**EEL 5666 INTELLIGENT MACHINES DESIGN LABORATORY**

# Search-n-Fetch

**By**

**Koushik Kalyanaraman**

**Instructors: Dr. A. Antonio Arroyo**

**Dr. Eric M. Schwartz**

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**UNIVERSITY OF FLORIDA**

**22nd April, 2008**

## TABLE OF CONTENTS

## Abstract

Search-n-Fetch locates and fetches balls of a given color. The robot is designed generically to fetch any ball since it has no ball-specific fetching mechanism. It uses a camera to detect the environment and Atmel MAVRIC-IIB microcontroller for the behaviors.

One of the objectives in the design is to keep the mechanism to pickup the balls as simple as possible and as generic as possible to any type of balls. The CMUCam is a handy sensor for blob detection and it is quite accurate in distinguishing contrasting colors.

The design allows to gather balls less than or equal to the size of tennis ball. All fetched balls are released near the nearest wall the robot encounters during obstacle avoidance.

I chose to design this robot because I wanted the robot to be simple, fully functional and achievable within the given time frame.

## Introduction

In a game of tennis, or cricket, fetching a ball can interrupt play for considerable lengths. The autonomous robot Search-n-Fetch can locate and fetch balls there by letting the players concentrate on their game. The ball fetching does not need arms because the balls can roll with the robot. I thought an arm mechanism would be mechanically more challenging but would not improve the performance by any factor. From videos of existing ball fetching robots I found that arms make the fetching process slow and make the robots suitable only for certain size of balls.

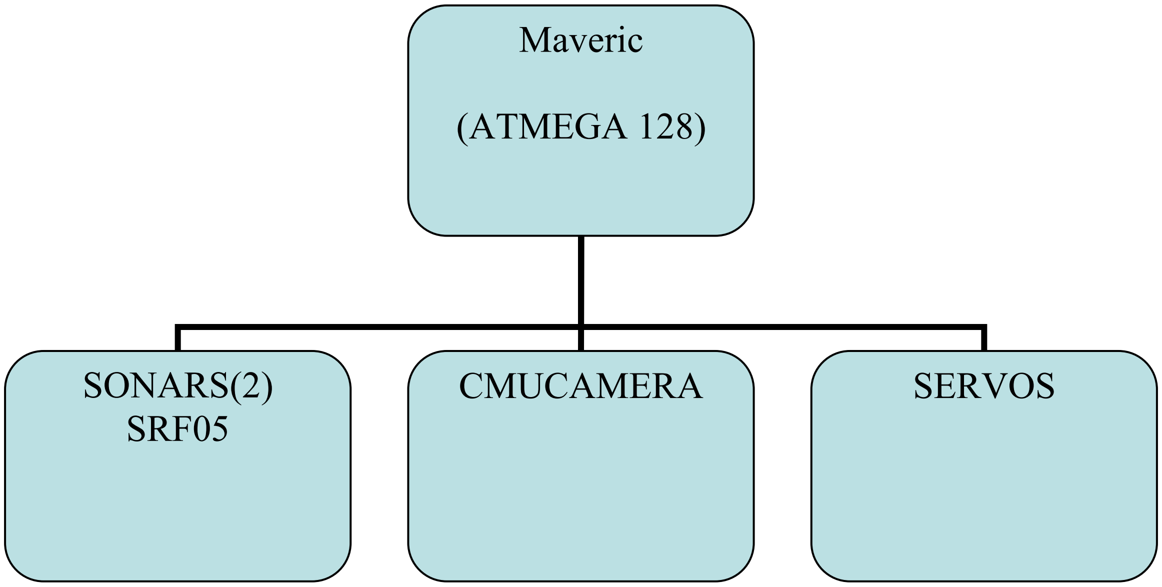
I chose to make the design as generic as possible. For the initial part of the semester, I was constantly brainstorming ideas to increase the numbers of balls that I can collect with the given size design. The final design was chosen more on account of ease of development rather than concentrating on the number of balls being collected.

This robot was designed with an intention to learn and get an insight into the field of robotics instead of focusing on trying to create something new and unimaginable (which was my initial wishful thought!). My focus was always on simplicity and elegance during the design and implementation.

