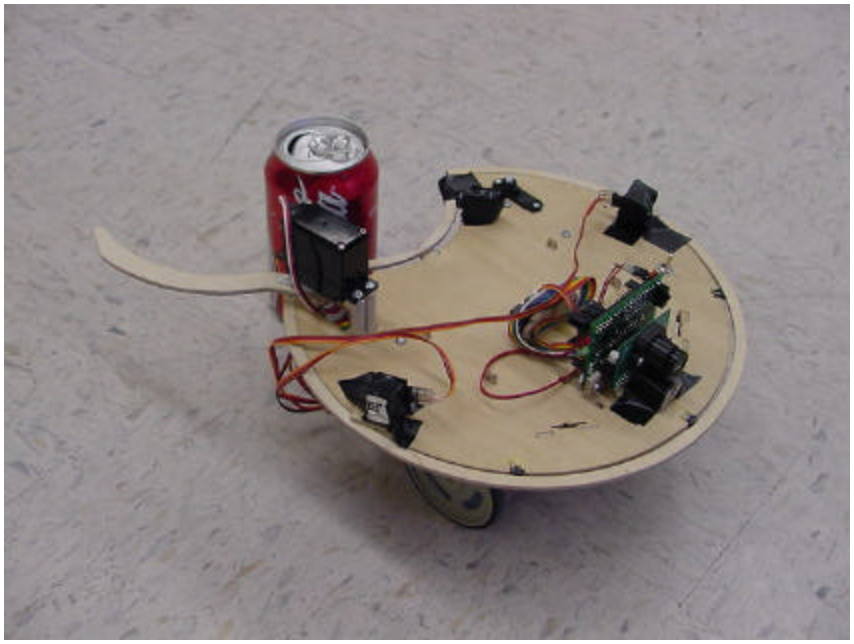

Kyle Tripician

EEL5666 – Intelligent Machines Design Laboratory

The Aluminator

03/22/03

Final Report

Teacher: Dr. A. A. Arroyo

TAs: Jason Plew and Uriel Rodriguez

Index

Abstract

The Aluminator was designed as an autonomous service robot. It roams an area looking for red aluminum cans. When one is found, it will approach it and grab it with its claws. Once it has a firm grip on it, it will take it to a designated trash area and release the can. The following paper has a detailed description of what I utilized to design and build this robot.

Executive Summary

When I first signed up for IMDL, I knew that it was very time consuming and stressful from past classes. However, that did not stop me. I was interested in integrating what I had learned in EEL4744, Microprocessors, with real life situations in hopes to get a better understanding of intelligent machine design. By designing the Aluminator, I not only have a better understanding of microprocessors in general, but also of behaviors of robots. My main goal in IMDL was to design an inexpensive robot that would serve a purpose in my life. The Aluminator does just that. While it was a very time consuming class, the end product was worth the time.

Introduction

Moving away for college brought about many new experiences for me. The freedom of living outside of the restrictions of my parents and the ability to do whatever I wanted, whenever I wanted was a great adventure for me. I was able to eat, study, and party at my leisure, with little consequence other than poor grades. Four years, one dorm room, and two apartments later, I have realized that my freedom came with a price. Having two other roommates, and no parents to tell us to clean, living has become somewhat of mess. The biggest cause of this mess has been aluminum cans, scattered everywhere.

I created the Aluminator for this sole reason. The Aluminator is designed to roam a living area, searching for red aluminum cans. Once found, it will grab the can with it's claws and bring the can to a trash area. This makes putting the cans in the garbage a much simpler task, that both my roommates and I are willing to do.

When I first signed up for IMDL I imagined that robotics would be an interesting and fun class that would utilize my understanding of computers and electronics. The class gave me a more in depth look as well as hands on experience with microprocessors. Programming in C was also very new to me. I had taken a class in C++ as a freshman, but I never expanded my knowledge in it. Overall, the class taught me an enormous amount in one semester, from programming in C to integrating systems together. This paper is an in depth look of all the tools and hardware I used to create the Aluminator

Integrated System

For my design I chose to use the Progressive MegaAVR DEVelopment board with an Atmel ATmega 323 microprocessor. The processor comes equipped with 32kB of flash RAM, 1k of EEPROM and 2K of SRAM. I found this to be suitable for my needs and was relatively inexpensive. The chip had 1 8-bit A/D conversion channel, 4 pulse width modulators, and a full duplex USART, all of which I utilized.

The Progressive board was relatively user friendly with a "boot-loader" program already programmed into the chip that mad programming the chip simple with the use of an interface designed by Progressive Resources. However, I did discover a major problem in the design of the board, which caused me many headaches. When interfacing with the USART, using the designated Transmit and Receive pins other than on the actual DB9 serial connector proved to be impossible, due to the fact they are not actually tied to the receive and transmit pins on the processor. Figure 1 is part of a schematic of the Progressive Dev board, and what I found to be wrong with it. This design flaw cost me two weeks of work when interfacing the CMU Camera, not to mention tons of frustration. After figuring out this out I would definitely not recommend this board to anyone who is taking IMDL in the future.

I interfaced three Sharp range sensor, a voltage divider network and three Lite-On Infrared detector cans using the A/D port (port A) on the board. I also interfaced a CMU Camera to the USART. A more in depth description of these sensors are in the sensor section of this paper.

