Date: 8/7/2007
Student: Chad J. Fralick
TA: Kevin Claycomb
Adam Barnett
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz
Robot: WaterBot

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

WaterBot
Final Project Report

Table of Contents

Abstract

This report covers the design, building, and testing of WaterBot. WaterBot is an autonomous, household robot that will move around a room watering specific plants that have magnetic panels in front of them. WaterBot avoids hitting obstacles in the process of finding plants and can return home once it is finished watering. It will follow a line to complete its task of watering.

Executive Summary

WaterBot, through the course of the semester has progressed from an idea and a mound of used parts to a relatively smart helper that obeys. WaterBot does several things in its job of watering plants.

Throughout the process of watering, WaterBot follows a line. Four IR/phototransistor pairs on the underside of the robot let WaterBot know when it is centered over the line. The line guides the robot next to each plant. A panel with a binary sequence of magnets placed at each plant gives WaterBot something to look for. Magnetic sensors lined up on the side of WaterBot interact wirelessly with the magnets on the panels. Once sensing and lining up with the panel, WaterBot knows when to stop. The robot then reads the plant identification number off the panel and decides how much to water the plant, if it waters it at all. This information is placed in the robot's memory before using. If WaterBot determines that Plant # *X* needs water, it turns on the pump and squirts water out of its tube into the plant's pot. WaterBot then continues on the line to the next plant.

While moving on the line, if at any point WaterBot recognizes something directly in front of it, it stops and waits for the obstacle to move. WaterBot efficiently and automatically takes care of the task of watering the plants.

Introduction

Plants are a pleasing addition to any house or patio. At the very least, they add color and interesting shapes to a bland room. Unfortunately, many plants die each year due to insufficient watering techniques. Many have seen the decay: brown, wilted leaves about to fall off their stalks. Owners either forget to take care of their precious possessions, or they just don't know what to do with them.

Fortunately, mindless and forgetful owners will no longer be responsible for involuntary plant starvation. WaterBot, the plant-watering robot, will take over the task of adequately watering each plant at pre-determined intervals. WaterBot will find and water the plants that are scheduled for watering. It will navigate around common obstacles found at a normal home until it reaches a thirsty plant. Once confirming the plant is on the list of 'to-be-watered plants', the robot dispenses a certain amount of water into the plants pot. After watering one plant, it finds other thirsty plants, remembering which ones still need watering. After watering all the plants, the robot returns to its home base and waits until it is time to water plants again.

WaterBot will follow a predetermined path that will bring it in close proximity to plants. Using its magnetic sensor array, it will detect the nearby plant, line up with the plant, water it, and continue on to the next plant. A unique number will be assigned to each plant. This number will be represented with a panel of magnets. The magnets will assist the robot in lining up with the plant and confirming each plant's identity. A small database in the robot's memory will contain the schedule and identity of the pots to be watered and if practical, the approximate amount to water each pot.

A small, electrical water pump (windshield wiper fluid pump) will deliver water from the on-robot reservoir to the pot. A tube will direct the water from the pump to inside the pot. Once the robot exhausts its water supply, it will need to return to its home base and await refilling.

This report contains a detailed description of all the parts of WaterBot. First, the Integrated System describes the overall system and the general theory of operation. Next, the physical subsystems are described in greater detail: Mobile Platform, Actuation and Sensors. Following that, WaterBot's Behaviors are described followed by Experimental Layout and Results.

Integrated System

WaterBot's Micro Controller is the Atmel ATmega 128 on a Maverick IIB board. All the electrical components of WaterBot will be connected to this board (see Figure 1).

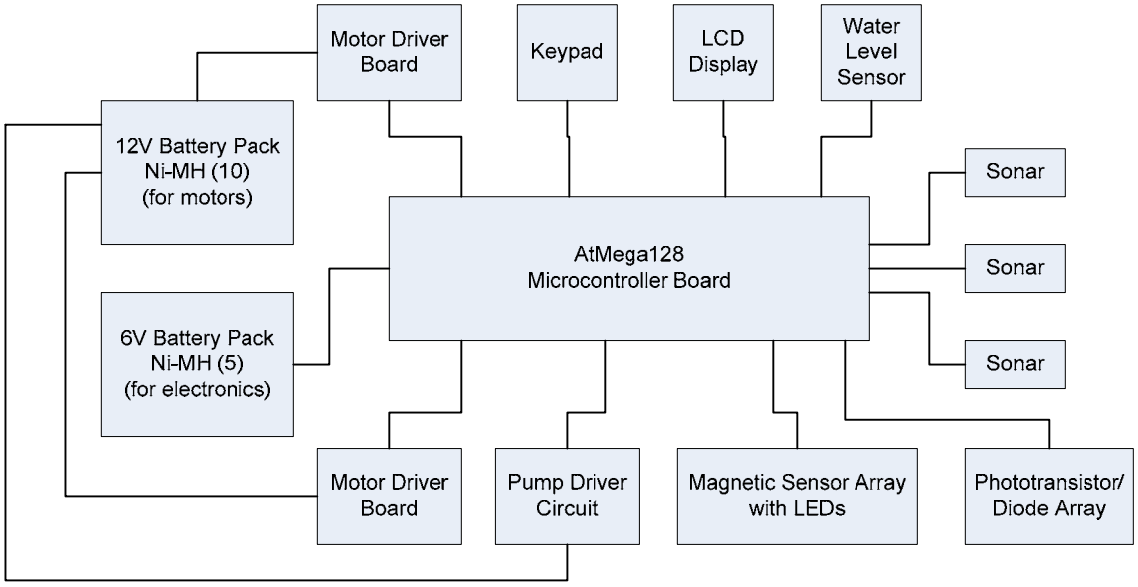


Figure 1 Electrical Components

The electrical components include:

- 1 AtMel Maverick 2B AtMega 128 Microcontroller board
- 2 battery packs (one for motors at 12V and one for electronics at 6V)
- 3 sonar modules (Devantech SRF05)
- 1 telephone keypad for input
- 1 LCD display
- 1 Water level switch (wasn't incorporated in final robot)
- 4 phototransistor/diode pairs
- 3 motors with appropriate driver circuits (2 for drive and 1 as a pump)
- 8 magnetic sensors
- Several indicator LEDs

A concept schematic top and side view of WaterBot is shown in Figure 2. Instead of RFID (as noted in the figure), WaterBot uses its magnetic sensor array to find, line up with, and identify plants.