## Integrated System

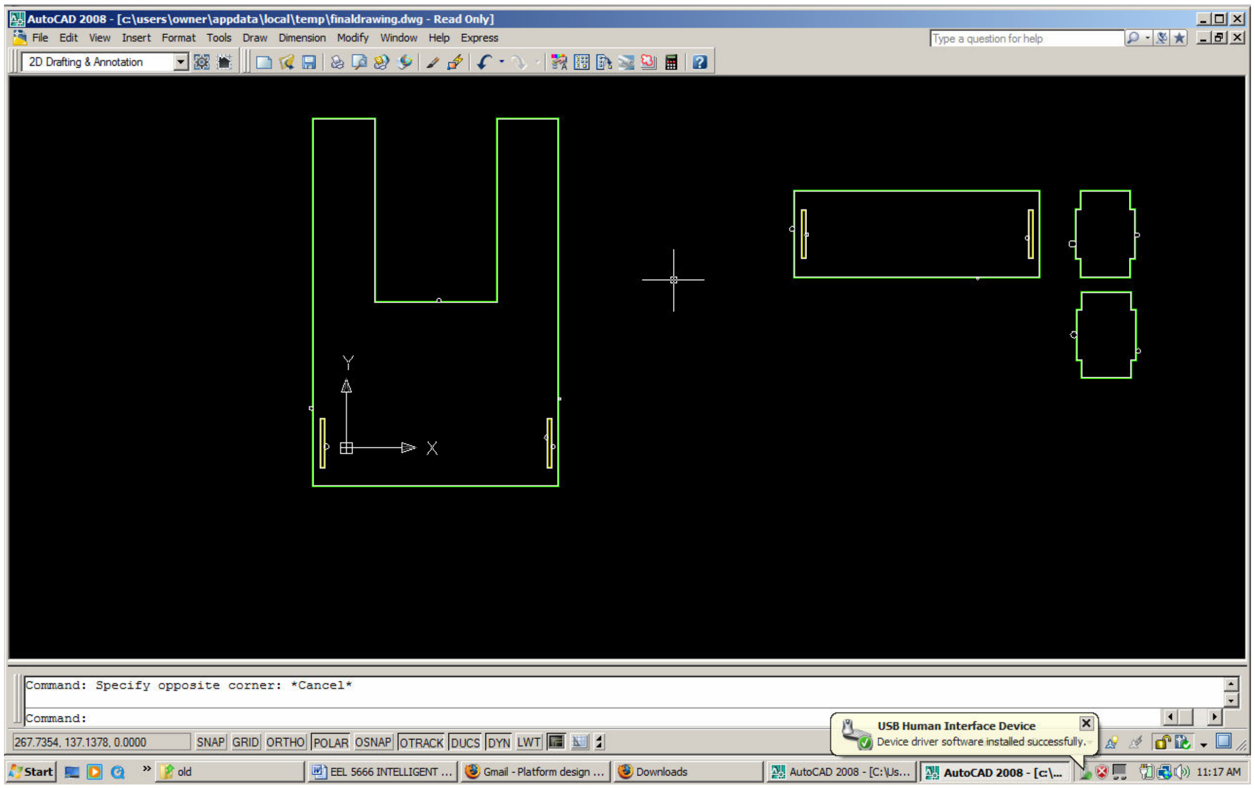
Search-n-Fetch has a MAVRIC-IIB microcontroller as the brain and a CMU camera as its. It has Sonar sensors to avoid obstacles and a simple hacked servo (as a door) to collect the balls. It also has an LCD to display the robot's status to the user.

The CMUcamera is used to detect the ball and give the co-ordinates at which the ball is, so that the robot can turn accordingly. Since, it has a two servo powered wheels at the middle of the robot and a caster wheel at its rear, its turning angle is quite big.



## Mobile Platform

The platform, for the most part is just a U shaped wooden piece with dimensions calculated so that there is just enough space to keep all the components.


AutoCAD 2008 - [c:\users\owner\appdata\local\temp\finaldrawing.dwg - Read Only]
File Edit View Insert Format Tools Draw Dimension Modify Window Help Express
2D Drafting & Annotation
Type a question for help
Y
X
267.7354, 137.1378, 0.0000
SNAP GRID ORTHO POLAR OSNAP OTRACK DUCS DYN LWT
Start
old
EEL 5666 INTELLIGENT ...
Gmail - Platform design ...
Downloads
AutoCAD 2008 - [C:\Us...
AutoCAD 2008 - [c:\...
11:17 AM

```
Command: Specify opposite corner: *Cancel*

Command:
```

USB Human Interface Device
Device driver software installed successfully.

The one big design decision which I made quite wrongly until demo day (April 16th 2008) was the placement of CMUCam. I wanted flexibility in the cmucam's position , so I planned to keep it well into the platform. This design proved to be worse for the following reasons

1) most of the camera's range is lost by seeing the balls that are already collected
2) The camera itself was placed in a thin piece of wood which was constantly shaking. This made calibrating the center values and interpreting the values from CMUCam next to impossible.

The reason why I wanted the camera to be mounted like that was mainly to experiment with all possible positions and orientations. Moreover, I had that arrangement so that I can connect another unhacked servo to the end of the rod. This servo could control the CMUCam's angle of vision. Though this arrangement of CMUCam gives way to many interesting and exciting behaviours with almost no extra coding, I couldn't figure out a way to mount it in a fixed position to get it working. (I still have the wooden piece in the platform with hope of figuring out a way to do it).

## Actuators

I started of working with DC motors. I made a soldering error which rendered one of my DC motors unusable. I figured that servo motors where cheaper and easier to program and would not affect my goals in anyway. The DC motors also needed opto-isolator circuits which I did not cut. All these factors and the lack of time, made me switch to servos instead of DC motors.

