Racer-X

Mike Bonestroo

Written Report

EEL 5666 – Intelligent Machines Design Lab

Spring 2003

March 15, 2004

Table of Content

Abstract

The purpose of the Racer-X is to race against a R/C car for practice. This way a young child or adult could practice their racing without having to have another person to compete against. The car will be able to increase speed on straight-aways and decrease speed for the turns while keeping track of the number of laps that the car has completed.

Executive Summary

Racer-X is a high-speed line follower. It has the ability to count the amount of laps it has taken and display them on the on-board lcd screen. Racer-X's also changes speeds according to the type of track it is on; if a straightaway, Racer-X speeds up, if a curve or a turn, Racer-X slows down. Racer-X is equipped with a 595RPM motor that allows him to reach high speeds, giving a R/C car a good hard competitor. With the front wheel steering and single wheel drive, it leaves very few parts to go bad, so the durability of Racer-X will be high. All a person would have to do is lay down a strip of electrical tape around his/her track and they instantly have some competition to practice with. Plus the racer would always know who wins the race because Racer-X would stop after the pre-determined amount of laps, telling the racer he/she has completed the race.

Introduction

Think of a board young child wanting to race his R/C car but not being able to because none of his friends are available to play. With Racer-X the child would have his problem solved, he/she could just pop some batteries in Racer-X and then he/she is ready to race.

Integrated System

Racer-X's brain is the ATmega128 on a Mavric-II development board. Features of the board are 128K Program FLASH, 4K Static RAM, 4K EEPROM • dual level shifted UARTs • RS485 on-board • I2C ready w/pull-up resistors installed • up to 53 digital I/O pins • Selectable clock frequency of 16 MHz or 14.7456 MHz (select at order time) • Advanced, low drow-out voltage regulator on-board accepts 5.5-15V input with reverse polarity hookup protection • Small size at 2.2 x 3.6 inches

The robot also will be equipped with an LCD display that will give back information on what is happening to the robot. There will also be 2 IR from Sharp (GP2D12). Then there is Hamamatsu Photoreflector (4) to follow the line on the ground. The robot will have multiple bump switches to kill the system in the case of a collision. The motor that I am using for this robot is a Gear Head Motor - 12vdc 43:1 290rpm (4mm shaft). Description - Voltage = 12vdc, RPM = 290, Reduction = 43:1 metal gear, Stall Torque = 74.8 oz-in (5.4 kg-cm). Sozbots Dual RC H-Bridge Motor Controller was used with the motor to handle the forward and backward motion. Description - Channels = Dual, Voltage = 6v - 18vdc, Peak Current = 5.0 amp.

Mobile Platform

The platform that I am using is a specially designed platform modeled from an existing R/C car (TC3 Team Associated). I designed the car in autoCAD and use the T-tech machine to cut it out of 1/8th inch think balsa wood. The design is to be low to the ground to look like a normal version of the R/C car that it is modeled from. The car body will also be the same dimensions so that it can use the cover film that is designed to sit on the model R/C. The size of Racer-X may be a little larger then some other robots built in the lab before but will hold the sensors perfectly.

The robot is around 12.5” long and 5” wide with the front of the body brought in an inch for the turning ability of the front wheels. Talking about the front wheels, they will be the source of turning; like a true car would turn the wheels can change the direction of the vehicle. The back wheels will only have one as the driving force of the robot. The other back wheel will simply just turn on a ball bearing. The reason for this is to eliminate the need for a back differentiacal axle.

Actuation

The robot is driven by a single motor connected to the rear right wheel for forward and backward movement, instead of the conventional two motors or axle. I did this to simplify the building of the car so I can get on to programming since that is not my strong suit. The motor that I am using for this robot is a Gear Head Motor - 12vdc 43:1 290rpm (4mm shaft). Description - Voltage = 12vdc, RPM = 290, Reduction = 43:1 metal gear, Stall Torque = 74.8 oz-in (5.4 kg-cm). Sozbots Dual RC H-Bridge Motor Controller was used with the motor to handle the forward and backward motion. Description - Channels = Dual, Voltage = 6v - 18vdc, Peak Current = 5.0 amp.

The steering of the front wheels will be done be the means of a servo. The servo is HITEC 311 STANDARD SERVO that you can find at any hobby shop or robot site. (http://www.go-advanced.com/ADV_ROBOT/electronics.htm). The use of this servo and of mechanisms that are used to turn the R/C car combined with some autoCAD designed parts will complete the turning device.

Special Sensor Report

IR Package

The front of the robot is equipped with two IR Sharp GP2D12s. This sensor takes a continuous distance reading and reports the distance as an analog voltage with a distance range of 10cm (~4") to 80cm (~30"). The interface is 3-wire with power, ground and the output voltage and requires a JST 3-pin connector. They are used in the obstacle avoidance behavior and insure that the robot doesn't run into a wall when it goes off the track. The IR input is read by the microprocessor and if the IR reads a number higher then the threshold that is declared in the code the robot takes action according to which IR says there is an object in front of it.

Bump Switch Package

The bump switches/sensors are located in the front and the rear of the robot. As of now I am only going to use one on each ends, but this may change as I learn more about the way the robot moves and gets stuck in corners. The way that these work is simple, the sensor send the microprocessor a signal that is given to the sensor with the use of resistors. Once the microprocessor receives a signal from the switch it knows that there is something in front of it or behind it and the robot takes action on moving away from the object. If there is no signal coming from the sensor then the robot carries on with its normal behaviors.