I used the AVR Studio to program in C and compiled with AVRGCC. I chose this method mainly because it was free and my TAs, Uriel and Jason seemed to have a vast knowledge of the platform. This made support a lot simpler. Although difficult to understand at first, the GCC compiler was a useful task in interfacing my program with

my chip due to the built in include file that specified all the registers of the mega323 Chip.

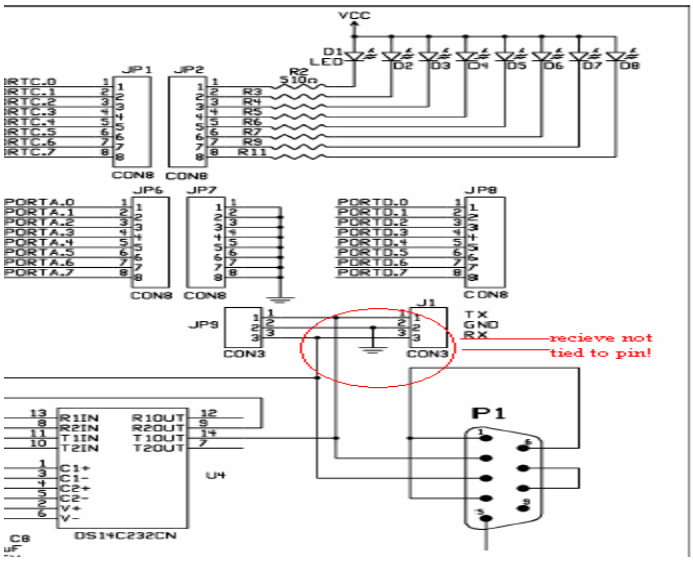

Figure 1. Progressive board diagram.

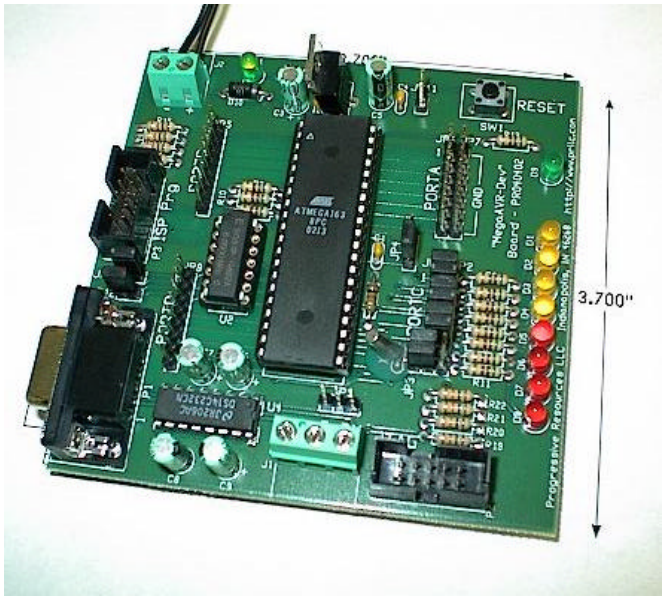

Figure 2. Progressive MegaAVR Dev board w/ Atmel 323 Chip.

Mobile Platform

When designing a board, I recognized that I would need a few features. First I would need a platform that would be easy to navigate around corners and objects, so picking a circular design suited me best. Next, I decided I needed a concave slot in my board where the can could be pulled in easily to. I designed my platform in Auto Cad 2002 and had the T-tech machine in the MIL Lab cut it out for me. The platform is 4.5" in radius and has a 2" circular indentation in the back. The robot is approximately 4" tall and the arms span out about 2.5".

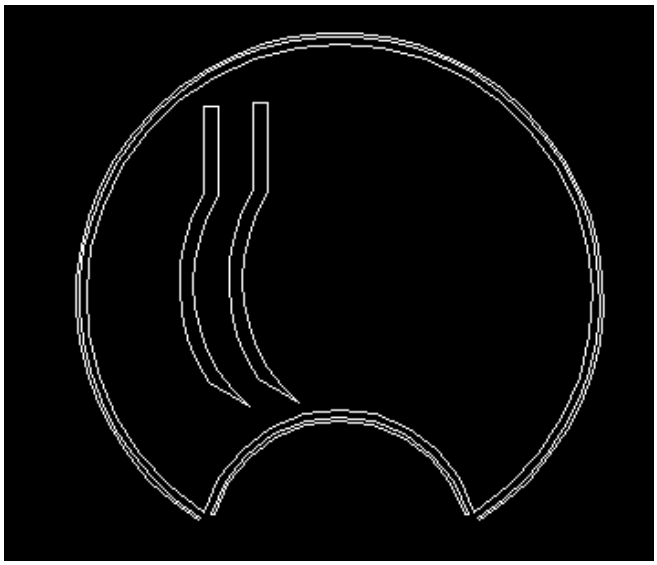

Figure 3. Main platform, skirt, and arm design.