I use an unhacked servo as a door mechanism in order to retain the collected balls within the robot. I wanted to use some mechanism which does not require any actuation to act as a door- but such a design was not feasible.

One reason I am dissatisfied with the servos that I use is that, I could not get them to move straight. This flaw sometimes causes my robot to miss the ball which it is supposed to fetch.

I bought the castor wheel from http://www.pololu.com/catalog/product/955 which is a very good wheel and I would strongly recommend it for anyone who wants an elegant castor.

The PWM signal coding done for actuation turned out to be quite interesting for me. I messed up my fuse bits and accidentally changed my clock frequency to 8MHz. This made my board oblivious to the class handout source codes. I figured out the new PWM values to work with all three servos. The timer and clock synchronization was a good learning exercise.

## Sensors

CMU Camera – SPECIAL SENSOR

The CMU camera has been successfully used in many past projects and ample code and theory references are available for this sensor. The CMU camera operates through the USART0 from the Atmega128. PE0 and PE1 are used for the USART Rx and Tx respectively.

The CMU camera is used in middle mass mode in which the centroid of the blob of a given color is returned as M packets with other additional details like the bounding rectangle, the number of pixels and the confidence value.

The CMU camera was purchased from Seattle Robotics

**EXPERIMENTS**

The CMU camera experiments include detecting and giving proper x-y coordinates based on the objects position in its screen. The CMU camera was connected to the computer in imdl lab using a serial cable and the java GUI that comes with the camera.

The following are the settings used for this experiment.

1) White balance : Off
2) Auto gain : On
3) Red color : 116 to 136
4) Green color : 118 to 138
5) Blue color : 23 to 43

With this settings, the assumptions mean that the camera should look for balls of the given RGB range and the resultant color blob would mean the presence or absence of the tennis ball. The color values are used with TC (track color) command.

The experiment worked fine and the following data was collected from it.

| Distance from Ball to Camera (inches) | Centroid X | Centroid Y |
|---|---|---|
| 19 | 46 | 135 |
| 20 | 47 | 123 |
| 21 | 48 | 114 |
| 22 | 48 | 108 |
| 23 | 50 | 100 |
| 24 | 50 | 93 |

The camera is mounted in its inverted position which is the reason why the Y co-ordinates are decreasing instead of increasing with distance. The X coordinates are fairly same because the ball was moved almost along the same x-axis with respect to the camera.

Though, these readings were consistent and was repeated for 3 trails, the results were disappointing when lighting conditions changed. The same tennis ball was not detected in the lab in another system near which there was only one tube light.

Instead of counting on a “magic” color configuration for detecting balls and to keep the mechanism to be general for any color and sized balls, a generic command like TW should be used for initialization.

TW command is executed during the initialization procedure during which the color of the ball in that particular light source is noted. This color is used to track the ball in subsequent calls to

TC (without any arguments). The TW command gets the color which is dominant in the center of the camera which should suffice our purposes.

The CMU camera will be run in poll mode to avoid the camera from over loading the microcontroller with data. The camera will be run in raw mode without any acknowledgements to make the data processing simple.

SONARS (SRF05)

The sonar values are received from SRF05 by sending a trigger input and receiving the echo output. The value of the sonar  is used for obstacle avoidance

## Behaviors

**Obstacle Avoidance:**

I tried the following obstacle avoidance strategies

1) Fuzzy logic:

I dropped the idea because the values were not consistent and serious questions about whether the obstacle was avoided or not were raised.

2) Simple closest obstacle detection:

I detect if the obstacle is too close and take a sharp turn in the direction away from the obstacle.

The latter approach was simpler and more elegant.

**Ball Collection:**

The CMUCam triggers the board with the x and y coordinates of the ball of the color required. During initialization, the color of the ball is given as input by holding the ball right before the camera’s window.

Once the ball is in the field of view of the camera , the robot attempts to fetch the ball and collects the ball in the inside of the robot. Once, the ball is collected, the robot does obstacle avoidance with the ball.

**Ball release:**

When faced with a wall, the robot releases the ball and moves back from the wall. If the robot detects the released ball again, it attempts to fetch it again.

## Experimental Layout and Results

I tested each of my sensors and servos separately and in combination with each other before I added them to my robot.

The maximum time of experimentation was spent in understanding how to

1)mount the CMUcam

2)interfacing the CMUcam with the maveric board.

I have calibrated the CMUCam and made it work with the servo to collect the balls.

## Conclusion

I completed all the requirements that I initially had in mind and learnt a lot from this course. But I have not implemented the more complex behaviors that I had in mind in the beginning of the course. This course has been more difficult than most courses because in spite of the efforts that I put in, I do not seem to get anywhere for most part of the class.

I learnt the basics of robotics design which is the reason I took this course, though the course was not anyway easy, my robot is worth the pain and I am happy about the course.