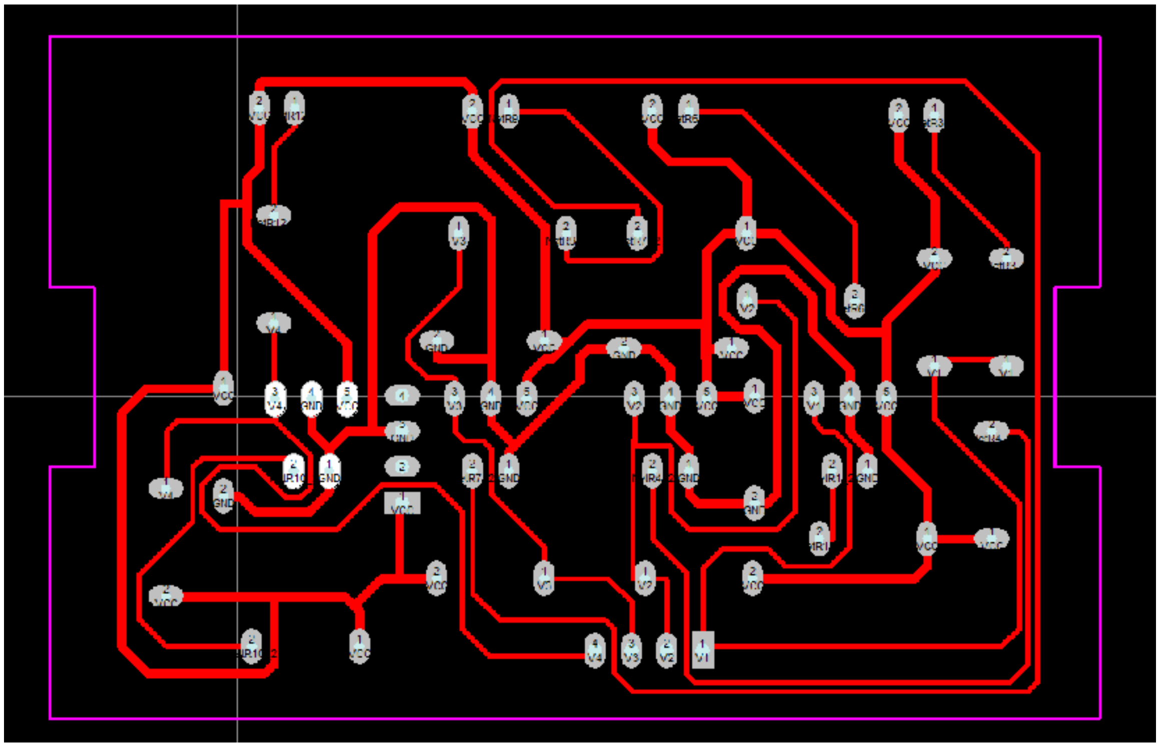
CDS cell Package

I am using these sensors to tell the robot to speed up or decrease speed. There is two CDS cells located on either side of the robot on the undercarriage. CDS work by having varied resistance according to the amount of light that they are subjected to. Under dim to low light, there is high resistance. Under high intense light, there is low resistance. With this varied resistor and a fix resistor you can use a calculation that will help you create a varied value input to your microprocessor using an A/D port. With the varying inputs you can declare a threshold to make a certain behavior run if broken. These sensors were going to be running under the car to change the speeds of the vehicle as it went around the track. Unfortunately the CDS cells are way to slow to react to a simple led on the track. So with the CDS not able to sense an LED to slow down the car I had to move to another choice. I would caution anyone that is choosing this sensor, if you are moving at any rate of speed faster then a servo would move, don't use a CDS cell.

