IMDL Final Report

James Landry
December, 10 2002

EEL5666
A. Antonio Arroyo

Table of Contents

Abstract

I am proposing to build a robot that will locate a rope, wooden dowel or other similar device. It will attach itself and ascend until it recognizes that it cannot climb anymore. It will measure the distance that it has climbed, and report the distance via speech. It will then descend safely to the bottom where it will detach itself and go about looking for another similar device to climb. It will carry out obstacle avoidance, ignoring any obstacles that it is unable to climb.

Executive Summary

When designing Tarzan, my main design consideration was to have a lot to work with at a minimal cost. Meaning that I wanted to have the best of speed, torque, memory, at the least cost of weight, being that it was gong to be climbing, and money, being that I am a broke college student. The Botgoodies AM128 development board, fig. 1, was chosen based on its massive capabilities in a compact, light package. It has a 5 V voltage regulator, barrel type idiot proof power connector, and a reset switch all built in. The processor has 128 Kbytes program flash (never fill that up), 4KByte SRAM, 4KByte EEPROM, 8 channel 10 Bit A/D converter, 6 ports (32 I/O, 8 output, 8 input), external and internal interrupt sources (timer,comp,etc ..), "and-a-partridge-in-a-pear-tree."

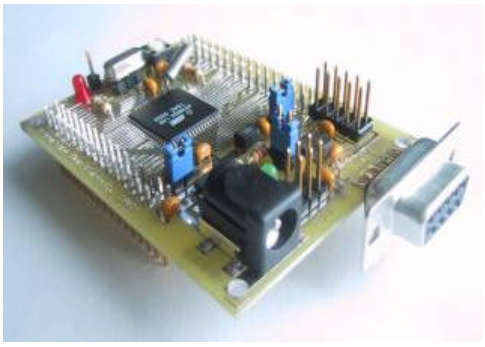

Fig. 1Botgoodies ATmega 128 board

The platform was designed around the heaviest components, to keep the robot balanced when climbing up and down the rope. I also wanted to keep the platform simple, to cut down on the weight (see fig. 2). Therefore it was only a single sheet of wood, with the components mounted on the top and bottom. I chose a circle to allow for a bump skirt to be allowed for bump detection, and to allow the components to be oriented around the center of the bot, where the rope was going to be during climbing, also for balance.

fig.2 platform design

For rope detection and obstacle avoidance, I wanted a sensor that would allow for very close detection and was very easy to use. Therefore I chose the Sharp GP2D120. It produces an analog voltage corresponding to distances between 4 to 30 cm.

For motion of the robot, I chose two hitec HS-303's, for their small size and adequate torque. For climbing I needed something powerful enough to take the entire platform and integrated system to the top. For this task I chose the Hitec HS-805BB. There really is no substitute. Before building the actual robot I designed a test fixture to see if it would have sufficient torque, and it supported more than enough weight (see fig 3).

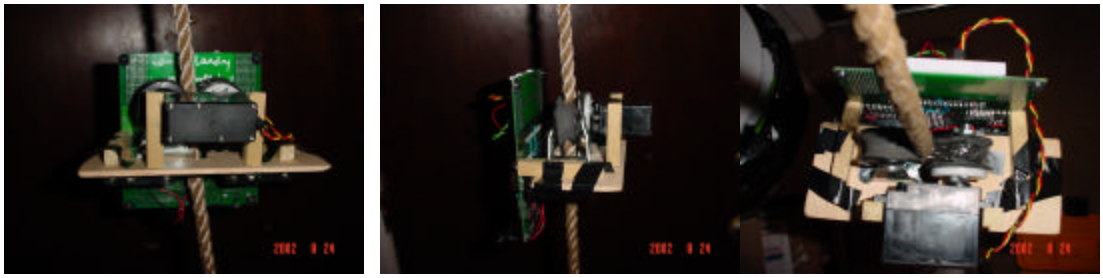

fig. 3 rope climbing test fixture

To convey information (ie: distance climbed), I opted for speech, because it's both simple and entertaining. I used the Quadravox Voice Output Module. It has Digital volume control, built in 0.3 Watt amplifier, and 240 prerecorded digital audio phrases. 5/13

To measure the distance climbed, I used a wheel encoder composed of a Hamatsu IR emitter detector pair and a disc of alternating white and black lines. The processor counted the pulses and determined the distance climbed.