APPENDIX A:

FINAL CODE:

```
//SimpleAvoidance.h
#include <avr/io.h>
#include <stdlib.h>
#define S1R (1<<0) // Echo output 1
#define S1 (1<<1) // Trigger input 1
#define S2R (1<<6) // Echo output 2
#define S2 (1<<7) // Trigger input 2
#define TRAILS 10
// PORTD , PIND == PORTA, PINA
#define MIN(x,y) ((x) < (y)) ? (x) :(y)
#define MAX(x,y) ((x) > (y)) ? (x) :(y)

int getSonar1();              // use to read value from sonar module
int getSonar2();
void doorOpen();

void doorClose();


void pwm_init();


void sonar_init();
void stop ();
void move() ;

void turn ();

int guess ();

void lcd_delay();

int turnTowardsBall(int x);
```

//SimpleAvoidance.c

```
#include "SimpleAvoidance.h"
#include "lcd.h"


void sonar_init()
{
      DDRD = S1|S2;
}

int getSonar1()
{
	unsigned long total = 0;
	unsigned int n,i;
	unsigned int min,max=0;

	min = 0xffff;

	for (i = 0;i < TRAILS; i++)
	{
		n = 0;
		PORTD =  (PIND & (~S1)); //(PIND & 0x7F); // these two lines
create a rising edge
		PORTD =  (PIND | S1); //(PIND | 0x80);    // on PORTD pin 7 (2)

	//lcd_clear();
	//lcd_string("rising edge done");

		while (n < 40)
		{
			//waste enough clock cycles for at least 10us to pass
			n += 1;
			n++;
		//	lcd_clear();
		//	lcd_int(n);
		}

		PORTD =  (PIND & (~S1));//	PORTD = (PIND & 0x7F);  // force
PORTD pin 7(2) low to create a falling edge
							// this sends out the trigger

		while (!(PIND & S1R))
		{
			// do nothing as long as echo line is low
		}

		n = 0;		//re-use our dummy variable for counting

		while (PIND & S1R)
		{
			n += 1;			// add 1 to n as long as PORTD pin 0 is
high
		}

		//when we get here, the falling edge has occured
		total +=n;
		min = MIN (n,min);
		max = MAX (n,max);
	}
	total-= (min + max);
	n = (int) (total/TRAILS - 2);

	return n;

}

int getSonar2()
{
	unsigned long total = 0;
	unsigned int n,i;
	unsigned int min,max=0;

	min = 0xffff;

	for (i = 0;i < TRAILS; i++)
	{
		n = 0;
		PORTD =  (PIND & (~S2)); //(PIND & 0x7F); // these two lines
create a rising edge
		PORTD =  (PIND | S2); //(PIND | 0x80);    // on PORTD pin 7 (2)

	//lcd_clear();
	//lcd_string("rising edge done");

		while (n < 40)
		{
			//waste enough clock cycles for at least 10us to pass
			n += 1;
			n++;
		//	lcd_clear();
		//	lcd_int(n);
		}

		PORTD =  (PIND & (~S2));//	PORTD = (PIND & 0x7F);  // force
PORTD pin 7(2) low to create a falling edge
						// this sends out the trigger

		while (!(PIND & S2R))
		{
			// do nothing as long as echo line is low
		}

		n = 0;		//re-use our dummy variable for counting

		while (PIND & S2R)
		{
			n += 1;			// add 1 to n as long as PORTD pin 0 is
high
		}

		//when we get here, the falling edge has occured
		total +=n;
		min = MIN (n,min);
		max = MAX (n,max);
	}
	total-= (min + max);
	n = (int) (total/TRAILS - 2);

	return n;

}

	int ACENTER = 1500;
	int BCENTER = 1510;
```

```
      int FORCE = 700;
      int CFORCE = 700;

      int STEP = 30;

void stop ()
{
      OCR1A = ACENTER;
      OCR1B = BCENTER;
      FORCE = 0;
}

void move()
{
      int i;
      //lcd_clear();
    //lcd_string ("Move ");

      if (FORCE > 700)
            FORCE = 700;
      if (FORCE < -700)
            FORCE = -700;
      //lcd_int (FORCE);
      //OCR1A = ACENTER - FORCE;
      //OCR1B = BCENTER + FORCE;

//    int diff = abs (FORCE - CFORCE);
//    int sign = (FORCE > CFORCE) ? 1: -1;
//    CFORCE += sign * (diff/10);
//    OCR1A = ACENTER - CFORCE;
//    OCR1B = BCENTER + CFORCE;


      OCR1A = ACENTER - FORCE;
      OCR1B = BCENTER + FORCE;


/*
      if (CFORCE <= FORCE)
      {
      CFORCE += STEP;
      OCR1A = ACENTER - CFORCE;
      OCR1B = BCENTER + CFORCE;
      }
      if (CFORCE > FORCE)
      {
      CFORCE -= STEP;
      OCR1A = ACENTER - CFORCE;
      OCR1B = BCENTER + CFORCE;
      }
*/
}

void turn () //Right +ve FORCE , Left -ve FORCE
{
      int i;
      //lcd_clear();
      //lcd_string ("Turn ");
      if (FORCE > 700)
            FORCE = 700;
      if (FORCE < -700)
            FORCE = -700;
      //lcd_int (FORCE);
      /*
      int diff = abs (FORCE - CFORCE);
      int sign = (FORCE > CFORCE) ? 1: -1;
      CFORCE += sign * (diff/10);
      OCR1A = ACENTER + CFORCE;
