC pipe sheath to increase rigidity. On top is a dense foam ball for weight and a cushion for when the pendulum falls.

Introduction:

SEAL consists of a 9x5x3 rectangular box that houses the majority of the electronic components. On top, the pendulum apparatus is bolted to the lid and extends upward when SEAL is balancing.

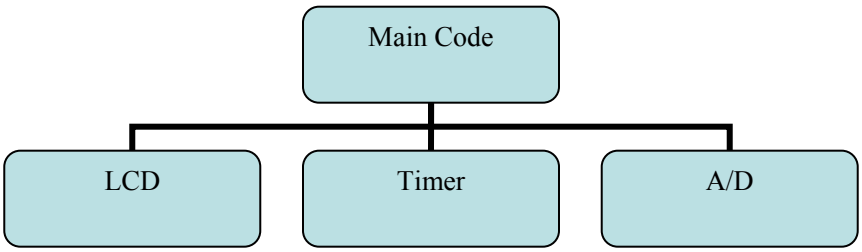
SEAL can only balance the pendulum when the ball pendulum stays within a few degrees of center. Once the pendulum is outside of this zone SEAL is not fast enough to catch up with the ball before it falls to the ground. Therefore SEAL must react very quickly to the movement of the ball to keep it within the required range.

SEAL's software runs on the M128 board from BDMicro. This board has an Atmel ATMega128 microcontroller with a 16Mhz clock. The software is written entirely in C and is compiled using the avr-gcc compiler. Using C, I was able to program SEAL much more quickly, than if I was using assembly. Also, my code is much more readable and can be changed quite easily.

MAIN BODY

Integrated System:

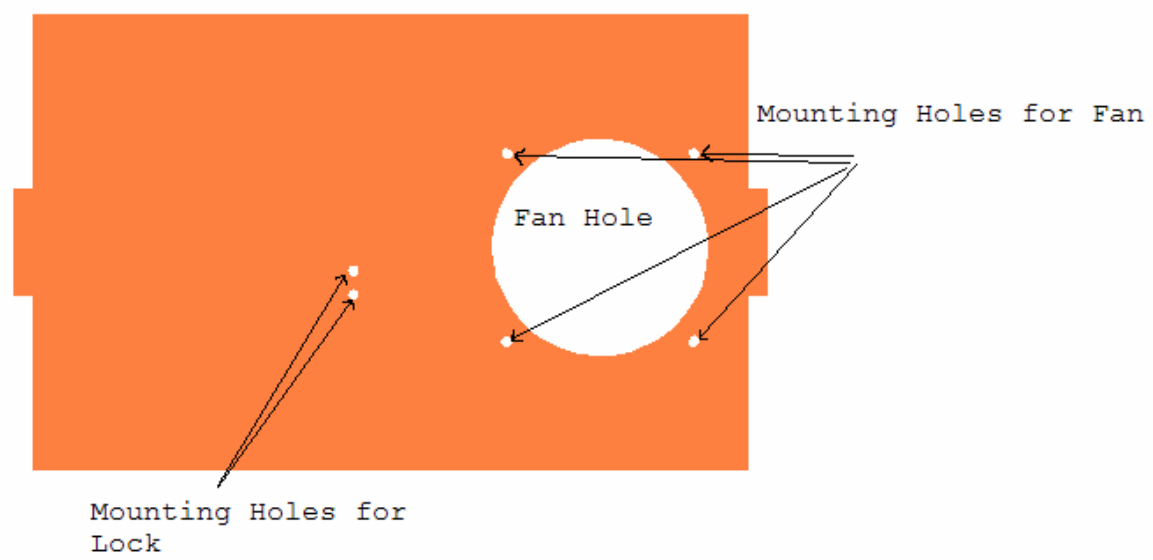
SEAL's code is written entirely in the C programming language. In total, it is about 450 lines. The code is broken down into different subroutines to do different jobs. The different parts of the code are LCD control, A/D, and timers. Each of these subsystems were written and tested before combining them together to create the SEAL's main code. The following figure gives a graphical representation of how the code is put together.