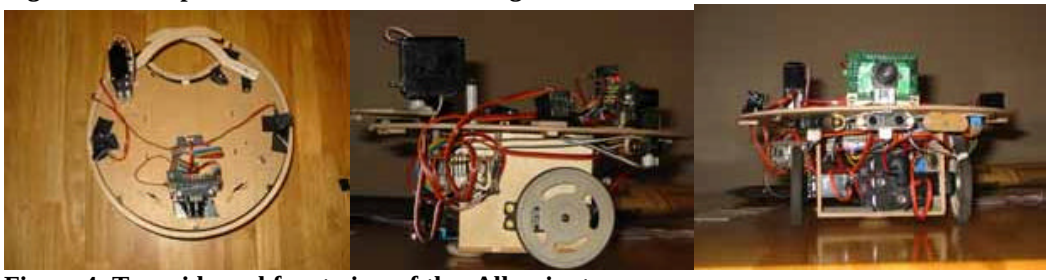

Figure 4. Top, side and front view of the Alluminator.

Actuation

The Alluminator used two standard servos for locomotion. . The servos were purchased at http://www.acroname.com and are model MX-400. In order to use these servos, I had to "hack" them by cutting off a plastic stopper on the main gear to get it to do a full 360 degree rotation. I then adjusted the potentiometer by setting it to a 12 microsecond neutral. I used two of the four pulse width modulators (PWM) on my mega323 chip to produce the proper signal to actuate the servos. The code to test the two servos for actuation can be found in Appendix A.

I also used two other two GWS pico-ball bearing servos to actuate my arms. I did not need to hack these servos because I only needed 180 degrees of rotation. I had to purchase three pico-servos because I found that they were very easy to break. The plastic gears inside the servo tend to strip very easily. I had both pico-servos working and functional with the other two PWM ports on my board. However, the night before demo day one of my micro servos caught fire in lab, so I needed to use a standard servo for one of my arms. This turned out to be a blessing in disguise, because the micro servos alone did not have enough torque to pull in the coke can and trigger the can sensor. The code to test the servos for the arms can also be found in appendix A.

Sensors

Sensor Suite

Qt.	Type	Use
3	Sharp GPD12 IR Rangers	Obstacle Avoidance
1	Sharp GPD12 IR Ranger	distance from can detection
3	Bump Switch	Obstacle detection/ Avoidance
1	Bump Lever	Can touch detection
3	Lite-ON IR Detectors	56.6kHz IR Detection
1	CMU Camera	Color Detection/Tracking

Infrared Rangers

In order for the Aluminator to avoid objects, I interfaced three Sharp GPD12 distance rangers to my A/D port. These sensors were very easy to integrate, only having to connect power, common ground and signal out to them. Once the Aluminator received a value corresponding to five inches away (approximately 60 Hex), from one of the three range sensors, it would change its path accordingly, as to avoid collision. Each sensor returns an analog value that is accurate to distance between 10cm and 80cm, which is then converted through the A/D port on the mega323 chip to a digital value. The A/D port was also very self explanatory, converting the analog voltage to a digital signal took me no time to integrate.

I also used a one more distance ranger attached to the back of my robot. This made it possible to back into the can without knocking it over. A graph of the distance versus voltage returned can be found in figure 5. The code to test the range sensors, as well as all other sensors using the A/D port, including IR detectors and bump skirt is included in appendix A.

Distance(in.)	Digital output (Hex)
3	80
4	78
5	66
6	54
7	48
8	43

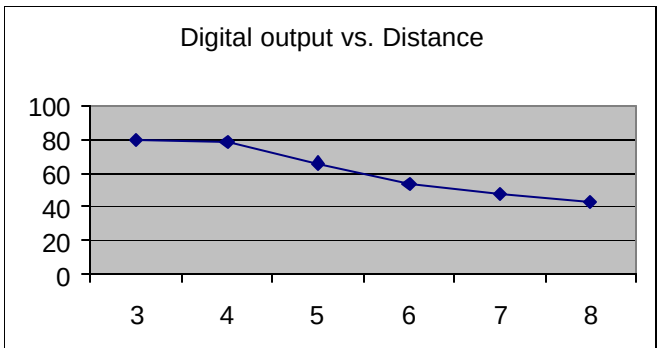

Figure 5. Sharp GPD12 Distance Ranger

Bump Switches

The Sharp GP2D12 IR Rangers take an average of the array of the distance that is seen. This makes it possible for some smaller and thinner objects to be neglected. Because of this, the Aluminator could possibly continue on its path and hit the object. In order for the robot to change its path, an array of bump switches needs to be implemented. The push switches provided by the IMDL lab will be aligned on the left, center and right of the robot and a thin layer of wood attached as a "skirt" to detect where the collision took place. The three regions (left, right and center) wires are sent through a voltage divider network as shown in Figure 6. The signal is then sent to an analog port and converted to determine which region was bumped. The code to test the bump switches is the same code used to test the IR Rangers and the Detectors, and can be found in Appendix A.

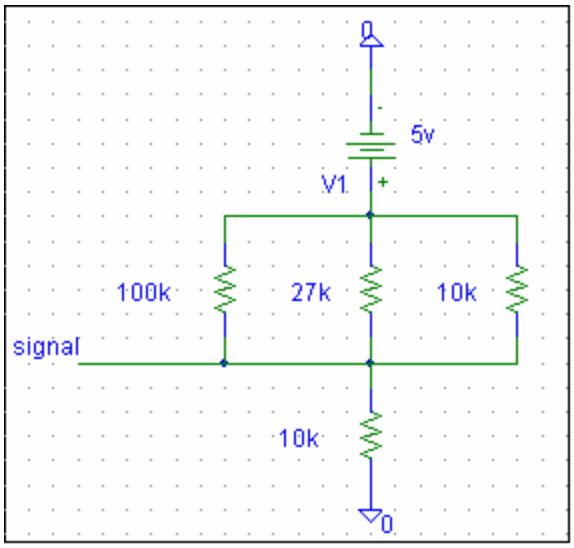

Figure 6. Voltage divider network used for bump switches.