      OCR1B = BCENTER + CFORCE;
      */

      OCR1A = ACENTER + FORCE;
      OCR1B = BCENTER + FORCE;


/*
      if (CFORCE <= FORCE)
      {
      CFORCE += STEP;
      OCR1A = ACENTER + CFORCE;
      OCR1B = BCENTER + CFORCE;
      }
      if (CFORCE > FORCE)
      {
      CFORCE -= STEP;
      OCR1A = ACENTER + CFORCE;
      OCR1B = BCENTER + CFORCE;
      }
*/
}

int guess ()
{
      double a_number;

    a_number = (float) random();

    a_number = (float) random() / (float) 0x7fffffff;

      if (a_number > 0.5)
            return 0;
      return 1;
}
/*
int main(void)
{

      int s1 = 0, s2 = 0;
    int i;

      sonar_init();

   DDRC = 0xFF;               // set portC to output  (could also use DDRC =
0b11111111)
```

```
   DDRE = 0xFF;         // set portE to output
 //DDRE = 0b11111111;   // set portE to output
   DDRB = 0xFF;                // set portB to output

   lcd_init();     // set lcd in 4 bit mode, 2-line mode, with cursor on and
set to blink

   //lcd_string("Your LCD is working.");   // if your LCD is wired up
correctly, you will see this text
                                         // on it when you power up your
Micro-controller board.
   //lcd_row(1);
   //lcd_string("PWM Generator Test");


   pwm_init();

      FORCE = 500;
      OCR1C = 1950;

      //turn();
      while(1)
      {
            int f1,f2;
            s1 = getSonar1(); // RIGHT
            s2 = getSonar2(); // LEFT

            s1 /= 100;
            s2 /= 100;
            int OBSTACLE = 7;

            if (s1 < OBSTACLE && s2 < OBSTACLE)
            {
                  FORCE = -700;
                  move();
                  for (i = 0; i < 2; i++)
                        lcd_delay();
                  FORCE = 700;
                  int v = guess ();
                  if (v)
                        FORCE *= -1;
                  turn();
                  continue;
            }
            else if (s1 < OBSTACLE) //.Right Sonar
            {
                  FORCE = -700; // Turn left
                  turn();
                  lcd_delay();
                  continue;
            }
            else if (s2 < OBSTACLE) // Left Sonar
            {
                  FORCE = 700;
                  turn(); // Turn Right
                  lcd_delay();
                  continue;
            }
            else
            {
                  FORCE = 700;
                  move();
                  lcd_delay();
            }

      }
return 0;
} // main*/

void  pwm_init(void)
{
      TCCR1A = 0b10101000;    // ......00:P&FC PWM //
11......,..11....:OC1A,OC1B inv
      TCCR1B = 0b00010010;    // ...10...:P&FC PWM // .....010:bclk/8 (~8kHz)
      DDRB = 0xFF;                  // set PWM pins as outputs, PB5 (OC1A) &
PB6 (OC1B)
    ICR1= 0x8f9c;//20000;
      TCNT1=0x0000;
      OCR1A = 0x0000;               // 1/2 duty, ~2.5v dc level, motor 1 on
PB5 (OC1A)
      OCR1B = 0x0000;               // 0/256 duty, ~2.5v dc level, motor 2 on
PB6 (OC1B)

      doorClose();
} // pwm_init

int turnTowardsBall(int x)
{
      int f = (x - 40);
      if ( abs (f) < 15)
            return 1;
      FORCE = 50;


      if ( f > 0 )
            FORCE *= -1;

      lcd_clear ();
      lcd_string ("X = ");
      lcd_int (x);
      lcd_string ("Force = ");
      lcd_int (FORCE);

      turn();
      delay (10);
      return 0;
}


void doorOpen()
{
      OCR1C = 800;
      int i;
      for ( i = 0; i < 40; i++)
```

```
            lcd_delay();
}

void doorClose()
{
      OCR1C = 1950;
      int i;
      for (i = 0; i < 40; i++)
            lcd_delay();
}
```

//lcd.h

```
/*************************** Prototypes ********************************/
void lcd_delay();         // short delay (50000 clocks)
void lcd_init();          // sets lcd in 4 bit mode, 2-line mode, with cursor
on and set to blink
void lcd_cmd();           // use to send commands to lcd
void lcd_disp();          // use to display text on lcd
void lcd_clear();         // use to clear LCD and return cursor to home
position
void lcd_row(int row);    // use to put the LCD at the desired row
void lcd_int(int value) ;
void lcd_string(char *a) ;
void delay (int);
```