All of this together comprises Tarzan, the autonomous rope detection and climbing unit. Coming soon to a store near you.

Introduction

I have been looking forward to this class since I was introduced to the class in eel4744 with Dr. Schwartz. Since that class, I have been very interested in microprocessor controls and digital signal processing, and would like to pursue a career in this area. I had a difficult time deciding on an idea for this IMDL project. I wanted to do something different and interesting. I came across a website of an annual contest for rope climbing robots. The contest only required that the robot climb the rope. It did not state that it had to find the rope first. So "bam", I decided to kick it up a notch and design and build an autonomous mobile agent that could locate and climb a rope. This paper discusses the integrated system necessary to accomplish this task, including the necessary sensors, actuation and design considerations.

Integrated System

The system will include an atmel ATmega 128 microprocessor, two Hitec HS-303 servos for movement (weighs 1.7oz with 3.7 kg-cm torque), a Hitec HS-805BB servo for the climbing device (weighs 4.2oz with 22.96 kg-cm torque), five AA batteries for power, and three Sharp GP2D120 IR emitter detector pairs for obstacle avoidance, buttons and a bump skirt for bump detection, a break beam sensor to detect the device to be climbed, and a shaft encoder using the hamamatsu emitter detector pair to measure the distance climbed.

Mobile Platform

The mobile platform will be much like a compact version of the TJ robot with the heavy components stacked, to keep the weight close to the climbing apparatus, (see fig 2). This is done to keep the weight to a minimum, and keep the weight close to the rope to cut down on undesired movement of the robot while climbing. It will be a round, single sheet of wood, about 3-4 inches in diameter. The platform will be designed around its largest component, the HS-805 servo. There will be an opening on the front to allow itself to attach to the device to be climbed. It will be a "tail dragger" with two Hitec HS-303 servos on the bottom of the board for movement. The HS-805BB servo, used for climbing, will be located on the top of the board. The atmel board, electronics and batteries will be located wherever they best balance the platform for climbing.

Actuation

For movement two HS-303 servos have been chosen. For climbing, the HS-805BB combined with a spiked wheel was chosen (fig 5). They were chosen because they are the best compromise of torque, weight and cost. The 805 attaches to the rope via a servo mechanism (fig 4).

fig 4 servo mechanism to attach to rope

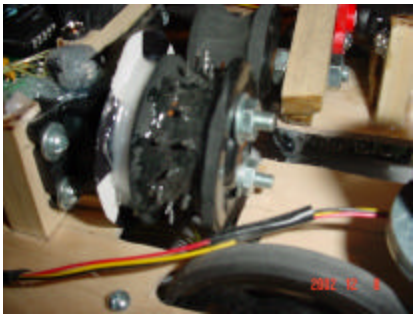

fig 5 rope climbing wheel

Sensors

IR

Sharp GP2D120 IR emitter detector pairs will be used for distance measurement. These sensors will be used for object avoidance and to detect when the robot cannot climb anymore. These sensors create an analog voltage corresponding to a distance. They can detect distances of 4 to 30 cm.

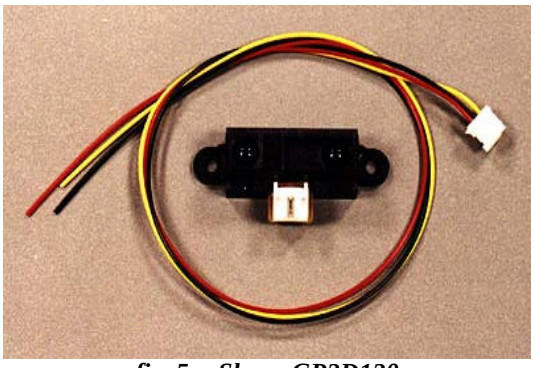

fig. 5 Sharp GP2D120

Bump Switches

There are 4 button switches surrounded by a single ring for collision avoidance, like that on the TJ robot.