IR Detector Cans

In order for the Aluminator to return to a trash area after retrieving a can, I needed to use an Infrared beacon to emit IR at 56 kHz. In order to detect this signal, I used three Lite-On Infrared Detector cans. These cans returned a digital signal and therefore needed to be hacked to return an analog signal. By following the instructions on Michael Hatterman's report, it was easy to hack the sensors to enable them to send analog voltages out. The can that returns the higher analog voltage is the can that is receiving the most IR. This made it simple to write code to go in the direction of the IR beacon.

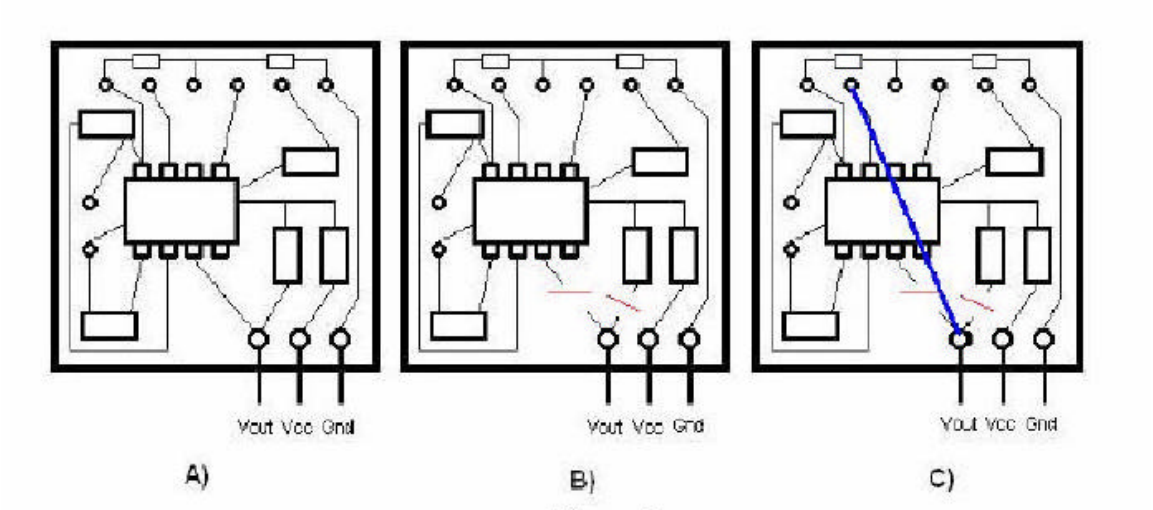

Figure 7. Diagram to hack IR sensors. Regards to Michael Hatterman.

After looking through many designs from past reports in IMDL, I could not find a schematic that was fully functional. The IR beacon I made was with the help of Steve Vander Pleog. The IR beacon is a simple circuit that utilized a 555 timer and a Darlington transistor. Ian St. John gave me the idea to use a Darlington transistor to amplify my signal. I have drawn the schematic in figure 8.

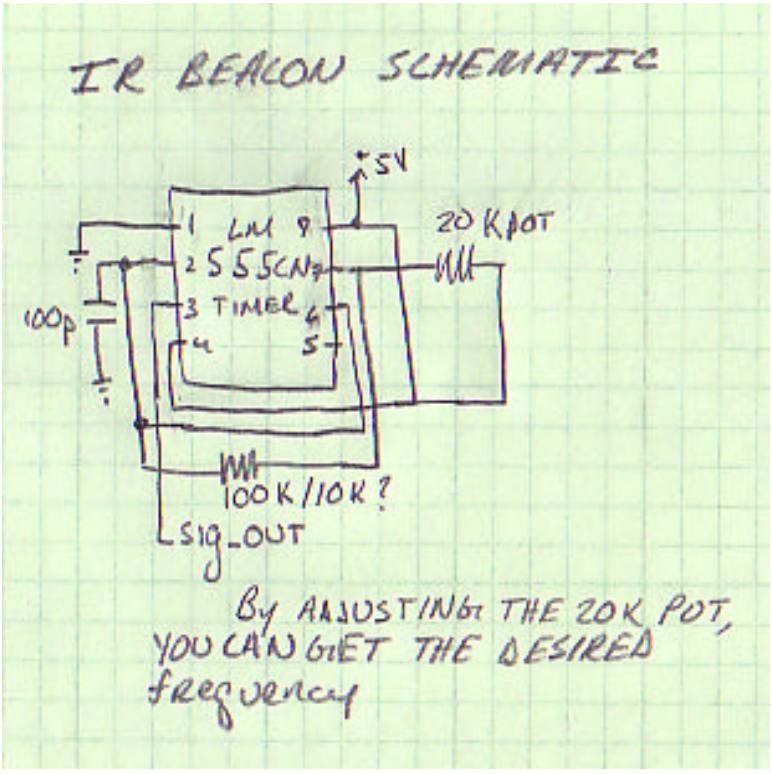

Figure 8. IR beacon Schematic designed with the help of Steve Vander Pleog.