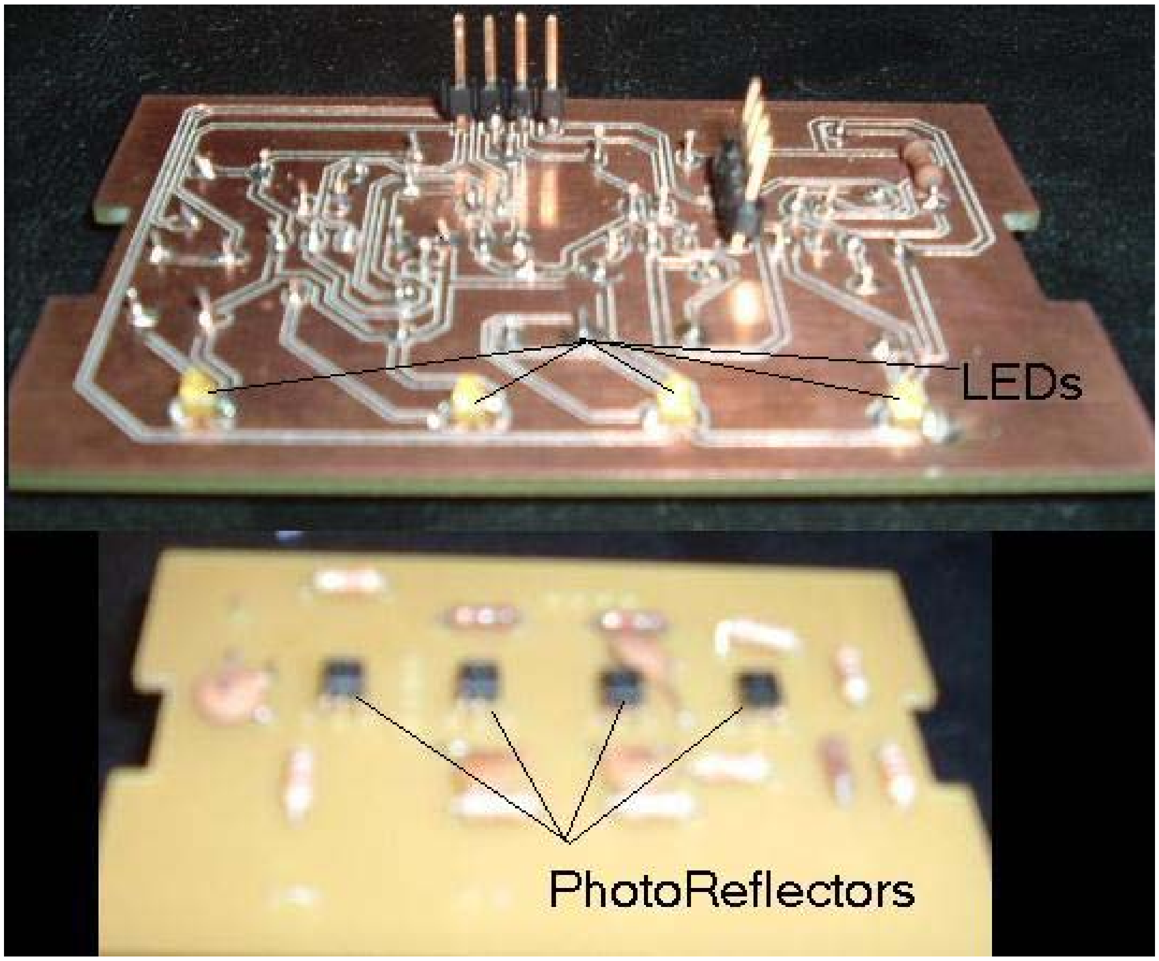
Line Following Steering Suite

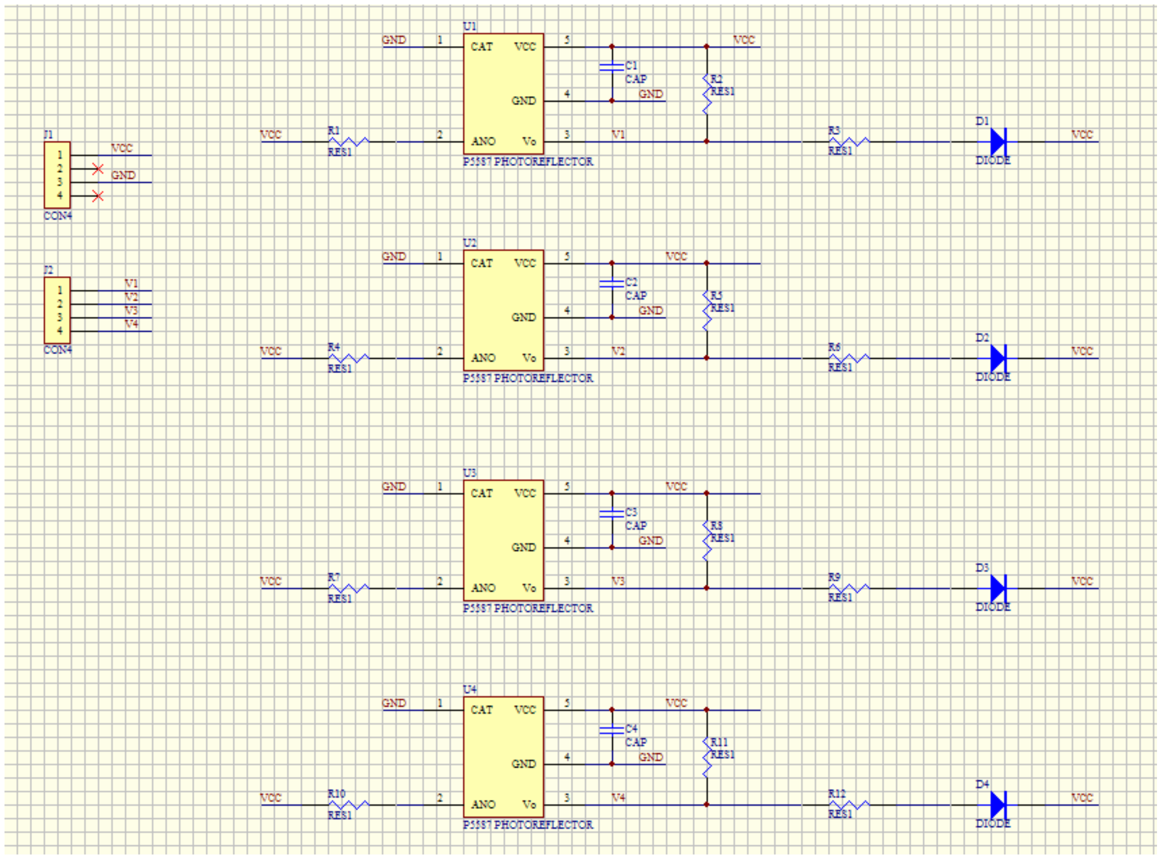
With not having much background in engineering, my special sensor is not much of a special sensor to most. In my opinion my whole project is a special sensor because it is a victory every time I get any results from my robot. But there is one unusual part to my car robot that interacts with a sensor that is not seen on most robots. So the combination of the two is going to be what my special sensor report is on.

My robot is a line following robot that is to be a second car for a young child to race when no other children are around to race him. To achieve the task of following a line I have used four Hamamatsu Photoreflectors. These four photoreflectors have been mounted onto a Protel board that was designed by Steven Pickles and Myself, which helps show if the photoreflector's see a line or not (below).



This board has four LED lights that are at the top of the board that signal it the line is being picked up by the Photoreflector. This is mainly a way for us to see if the sensor is working properly. Also on the board is the four sensors themselves, the five whole arrangements are there locations, and a series of resistors and capacitors. The schematics are as follows for this board.