Rope Locater (Special Sensor)

My special sensor is a sensor to detect if an object is capable of being climbed. In my demonstration, I am using a rope. The sensor uses two Sharp GP2D120 IR emitter detector pairs, same used for obstacle avoidance, positioned in such a configuration to allow a rope to become undetected if positioned in a one half inch area between the sensors, allowing the robot to approach the rope in such a way to climb it (see figure 2). It will only allow objects of the ropes diameter, about one and a half inches, to pass into the climbing mechanism, objects bigger than this will be handled by the obstacle avoidance subroutine.

The code works as follows (code fragment id figure 3). If an object enters the field of vision of one of the IR detectors, than the robot attempts to position it in the blind spot by turning until it disappears. It then continues forward. If the object is again detected by one of the two IR detectors, then it is again realigned into the blind spot. It then continues forward until the rope is passed into the climbing area where a break beam sensor is triggered to recognize that it is ready to be climbed.

fig. 6 Configuration of sensors to allow blind spot.

```
//NB! This fragment was taken from main loop, polling all sensors to determine behavior
else if((read_adc(3)  > 500) && (read_adc(6)  < 500))  //right front sensor triggered
{
            OCR1AL=0;                           //stop
            OCR1BL=0;
            delay_ms(50);                       //wait 50 ms

           while((read_adc(3)  > 500) && (read_adc(6)  < 500))  //while right front sensor triggered, turn left
           {
                  OCR1AL=100;      //turn left, till sensor does not detect anymore then continue straight
                  OCR1BL=0;
           }
           OCR1AL=0;                    //stop
           OCR1BL=0;
           delay_ms(50);                //wait 50 ms
 }
else if((read_adc(6)  > 500) && (read_adc(3)  < 500))  //left front sensor triggered
{
            OCR1AL=0;                    //stop
           OCR1BL=0;
           delay_ms(50);                //wait 50 ms
          while((read_adc(6)  > 500) && (read_adc(3)  < 500))    //while left front sensor triggered, turn right
          {
                  OCR1AL=0;            //turn right, then continue straight
                  OCR1BL=100;
           }
           OCR1AL=0;                    //stop
           OCR1BL=0;
           delay_ms(50);                //wait 50 ms.
}
```

Fig. 7 code fragment

Wheel Encoder

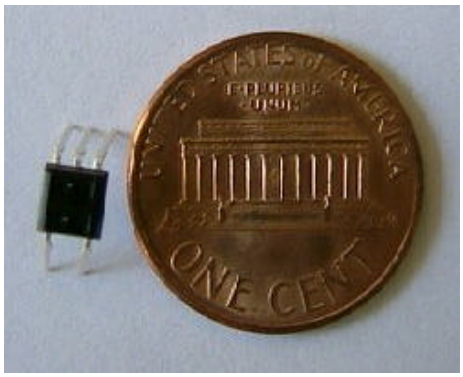

fig. 8 Hamamatsu P5587 Photo-reflector

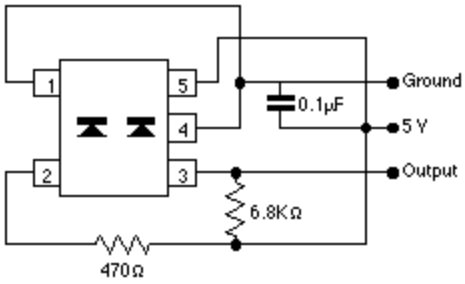

fig. 9 Typical Circuit Configuration

I am using a wheel encoder to calculate how far the robot has climbed up the rope. The climbing wheel has a disc fixed to it with alternating black and white segments. The Hamamatsu photo-reflector package contains an infrared ("IR") LED and a matching IR phototransistor, both mounted on the top of the device in such a way that the phototransistor will detect reflected IR light emitted from the LED when a bright surface (such as white card) is positioned within a few millimeters. As the wheel turns, and the photo-reflector "sees" the white segments between the black segments, the phototransistor outputs a digital pulse train. By counting the pulses, and knowing the number of segments and diameter of the wheel, the robot can compute distance traveled by the wheel.