CMU Camera

The CMU camera was used as my main sensor. I interfaced the camera with my processor in order to track the color red. I set the camera to send raw data back to the processor through the chip's USART. The camera came with a java GUI which was easy to use. In the GUI you could dump a frame in order to get the values of a certain color. You could then punch in ranges of colors and have the camera track them. I was only concerned with the middle x-coordinate because I just wanted the camera to track left and right.

Interfacing the camera became a very big headache. When interfacing the camera to the board, I encountered many problems, most of which could have been avoidable. After writing test code, I got my board to communicate with the camera by sending blinking the track light. Transmitting to the camera was simple. However, receiving packets from the camera turned out to be a headache. I found that at the current configuration, the receive complete flag was not polling, but I didn't know why. After writing four different sets of code, I realized it must be a hardware issue. Looking through the camera's, the processor's, and the board's documentation I discovered a flaw in the Progressive MegaAVR dev board. The receive pin tied to jumper nine was in fact, not tied. After making a new cable that used the DB9 serial port directly, I was able to receive packets, unflawed. This upset me greatly, because it could have been avoided had I realized it sooner. The code to configure the USART to work with the Camera can be found in appendix A.

The Aluminator would send a track color command to the camera and the camera would return an "M packet". If the camera did not see the color, it would return a 255 byte, followed by an 'M' followed by a white space. When it did see the given color, the

camera would send a 255 byte, and ‘M’, and the middle x coordinate, followed by the other coordinates.

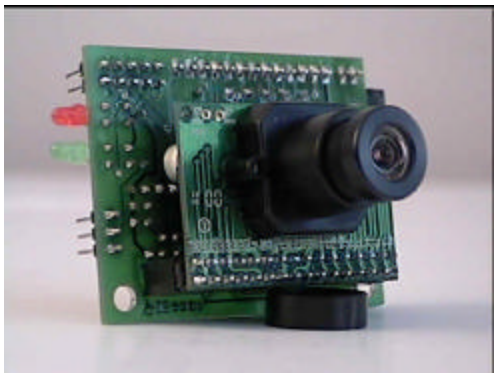

Figure 9. CMU Camera

Behaviors

The Aluminator exhibits three specific behaviors: tracking, retrieval, and disposal. Below I will discuss all three behaviors in detail.

Tracking:

When the Aluminator comes back from dispensing a can, or is first initialized, it enters this mode. He will slowly do 2 complete rotations if nothing to search for a red can. If no can is found he will move forward in a random pattern, while avoiding obstacles and stop once again to search for a can. When a can is found he will then go into retrieval mode.

Retrieval:

When the Aluminator finds a can, he enters this mode. The camera will lock on the middle of the can and the robot will approach accordingly to the object. When it becomes 4 inches away from the object, the robot will turn 180 degrees away from the can. Using a range sensor, the robot will back up towards the can until it is in grasping distance. Once the camera is in grasping range, the Aluminator closes it's arms and grabs for the can. If he misses the can, the robot runs away and goes into tracking mode to search for another can. If he grabs the can successfully, he will then go into dispensing mode.

Dispensing:

After successfully grabbing and securing a can, the Aluminator goes into dispensing mode. In this mode, the robot looks for the designated trash area. The robot will detect the beacon located at the trash area and guide itself towards the area. Once it is at a certain distance away from the beacon, the robot will open his arms and release the can. It will then turn 180 degrees and go in the opposite direction and enter tracking mode, in order to look for another can.

Conclusion

Being at the conclusion of this course, I am not sure how I learned so much in one semester. In order to be successful in this course I had to do a number of things: Learn C Programming, interface a new microprocessor, and learn how to use and interface sensors. Coming into this class, I only took EEL4744, in which we learned the Motorola MC68HC12 processor and programmed in assembly. Being an undergraduate and taking this course proved to be very time consuming. Juggling this course with all my others was a difficult task. Many hours a day were spent in the MIL lab, and lots of grey hairs were formed. Overall, the experience has been astounding and mostly, worthwhile. I can truly say that I have learned more in this class than any other At the University of Florida.

Acknowledgements

Below is a list of people that I would like to thank for their help and contributions to my success in this course in no particular order:

Uriel Rodriguez and Jason Plew – unrelenting help in the lab
Brian Ruck – Various ideas and motor actuation
Steve Vander Ploeg – IR beacon design and 555 timer schematic
Jordan Wood – Various ideas
Hubert Ho – CMU camera help
Louis and Belleza Tripician - monetary support
Dr. Arroyo and Dr. Schwartz – for giving me the opportunity to take such an interesting and enlightening class.

```
**********************************************************************
Kyle Tripician EEL 5666
C Code, compiled with AVR Studio and AVRGCC
All of the code may be used and modified for educational purposes with proper
acknowledgment.
**********************************************************************
****TEST CODES****
********A/D Initialization and test code*********
#include <io.h>
#include <sig-avr.h>
typedef unsigned char  u08;
int main( void )
{
        u08 led,i,j,k;

        outp(0x00,DDRA);
        outp(0xFF,DDRC);            /* use all pins on Portc for output */
        outp(0x80,ADCSR);           /* turn A/D on */
        outp(0x25,ADMUX);           //left adjusted, channel 6

        while(1){                   //loop conversion
        sbi(ADCSR,ADSC);                        /* start conversion */
        loop_until_bit_is_clear(ADCSR,ADSC);/* Checking if done */
        led=inp(ADCH);                  /*getting value */
        outp(led,PORTC);                //output value on portc
        }
**********PWM Initialization and test code***************
#include <io.h>
#include <sig-avr.h>
int main(void)
{
outp(0xFF,DDRD);    //set up PWM, PORTD sent send signals to motors//
outp(0xFF,DDRC);    //set outputs for leds
outp(0xA1,TCCR1A);          //8 bit PWM//
outp(0x04,TCCR1B); //prescale clock by 256//
outp(0x00,OCR1BH);//leave empty
outp(0x12,OCR1BL); //12 us neutral
outp(0x00,OCR1AH);//leave empty
outp(0x12,OCR1AL);  //12 us neutral
for(;;) {}

}
********PWM TCNT0 and TCNT2 channels test code*************
//Courtesy – Brian Ruck
int main(void)
{
outp(0xFF,DDRD);
outp(0xFF,DDRB);
outp(0x64,TCCR0);    //enable PWM in 8bit timer/counter control register0
```

```
outp(0x66,TCCR2);    //enable PWM in 8bit timer/counter control register2
outp(0x00,TCNT0);
outp(0x00,TCNT2);
outp(0x09,OCR0);     //PORTB pin 3
outp(0x23,OCR2);     //PORTD pin 7
for (;;) {}
}
```

**********Main Code**********

****globals.h****

```
#ifndef GLOBALS_H
#define GLOBALS_H

typedef unsigned char u08;
typedef unsigned int u8;
typedef          char s08;
typedef unsigned short u16;
typedef          short s16;
volatile u8 adval[8];
volatile int muxc[8];
volatile u08 irleft, irright, irmid, bump, leftc, midc, rightc, backir;
volatile u08 forw, rev, stop, sforw, srev, arms, leftm, rightm, open, close;
volatile u8 cmudat[9];
#define forw 10
#define rev 0
#define stop 5
#define sforw 7
#define srev 3
#define open 20
#define close 30
#define leftm 1
#define rightm 2
#define arms 3

#endif
```

******uart.c*******//UART initialization code for CMU CAMERA AND USART

```
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <progmem.h>
#include <sig-avr.h>
#include "globals.h"
//Initialize UART
//'baud' is the baud register divider
void uartinit(void)
{
  /* Set baud rate */
  outp(0x98,UCSRB);
```

```
  outp(0x86,UCSRC);
  outp(0x00,UBRRH);
  outp(0x09,UBRRL);
  /* Enable Receiver and Transmitter */

        // 1 stop bit
}

//Send a single byte of data
//'data' is the byte sent
void uarttransmit(unsigned char data)
{
  /* Wait for empty transmit buffer */
  while (!(UCSRA & (1<<UDRE))){}
  /* Put data into buffer, sends the data */
  outp(data,UDR);
}


//Send a given EOS terminated string
void uartstring(unsigned char * myStringIn)
{
  unsigned char *myString = myStringIn;
  unsigned char ch1;
  unsigned char gotNULL = 0;
  ch1= *myString++;
  while(!gotNULL){
    uarttransmit(ch1);
    ch1 = *myString++;
    if(ch1 == '\r'){
      gotNULL = 1;
      uarttransmit(ch1);
      }

    }
}

void caminit(void){
        uartstring("PM 1\r");           //put camera into pole mode
        delay(5000);
        uartstring("RM 3\r");           //suppress ACK NCK, and Send raw packets.
        delay(10000);
}

void blink(void){
        uartstring("L1 1\r");
        delay(3000);
        uartstring("L1 0\r");
        delay(3000);
```

```
        }

**********ad.c**********
//A/D Conversion array and obstacle avoidance code
#include <io.h>
#include <stdlib.h>
#include <progmem.h>
#include <sig-avr.h>
#include "globals.h"
#include "motors.h"

u08 dist=0x50;

void adinit(void){      //initialize the A/D
outp(0x00,DDRA);
outp(0x80,ADCSR);   //turn on A/D
sbi(ADMUX,ADLAR);         //left adjust the A/D conversion
}


void adconv(void){
int c=0x00;
int b;
        muxc[0]=0x20;                           //left adjusted channel 0//
        muxc[1]=0x21;                           //left adjusted channel 1//
        muxc[2]=0x22;                           //left adjusted channel 2//
        muxc[3]=0x23;
        muxc[4]=0x24;
        muxc[5]=0x25;
        muxc[6]=0x26;                           //left adjusted channel 3//
        muxc[7]=0x27;
        for(b=1;b<=8;b++){
                outp(muxc[c],ADMUX);
                sbi(ADCSR,ADSC);                // start conversion
                loop_until_bit_is_clear(ADCSR,ADSC);      // checking if done
                adval[c]=inp(ADCH);             //place value in
                c++;
                }
irleft=adval[0];
irmid=adval[2];
irright=adval[1];
bump=adval[3];
rightc=adval[4];
midc=adval[5];
leftc=adval[6];
backir=adval[7];
outp(backir,PORTC);
}
```

```
void bumps(void){
		if(bump>0x0A && bump<=0x2D){		//middle bump
			motor(leftm, rev);
			motor(rightm,rev);
			delay(50000);
			motor(leftm, stop);
			motor(rightm, forw);
			delay(50000);
		}
		if(bump>0x2D && bump<=0x3D){		//left bump
			motor(leftm, rev);
			motor(rightm,rev);
			delay(50000);
			motor(leftm, forw);
			motor(rightm, stop);
			delay(50000);
		}
		if(bump>0x3D && bump<=0x59){		//mid & left bump
			motor(leftm, rev);
			motor(rightm,rev);
			delay(50000);
			motor(leftm, forw);
			motor(rightm, stop);
			delay(50000);
		}
		if(bump>0x59 && bump<=0x9D){		//right bump
			motor(leftm, rev);
			motor(rightm,rev);
			delay(50000);
			motor(leftm, stop);
			motor(rightm, forw);
			delay(50000);
		}
		if(bump>0x9D && bump<=0xA4){		//middle & right bump
			motor(leftm, rev);
			motor(rightm,rev);
			delay(50000);
			motor(leftm, stop);
			motor(rightm, forw);
			delay(50000);
		}
}

void range(void){
	/*if(irmid<dist && irleft<dist && irright<dist){
		motor(leftm, forw);
		motor(rightm, forw);
		delay(5000);
		}*/
```

```
        if(irmid<dist && irleft<dist && irright>=dist){
                motor(leftm, stop);
                motor(rightm, forw);
                delay(5000);
        }
        if(irmid<dist && irleft>=dist && irright<dist){
                motor(leftm, forw);
                motor(rightm, stop);
                delay(5000);
        }
        if(irmid<dist && irleft>=dist && irright>=dist){
                motor(leftm, forw);
                motor(rightm, forw);
                delay(5000);
        }
        if(irmid>=dist && irleft<dist && irright<dist){
                motor(leftm, rev);
                motor(rightm, forw);
                delay(5000);
        }
        if(irmid>=dist && irleft<dist && irright>=dist){
                motor(leftm, rev);
                motor(rightm, forw);
                delay(5000);
        }
        if(irmid>=dist && irleft>=dist && irright<dist){
                motor(leftm, forw);
                motor(rightm, rev);
                delay(5000);
        }
        if(irmid>=dist && irleft>=dist && irright>=dist){
                motor(leftm, rev);
                motor(rightm, rev);
                delay(5000);
                motor(leftm, rev);
                motor(rightm,stop);
                delay(5000);
        }
}

void beacon(void){
        if(midc>0x20 && midc>leftc && midc>rightc){
                motor(leftm, forw);
                motor(rightm, forw);
                delay(10000);
        }
        else if(rightc>0x20 && rightc>leftc && rightc>midc){
                motor(leftm, forw);
                motor(rightm, stop);
```

```
                delay(10000);
                }
        else if(leftc>0x20 && leftc>rightc && leftc > midc){
                motor(leftm, stop);
                motor(rightm, forw);
                delay(10000);
        }
        else{
                motor(leftm,sforw);
                motor(rightm,srev);
                }
        }
```