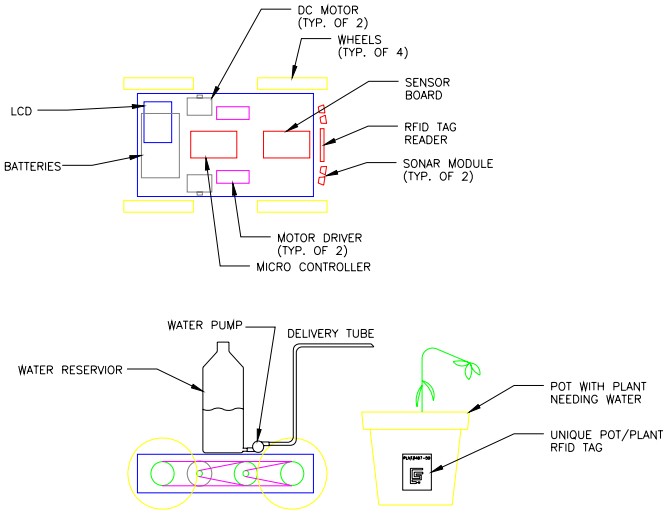


Figure 2 Concept schematic

Mobile Platform

WaterBot's platform is a box slightly larger than a cereal box. All the sides of the box, except for the top have been cut from wood. It houses most of the electronic components including the motors, driver circuits, LCD screen, controller board and batteries. The top of the box is a clear pane of plexiglass. This allows all the components inside to be seen from the outside while still being protected from water in the event of leaking or spilling. The top is held about ¼" off the top of the frame to allow for adequate ventilation of the batteries.

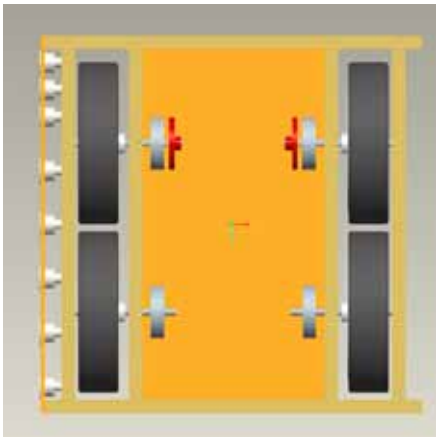


Figure 3 Top View Platform (Original design)

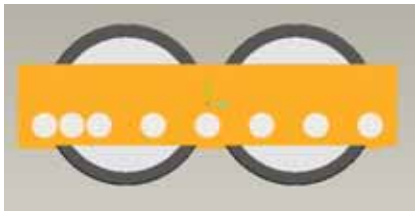


Figure 4 Side View Platform (Original design)

The original design called for four, 6" diameter wheels attached on two opposite sides of the rectangular box. I built this design first. The wheels were driven like tracks on a tank: the two wheels on each side were connected and spun at the same speed as each other. As was predicted, WaterBot had difficulty turning with the original design. The motors could not produce enough torque to overcome the friction of rubber wheels sliding sideways. Therefore, I modified the design by removing the belts and replacing the front wheels with casters. The final wheel design works much better; WaterBot can turn freely. See figure 5, a bottom view picture of the new/final wheel design.

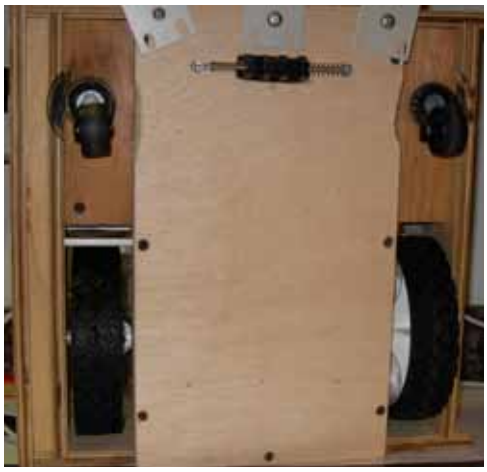


Figure 5 New Wheel Design

The two large drive wheels and two casters are placed far enough apart to give WaterBot a low center of gravity which means exceptional stability. The ½" thick plywood used for the frame provides ample strength for carrying the water reservoir. The frame and wheels can actually support at least 160 pounds. This was proven by standing on it. The water reservoir (used apple jug) sits casually on top of the pane of plexiglass. The jug holds approximately two quarts of water.

Actuation

Two, identical stepper motors provide torque to the 6" inch diameter wheels. The motors were salvaged from two similar, HP printers. The motors were made by NMB (model number PM55L-048). They are 18V, two phase, unipolar, stepper motors. They have 48 steps per revolution. I drive the motors with 12V and they seem to work just fine at the less-than-rated voltage. A small gear was already attached to each of the stepper motors.

Larger gears, with 6 times as many teeth as the small gear are connected to the shafts of the 6" wheels. The large gears and shafts were also retrieved from old printers. The shafts were cut to down to size.

Stepper motors were chosen over DC motors to enable more precise control of WaterBot's movement. WaterBot consistently lines up within 1/2" of a plant's magnetic panel and stops within about ¼" of its target. This is easily achieved through the use of stepper motors. The microcontroller can precisely control the number of steps to make and the speed to make the steps.

The disadvantage to using stepper motors on this project was the vibration they cause as low speed. Another problem is the ability to go fast. The stepper motors on WaterBot cannot be stepped too quickly, or they will malfunction.

The drive motors are controlled by the microcontroller, but powered through a separate power supply. Since stepper motors require constant switching on and off, complete isolation was needed between the microcontroller and the motor drivers to protect the sensitive components. This was accomplished through the use of an opto-isolator chip.