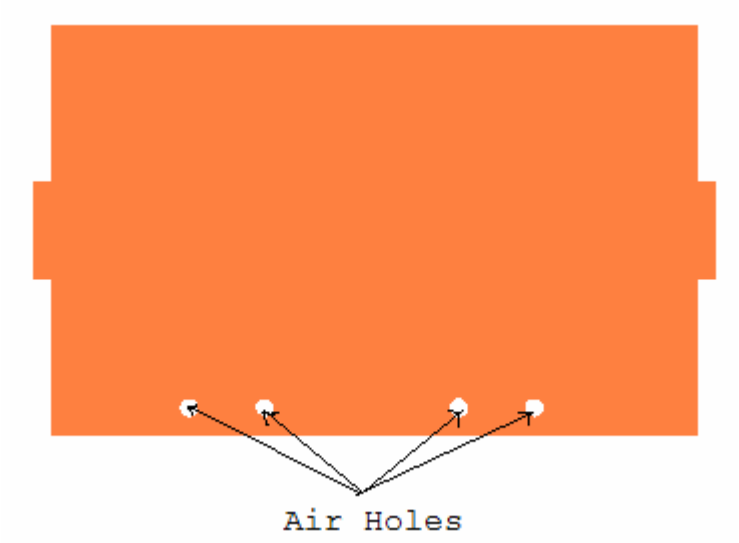
Mobile Platform:

SEAL's main body consists of 9 wooden pieces put together to form a box with a hinged lid. The box is 9 inches long 5 inches wide and 3 inches tall. The entire box is glued together except for the top which is held in place by hinges for easy access to the inside. Also SEAL as a pendulum apparatus attached to its top and held in place by steel bolts. The pendulum consists of many different materials some of which are wooden components cut out in class. More details on this will be discussed later in the report

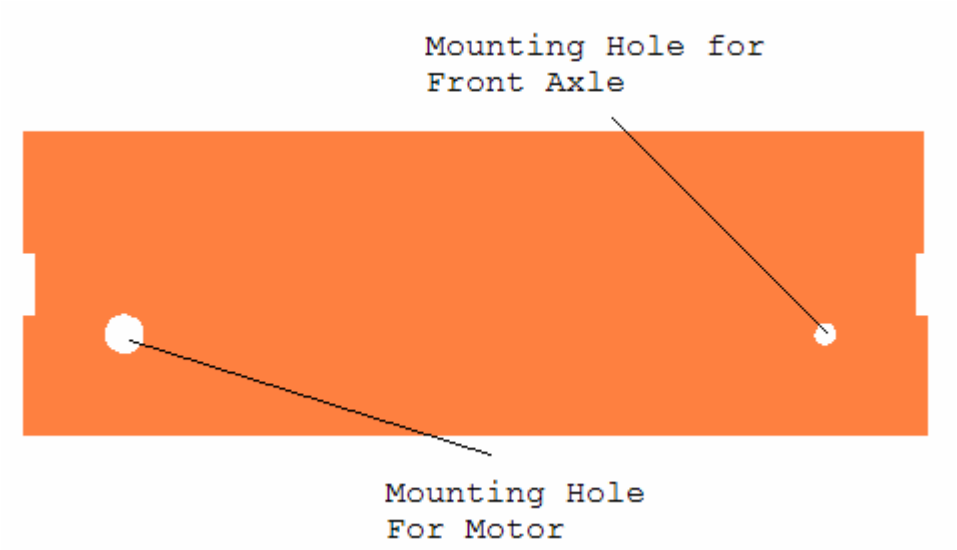
The front of the robot has a whole in it for a cooling fan to suck in air as well as a lock to hold the top down. Below is a graphical representation of the front of the robot. On it you can see the hole for the fan and the mounting holes for both the fan and the lock. Also you can see the two tabs on the side which are used to hold the piece together with the other pieces of the robot.



The back of the robot is the same size as the front. Instead of fan and lock mounting holes, it has for holes in the bottom to help the air brought in by the fan in front to circulate throughout the robot and exit in the back. The following is a figure showing the design of this piece.