**********Motors.c**********

```
//Servo initialization code, and motor actuation code
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <progmem.h>
#include <sig-avr.h>
#include "globals.h"

void motorinit(void){
        /* set up portd for PWM from counter 1. this is for my motors.*/
outp(0xFF,DDRD);    //set up PWM, PORTD sent send signals to motors//
outp(0xFF,DDRC);    //set outputs for leds
outp(0xA1,TCCR1A);          //8 bit PWM//
outp(0x04,TCCR1B); //prescale clock by 256//
outp(0xFE,DDRB);
outp(0x64,TCCR0);    //enable PWM in 8bit timer/counter control register0
outp(0x66,TCCR2);    //enable PWM in 8bit timer/counter control register2
outp(0x00,TCNT0);
outp(0x00,TCNT2);
}


void motor( u08 x, u08 y) {
        // x is which motor(1 left, 2 right), y is speed -100, 0, or 100
        if(x==rightm){
                if(y==rev){
                        outp(0x00,OCR1AH);//leave empty
                        outp(0x23,OCR1AL);//full reverse
                }
                else if(y==srev){
                        outp(0x00,OCR1AH);//leave empty
                        outp(0x13,OCR1AL);//slow rev
                }
                else if(y==stop){
                        outp(0x00,OCR1AH);//leave empty
```

```
                outp(0x12,OCR1AL);//15us neutral
        }
        else if(y==sforw){
                outp(0x00,OCR1AH);//leave empty
                outp(0x11,OCR1AL);//slow forward
        }
        else if(y==forw){
                outp(0x00,OCR1AH);//leave empty
                outp(0x0D,OCR1AL);          //full forward
        }



}
else if(x==leftm) {
        if(y==rev){
                outp(0x00,OCR1BH);//leave empty
                outp(0x0D,OCR1BL);          //full reverse
        }
        else if(y==srev){
                outp(0x00,OCR1BH);//leave empty
                outp(0x11,OCR1BL); //slow rev
        }
        else if(y==stop){
                outp(0x00,OCR1BH);//leave empty
                outp(0x12,OCR1BL); //15us neutral
        }
        else if(y==sforw){
                outp(0x00,OCR1BH);//leave empty
                outp(0x13,OCR1BL); //slow forward
        }
        else if(y==forw){
                outp(0x00,OCR1BH);//leave empty
                outp(0x23,OCR1BL); //full forward
        }

}
else if(x==arms){
        if(y==open){
                outp(0x23,OCR0);
                outp(0x0E,OCR2);
        }
        else if(y==close){
                outp(0x0A,OCR0);
                outp(0x15,OCR2);
        }
        else if(y=stop){
                outp(0x12,OCR0);
                outp(0x11,OCR2);
```

```
                }
        }
}
```