Each motor is controlled with 4 step wires and 1 common wire. The single common wire is connected to V+, which for WaterBot, is 12V. Each of the 4 step wires are switched to ground in a specific sequence. The speed of the switching controls the RPM of the motor. Each step is switched indirectly by the microcontroller through a circuit described below. The 8 circuits required to control 4 step wires for each motor is picture in figure 6.

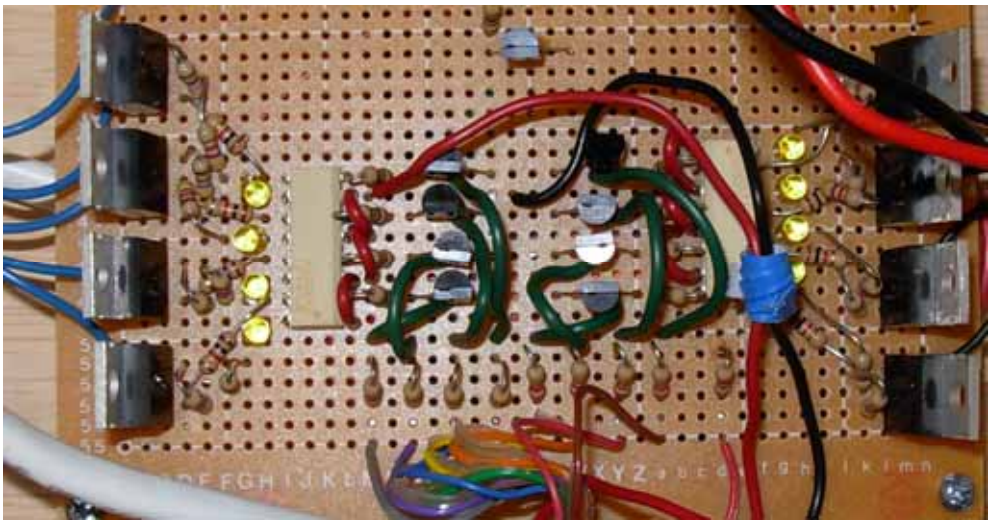


Figure 6 Drive Motor Driver Circuits

All 8 circuits are exactly the same, one of the circuits is described here. The microcontroller pin directly powers the base of a small 2n2222 transistor through a resistor. The resistor limits the current from the pin in the event of a junction breakdown inside the transistor. The transistor controls the on/off status of the IR LED located inside the white optoisolator chip. The LED needs about 16mA of current. The 2n2222 transistor is used to minimize the amount of current the microcontroller pin will have to source. These two devices are powered from the 5-6V source, the same source that powers the microcontroller board. The opposite side of the optoisolator contains a photo transistor. I connected this externally to another transistor, the high power TIP120 darlington transistor. The TIP120 can handle a current of 5A, more than enough for the 1.5A stepper motors. The TIP120 also contains a clipping diode to take care of some of the spikes produced during the used of the motor it controls. Logic high on the microcontroller pin drives the 2n2222 transistor into saturation, which turns on the IR LED inside the optoisolator, which optically excites the phototransistor into saturation, which allows current to flow into the base of the TIP120, which in turn connects the step wire to ground. Each motor requires four data pins, one for each step wire. The circuit schematics can be found in the lab notebook section of the appendix.

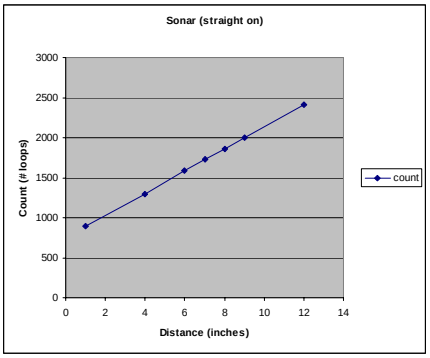
The drive motors were driven with the use of interrupts. The main program sets the delay for the interrupts and the interrupt sub routine updates the motors once the delay has expired. The control code can be seen in the appendix.

Sensors

Sonar

Devantech SRF05 from www.acroname.com

WaterBot has three sonar modules looking forward. They sense objects in the path of WaterBot. For plant watering, WaterBot only uses the center sonar module to keep from hitting objects on or near the path line directly in front of it. Preliminary tests reveal the sonar's linear characteristic when comparing the distance from the wall to the module (see Chart 1).



Graph 1 Sonar Loop Count vs. Distance

The test was conducted with a vertical board held at different distances away from the sensor. For this experiment, I placed the board directly in front of the sensor.

The sonar module sends a pulse and awaits its return. The time it takes for the pulse to return linearly corresponds to the distance that the module is from an obstacle. After the microcontroller tells the sonar to send a pulse, a loop is entered that increments a variable until the signal returns. The number of loops completed corresponds to the distance the module is from the nearest object within its range of vision (about 30 degrees on each side). The sonar pinging code can be found in the appendix.

Figure 7 Sonar Test

Bump Sensors

WaterBot was initially designed to have bump sensors in the rear so it will know when it runs into something while going in reverse. I decided WaterBot would always need to go forward to accomplish its goal of watering plants. Bump sensors will be a good addition to WaterBot to make it more versatile in the future.

Magnetic Sensor Array

120t-12-W from GRI (security system sensors)

A magnetic sensor array located on WaterBot's right side will be used to uniquely identify each plant and line up with it. The largest working range between a sensor and a magnet is 1" along the axis of the sensor. It has a radius around the sensor of ½" where magnets will still be sensed. WaterBot, when passing close by a plant with a magnetic panel, will recognize and identify the pot after lining up with it and decide whether or not to water it.