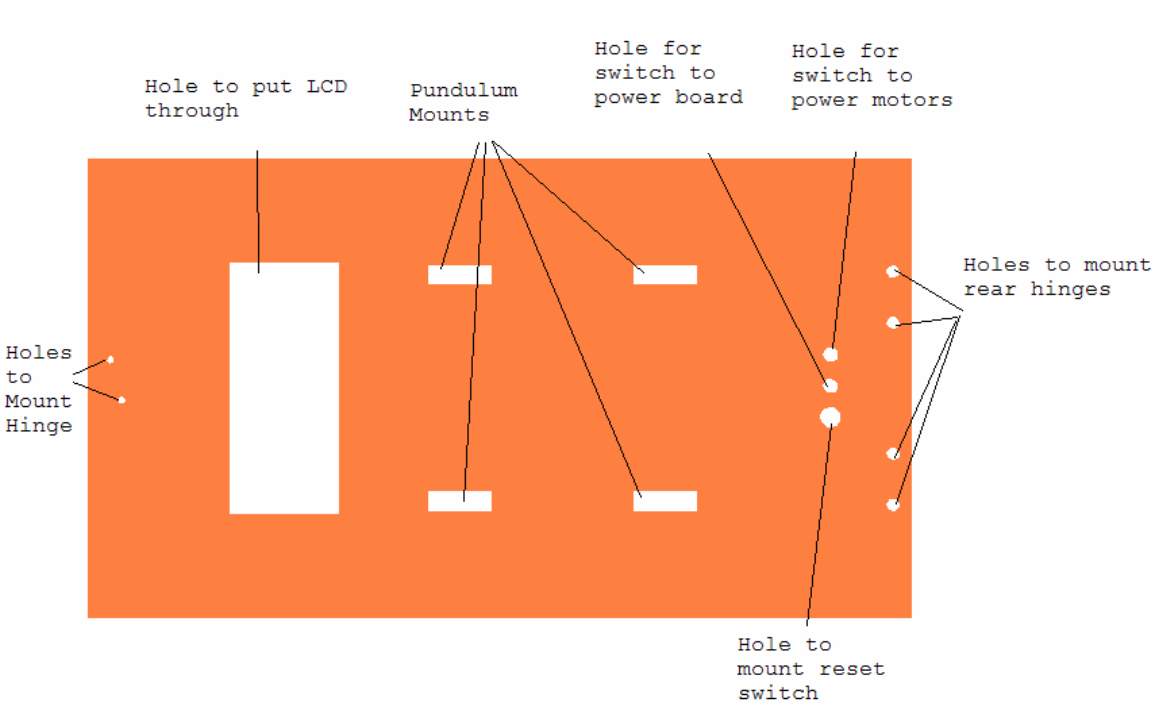


Next, SEAL has two sides both made using the same design. The side of the robot has two holes in it. One hole is to mount the motors the other is to mount the axle for the front wheel. Below, you can see the side piece with the holes discussed above.



The bottom of SEAL's body consists of a 9x5 piece of wood that has been glued to the four side pieces.

SEAL's top is by far the most intricate piece. It consists of a 9x5 sheet of wood that has holes cut in it for the LCD display, pendulum mounts, switches, and hinge mounts. The following figure shows a diagram of SEAL's top.



The final piece of SEAL's platform is the pendulum. The main pendulum mounts are constructed using four pieces of wood. Two of the pieces are then glued together to form one mount and likewise for the other two. An example of the pendulum mounts can be seen below.

The Pendulum itself consists of a ½" wooden dowel shaft that has been bored out about 2" into the end. The shaft of the potentiometer is then put into this hole while the body of the potentiometer is held in place by a wooden plate that is then screwed to the wooden pendulum mounts. Also a whole is drilled perpendicularly through the middle of the dowel in order to mound the rod of the pendulum. The rod of the pendulum is a 3/8" steel threaded rod that is pushed through the hole mentioned about and held in place by two lock washers and two bolts. Around the threaded rod is a PVC pipe sheath to increase the rigidity of the rod. On top to act as a weight is a dense foam ball.

Actuation:

SEAL's movement is handled by two 12VDC motors and a motor driver. The motor driver circuit developed by Max Koessick uses two LMD18200T motor drivers, one for each motor. Two 470µF capacitors help to alleviate voltage spikes between power and ground while two large heat sinks one attached to the back of each motor driver chip

regulate help to cool down the motor drivers during large changes in motion. Also to aid in the cooling of the motor drivers a 12V fan was mounted to the front of the robot and the motor driver board is positioned in such a way that the air from the fan blows across the two heat sinks.

I used pulse width modulation (PWM) to send movement signals to the motors. I small trick was used to make this easier than what the motor driver is built for. The motor driver chips have a direction pin as well as a power pin. Under normal operation, one would set the direction pin in the direction desired and then control how fast to go by the power pin. However, since motors are integrators, by sending signals to go forward and then reverse in rapid succession the motor averages these signals out and responds to which ever signal was more prevalent. So what is done, is to set the power pin to VCC which will signal the motor to go at full speed and then send the PWM signal to the direction pin which will then control speed and direction.