**********Main Code********alum.c**********

```
#include <io.h>
#include <string.h>
#include <interrupt.h>
#include <stdlib.h>
#include <progmem.h>
#include <sig-avr.h>
#include "globals.h"
#include "uart.h"
#include "ad.h"
#include "motors.h"
#include "delay.h"

volatile u8 temp;
volatile u8 i=0;

u8 j=0;
u8 k=0;
SIGNAL(SIG_UART_RECV){  //Recieve complete interrupt function
        temp=inp(UDR);
        if(temp != 0x3A){
                if(temp == 0x20 || i==9){
                        i=0;
                }
                else if(i==0){
                        if(temp == 0xFF){
                                cmudat[i]=temp;
                                i++;
                        }
                }
                else if(temp !=0x20 && i<9){
                        cmudat[i]=temp;
                        i++;
                }
        }
}
void track(void){
        if(cmudat[2]>0x01 && cmudat[2]<0x15){
                motor(leftm,sforw);
                motor(rightm,stop);
        }
```

```
        if(cmudat[2]>=0x15 && cmudat[2]<=0x3C){
                motor(leftm,sforw);
                motor(rightm,sforw);
        }
        if(cmudat[2]>0x3B && cmudat[2]<=0x80){
                motor(leftm,stop);
        motor(rightm,sforw);
                }
}
void scan(void){
        motor(leftm,sforw);
        motor(rightm,srev);
        delay(9000);
        motor(leftm,stop);
        motor(rightm,stop);
        delay(40000);
        }

void turn(void){
        motor(leftm, sforw);
        motor(rightm,srev);
        motor(arms,open);
        delay(190000);
        delay(190000);
        delay(115000);
        motor(leftm, stop);
        motor(rightm, stop);
        }
void scram(void){
        motor(leftm,forw);
        motor(rightm,rev);
        delay(90000);
        for(j=0;j<20000;j++){
                adconv();
                bumps();
                range();
                motor(leftm,forw);
                motor(rightm,forw);
                motor(arms,close);
        }
        motor(arms,stop);
        }
int main(void){
        outp(0xFF,DDRC);    //set outputs for leds
        outp(0x00, DDRB);
        outp(0x00,PORTB);
        adinit();
        motorinit();              //initialize servos, and arms.
        delay(50000);
```

```
	uartinit();		//enable uart 38400 baud, no parity, 1 stop bit, 8 data bits
	caminit();		//initialize camera, raw mode, poll mode
	sei();
	delay(20000);
while(1){
	//local variables
	volatile u8 lost=1;
	volatile u8 around=0;
	volatile u8 time=0x1C;
	volatile u8 imclose = 0;
	volatile u8 backup = 0;
	volatile u8 lever=0x00;
	volatile u8 gotir=0x00;
	volatile u8 aopen=0x00;
	adconv();
	while(lost){			//search mode...spin in circles for a certain amount
of time,
					//if nothing found, move to another spot and spin
again
		adconv();		//convert all A/D channels
		bumps();
		uartstring("TC 100 250 16 16 16 16\r");
		outp(cmudat[2],PORTC);
		scan();
		if(cmudat[2] > 0x01){
			motor(leftm,stop);
			motor(rightm,stop);
			lost=0;
		}
		else{
		time--;
		if(time <=0x01){	//move to another location if nothing is found
				time= 0x1C;
				motor(leftm,forw);
				motor(rightm,forw);
				for(j=0; j<40000; j++){
					adconv();
					range();
					bumps();
					motor(leftm,forw);
					motor(rightm,forw);
				}
			}
		}
	}
	while(!imclose){
		uartstring("TC 150 250 16 16 16 16\r");
		outp(cmudat[2],PORTC);
		adconv();
```

```
		track();
		if(irmid>=0x77){
		motor(leftm, stop);
		motor(rightm,stop);
		imclose=1;
		adconv();
			while(irmid>0x70){
			adconv();
			motor(leftm,srev);
			motor(rightm,srev);
			}
		motor(leftm, stop);
		motor(rightm,stop);
			}
		delay(10000);
		}

while(around==0){
		turn();
		around=0x01;
		}

while(!backup){
adconv();
		if(backir<0x75){
			motor(leftm,srev);
			motor(rightm,srev);
		}
		else{
			motor(leftm,stop);
			motor(rightm,stop);
			backup=1;
		}
}
while(!aopen){
		motor(arms, close);
		delay(190000);
		delay(190000);
		delay(105000);
		if(PINB <= 0xFE){
		aopen=0x01;
		}
		else{
			scram();
			lost=0x01;
			around=0;
			time=0x1C;
			imclose = 0;
			backup = 0;
```

```
                        lever=0x01;
                        gotir=0x01;
                        aopen=0x01;
                }
        }
        while(!gotir){
        adconv();
        bumps();
        beacon();
                if(midc>=0x81){
                        gotir=1;
                        motor(leftm, stop);
                        motor(rightm,stop);
                        motor(arms,open);
                        delay(60000);
                        motor(arms,stop);
                        scram();
        }

        }
        }
}
```