//lcd.c

```
#include "lcd.h"
#include <avr/io.h>


/* IMPORTANT!

Before using this code make sure your LCD is wired up the same way mine is,
or change the

code to
match the wiring of your own LCD.  My LCD is connected to PortC of the At-
Mega128 in the

following manner:

PortC bit 7   :   LCD data bit 7 (MSB)
PortC bit 6   :   LCD data bit 6
PortC bit 5   :   LCD data bit 5
PortC bit 4   :   LCD data bit 4 (LSB)

PortC bit 3   :   (not connected)
PortC bit 2   :   LCD enable pin (clock)
PortC bit 1   :   LCD R/W (read / write) signal
PortC bit 0   :   LCD RS (register select) pin

Also remember you must connect a potentiometer (variable resistor) to the
vcc, gnd, and

contrast pins on the LCD.
The output of the pot (middle pin) should be connected to the contrast pin.
The other two

can be on either pin.

*/


void lcd_delay()      // delay for 10000 clock cycles
{
   long int ms_count = 0;
   while (ms_count < 1000)
   {
      ms_count = ms_count + 1;
   }
}

void delay (int k)
{
      int i;
      for (i = 0; i < k; i++)
            lcd_delay();
}

void lcd_cmd( unsigned int myData )
{

   /* READ THIS!!!

   The & and | functions are the BITWISE AND and BITWISE OR functions
respectively.  DO NOT
   confuse these with the && and || functions (which are the LOGICAL AND and
LOGICAL OR functions).

   The logical functions will only return a single 1 or 0 value, thus they do
not work in this scenario
   since we need the 8-bit value passed to this function to be preserved as
8-bits
   */


   unsigned int temp_data = 0;

   temp_data = ( myData | 0b00000100 );         // these two lines leave the
upper nibble as-is, and set
```

```
   temp_data = ( temp_data & 0b11110100 );  // the appropriate control bits
in the lower nibble
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110000);            // we have written upper
nibble to the LCD

   temp_data = ( myData << 4 );           // here, we reload myData into our
temp. variable and shift the bits
                                                // to the left 4 times.  This
puts the lower nibble into the upper 4 bits

   temp_data = (temp_data & 0b11110100);   // temp_data now contains the
original
   temp_data = (temp_data | 0b00000100);   // lower nibble plus high clock
signal

   PORTC = temp_data;                     // write the data to PortC
   lcd_delay();
   PORTC = (temp_data & 0b11110000);      // re-write the data to PortC with
the clock signal low (thus creating the falling edge)
   lcd_delay();

}

void lcd_disp(unsigned int disp)
{

   /*

   This function is identical to the lcd_cmd function with only one
exception.  This least significant bit of
   PortC is forced high so the LCD interprets the values written to is as
data instead of a command.

   */

   unsigned int temp_data = 0;

   temp_data = ( disp & 0b11110000 );
   temp_data = ( temp_data | 0b00000101 );
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110001);
   lcd_delay();                  // upper nibble

   temp_data = (disp << 4 );
   temp_data = ( temp_data & 0b11110000 );
   temp_data = ( temp_data | 0b00000101 );
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110001);
   lcd_delay();                  // lower nibble

}
```

```
void lcd_init()
{
      DDRC = 0xff;
   lcd_cmd(0x33);      // writing 0x33 followed by
   lcd_cmd(0x32);      // 0x32 puts the LCD in 4-bit mode

   lcd_cmd(0x28);      // writing 0x28 puts the LCD in 2-line mode

   lcd_cmd(0x0F);      // writing 0x0F turns the display on, curson on, and
puts the cursor in blink mode

   lcd_cmd(0x01);      // writing 0x01 clears the LCD and sets the cursor to
the home (top left) position

   //LCD is on... ready to write

}

void lcd_string(char *a)
{

   /*

   This function writes a string to the LCD.  LCDs can only print one
character at a time so we need to
   print each letter or number in the string one at a time.  This is
accomplished by creating a pointer to
   the beginning of the string (which logically points to the first
character).  It is important to understand
   that all strings in C end with the "null" character which is interpreted
by the language as a 0.  So to print
   an entire string to the LCD we point to the beginning of the string, print
the first letter, then we increment
   the pointer (thus making it point to the second letter), print that
letter, and keep incrementing until we reach
   the "null" character".  This can all be easily done by using a while loop
that continuously prints a letter and
   increments the pointer as long as a 0 is not what the pointer points to.

   */

   while (*a != 0)
   {
      lcd_disp((unsigned int) *a);   // display the character that our
pointer (a) is pointing to
      a++;                        // increment a
   }
   return;

}


void lcd_int(int value)
{

   /*
   This routine will take an integer and display it in the proper order on
   your LCD.  Thanks to Josh Hartman (IMDL Spring 2007) for writing this in
lab
   */

   int temp_val;
   int x = 10000;         // since integers only go up to 32768, we only need
to worry about
                           // numbers containing at most a ten-thousands
place

   while (value / x == 0)   // the purpose of this loop is to find out the
largest position (in decimal)
   {                          // that our integer contains.  As soon as we
get a non-zero value, we know
      x/=10;                // how many positions there are int the int and x
will be properly initialized to the largest
   }                          // power of 10 that will return a non-zero
value when our integer is divided by x.

   if (value==0) lcd_disp(0x30);

   else while (x >= 1)            // this loop is where the printing to the
LCD takes place.  First, we divide
   {                          // our integer by x (properly initialized by
the last loop) and store it in
      temp_val = value / x;     // a temporary variable so our original value
is preserved.Next we subtract the
      value -= temp_val * x;    // temp. variable times x from our original
value.  This will "pull" off the most
      lcd_disp(temp_val+ 0x30); // significant digit from our original
integer but leave all the remaining digits alone.
                                    // After this, we add a hex 30 to our
temp. variable because ASCII values for integers
      x /= 10;                // 0 through 9 correspond to hex numbers 30
through 39.  We then send this value to the
   }                          // LCD (which understands ASCII).  Finally, we
divide x by 10 and repeat the process
                                    // until we get a zero value (note: since
our value is an integer, any decimal value
   return;                          // less than 1 will be truncated to a 0)

}

void lcd_clear()      // this function clears the LCD and sets the cursor to
the home (upper left) position
{
   lcd_cmd(0x01);

   return;
}

void lcd_row(int row)   // this function moves the cursor to the beginning of
the specified row without changing
{                             // any of the current text on the LCD.

   switch(row)
   {
      case 0: lcd_cmd(0x02);
      case 1:   lcd_cmd(0xC0);

   }

   return;
}
```