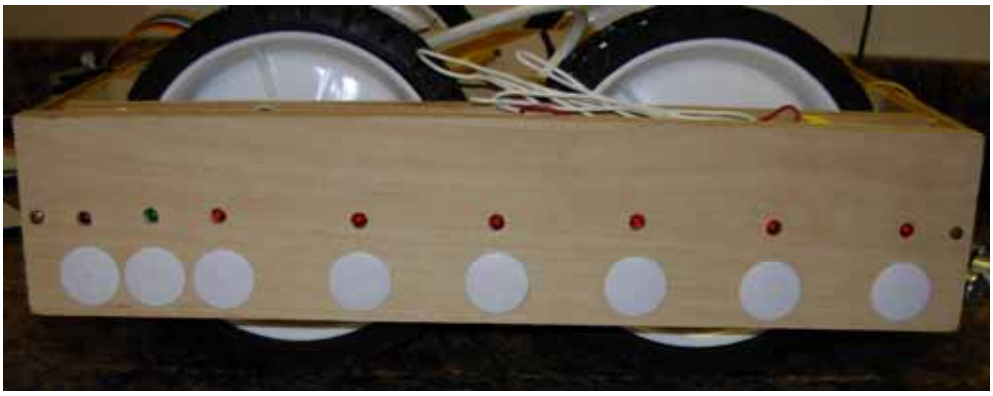


Figure 8 Magnetic Sensor Array with LEDs above sensors

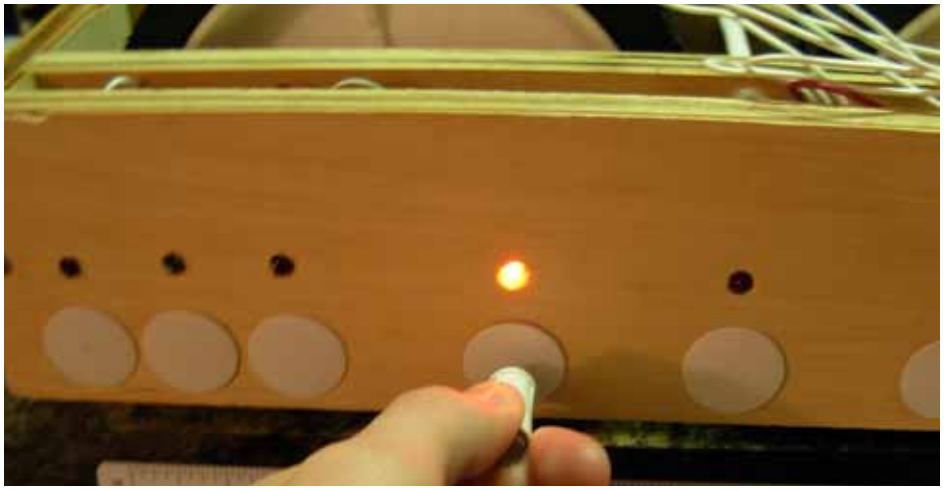


Figure 9 Magnetic Sensor in use

Figure 10 Magnet Panel

Water Level Detector

The complete design includes a water level sensor is in the water container to alert WaterBot when it is almost empty. It includes a bobber, a string, and a micro-switch. The weight of the bobber engages the switch once the water drops too low. This will be added to WaterBot in the future.

Phototransistors/Diodes

OPB745 from www.digikey.com (IR/Phototransitor pairs)

Diodes and phototransistors are used to keep WaterBot following a line. 4 sensors are mounted on the bottom of WaterBot. They detect the presence of a dark line, which will be ¾" wide electrical tape for WaterBot. The phototransistors are connected to the A/D converter pins of the microcontroller. WaterBot turns either right or left based on which of the A/D channels (which are connected to the phototransitors) are above a 'line sensing' threshold value. The code for this can be found in the appendix.

The sensors were mounted on the underside of the robot to look at the tape on the floor. The custom mounting hardware allowed for easy tweaking and adjusting during setup. See figure below.

Figure 11 Line Following Circuit

The circuits used for the LED/Phototransistor pairs were adapted from the circuits provided by William Dubel in "Reliable Line Tracking".

Behaviors

Line Following

WaterBot follows a line made from electrical tape in order to properly align itself with the plants. After passing through the A/D converter, the values from the IR/Phototransistor sensors are converted into either a 1 or 0. A 1 corresponds to the presence of a line and a 0 corresponds to no line. Depending on which sensors see lines, the motor speeds are changed. A motor has three speeds: slow, medium and fast. If the robot sees a line on only one of its outer sensors, it makes one motor go fast and the other slow. The code for the line following can be found in the appendix. It was found that smaller differences in speed change greatly improve the smoothness of the line following process.

Lining UP

WaterBot will have to line up with each plant to ensure that water will not be spilled on the floor. The magnetic sensor array will be used to accomplish this. One magnet, the first one, on each plant's panel is used solely for alignment. WaterBot will sense the magnet on bit 0 of PORTE as it drives into position. It will sense the magnet as it goes from bit 0 all the way to bit 6. The three closely spaced sensors near the rear of WaterBot, bits 5-7 are solely used for aligning with each plant's panel. The goal is to activate only bit 6 of the three bits. This is the only sensor with a green light above it. A magnet can activate two sensors at one time, so WaterBot must drive forward and reverse as needed to ensure that only the bit 6 sensor is engaged.

After only the bit 6 sensor is engaged, WaterBot is ready to read the number on the magnetic panels.

Pumping

If WaterBot has found a plant that is in the internal list of plants, its controller will indirectly control a relay that turns the water pump on. The water pump will stay on for a delay specified by the internal plant database. Two, blue LEDs illuminated the pumping process.

Experimental Layout and Results

Magnetic Sensor tests

The magnetic sensors were tested by hooking them in series with an LED circuit and moving a magnet around the sensor. Along the axis of the sensor, the contact engages when the magnet is 1" away from the sensor. Within this distance, the magnet can also be moved a maximum of ½" from the center axis of the sensor. This range is the fudge range associated with the sensor and is used to the advantage of the robot when lining up. This range was used to design the layout of the sensor array and magnet spacing on the plant panels.