Break Beam Sensor

The IR Break Beam is used to detect when the rope is in a position to attach to. It is a simple IR Emitter LED and IR Detector Transistor pair. Heat shrink tubing is used to collimate the light from the emitter and aim it directly at the detector. When the beam is broken the voltage raises from 3 volts to almost 5 volts.

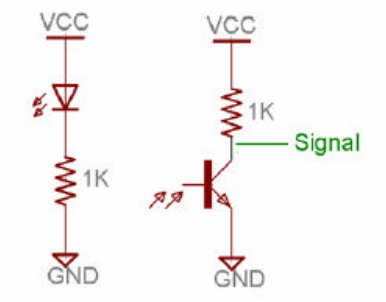

fig. 10 Break Beam Circuit Diagram

Behaviors

The robot wonders until it finds an apparatus to climb, recognizes that it can climb it, and ascends it till it cannot anymore. It measures the distance climbed and relays the distance via speech. It will then descend safely where it detaches and continues looking for another apparatus to climb.

Conclusion

From the beginning I have encountered many problems and learned many important lessons.

Originally I had purchased an ATmega 103 development board from Aikida. I never successfully accomplished anything with this board. I don't know what the problem was. Akida willingly sent me an additional board that was "loaded with a working program." I still was unable to get this board to work. I returned this board and purchased the current Botgoodies board, which successfully ran after 5 mins. I also was never able to get the CMU cam to communicate with my processor. There was 3 weeks left until the deadline, so I chose an alternative and decided to come back to it if time permitted. It didn't. My alternative was using IR to detect the rope. I originally had four IR pairs on the front of my robot, but two burnt out two days before demo day. I did not want to spend any more money on this project, so I struggled to get my algorithm working with only two. It worked. My biggest problem, was getting the bot to grip the

rope to climb. I tried to mechanisms. I first tried to use a push-pull servo assembly, but this did not produce enough tension to grip the rope with the 805. My next alternative was a lifting and lowering trap door approach. This produced enough tension, but when the robot began climbing, it almost twisted the platform in half. I guess it produced too much tension. I tried sanding down components and rearranging components, but nothing could fix the problem.

In conclusion, I accomplished a majority of my tasks. The robot does in fact locate a rope, and could in fact climb if it could attach to the rope, and can measure a distance climbed with the wheel encoder and communicate the information via speech. It can demonstrate obstacle avoidance and tell if an object is ready to be climbed. I should have built a slightly stronger platform, and developed a better way to attach to the rope. This lab taught me a lot of valuable lessons, and allowed me to demonstrate all that I have learned. This was one of the most valuable experiences I have had here at the University of Florida. One that will remain with me long into my career.

Documentation

Parts/Part Info:
-Acroname, http://www.acroname.com/
-The Robot Store, http://www.robotstore.com/
-Lynxmotion, http://www.lynxmotion.com/

Advice/Suggestions:
-Fall 2002 IMDL class
-Prof. A. Antonio Arroyo
-Prof. Eric Schwartz
-Uriel Rodriguez
-Jason Plew