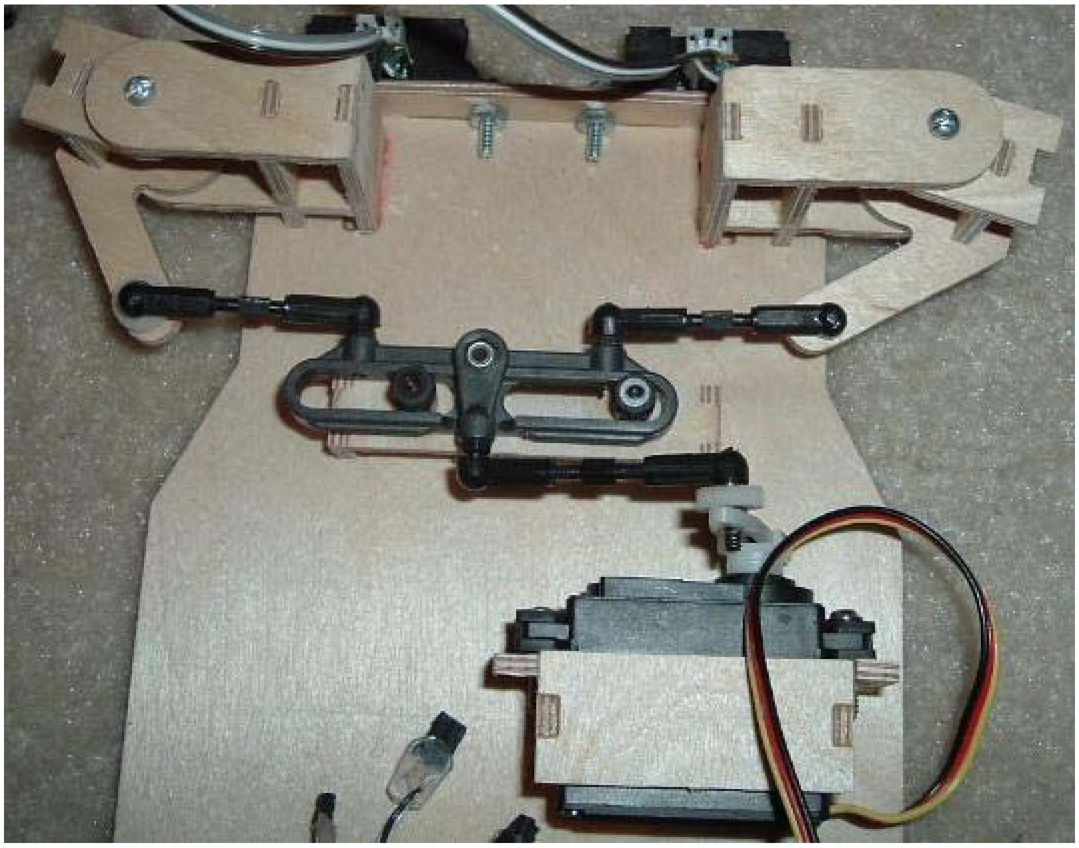

LEDs
PhotoReflectors



The arrangement of the photoreflectors is half of an inch between each one, so that I can get four different ranges of variables on how sharp to turn the robot. This configuration has proven to work extremely well with my robot. What I used as the line for the photoreflectors to pick up is three quarters inch electrical tape. Now you may be wonder what is different, this is all stuff that is the same as the past, well I moved in the outer two photoreflectors which is something that most robots haven't done, which has also turned into a great success. Another difference is explained in the next part of the report.

The steering in my robot is the part the sets it apart from the many robots done in this class. As it is going to steer like a true car that you drive. This type of steering has proved to be challenging, but I think that it has been done correctly and proves that front wheel steering is not as bad as one would assume.

The steering is off of a R/C car that I have had for some time, shown below, that I have converted to be used on my robot. It uses a series of ball and socket joints connected to dual eyelets to turn the front wheels. The servo is the actuator that creates the movement according to the input that it is given by the Microprocessor. There is seven different turning positions: straight, slight left, slight right, medium left, medium right, sharp left, and sharp right.



With the robot completely done, and steering tried and true, I would say that there are some bugs to be worked out. The steering system is limited by a few factors on my robot. In the picture above you can see that the steering is almost at it maximum right turn, but if you notice the servo is nowhere near it's maximum. The max right hand turn in this configuration is about a complete circle of ~4’8”. To decrease the amount needed to make a complete turn all one would need to do is increase the “eye loops” of the centerpiece of the system. If you where to do that, make sure that you move the “eye loop pins” farther apart so that the distance of movement is increased. I am not sure of how tight of a turn that this setup would be able to make, but I know that you could greatly improve on the radius of the turn if you were to do this little modification. I would also advise that you be careful, as if the car has too sharp of a turning radius that it will start to push the front end straight and not turn. I would just play with different turning values and find this point by trail and error.

One other bug that I have run into is a problem getting centered onto a line off a curve. This problem I believe could be corrected in code, but I know it could be corrected in the mechanics too. As I mentioned earlier, there is 7 turning degrees. This I though at first was going to be great because I had 3 different rates to turn each way. But once I had the robot moving, I found three is not enough. When following a line on a curve the robot is going full right or left, so once a straight section come up the robot seems to have a hard to centering on it. The robot will go from the full right turn and drive right over the line causing it to go to the full left turn. To combat this I feel if you were to add two more photoreflectors to the line following board you can gain two more different rates of turning, and this would bring your different turning values closer to one another in the programming as too not “over correct” on a straight-away. As I said before there might be a way to code your way out of this, but since I am not proficient at all with coding, this mechanical fix is how I would modify the robot to fix the problem.

Behaviors

Obstacle Avoidance-This is to avoid hitting objects in the path of the robot. To avoid obstacles I am using two IR's in the front of the vehicle. This a VERY basic setup because the main function of the robot is to follow a pre made path with only obstacles that I would make, which I didn't.

Line Following- The line following is done with the four photoreflectors on a circuit in the front of the vehicle. This circuit has proven to be very efficient and with the LEDs on the board it easy to see what the sensor is reading.