Sonar range tests

The sonar modules were attached to the robot and tested with a wall held at varying distances in front of it. The results can be found in the Sonar section above. The results indicate a linear relationship between the delay time and the distance from the module. This was expected due to the nature of sound reflection.

Line Following

Line following was tested by trial and error. At first, the difference in the motor speeds when turning were so large, the robot was weaving back and forth along the line. It weaved as much as 2 inches. After steadily decreasing the differences between the motor speeds, WaterBot became much smoother. The difference in the motor speeds when turning is now approximately 2% difference. This means WaterBot will not follow sharp turns well, but it will follow a straight line extremely well.

Through trial and error, it was found that the magnet panel should be placed 7-1/4" from the line. Any closer and the robot is likely to hit it, and any further away and the robot may not read all the magnets.

Conclusion

WaterBot can successfully approach plants, identify them based on a magnetic panel, water them a preset amount, and follow a line to the next plant. It can also notice an object obstructing its patch.

As of 8/6/2007, WaterBot is limited in its sensing abilities. It cannot sense when it is empty. The sensing apparatus is designed, but it has not been implemented. This is a limitation to WaterBot's ability to sense. WaterBot cannot leave the line to move around an obstacle yet. This was an initial goal that was abandoned. I think this will be a great area for future development on WaterBot.

I am very impressed with WaterBot's ability to follow a line and sense the magnetic panel. I initially feared that precision could not be achieved to a degree suitable for the magnetic array to work, but fortunately I was wrong.

Documentation

- LCD code for initialization and general use provided by Adam Barnett.
- Line tracking schematic adapted from William Dubel, "Reliable Line Tracking"