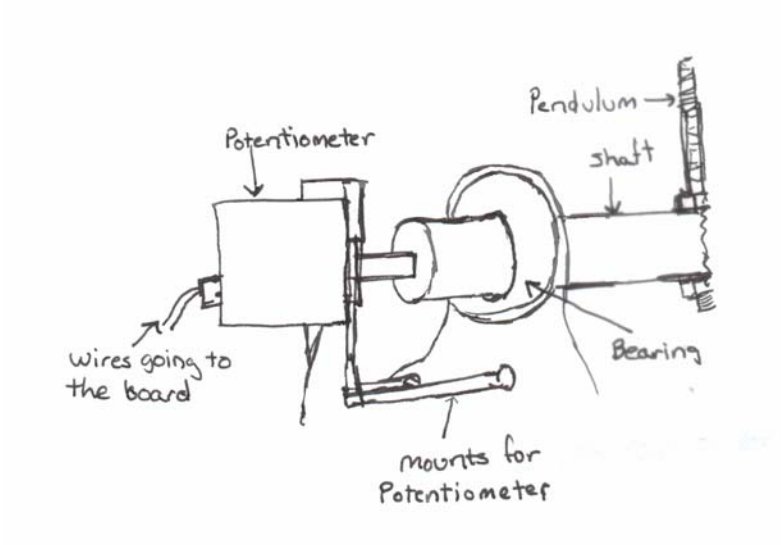
I used 4 inch rubber airplane wheels from hobby-lobby.com attached to the motors via 4mm aluminum hubs from lynxmotion. I chose then wheels because of their traction and size. In the front, I used 3 inch wheels from hobby-lobby.com. These wheels were attached to independent axles and allowed to turn freely. It also gave the robot a nice appearance with larger wheels in the back than in the front.

The motors used to generate SEAL's power are two 178rpm and 150 ozin of torque GH12 motors from Lynxmotion. These motors have a good combination of rpm and torque necessary for changes direction quickly and then catching the falling pendulum. However, the gears inside the gear-head are very weak. During the testing of SEAL, I stripped multiple gears and ultimately destroyed 4 of these motors. These motors seem to

be good for projects in which you will not be changing directions very quickly but are not well suited for my purposes. The constant problems with these motors ultimately led to my lack of completion of the desired operation of SEAL.

Sensors:

The Potentiometer, in this case a Alpha model number B5K, is a common device used that produced an adjustable voltage divider circuit. In this case, the potentiometer is mounted to a shaft such that when the shaft turns, the voltage divider circuit within the potentiometer changes. The following is an illustration of how the device is hooked up.



When attached in this manor, the potentiometer is able to sense the position of the pendulum as the ball falls forward or backwards. The bearing provides the shaft with the ability to move smoothly, and the mounts for the potentiometer stop the potentiometer from moving when the pendulum rotates.

Advantages and Disadvantages:

Another sensor option would be to use a shaft encoder to since the position. The advantages of that sensor technology are that they can read the exact position of the sensor without having the use an analog to digital converter. Also, this type of sensor can have a greater resolution than a potentiometer. However, in order to get a shaft encoder that is good enough to be useful for this type of application one would have to spend at least $50. A potentiometer however offers ample accuracy and the A/D converter in the Atmel ATMicro128 is fast enough that readings can be taken fast enough in order to adjust to the moving pendulum. Also, a potentiometer is much less expensive than an encoder. The one that I am using was $2.79, some very good potentiometers can be found for $25.

Specifications:

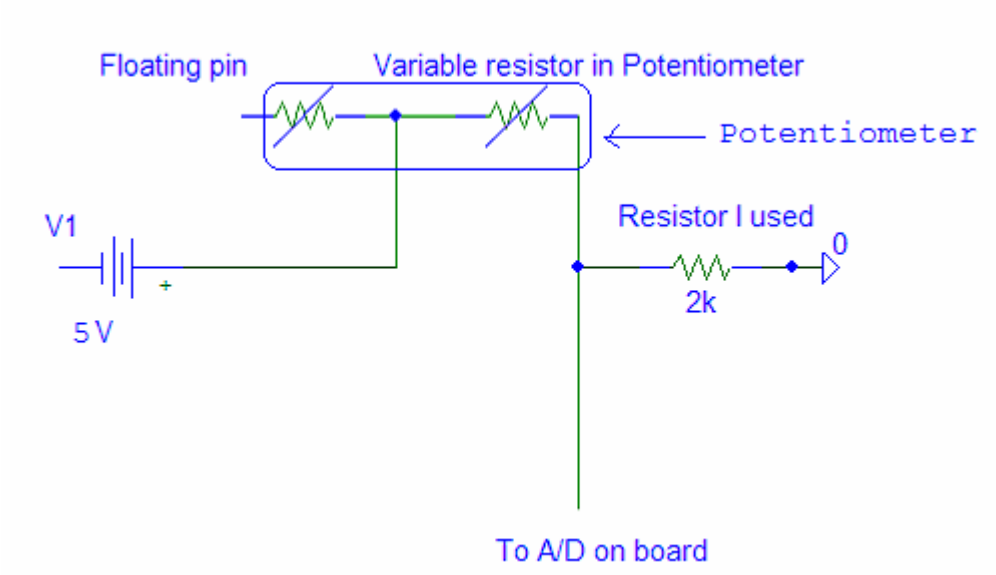
The potentiometer I am using was purchased from Radio Shack model number 271-1714 http://www.radioshack.com/product.asp?catalog%5Fname=CTLG&category%5Fname=CTLG%5F011%5F002%5F011%5F000&product%5Fid=271%2D1714 . It is a 5K potentiometer which means that the resistance across the entire voltage divider circuit is always 5 Kilo Ohms. It has a linearity tolerance of 20% which indicates that as you turn the potentiometer the resistor value the circuit generates will change linearly within 20%. My potentiometer has a 5% resistance tolerance which indicates the amount of resistance that is experienced as you try to turn the shaft of the potentiometer. This potentiometer is not an exact sensor. Much better potentiometers can be obtained that have a smaller linearity tolerance which will give more exact data. However, I found that since there is

a very small angle in which the pendulum can move before it is unrecoverable, the values obtained from the potentiometer in this range are adequate for this application.

Also the potentiometer I am using has a 6mm diameter shaft, which was originally an inch and a half long. I cut the shaft down to about a half inch in length and then pushed it into my pendulum assembly.

Usage:

A Potentiometer normally has 3 leads one for power, one for ground, and the other for the output voltage. In my application I used the potentiometer only as a variable resistor. So I left the power pin floating and connected the output pin to +5v and the ground pin to the middle of a voltage divider I designed. I then combined this with another resistor which I selected and used the voltage between the potentiometer and the other resistor as my input voltage to the A/D. The circuit to use this sensor is shown in the figure below.



This is a very simple circuit that connects the variable voltage of the potentiometer to the A/D converter on the board. The board is then programmed to read the A/D and react accordingly. The 2K resistor was chosen by moving the pendulum to the front tolerance and getting a resistance value, and then moving it to the back tolerance and recording a resistance. Then using the equation $\sqrt{R1}\sqrt{R2}$ given to my by the TA, I was able to find a resistor value for my voltage divider that would give me the best resolution for my design. Using this circuit I was able to obtain a resolution of less than a half of a degree of movement of the pendulum. This allowed me to react very quickly to small changes in to position of the pendulum and get under it before it fell so far that I could not recover.

Behaviors:

SEAL has two basic behaviors. The first is to determine where the center point is for the pendulum and the second is to move forward and backwards attempting to stay under the pendulum and keep it from falling.

The first behavior begins as soon as the board is turned on. SEAL will display "Centering" on the LCD and during that time SEAL will take many different readings from the potentiometer and average them together. However, SEAL needs a little help determining the center point, so the user must hold the pendulum as close to the center as possible in order for SEAL to get a good idea of center. This behavior lasts 8 seconds in which time SEAL takes 40 different readings from the potentiometer and averages them all together.

The second and primary behavior of SEAL is to balance the pendulum. This is accomplished by reading the location of the pendulum and comparing it to the center position to get an offset from center. It then uses this offset directly to find the position

of the pendulum and subtracts the old position from it to find a velocity component. After both a position and a velocity component have been found each are multiplied by a different multiplier and then added together. This final value is then sent to the motors. This loop of is then repeated about every 40ms.

Experimental Layout and Results:

The first test was to attach the pendulum to the A/D and see what values I was receiving. I used code found on the BDMicro site to poll the A/D for values. From this I was able to determine whether the values I was receiving would be precise enough for my application. From this test I did find that I was receiving about 3 values per degree of change which is more than enough for the application.