Lap Counting- The laps are being counted by the same sensor being used to follow the line. It counts the laps only when a certain value is being read in by the sensor around the track.

Speed Regulating- Again the line following device regulates the speed as well. When certain values are read by the sensor, the speed changes dependent on them.

Experimental Layout and Results

The robot chassis is built, but not totally final. The layout of where each component as it currently is, is final, but there is still a few small things to be added. I am currently working on the programming of the PWMs, once this is done the Obstacle avoidance should be completed within the day.

Conclusion

In conclusion of me completing the class and my robot, I would say that this is one of the most information classes I have ever taken. My robot has taught me more things about electrical issues and simple behavior problems. Even only being a business major I pulled it off. I feel that the robot is about as good as anyone else could have gotten it, other then maybe better coding. I think that mechanically it works for what it is meant to perform and I am pleased with the results that I have come out with.

Things that I would have like to differently if I were to start the project over are the two I mentioned before about the steering in the sensor report. I feel with those modifications that the robot would steer a lot better without having problems re-centering on a straightaway. Another change I would make is using a sonar sensor in front and not the two IR's. This way there wouldn't be any blind spot in the front protecting the line following circuit better. From experience, you want to do this, with my two IR's still ran full blast into a chair leg, breaking my circuit. As for the rest of the robot I am VERY pleased with the results. I would say that the robot as a whole has come together better then I had imagined, I thought since I was a business major that I would not be able to achieve all the goals that I wanted too, but the final product has proven that I can and did. I am impressed at how well I thought out the chassis, only cutting it out once even with the front wheel steering.

I was limited in my project on the coding side. Since I have never used C++ before I had no idea how to even start. Thankfully I had a friend in the class with me that help out on the coding. Since he had his own project he couldn't help all the time and I did end up coding a good amount on my own. So I feel that my code could be improved a bit by someone that knows more about C++ then I do.