Appendices

The following appendices include all the code and lab notebook entries.

```
/*
Chad Fralick
WaterBot code
EEL5666C, University of Florida
LCD setup code adapted from code by Adam Barnett (and one method written by Josh Hartman, noted below)
*/

#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "globals.h"
#include "lcd.h"
#include "motors.h"
#include "adc.h"
#include "sonar.h"
#include "water.h"
#include "linefollow.h"
#include "arbitrator.h"

volatile uint16_t ms_count; //used for timer interrupts

//HELPER FUNCTIONS.....
//.....
void interrupt_test(void)
{
while(1)
 {
   lcd_row(0);
   char echo0_str [8];
   lcd_string(itoa(ms_count, echo0_str, 10));
   lcd_string("  ");

   lcd_row(1);
   lcd_string("  Light blink");
  }
}

void init_timer0(void)
{
  TCCR0 = 0;
  TIFR  |= _BV(OCIE0);  //|_BV(TOIE0)
  TIMSK |= _BV(TOIE0)|_BV(OCIE0);          /* enable output compare interrupt */
  TCCR0  = _BV(WGM01)|_BV(CS02);//|_BV(CS00); // CTC, prescale = 128
  TCNT0  = 0;
  OCR0   = 254;                    /* */
}


ISR(SIG_OUTPUT_COMPARE0)
{
//the right motor uses the lower 4 bits
    count_L++;
    count_R++;
```

```
    if(turn_flag == 0)
    {
        if(m_speed_L < count_L)
        {
            count_L =0;
            switch(stat_L) {  //increments motor one step
                case 0x00: PORTB = 0x60 | stat_R; break;
                case 0x30: PORTB = 0x60 | stat_R; break;
                case 0x60: PORTB = 0xc0 | stat_R; break;
                case 0xc0: PORTB = 0x90 | stat_R; break;
                case 0x90: PORTB = 0x30 | stat_R; break;
                default: break;
                }
        }
        stat_L = PINB & 0xf0;

        if(m_speed_R < count_R)
        {
            count_R =0;
            switch(stat_R) {  //increments RIGHT motor one step
                case 0x00: PORTB = 0x06 | stat_L; break;
                case 0x03: PORTB = 0x06 | stat_L; break;
                case 0x06: PORTB = 0x0c | stat_L; break;
                case 0x0c: PORTB = 0x09 | stat_L; break;
                case 0x09: PORTB = 0x03 | stat_L; break;
                default: break;
                }
        }
        stat_R = PINB & 0x0f;
    }
    if(turn_flag == 1)
        {
        if(m_speed_L < count_L)
        {
            count_L =0;
            switch(PINB) {  //increments motor one step
                case 0x00:       PORTB = 0b00111100; break;
                case 0b00111100: PORTB = 0b01100110; break;
                case 0b01100110: PORTB = 0b11000011; break;
                case 0b11000011: PORTB = 0b10011001; break;
                case 0b10011001: PORTB = 0b00111100; break;
                default: PORTB = 0; break;
                }
        }

        }
}

//MAIN FUNCTION.....
//.....

int main(void)
{
```

```
//>>>>>>__________INITIALIZATION_____________________________________________________
    DDRB = 0xff; //motor port initialize to all outputs
    PORTB = 0x00;

    DDRC = 0xff;      // set all pins on portC to output  (could also use DDRC = 0b11111111) for LCD
    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(0);
    //lcd_string("Initialize...      ");

    //lcd_row(1); lcd_string("PortA - Keypad.....");
    DDRA = 0xf0; //set pins 7-4 as output and pins 3-0 as input for keypad
    PORTA = 0x0f; //sets pins 3-0 to be pulled up and all columns to 0's
    //lcd_row(1); lcd_string("PortA - DONE       ");

    //lcd_row(1); lcd_string("PortE - MagSwtches..");    //setting up port E for Magnetic Switches
    DDRE = 0x00; //sets all pins to be inputs, magnetic sensor array
    PORTE = 0xFF; //set up all input pins to be pulled up
    //lcd_row(1); lcd_string("PortE - DONE       ");

    //lcd_row(1); lcd_string("PortD - Sonar.....");
    DDRD = 0b11101010;      //set pins 7,6,5,3,1 as output and pins 4,2,0 as input for sonar
    PORTD = 0x00; //set all pins to tristate (if input) and 0 (if output)
    //lcd_row(1); lcd_string("PortD - DONE       ");

    //lcd_row(1); lcd_string("PortF - LineFollow..");
    adc_init(); //initializes A/D Converter
    //lcd_row(1); lcd_string("PortF - A/D Config'd");

    DDRG = _BV(DDG0);
    PORTG = 0x00;

    lcd_clear(); lcd_string("WaterBot is Ready!  ");
    lcd_delay(hello);

    //while(PINA == 0x0f) //Loops until any # is pressed
    //{
    //lcd_delay(25000);
   //}
   //lcd_delay(100000);

//>>>>>>>>>>________SELECT A TEST_____________________________
    lcd_clear(); lcd_string("Select #for Task    ");
    lcd_row(1); lcd_string("1:a 4:Sr 7:lf *:mr");

    PORTA = 0xef; // keypad set to look for 1, 4, 7 or *

    while (1)
    {
     if(PINA == 0xed) //if '4' was pressed
      {
       lcd_clear(); lcd_string("4 was pressed.......");
```

```
      lcd_row(1); lcd_string("Prepare for sonar   ");
      lcd_delay(hello);
      sonar_test();
      break;
     }

    if (PINA == 0xee) //if '1' was pressed
     {
      lcd_clear(); lcd_string("1 was pressed.......");
      lcd_row(1); lcd_string("Prepare for all test");
      lcd_delay(hello);
      break;
     }

    if (PINA == 0xeb) //if '7' was pressed
     {
      lcd_clear(); lcd_string("7 was pressed.......");
      lcd_row(1); lcd_string("Prepare for LF     ");
      lcd_delay(3*hello);
      PORTB = 0x00;
      sei();
      init_timer0();
      lcd_clear();
      while(1) {LF_test();}
      break;
     }
     if (PINA == 0xe7) //if '*' was pressed
     {
      lcd_clear(); lcd_string("* was pressed.......");
      lcd_row(1); lcd_string("Prepare for ITR test");
      lcd_delay(3*hello);
      PORTB = 0x00;
      motors_test();
      break;
     }

   }

   lcd_clear();

//this loop outputs value of sensors on LCD
   while(1)
   {
   lcd_row(0);
//  lcd_string("1:");
   char switch_str [8];
   lcd_string(itoa(PINE, switch_str, 16));
   lcd_string(" ");

//  lcd_string("2:");
   char keypd_str [8];
   lcd_string(itoa(PINA, keypd_str, 16));
   lcd_string(" ");
```

```
//  lcd_string("3:");
    char echo0_str [8];
    lcd_string(itoa(sonar_ping(0), echo0_str, 16));
    lcd_string(":");

//  lcd_string("3:");
    char echo1_str [8];
    lcd_string(itoa(sonar_ping(1), echo1_str, 16));
    lcd_string(":");

//  lcd_string("3:");
    char echo2_str [8];
    lcd_string(itoa(sonar_ping(2), echo2_str, 16));
    lcd_string(" ");

     adc_chsel(0);
     lcd_delay(10000);
     LF_0 = ADCH; //right
     lcd_delay(10000);
     adc_chsel(1);  //right mid
     lcd_delay(10000);
     LF_1 = ADCH;
     lcd_delay(10000);
     adc_chsel(2);  // L mid
     lcd_delay(10000);
     LF_2 = ADCH;
     lcd_delay(10000);
     adc_chsel(3);  //L
     lcd_delay(10000);
     LF_3 = ADCH;

    lcd_row(1);
    lcd_string(" ");
    char adc0_str [8];
    lcd_string(itoa(LF_0, adc0_str, 16));

    lcd_string(" ");