//Main

```
/*************************** Includes ********************************/


#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>

#include "SimpleAvoidance.h"
#include "lcd.h"
void port_init();         // initialize I/O of ports
void interrupt_init();         // initialize interrupt routines
void process();

extern int FORCE;
int tr,tl;
tr = 0;
tl = 0;

// Define globals
int ledOn = 0;
int lineTPos = 0;
char *lineT;
int lineRPos = 0;
char lineR[20];

void send (unsigned char * text)
{
      int i;
      lineT = text;
      for (i = strlen(text); i < 20; i++)
            lineT[i] = ' ';
    UCSR0B |= 0b00100000;        // turn on transmission
      UCSR0B &= 0b01111111;        // turn off receive
    lineTPos = 0;
       for (i = 0; i < 500; i++) {
                        lcd_delay();          //delay to read LCD (humans
reading)
                }
      lcd_clear();
```

```
}

void avoid ()
{
      int i;
      int s1,s2;
                  s1 = getSonar1(); // RIGHT
            s2 = getSonar2(); // LEFT

            s1 /= 100;
            s2 /= 100;
            int OBSTACLE = 7;

            if (s1 < OBSTACLE && s2 < OBSTACLE)
            {
                  doorOpen();
                  FORCE = -700;
                  move();
                  delay(300);
                  doorClose();
                  FORCE = 700;
                  int v = guess ();
                  if (v)
                        FORCE *= -1;
                  turn();
            }
            else if (s1 < OBSTACLE) //.Right Sonar
            {
                  FORCE = -700; // Turn left
                  turn();
            }
            else if (s2 < OBSTACLE) // Left Sonar
            {
                  FORCE = 700;
                  turn(); // Turn Right
            }
            else
            {
                  FORCE = 700;
                  move();
            }

}

// Main routine
int main(void) {
        long i;
        int temp;

        port_init();                // set port directions

        lcd_init();     // set lcd in 4 bit mode, 2-line mode, with cursor on
and set to blink

                // if your LCD is wired up correctly, you will see this text
                                                                          // on
it when you power up your Micro-controller board.
```

```
        interrupt_init();                // initializes and activates
interrupt routines

            for (i = 0; i < 500; i++) {
                lcd_delay();          //delay to read LCD (humans reading)
        }
            for (i = 0; i < 500; i++) {
                lcd_delay();          //delay to read LCD (humans reading)
        }

            lcd_string ("Raw Mode");
            unsigned char newLine9[20] = "RS\r";
            send (newLine9);


            lcd_clear();

            unsigned char newLine5[20] = "CR 18 44\r";
            send (newLine5);

            lcd_string ("White Balance");
            lcd_row (1);
            lcd_string ("Place ball in front of cam");
            for (i = 0; i< 1000; i++)
                  lcd_delay();

            unsigned char newLine6[20] = "CR 18 40\r";
            send (newLine6);
            lcd_clear();
            lcd_string ("White Balance... Done");
            lcd_row (1);
            lcd_string ("Place ball in front of cam");

            unsigned char newLine2[20] = "MM 1\r";
            send (newLine2);

            lcd_clear();
            lcd_string ("Middle Mass mode");
            lcd_row (1);
            lcd_string ("Place ball in front of cam");

      lcd_string ("Raw Mode");
            unsigned char newLine1[20] = "RM 3\r";
            send (newLine1);


/*
            unsigned char newLine3[20] = "PM 1\r";
            send (newLine3);*/

            unsigned char newLine4[20] = "TW\r";
            send (newLine4);

            lcd_clear();

            UCSR0B |= 0b10000000; // Enable recieve
            pwm_init();
            sonar_init();

        while(1) {
                  avoid();
        }

        return 0;

}

SIGNAL(SIG_UART0_DATA) {
        if (lineTPos < 20) {
                UDR0 = lineT[lineTPos];
                  //    lcd_disp (lineT[lineTPos]);
                if (lineT[lineTPos] == 13) {        // if character is
carriage return
                        UCSR0B &= 0b11011111;                // then turn off
interrupt for transmission
                        int temp = UDR0;                        // clear out
receive buffer
                        UCSR0B |= 0b10000000;                // and turn on
interrupt for receiving
                }
                lineTPos++;
        }
}

SIGNAL(SIG_UART0_RECV) {
        if (lineRPos < 20) {
                lineR[lineRPos] = UDR0;
                if (lineRPos == 19|| lineR[lineRPos] == 13 || lineR[lineRPos]
== ':') {         // if character is carriage return
                        UCSR0B &= 0b01111111;                        // then
turn off interrupt for receiving
                        lineR[lineRPos] = 0;