Documentation

Appendices

The Makefile I used

```
# WinAVR Sample makefile written by Eric B. Weddington, Jörg Wunsch, et al.
# Released to the Public Domain
# Please read the make user manual!
#
# Additional material for this makefile was submitted by:
#  Tim Henigan
#  Peter Fleury
#  Reiner Patommel
#  Sander Pool
#  Frederik Rouleau
#  Markus Pfaff
#
```

```
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF (for use with AVR Studio 3.x or VMLAB).
#
# make extcoff = Convert ELF to AVR Extended COFF (for use with AVR Studio
#               4.07 or greater).
#
# make program = Download the hex file to the device, using avrdude.  Please
#               customize the avrdude settings below first!
#
# make filename.s = Just compile filename.c into the assembler code only
#
# To rebuild project do "make clean" then "make all".
#


# MCU name
MCU = atmega128

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = main

# Optimization level, can be [0, 1, 2, 3, s]. 0 turns off optimization.
# (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = 0


# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c

# If there is more than one source file, append them above, or modify and
# uncomment the following:
SRC += delay.c motor.c lcd.c adc.c avoid.c tracker.c bumped.c lapcounter.c

# You can also wrap lines by appending a backslash to the end of the line:
#SRC += baz.c \
#xyzzy.c
```

```
# List Assembler source files here.
# Make them always end in a capital .S.  Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/Win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.
ASRC =


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
EXTRAINCDIRS =


# Optional compiler flags.
#  -g:        generate debugging information (for GDB, or for COFF conversion)
#  -O*:       optimization level
#  -f...:     tuning, see gcc manual and avr-libc documentation
#  -Wall...:  warning level
#  -Wa,...:   tell GCC to pass this to the assembler.
#    -ahlms:  create assembler listing
CFLAGS = -g -O$(OPT) \
-funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums \
-Wall -Wstrict-prototypes \
-Wa,-adhlns=$(<:.c=.lst) \
$(patsubst %,-I%,$(EXTRAINCDIRS))


# Set a "language standard" compiler flag.
#   Unremark just one line below to set the language standard to use.
#   gnu99 = C99 + GNU extensions. See GCC manual for more information.
#CFLAGS += -std=c89
#CFLAGS += -std=gnu89
#CFLAGS += -std=c99
CFLAGS += -std=gnu99



# Optional assembler flags.
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -ahlms:    create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#            files -- see avr-libc docs [FIXME: not yet described there]
ASFLAGS = -Wa,-adhlns=$(<:.S=.lst),-gstabs



# Optional linker flags.
#  -Wl,...:   tell GCC to pass this to linker.
#  -Map:      create map file
#  --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref



# Additional libraries

# Minimalistic printf version
#LDFLAGS += -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires -lm below)
#LDFLAGS += -Wl,-u,vfprintf -lprintf_flt

# -lm = math library
LDFLAGS += -lm




# Programming support using avrdude. Settings and variables.

# Programming hardware: alf avr910 avrisp bascom bsd
# dt006 pavr picoweb pony-stk200 sp12 stk200 stk500
#
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = avrisp


AVRDUDE_PORT = com1      # programmer connected to serial device
#AVRDUDE_PORT = lpt1    # programmer connected to parallel port

AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c
$(AVRDUDE_PROGRAMMER)
```

```
# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE += -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_FLAGS += -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_FLAGS += -v -v




# ---------------------------------------------------------------------------

# Define directories, if needed.
DIRAVR = c:/winavr
DIRAVRBIN = $(DIRAVR)/bin
DIRAVRUTILS = $(DIRAVR)/utils/bin
DIRINC = .
DIRLIB = $(DIRAVR)/avr/lib


# Define programs and commands.
SHELL = sh

CC = avr-gcc

OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size


# Programming support using avrdude.
AVRDUDE = avrdude


REMOVE = rm -f
COPY = cp

HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
```

```
ELFSIZE = $(SIZE) -A $(TARGET).elf



# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:




# Define all object files.
OBJ = $(SRC:.c=.o) $(ASRC:.S=.o)

# Define all listing files.
LST = $(ASRC:.S=.lst) $(SRC:.c=.lst)

# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)



# Default target.
all: begin gccversion sizebefore $(TARGET).elf $(TARGET).hex $(TARGET).eep \
	$(TARGET).lss $(TARGET).sym sizeafter finished end


# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
```

```
begin:
	@echo
	@echo $(MSG_BEGIN)

finished:
	@echo $(MSG_ERRORS_NONE)

end:
	@echo $(MSG_END)
	@echo


# Display size of file.
sizebefore:
	@if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_BEFORE);
$(ELFSIZE); echo; fi

sizeafter:
	@if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE);
echo; fi



# Display compiler version information.
gccversion :
	@$(CC) --version




# Convert ELF to COFF for use in debugging / simulating in
# AVR Studio or VMLAB.
COFFCONVERT=$(OBJCOPY) --debugging \
	--change-section-address .data-0x800000 \
	--change-section-address .bss-0x800000 \
	--change-section-address .noinit-0x800000 \
	--change-section-address .eeprom-0x810000


coff: $(TARGET).elf
	@echo
	@echo $(MSG_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
```

```
	@echo
	@echo $(MSG_EXTENDED_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof




# Program the device.
program: $(TARGET).hex $(TARGET).eep
	$(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH)
$(AVRDUDE_WRITE_EEPROM)




# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
	@echo
	@echo $(MSG_FLASH) $@
	$(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
	@echo
	@echo $(MSG_EEPROM) $@
	-$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
	--change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.lss: %.elf
	@echo
	@echo $(MSG_EXTENDED_LISTING) $@
	$(OBJDUMP) -h -S $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
	@echo
	@echo $(MSG_SYMBOL_TABLE) $@
	avr-nm -n $< > $@



# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
	@echo
	@echo $(MSG_LINKING) $@
	$(CC) $(ALL_CFLAGS) $(OBJ) --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
%.o : %.c
	@echo
	@echo $(MSG_COMPILING) $<
	$(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
	$(CC) -S $(ALL_CFLAGS) $< -o $@


# Assemble: create object files from assembler source files.
%.o : %.S
	@echo
	@echo $(MSG_ASSEMBLING) $<
	$(CC) -c $(ALL_ASFLAGS) $< -o $@





# Target: clean project.
clean: begin clean_list finished end

clean_list :
	@echo
	@echo $(MSG_CLEANING)
	$(REMOVE) $(TARGET).hex
	$(REMOVE) $(TARGET).eep
	$(REMOVE) $(TARGET).obj
	$(REMOVE) $(TARGET).cof
	$(REMOVE) $(TARGET).elf
	$(REMOVE) $(TARGET).map
	$(REMOVE) $(TARGET).obj
	$(REMOVE) $(TARGET).a90
	$(REMOVE) $(TARGET).sym
	$(REMOVE) $(TARGET).lnk
	$(REMOVE) $(TARGET).lss
	$(REMOVE) $(OBJ)
	$(REMOVE) $(LST)
	$(REMOVE) $(SRC:.c=.s)
	$(REMOVE) $(SRC:.c=.d)


# Automatically generate C source code dependencies.
# (Code originally taken from the GNU make user manual and modified
# (See README.txt Credits).)
#
# Note that this will work with sh (bash) and sed that is shipped with WinAVR
# (see the SHELL variable defined above).
# This may not work with other shells or other seds.
#
%.d: %.c
	set -e; $(CC) -MM $(ALL_CFLAGS) $< \
	| sed 's,\(.*\)\.o[ :]*,\1.o \1.d : ,g' > $@; \
	[ -s $@ ] || rm -f $@


# Remove the '-' if you want to see the dependency files generated.
-include $(SRC:.c=.d)



# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion coff extcoff \
	clean clean_list program
```

A/C code that I used from Bdmicro. (has some other good helpful codes)

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

#ifndef __adc_h__
#define __adc_h__

void	adc_init(void);

void	adc_chsel(uint8_t channel);

void	adc_wait(void);

void	adc_start(void);

uint16_t adc_read(void);
```

```
uint16_t adc_readn(uint8_t channel, uint8_t n);

#endif
```

■■

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

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

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


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

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


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

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


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


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


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

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

  adc_start();

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

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

This is my avoidance behavior code.

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

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

#include <avr/io.h>
#include <stdio.h>
#include "adc.h"
#include "motor.h"
#include "lcd.h"
#include "delay.h"

#ifndef __avoid_h__
#define __avoid_h__

void avoid_obst(void);

#endif
```

```
/*
```

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

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

#include <avr/io.h>
#include <stdio.h>
#include "adc.h"
#include "motor.h"
#include "delay.h"
#include "lcd.h"


void avoid_obst(void){
        uint8_t threshold = 225;
        uint16_t IR_right;
        uint16_t IR_left;

        //Test for IR-right.  If IR_Right >=threshold, avoid obstacle
        IR_right=adc_readn(0, 10);
        //tst for IR_left
        IR_left=adc_readn(1,10);

        if((IR_right>=threshold) && (IR_left>=threshold)){
                lcd_send_command(0x01);
                lcd_send_str("ahead.");
                neutral();
                wait(350);
                fullRight();
                reverse();
                wait(1500);
                neutral();
                fullLeft();
                wait(350);
                halfForward();
                wait(250);
                straight();
                        lcd_send_command(0x01);
        }

        else if (IR_right>=threshold){
        lcd_send_command(0x01);
                lcd_send_str("Obstacle right.");
                neutral();
```