I also experimented to find the tolerances for which the pendulum could fall forward or backward and the robot will still be capable of catching it. I did this by changing the value at which the motors began driving full speed. I then let the ball fall from the center point and saw whether or not the ball hit the ground in the direction of its original fall or if the robot caught the falling pendulum and caused it to fall in the opposite direction. From this experiment I was able to determine that the robot must start moving while the pendulum is within about 3 degrees of center or the pendulum would be unrecoverable.

CLOSING

Conclusion:

Unfortunately, I feel was unable to complete the project. SEAL at its best was only able to balance the pendulum for 4 to 5 seconds. I feel that this lack of balancing ability is because the code is not written well enough to control the pendulum. This is the result of lack of opportunity to code the robot. Every time I tried to code the robot for over about an hour at a time the motors would strip gears. I ordered two sets of motors and was able to destroy enough gears in both that gear replacements were no longer possible. I feel that if these motors had not continued to break I would have had the time I needed to write the code necessary to balance the pendulum.

In the future, I think it would be interesting to write self learning code for SEAL. I feel this method would produce a robot that would be much better at balancing than any code that I could derive.

To anybody who tries this project in the future, I would recommend starting the project early. Something will go wrong with your hardware or your software will take longer to write than you originally think. For the most part my design was solid; some things that I was unhappy with are the pendulum rod, the potentiometer attachment, and motors. First, the rod of the pendulum was a little flimsy; a better choice may have been a copper pipe which would be stronger and lighter. The pendulum attachment was not exactly correct either. Finding a way to hold it in place in the center of the shaft would have been a big help. Finally, I would have ordered better motors or built a gear box for the ones that I have. The motors were eventually the reason was not successful. The constant shredding of the gears caused me to be unable to program the robot. I would not

recommend using any of the motors on the lynxmotion website that have a 4mm shaft. Thanks to William I was able to use two motors with 6mm shafts. They seem to be a little more robust and allowed me to make much more progress. The motors are the most important part of this design. You will want to find motors with high torque and high rpm that are capable of catching the pendulum over the greatest range of motion.

Another problem with this project is deciding what code to change. Once you get close to having the pendulum balanced you will begin trying many different things and nothing really works. Also, certain conditions may be constantly changing such as battery charge and surface. It would probably be wise to regulate the power going to the motors. This will allow you to have one more constant in your equation. Another problem is the surface on which the robot moves. To solve this problem I recommend buying some sort of floor covering (I found that outdoor carpeting would probably make a good surface) and sticking with it. One trick I learned from fellow classmate Anne Harmeson is too put soda on your tires and the floor in order to increase traction.

Documentation:

I'd like to thank Dr. Arroyo, Dr. Schwartz for allowing me to have this experience. Not only do I feel this class helped me in interviews but gave me much more confidence in myself as an engineer. I would also like to thank Luis, Max, and William for all of the help in the lab. Your expertise in the area of robotics was greatly appreciated.

I would also like to credit a number of people whom I have used their code, designs, tools, or ideas in the completion of my robot:

- Louis Brandy for informing me of the equation to find the resistor value used in my voltage divider equation and explaining how to wire my potentiometer to best suit my needs
- Max Koessick for the motor driver circuit he designed for me and for helping create a centered hole in the shaft for my pendulum
- William Dubel for letting me borrow his motors. These motors made it possible to demo my robot on media day.
- Chris Taylor for giving me the initialization code for the PWM and letting me borrow numerous parts
- Max Billingsley for his LCD code
- Wade Greeson for letting me borrow his 6mm hubs and innovative ideas
- Jon Matheny for letting me use numerous tools of his throughout the semester
- Anne Harmeson for her idea on how to gain traction.
- BDMicro Sample code page for providing the A/D code and Timer code

Parts List:

Wood for body obtained in lab.