    char adc1_str [8];
    lcd_string(itoa(LF_1, adc1_str, 16));

    lcd_string(" ");
    char adc2_str [8];
    lcd_string(itoa(LF_2, adc2_str, 16));

    lcd_string(" ");
    char adc3_str [8];
    lcd_string(itoa(LF_3, adc3_str, 16));
  }
    return 0;
}
```

```
#ifndef _globals_h
#define _globals_h

//Motor Globals     >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
#define max_speed        19  // speed inverse (step delays)
#define min_speed        150
#define max_speed_inc    10   //percent
#define max_speed_dec    20   //percent
#define R 0 //Right mtr_label
#define L 1 //Left mtr_label
#define S 0 //Stop
#define F 1 //Forward
#define B 2 //Backward
#define init_speed       60

int m_lots = 20;
int m_some = 15;
int m_litt = 10;

int m_fast = 22; //40, 70, 100 works great //60, 70, 80 is better //65 70 75 even better
int m_medi = 24; //75 80 85 works  //85 90 95 too slow
int m_slow = 26; //50 55 60 works fine  //35 40 45 is great  //30 35 40 is greater  //see notebook for rest
int m_stop = 100; //goes really slow

int turn_flag = 0; //true if turning around
int done_flag = 0;
long int time_off = 1000; //time to wait before watering again

volatile uint8_t dir_L = 0;
volatile uint8_t dir_R = 0;
volatile uint8_t target_dir_L = 0;
volatile uint8_t target_dir_R = 0;
volatile uint8_t speed_L = 0;
volatile uint8_t speed_R = 0;
volatile uint8_t target_speed_L = 0;
volatile uint8_t target_speed_R = 0;
volatile uint8_t speed_diff_L = 0;
volatile uint8_t speed_diff_R = 0;

//Motor Interrupts >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
int stat_L = 0;
int stat_R = 0;
int m_speed_L = init_speed;
int m_speed_R = init_speed;
int count_L = 0;
int count_R = 0;

//LCD Globals     >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
long int lcd_delay_time = 1000;       //to use in LCD delay function

//SONAR Globals     >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
int end_line = 0x0500; //used for turning around
int obstacle = 0x0400; //used for seeing obstacle
```

```
//Init Globals      »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
long int hello = 100000;            //initialization delays

//LF Globals      »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
int LF_thresh = 0x95;
int LF_0 = 0;
int LF_1 = 0;
int LF_2 = 0;
int LF_3 = 0;
int LF_status = 0;
int LF_status_old = 0;

int LF_L = 0;
int LF_R = 0;
int LF_P = 0;

int solid_cnt = 0;


//A/D Converter Globals      »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
long int adc_delay = 1000;


//Arbitrator Globals      »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
int new_speed_L = init_speed;
int new_speed_R = init_speed;

//Motor Globals      »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»
#endif /*globals.h*/
```

```
// LINEFOLLOW.H line following code


#ifndef _linefollow_h
#define _linefollow_h

#include "arbitrator.h"
#include "motors.h"

void turn_around_init();
void turn_around();


void LF_test()
{
    if((PINE & 0b11100010) ==0b10100000)
        {  water_plant();  }

    LF_status = 0;

    adc_chsel(0); lcd_delay(adc_delay);
    LF_0 = ADCH; //right
    if(LF_0 > LF_thresh) LF_status = LF_status | 0x01;

    adc_chsel(1); lcd_delay(adc_delay);
    LF_1 = ADCH;//right mid
    if(LF_1 > LF_thresh) LF_status = LF_status | 0x02;

    adc_chsel(2); lcd_delay(adc_delay);
    LF_2 = ADCH;// L mid
    if(LF_2 > LF_thresh) LF_status = LF_status | 0x04;

    adc_chsel(3); lcd_delay(adc_delay);
    LF_3 = ADCH;// L
    if(LF_3 > LF_thresh) LF_status = LF_status | 0x08;

    lcd_row(0);
    //char strn [8];  //debug output
    //lcd_string(itoa(LF_status, strn, 2)); //debug output

    if(turn_flag == 0)
    {
        if(sonar_ping(1) < obstacle)
        {   stop_motors();
            lcd_clear(); lcd_string("      Obstacle      ");
            while(sonar_ping(1) < obstacle)
                {stop_motors(); lcd_delay(100000);}
            m_speed_L = init_speed; m_speed_R = init_speed; //go if obstacle moves
        }
    switch(LF_status)
    {
        case 0x00: lcd_string("   No Line          ");
            if(LF_status_old == 0x01) {LF_L = m_fast; LF_R = m_stop; //m_slow
                LF_status = 0x01; break;}
```

```
            if(LF_status_old == 0x08) {LF_L = m_stop; LF_R = m_fast; //m_slow
                LF_status = 0x08; break;}//go straight (no line found)
        case 0x01: lcd_string("  Go Right (lot)    ");
            LF_L = m_fast; LF_R = m_slow;  break;//go right (a lot)
        case 0x03: lcd_string("  Go Right          ");
            LF_L = m_fast; LF_R = m_medi;  break;//go right
        case 0x02: lcd_string("  Go Right (little) ");
            LF_L = m_medi; LF_R = m_slow;  break;//go right(a little)
        case 0x06: lcd_string("  Go Straight       ");
            LF_L = m_fast; LF_R = m_fast;  break;//go straight (on line)
        case 0x04: lcd_string("  Go Left (little)  ");
            LF_L = m_slow; LF_R = m_medi;  break;//go left(alittle)
        case 0x0c: lcd_string("  Go Left           ");
            LF_L = m_medi; LF_R = m_fast;  break;//go left
        case 0x08: lcd_string("  Go Left (lot)     ");
            LF_L = m_slow; LF_R = m_fast;  break;// go left (alot)
        case 0x0f:
            solid_cnt++;
            if (solid_cnt > 3 && LF_status_old == 0x0f &&
            sonar_ping(0) < end_line &&
            sonar_ping(1) < end_line &&
            sonar_ping(2) < end_line)
            {
                solid_cnt = 0;
                lcd_string("  Turn Around       ");
                stop_motors();
                turn_around_init();
            }

            break;
        default:   lcd_string("  Whoa!!            ");
            LF_L = m_slow; LF_R = m_slow;  break;//go straight
    }
    LF_status_old = LF_status;
    arbitrate();
    } //end of IF turn flag == 0
    if(turn_flag == 1) turn_around();
}

void turn_around_init()
{
    PORTB = 0x00;
    m_speed_L = 30000; m_speed_R = 30000;
    lcd_delay(500000); //delay just because it looks good
    turn_flag = 1;
    m_speed_L = m_slow; //sets turn speed for both motors in this function
    lcd_delay(1000000); //give robot enough time to leave the line
}

void turn_around()
{
    if(//LF_status == 0x01
    //|| LF_status == 0x02
    //|| LF_status == 0x03
```

```
     0 || LF_status == 0x02
    || LF_status == 0x06
    || LF_status == 0x04
    //|| LF_status == 0x0c
    //|| LF_status == 0x08
    )
    {
        stop_motors();
        PORTB = 0;
        turn_flag = 0;
        switch(done_flag) {
            case 0: done_flag = 1; break;
            case 1:
                { long int y = 0;
                while(y < time_off) { stop_motors(); lcd_delay(10000); y++; }
                done_flag = 0;}