```
		wait(250);
		fullRight();
		reverse();
		wait(1000);
		neutral();
		fullLeft();
		wait(250);
		halfForward();
		wait(250);
		straight();
		lcd_send_command(0x01);
	}




	else if (IR_left>=threshold){
		 lcd_send_command(0x01);
		lcd_send_str("Obstacle left.");
		neutral();
		wait(250);
		fullLeft();
		reverse();
		wait(1000);
		neutral();
		fullRight();
		wait(250);
		halfForward();
		wait(250);
		straight();
		lcd_send_command(0x01);
	}

	else{
		straight();
		halfForward();
		lcd_send_command(0x01);
		lcd_send_str("Dandy");
	}


}
```

I used this for my bump switches

```
#include <avr/io.h>
#include <stdio.h>
#include "adc.h"
#include "motor.h"
#include "lcd.h"
#include "delay.h"

#ifndef __bumped_h__
#define __bumped_h__

void hit(void);

#endif
```

```
#include <stdio.h>
#include <avr/io.h>
#include "adc.h"
#include "motor.h"
#include "delay.h"
#include "lcd.h"
#include "bumped.h"

void hit(void){
	uint16_t Bump;

	//Test for Front Bump
	Bump=adc_readn(2, 1);


	lcd_send_command(0x01);
	if(Bump > 310 && Bump < 330){
		lcd_send_str("Hit in Front!");
		slowReverse();
		fullRight();
		wait(1000);
		}
	else if(Bump > 505 && Bump < 525){
		lcd_send_str("Hit in Back!");
		slowForward();
		wait(500);
		}
	else{
		}
	wait(100);
}
```

This is the delay code so I could use wait times.

```
/*	EEL 5666C - Intelligent Machine Design Laboratory
**	Department of Electrical and COmputer Engineering
**	University of Florida - Spring 20004
**
**	Professor:	Dr. A. A. Arroyo
**	TAs:		Louis Brandy
**			Max Koessick
**			William Dubel
**
**	Name:		Steven Pickles
**	Class:		5EG
**	UF-ID:		4129-2230
**
**	Project:	Forklift Robot
**	Robot Name:	<none yet>
**	Function:	Follows a line and reads "road signs" in the form of barcodes on
**			the warehouse floor.  The robot will either pick up or drop
off
**			a pallette from a specified place in the warehouse.
**
**	Filename:	delay.h
**	Purpose:	Definition file for the "wait" routine.  When the wait function
**			is invoked, a number of milliseconds is passed and
execution
**			of other functions is prevented until the delay is finished.
**
**	Source Code References:
**		01)		The timer was inspired by similar code found at
www.bdmicro.com
*/



/*
**	Delay.h File Identifier
*/
#ifndef __delay_h__
#define __delay_h__



/*
**	Compiler Include Directives
*/
#include <avr/pgmspace.h>
#include <avr/signal.h>
```

```
/*
**      Global/External Variables
*/

//      Number of milliseconds that have gone by since being reset
extern volatile uint16_t msCount;



/*
**      Interrupt Vector
**      Type:           Output Compare 0 Match
**      Purpose:        Since output compare 0 has matched, increment msCount.
*/
SIGNAL(SIG_OUTPUT_COMPARE0);



/*
**      Function:       wait
**      Purpose:        Delay execution of other functions for a specified number of
**                              milliseconds.  Note that interrupts are still ENABLED
during
**                              the execution of this function.
**      Parameters:     uint16_t msDelay - length of delay in milliseconds
**      Variables:      uint16_t msCount - current millisecond count
**      Registers:      TCNT0 - Timer 0 Counter Register
**      Returns:        <none>
*/
void wait(uint16_t msDelay);


/*
**      Function:       initializeWaitTimer
**      Purpose:        Initializes the timer 0 control and mask registers so that
**                              an interrupt occurs roughly every 1 millisecond**.  This is
**                              essential because the wait function depends upon timer 0
**                              initialization to work properly.
**                              ** currently 0.9765625 milliseconds
**      Parameters:     <none>
**      Variables:      <none>
**      Registers:      TIFR -          Timer Interrupt Flag Register
**                              TIMSK - Timer Interrupt Mask Register
**                              ASSR -          Asynchronous Source Register
**                              OCR0 -          Output 0 Compare Register
**                              TCCR0 -         Timer 0 Control Register
*/
void initializeWaitTimer(void);

#endif
```

```
/*      EEL 5666C - Intelligent Machine Design Laboratory
**      Department of Electrical and COmputer Engineering
**      University of Florida - Spring 20004
**
**      Professor:      Dr. A. A. Arroyo
**      TAs:            Louis Brandy
**                              Max Koessick
**                              William Dubel
**
**      Name:           Steven Pickles
**      Class:          5EG
**      UF-ID:          4129-2230
**
**      Project:        Forklift Robot
**      Robot Name:     <none yet>
**      Function:       Follows a line and reads "road signs" in the form of barcodes on
**                              the warehouse floor.  The robot will either pick up or drop
off
**                              a pallette from a specified place in the warehouse.
**
**      Filename:       delay.c
**      Purpose:        Implementation file for the "wait" routine.  When the wait
**                              function is invoked, a number of milliseconds is passed and
**                              execution of other functions is prevented until the delay is
**                              finished.
**
**      Source Code References:
**              01)             The timer was inspired by similar code found at
www.bdmicro.com
*/



/*
**      Compiler Include Directives
*/
#include "delay.h"
```

```
/*
**	Interrupt Vector
**	Type:		Output Compare 0 Match
**	Purpose:	Since output compare 0 has matched, increment msCount.
*/
SIGNAL(SIG_OUTPUT_COMPARE0)
{
	// Increment the millisecond counter because an interrupt just occured
	msCount++;
}