ATMega 128 board from BDMicro www.BDmicro.com

LMD18200T H-bridges samples from National Semiconductor www.national.com

Potentiometer from Radio Shack

LCD from microprocessors class

½" wooden dowel from Wal-mart

6’ threaded rod from Lowes

PVC pipe from Lowes

Ball from Palos Sports www.PalosSports.com

Assorted screws and bolts from the IMDL lab and Lowes

Cooling fan from E-plus electronics

Heatsinks from Radio Shack

Motors from lynx motion www.lynxmotion.com

Bearings from Lowes

Hinges and Latch from Lowes

Switches from IMDL lab

Appendices:

LCD.h

```
/*
* lcd.h
*
* Author: Max Billingsley
* Adapted by: Greg Beckham
*/
#include <avr/io.h>
#include <avr/signal.h>
#include <inttypes.h>


#define LCD_PORT PORTA
#define LCD_DDR DDRA
#define ENABLE 0x08
/* function prototypes */
void lcd_set_ddr(void);
void lcd_init(void);
void lcd_delay(void);
void lcd_send_str(char *s);
void lcd_send_byte(uint8_t val);
void lcd_send_command(uint8_t val);
```

LCD.C

```
/*
* lcd.c
*
* Author: Max Billingsley
* Adapted by: Greg Beckham
*/
/*
* LCD_PORT1  = RS
* LCD_PORT2  = R/W
* LCD_PORT3  = EN
* LCD_PORT4  = DB4
* LCD_PORT5  = DB5
* LCD_PORT6  = DB6
* LCD_PORT7  = DB7
*
```

```
* RS: Register Select:
*
* 0 - Command Register
* 1 - Data Register
*
*/
#include "lcd.h"
/* entry point */

void lcd_init(void)
{
lcd_send_command(0x83);
lcd_send_command(0x83);
lcd_send_command(0x83);
lcd_send_command(0x82);
lcd_send_command(0x82);
lcd_send_command(0x8c);
lcd_send_command(0x80);
lcd_send_command(0x0f);//0f
lcd_send_command(0x00);
lcd_send_command(0x01);
}

void lcd_set_ddr(void)
{
LCD_DDR = 0xff;
}

void lcd_delay(void)
{
uint16_t i;
for(i = 0;i<2000;i++){}
}

void lcd_send_str(char *s)
{
while (*s) lcd_send_byte(*s++);
}

void lcd_send_byte(uint8_t val)
{
uint8_t temp = val;
val &= 0xf0;
val |= 0x02;
LCD_PORT = val;
lcd_delay();
```

```
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
temp <<= 4;
temp |= 0x02;
LCD_PORT = temp;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
}

void lcd_send_command(uint8_t val)
{
uint8_t temp = val;
val &= 0xf0;
LCD_PORT = val;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
temp <<= 4;
LCD_PORT = temp;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
lcd_delay();
}
```

Timer.h

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#include <inttypes.h>

volatile uint16_t ms_count;

void ms_sleep(uint16_t ms);
void init_timer(void);
```

Timer.c

```
#include "timer.h"
/*
 * ms_sleep() - delay for specified number of milliseconds
 */
void ms_sleep(uint16_t ms)
```

```
{
  TCNT0  = 0;
  ms_count = 0;
  while (ms_count != ms)
    ;
}


/*
 * millisecond counter interrupt vector
 */
SIGNAL(SIG_OUTPUT_COMPARE0)
{
  ms_count++;
}


/*
 * initialize timer 0 to use the real time clock crystal connected to
 * TOSC1 and TOSC2 to generate a near 1 ms interrupt source
 */
void init_timer(void)
{
  /*
   * Initialize timer0 to use the 32.768 kHz real-time clock crystal
   * attached to TOSC1 & 2.  Enable output compare interrupt and set
   * the output compare register to 32 which will cause an interrupt
   * to be generated every 0.9765625 milliseconds - close enough to a
   * millisecond.
   */
  TIFR  |= BV(OCIE0)|BV(TOIE0);
  TIMSK |= BV(OCIE0);    /* enable output compare interrupt */
  TIMSK &= ~BV(TOIE0);   /* disable overflow interrupt */
  ASSR  |= BV(AS0);      /* use asynchronous clock source */
  TCNT0  = 0;
  OCR0   = 32;           /* match in 0.9765625 ms */
  TCCR0  = BV(WGM01) | BV(CS00); /* CTC, no prescale */
  while (ASSR & 0x07)
    ;
  TIFR  |= BV(OCIE0)|BV(TOIE0);
}
```

ADC.H

```
/*
 * $Id: adc.h,v 1.1 2003/12/11 01:35:00 bsd Exp $
```

```
 */
#include <avr/io.h>
#include <stdio.h>

#ifndef __adc_h__
#define __adc_h__

void    adc_init(void);

void    adc_chsel(uint8_t channel);

void    adc_wait(void);

void    adc_start(void);

uint16_t adc_read(void);

uint16_t adc_readn(uint8_t channel, uint8_t n);