                                    //lcd_clear();
                                    process();
                                    if(!tl&&!tr)
                                          avoid();
                        lineRPos = 0;
                                    UCSR0B |= 0b10000000;                //
and turn on interrupt for receiving
                }
                else lineRPos++;
        }
}

void test(int i)
{
      lcd_clear();
      lcd_int (lineR [i+1]);
      lcd_string (" ");
      lcd_int (lineR [i+2]);
      lcd_string (" ");
      lcd_int (lineR [i+3]);
```

```
        lcd_string (" ");
        lcd_int (lineR [i+4]);
        lcd_string (" ");
        lcd_int (lineR [i+5]);
        lcd_string (" ");
        lcd_row (1);
        lcd_int (lineR [i+6]);
        lcd_string (" ");
        lcd_int (lineR [i+7]);
        lcd_string (" ");
        lcd_int (lineR [i+8]);
        lcd_string (" ");
        for (i = 0; i < 1000; i++)
                lcd_delay();
}

void process()
{
        int i,j;

        //lcd_int (strlen(lineR));
        //lcd_string (lineR);
//      return;
        int length = strlen (lineR);
        for (i = 0; i < length; i++)
                        {
                                if (lineR[i] == 'M')
                                {
                                        if (i + 8 > length) return;
                                        //test(i);
                                        int cf = lineR[i+8];
                                        int x = lineR[i+1];
                                        int y = lineR[i+2];
                                        if (!(x>= 0 && x <= 80&& y>=0 && y <=
143&& cf >=0&&cf <=255))
                                                continue;
                                                /*
                                        lcd_clear();
                                        lcd_int (x);
                                        lcd_string (" ");
                                        lcd_int (y);
                                        lcd_string (" ");
                                        lcd_int (cf);
                                        lcd_row (1);
                                        lcd_string (lineR);
                                        delay (100);
*/
                                        if (cf >= 10)
                                        {
                                                stop();
                                                if (x <= 50 && x >= 35)
                                                {

                                                        doorOpen();

                                                        FORCE = 700;
                                                        move();
```

```
                                                  delay (1000);
                                                  stop();
                                                  doorClose();
                                                  tl = tr = 0;
                                            }
                                            else if (x < 35)
                                            {
                                                  FORCE = 200;
                                                  turn();
                                                  tr++;
                                                  tl = 0;
                                                  delay (20);
                                                  stop();
                                            }
                                            else if (x > 50)
                                            {
                                                  FORCE = -200;
                                                  tl++;
                                                  tr = 0;
                                                  turn();
                                                  delay (20);
                                                  stop();
                                            }

                                      }
                                      return;
                                }
                          }

}

void port_init() {
        DDRC = 0xFF;                            // set portC to output  (could
also use DDRC = 0b11111111)
        DDRB = 0xFF;                            // set portB to output
        DDRE = 0xFF;                            // set portE to output
        DDRA = 0b11111000;                    // set portA bits 2-0 to input
        DDRF = 0b00000000;                    // set port F to all input
}

void interrupt_init() {
        UCSR0B = 0b10011000;                    // enable tx and rx
        UCSR0C = 0b00000110;                    // asynch, no parity, 1 stop bit,
8 bit characters
        UBRR0L = 23;                                    // initialize transfer
speed for 38400 baud
        sei();                                            // enable all interupts
}
```