            default : break; }
        lcd_row(1); lcd_string("stop turning");
        lcd_delay(300000);

    }
    lcd_row(1); lcd_string(" YO");
}

#endif /*linefollow.h*/
```

```
#ifndef __motors_h__
#define __motors_h__

void motors_test(void);
void stop_motors(void);

//Functions>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void motors_test(void)
{
    long test_delay = 20000;
    PORTB = 0x00;

    while(1)
    {
        lcd_delay(test_delay);
        //PORTB = 0b00111100;
        PORTB = 0b00001100;
        lcd_delay(test_delay);
        //PORTB = 0b01100110;
        PORTB = 0b00000110;
        lcd_delay(test_delay);
        //PORTB = 0b11000011;
        PORTB = 0b00000011;
        lcd_delay(test_delay);
        //PORTB = 0b10011001;
        PORTB = 0b00001001;

        switch(PINA)
        {
            case 0xee: test_delay = test_delay + 500; break;
            case 0xed:
            {
            if(test_delay > 100) test_delay = test_delay - 100;
            if(test_delay <= 100 && test_delay > 10) test_delay = test_delay - 10;
            if(test_delay <= 10 && test_delay > 0) test_delay = test_delay - 1;
            break;
            }
        /*  case 0xeb:
            {
            PORTB = 0;
            char echo15_str [16];
            lcd_string(itoa(test_delay, echo15_str, 10));
            lcd_string(":");
            lcd_delay(1000000);
            }
        */
            default : break;
        }
    }
}

//>>>>>>>>>>>>>>>>>>>>>>>>>>>>
void stop_motors (void)
{
```

```
    m_speed_L = 30000; m_speed_R = 30000; //really slow motors
    count_L = 0; count_R = 0; //completely stop stepping
    PORTB = 0x00;
}

#endif /*motors.h*/
```

```
// waterplant.H line following code


#ifndef _water_h
#define _water_h

void water_plant();
void amount(long int);

void water_plant()
{
    m_speed_R = 30000;
    m_speed_L = 30000;
    PORTB = 0x00;
    lcd_clear(); lcd_string("Watering         ");


    int plant_num = 0;
    plant_num = (PINE & 0b00011100)/4; // only use 3 bits for plant #
    lcd_row(1);
    lcd_string("  Plant # ");
    char plant_strg [2];
    lcd_string(itoa(plant_num, plant_strg, 10));
    lcd_string("        ");
    PORTB = 0x00;

    switch(plant_num)
    {
        case 0: amount(1000000); break;
        case 1: amount(2000000); break;
        case 2: amount(3000000); break;
        case 3: amount(4000000); break;
        case 4: amount(5000000); break;
        case 5: amount(10000000); break;
        default: lcd_row(1); lcd_string(" Plant not on list. ");
            lcd_delay(1000000); break;
    }
    m_speed_L = m_medi; //reset motor speeds to straight forward and move on
    m_speed_R = m_medi;
    lcd_delay(600000);
    lcd_clear();
}
void amount(long int drops)
{
    PORTG = 0x01; //turn pump on
    lcd_delay(drops); //keep pump on during delay
    PORTG = 0x00; //turn pump off
    //lcd_row(1);
    //lcd_string("     Done      ");
}

#endif /*water.h*/
```

```
// LINEFOLLOW.H line following code


#ifndef _arbitrator_h
#define _arbitrator_h
void arbitrate();

void arbitrate()
{
    if(1)   //if LF
    {
        new_speed_L = LF_L;
        new_speed_R = LF_R;
        if(new_speed_L > max_speed && new_speed_L < min_speed)
            m_speed_L = new_speed_L;
        if(new_speed_R > max_speed && new_speed_R < min_speed)
            m_speed_R = new_speed_R;
    }
    return;
}

#endif /*arbitrator*/
```

```
// SONAR.H sonar code


#ifndef _sonar_h
#define _sonar_h


//////////////////////////////////////////////////////////////////////
void sonar_test(void);
int sonar_ping(int dir);     // pings specified sonar
//////////////////////////////////////////////////////////////////////
int sonar_ping(int dir)      //pings specified sonar and returns a count
{
   int pulse;
   int check;

   switch(dir)
   {
        case 0: pulse = 0b00000010; check = 0b00000001; break;
        case 1: pulse = 0b00001000; check = 0b00000100; break;
        case 2: pulse = 0b00100000; check = 0b00010000; break;
        default:  break;
    }
   PORTD = pulse; //set up a pulse
   int count_a = 0; //make the pulse 200 cycles long
   while (count_a < 200) //at least 12.5 uSeconds (must be at least 10)
   {
      count_a = count_a + 1;
   }
   PORTD = 0x00; //tells module to send pulse
   int time_count = 0;
   while (PIND != check)
   {
   time_count = time_count + 1;
   }
   while (PIND == check)
   {
   time_count = time_count + 1;
   }
   return(time_count);
}

void sonar_test(void)
{
while(1)
 {
   lcd_row(0);
   char echo0_str [8];
   lcd_string(itoa(sonar_ping(0), echo0_str, 10));
   lcd_string("  ");

   char echo1_str [8];
   lcd_string(itoa(sonar_ping(1), echo1_str, 10));
   lcd_string("     ");
```

```
   lcd_row(1);
   char echo2_str [8];
   lcd_string(itoa(sonar_ping(2), echo2_str, 10));
   lcd_string(" ");
  }
}


//////////////////////////////////////////////////////////////////
#endif /*sonar.h*/
```

```
//  ADC.H  analog to digital converter code


#ifndef _adc_h
#define _adc_h

//////////////////////////////////////////////////////////////////////
void adc_init(void);
void adc_chsel(uint8_t channel); //select AD channel

//////////////////////////////////////////////////////////////////////

void adc_init(void)
{
   /* configure ADC port (PORTF) as input */
   /*
adc_init() - initialize A/D converter
 Initialize A/D converter to free running, start conversion, use
 internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming
 16 MHz MCU clock)
 left adjust 10-bit value
 */
   DDRF  = 0x00;
   PORTF = 0x00;

   ADMUX = _BV(REFS0)|_BV(ADLAR);
   ADCSR = _BV(ADEN)|_BV(ADSC)|_BV(ADFR) | _BV(ADPS2)|_BV(ADPS1)|_BV(ADPS0);
}

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

//////////////////////////////////////////////////////////////////////
#endif /*adc.h*/
```

```
#ifndef __lcd_h__
#define __lcd_h__

void lcd_delay(long int delay);           // delay for specified amount (1000 is typical)
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_delay(long int delay)       // delay for specified amount (1000 is typical)
{
   long int ms_count = 0;
   while (ms_count < delay)
   {
      ms_count = ms_count + 1;
   }
}

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

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(lcd_delay_time);
   PORTC = (temp_data & 0b11110001);
   lcd_delay(lcd_delay_time);                    // upper nibble

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

void lcd_init()
{
   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.


   while (value > 0)              // 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); break;
      case 1:   lcd_cmd(0xC0); break;
      default: break;
   }

   return;
}




#endif
```