#endif
```

ADC.C

```
/*
 * $Id: adc.c,v 1.2 2003/12/11 02:15:39 bsd Exp $
 */

/*
 * ATmega128 A/D Converter utility routines
 */

#include "adc.h"


/*
 * adc_init() - initialize A/D converter
 *
 * Initialize A/D converter to free running, start conversion, use
 * internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming
 * 16 MHz MCU clock)
 */
void adc_init(void)
{
  /* configure ADC port (PORTF) as input */
  DDRF  = 0x00;
  PORTF = 0x00;
```

```
  ADMUX = BV(REFS0);
  ADCSR = BV(ADEN)|BV(ADSC)|BV(ADFR) |
BV(ADPS2)|BV(ADPS1)|BV(ADPS0);
}


/*
 * adc_chsel() - A/D Channel Select
 *
 * Select the specified A/D channel for the next conversion
 */
void adc_chsel(uint8_t channel)
{
  /* select channel */
  ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}


/*
 * adc_wait() - A/D Wait for conversion
 *
 * Wait for conversion complete.
 */
void adc_wait(void)
{
  /* wait for last conversion to complete */
  while ((ADCSR & BV(ADIF)) == 0)
    ;
}


/*
 * adc_start() - A/D start conversion
 *
 * Start an A/D conversion on the selected channel
 */
void adc_start(void)
{
  /* clear conversion, start another conversion */
  ADCSR |= BV(ADIF);
}


/*
 * adc_read() - A/D Converter - read channel
```

```
 *
 * Read the currently selected A/D Converter channel.
 */
uint16_t adc_read(void)
{
  return ADC;
}


/*
 * adc_readn() - A/D Converter, read multiple times and average
 *
 * Read the specified A/D channel 'n' times and return the average of
 * the samples
 */
uint16_t adc_readn(uint8_t channel, uint8_t n)
{
  uint16_t t;
  uint8_t i;

  adc_chsel(channel);
  adc_start();
  adc_wait();

  adc_start();

  /* sample selected channel n times, take the average */
  t = 0;
  for (i=0; i<n; i++) {
    adc_wait();
    t += adc_read();
    adc_start();
  }

  /* return the average of n samples */
  return t / n;
}
```

MAIN.C

```
#include <avr/pgmspace.h>
#include "adc.h"
#include "timer.h"
#include "LCD.h"
// Declare your global variables here
```

```
int main(void)
{
static int DIFF = 128;
DDRB = 0x60;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 62.500 kHz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0x0A;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
int direction;
int center = 0;
int offset = 0;
int velocity = 0;
int power,temp,vel_temp;
int previous = 128;
int previous_power = 0;
uint16_t i = 1;
init_timer();
sei();
adc_init();
lcd_set_ddr();
lcd_init();
fdevopen(lcd_send_byte,NULL,0);

OCR1A = 0x80;
OCR1B = 0x80;
```

```
printf("Centering");
while(i <= 8)
  {
  ms_sleep(1000);
  center =  center + adc_readn(0,5);
  i++;
  }
center = center >> 3;


while (1)
     {
       direction = adc_readn(0, 1);  /* sample channel 0 1 times*/
         if((direction <= (center - DIFF)) || (direction >= (center + DIFF)))
          {
          power = DIFF;
          //lcd_send_command(00);
          //lcd_send_command(01);
          //printf("Power: %3u", power);
          }
         else
          {
          offset = direction - center;
            temp = offset * 3;
            temp = DIFF + temp;
          velocity = offset - previous;
            velocity = velocity * 1;
            power =  temp + velocity;
          previous = offset;
          }
       if(power <= 60) power = 1;
         if(power > 200) power = 255;
       if(((power <= center + 2) && (power >= center - 2)) ||
((previous_power <= center + 2) && (previous >= center - 2))) power =
DIFF;
       //power = power + 128 - DIFF;
       power = power * .92 + previous_power * .08;
       previous_power = power;
       lcd_send_command(00);
```

```
        lcd_send_command(01);
        printf("Pow: %3d Vel: %3d", power,velocity);
        ms_sleep(40);
          OCR1A = power;         //Motor on Pin 6
          OCR1B = power;             //Motor on Pin 5

        }
return 0;
}
```