Appendices

```
/*****************************************

Project : Tarzan
Version : 10.5
Date    : 11/10/2002
Author  : James Landry
Company :
Comments:


Chip type           : ATmega128
Program type        : Application
Clock frequency     : 14.7456 MHz
Memory model        : Small
Internal SRAM size  : 4096
External SRAM size  : 0
Data Stack size     : 1024
*****************************************/

#include <mega128.h>
#include <delay.h>
#include <stdlib.h>
#include <math.h>
#define ADC_VREF_TYPE 0x40

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)



// Get a character from the USART1 Receiver
#pragma used+
char getchar1(void)
{
char status,data;
while (1)
      {
      while (((status=UCSR1A) & RX_COMPLETE)==0);
      data=UDR1;
      if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
        return data;
      };
}
#pragma used-
```

```
// Write a character to the USART1 Transmitter
#pragma used+
void putchar1(char c)
{
while ((UCSR1A & DATA_REGISTER_EMPTY)==0);
UDR1=c;
}
#pragma used-


// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
//check ADC port 4 to see if break beam sensor has been broken, this is to signal that a rope
//has been passed into the climbing area
unsigned int break_beam()
{
        if(read_adc(4)  > 600)
        {
                return 1;
        }
        else
        {
                return 0;
        }

}


void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=Out Func6=Out Func7=Out
// State0=T State1=T State2=T State3=T State4=T State5=0 State6=0 State7=0
PORTB=0x00;
DDRB=0xE0;

// Port C initialization
```

```
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out Func7=Out
// State0=0 State1=0 State2=0 State3=0 State4=0 State5=0 State6=0 State7=0
PORTD=0x00;
DDRD=0xFF;

// Port E initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTE=0x00;
DDRE=0xFF;

// Port F initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
// State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func0=In Func1=In Func2=In Func3=In Func4=In
// State0=T State1=T State2=T State3=T State4=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 3.906 kHz
// Mode: Normal top=FFFFh
// OC1A output: Toggle
// OC1B output: Toggle
// OC1C output: Toggle
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0xA1;
TCCR1B=0xA1;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
```

```
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
TCCR3A=0xA1;
TCCR3B=0xA1;
TCNT3H=0x00;
TCNT3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// USART1 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART1 Receiver: On
// USART1 Transmitter: On
// USART1 Mode: Asynchronous
// USART1 Baud rate: 9600
UCSR1A=0x00;
UCSR1B=0x18;
UCSR1C=0x06;
UBRR1L=0x5F;
UBRR1H=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AVCC pin3
// ADC High Speed Mode: Off
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x85;
SFIOR=0x00;

//quadravox initialisation , set volume (FC) to level 31 (highest 0-31)
putchar1(0xF0);
while(!PINA.1)
{}
putchar1(0xFC);
while(!PINA.1)
{}
putchar1(0x31);
while(!PINA.1)
{}
 delay_ms(2000);
            while (1) {
            int i,j = 0;                //initialize variables
            putchar1(0xF0);             //initialize quadravox voice module
             while(!PINA.1)
            {}                                  //turn the volume to max, level 31
            putchar1(0xFC);
             while(!PINA.1)
            {}
            putchar1(0x31);
             while(!PINA.1)
            {}




      /*while(1)  //used for testing desired system
       {

                        if(PINA.0)
                        {
                              PORTD.5 = 1;
                        }
                        else
                        {
                                    PORTD.5 = 0;
                        }
```

```
                if((!PINC.0) )
    {
        PORTD.5 = 1;
    }
    else
    {
                PORTD.5 = 0;
    }
} */

    delay_ms(5000);
while(1)  {




//pushbutton:
    //back       --> adc(1)
    //front right --> adc(2)
    //front left  --> adc(7)
//right front sensor is adc(3)
//left front sensor is adc(6)
    //top sensor is adc(5)

    if((read_adc(3)  > 600) && (read_adc(6)  > 600))//right and left sensor triggered
    {                                       //or right or left push button triggered
     //if yes, enter loop to avoid object.
                OCR3AL=0;
          OCR1AL = 0;     //reverse
                OCR1BL = 0;

                OCR1AL = 100;      //reverse
                OCR1BL = 100;
                delay_ms(2000);    //wait 2 seconds
                if(fmod(rand(), 1000)> 499)  //create random number to decide if left or right...
                {
                        OCR1AL = 0;        //turn right, then continue straight
                        OCR1BL = 100;
                delay_ms(4000);    //wait 1.5 seconds
    }
    else
    {
                OCR1AL = 100;         //turn left, then continue straight
                        OCR1BL = 0;
                delay_ms(4000);    //wait 1.5 seconds
    }
    }
    else if((read_adc(3)  > 500) && (read_adc(6)  < 500)) //right front sensor triggered
    {       OCR1AL=0;         //stop
                OCR1BL=0;
                delay_ms(50);

                while((read_adc(3)  > 500) && (read_adc(6)  < 500))
                {
                OCR1AL=100;        //turn left, till sensor does not detect anymore then continue straight
                        OCR1BL=0;
```

```
            }
            OCR1AL=0;        //stop
                OCR1BL=0;
                delay_ms(50);
}
else if((read_adc(6)  > 500) && (read_adc(3)  < 500)) //left front sensor triggered
{       OCR1AL=0;        //stop
                OCR1BL=0;
                delay_ms(50);
       while((read_adc(6)  > 500) && (read_adc(3)  < 500))
       {
                OCR1AL=0;        //turn right, then continue straight
                        OCR1BL=100;
                }
                OCR1AL=0;        //stop
                OCR1BL=0;
                delay_ms(50);
}
else if(read_adc(5) > 200)//if object detected above by top sensor
{    //delay_ms(1000);
     //j = 1;
     OCR1AL=0;      //stop movement
                OCR1BL=0;

}
     else if(break_beam()) //break beam sensor triggered
{
     delay_ms(1000);
      OCR1AL=0;        //stop
                OCR1BL=0;
                i = 0;
                OCR3BL =0x5666; //engage clampb servo to attach to rope
     While( read_adc(5) < 200)//climb until object detected above, then descend
     {
                        OCR3A L=40;
                if(PINA.0)//count pulses from wheel encoder to determine height climbed
                {
                                i++;
                                while(PINA.0)
                                {}//make sure it only counts pulse once
                        }
                else
                {} //do nothing
     }
                OCR3AL=00;//stop and output length climbed in centimeters


                putchar1(i);
                while(!PINA.1)        //#
                {}
                delay_ms(4000);
                putchar1(119);//centimereter

                delay_ms(4000);
```

```
                while(1)
                {//climb down, decrement distance to find bottome
                        if(PINA.0)//count pulses from wheel encoder to determine when at bottom
                {
                                i--;
                                while(PINA.0)
                                {}//make sure it only counts puls e once
                        }
                else
                                {} //do nothing


                }
}

  //pushbutton:
    //back        --> adc(1)
    //front right --> adc(2)
    //front left  --> adc(7)

else if(read_adc(2) > 1000)    //front right sensor hit
{
    OCR1AL = 100;      //reverse
                OCR1BL = 100;
                delay_ms(1000);     //wait 1 second
        // PORTD.5 = 1;
                OCR1AL = 0;         //turn right, then continue straight
                OCR1BL = 100;
     delay_ms(1000);         //wait 1 second

}
else if(read_adc(7) > 1000)      //front left sensor hit
{
                OCR1AL = 100;      //reverse
                OCR1BL = 100;
                delay_ms(1000);     //wait 1 second
        // PORTD.5 = 1;
                OCR1AL = 100;         //turn right, then continue straight
                OCR1BL = 0;
     delay_ms(1000);       //wait 1 second
}
else if(read_adc(1) > 1000)//back button hit... move forward
{
    OCR3AL=0;
                OCR1AL= -40;       //move straight
                OCR1BL=-40;
}
else
{       //else if nothing, continue straight
        // PORTD.5 = 0;
          OCR3AL=0;
                OCR1AL= -40;       //move straight
                OCR1BL=-40;
    }
```

```
      //for the reverse direction
      //OCR1AL=1;        //use oc1a for left servo
      //OCR1BL=-400;     //use oc1b for right servo
      //OCR3AL=-20;  //use oc3a for climbing servo
      //for the forward direction direction
      //OCR1AL=-400;        //use oc1a for left servo
      //OCR1BL=1;     //use oc1b for right servo
         }
  };
}
```