/*
**	Function:	wait
**	Purpose:	Delay execution of other functions for a specified number of
**				milliseconds.  Note that interrupts are still ENABLED
during
**				the execution of this function.
**	Parameters:	uint16_t msDelay - length of delay in milliseconds
**	Variables:	uint16_t msCount - current millisecond count
**	Registers:	TCNT0 - Timer 0 Counter Register
**	Returns:	<none>
*/
void wait(uint16_t msDelay)
{
  // Set timer counter 0 to 0 (reset TCNT0)
  TCNT0  = 0;

  // Set millisecond counter to 0 (reset msCount)
  msCount = 0;

  // Wait until millisecond counter = desired length of delay
  while (msCount != msDelay)
    ;
}


/*
**	Function:	initializeWaitTimer
**	Purpose:	Initializes the timer 0 control and mask registers so that
**				an interrupt occurs roughly every 1 millisecond**.  This is
**				essential because the wait function depends upon timer 0
**				initialization to work properly.
**				** currently 0.9765625 milliseconds
```

```
**	Parameters:	<none>
**	Variables:	<none>
**	Registers:	TIFR -		Timer Interrupt Flag Register
**				TIMSK - Timer Interrupt Mask Register
**				ASSR -		Asynchronous Source Register
**				OCR0 -		Output 0 Compare Register
**				TCCR0 -		Timer 0 Control Register
*/
void initializeWaitTimer(void)
{

	// Set timer interrupt flag register
	TIFR  |= BV(OCIE0)|BV(TOIE0);

	// Enable the output compare 0 interrupt
	TIMSK |= BV(OCIE0);

	// Disable timer 0 overflow
	TIMSK &= ~BV(TOIE0);

	// Use the asynchronous clock source
	ASSR |= BV(AS0);

	// Set time counter 0 to 0 (reset TCNT0)
	TCNT0 = 0;

	// Set output compare register 0 to match 0.9765625ms
	OCR0 = 32;

	// Set the timer 0 control register for CTC and no prescale
	TCCR0 = BV(WGM01) | BV(CS00);

	// Wait until the asynchronous control status register is ready
	while (ASSR & 0x07)
	;

	// Finally, set timer interrupt flag register
	//TIFR  |= BV(OCIE0)|BV(TOIE0);
}
```

This is the LCD code.

```
/*
* lcd.h
*
```

```
* Author: Max Billingsley
* Adapted by: David Wynacht
*/
#define LCD_PORT PORTA
#define LCD_DDR DDRA
#define ENABLE 0x40
/* function prototypes */
void lcd_set_ddr(void);
void lcd_init(void);
void lcd_delay(void);
void lcd_send_str(char *s);
void lcd_send_byte(uint8_t val);
void lcd_send_command(uint8_t val);
int main(void);
```

```
/*
* lcd.c
*
* Author: Max Billingsley
* Adapted by: David Wynacht
*/
/*
* LCD_PORT0 - DB4
* LCD_PORT1 - DB5
* LCD_PORT2 - DB6
* LCD_PORT3 - DB7
* LCD_PORT4 - RS
* LCD_PORT5 - r/w
* LCD_PORT6 - EN
*
* RS: Register Select:
*
* 0 - Command Register
* 1 - Data Register
*
*/
#include <inttypes.h>
#include <avr/io.h>
#include "lcd.h"
/* entry point */
void lcd_init(void)
{
lcd_send_command(0x43);
lcd_send_command(0x43);
lcd_send_command(0x43);
lcd_send_command(0x42);
lcd_send_command(0x42);
lcd_send_command(0x48);
lcd_send_command(0x40);
lcd_send_command(0x0f);
lcd_send_command(0x00);

lcd_send_command(0x01);
}
void lcd_set_ddr(void)
{
LCD_DDR = 0xff;
}
void lcd_delay(void)
{
uint16_t time1;
for (time1 = 0; time1 < 2000; time1++);
}
void lcd_send_str(char *s)
{
while (*s) lcd_send_byte(*s++);
}
void lcd_send_byte(uint8_t val)
{
uint8_t temp = val;
val >>= 4;
val |= 0x10; /* set data mode */
LCD_PORT = val;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
temp &= 0x0f;
temp |= 0x10; /* set data mode */
LCD_PORT = temp;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
lcd_delay();
}
void lcd_send_command(uint8_t val)

{
uint8_t temp = val;
val >>= 4;
LCD_PORT = val;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
```

```
temp &= 0x0f;
LCD_PORT = temp;
lcd_delay();
LCD_PORT |= ENABLE;
LCD_PORT &= ~ENABLE;
lcd_delay();
}
```

This is the code I used for following a line.

```
/*      EEL 5666C - Intelligent Machine Design Laboratory
**      Department of Electrical and COmputer Engineering
**      University of Florida - Spring 20004
**
**      Professor:      Dr. A. A. Arroyo
**      TAs:            Louis Brandy
**                              Max Koessick
**                              William Dubel
**
**      Name:           Steven Pickles
**      Class:          5EG
**      UF-ID:          4129-2230
**
**      Project:        Forklift Robot
**      Robot Name:  Kirby
**      Function:       Follows a line and reads "road signs" in the form of barcodes on
**                              the warehouse floor.  The robot will either pick up or drop
off
**                              a pallette from a specified place in the warehouse.
**
**      Filename:       tracker.h
**      Purpose:        Interface file for the line tracker and the external
**                              interrupts that it generates.
**
**      Source Code References:
**              01)
*/



/*
**      Tracker.h File Identifier
*/
#ifndef __tracker_h__
#define __tracker_h__
```

```
/*
**      Compiler Include Directives
*/
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <stdio.h>
#include "lcd.h"
#include "motor.h"



/*
**      Line Tracker Constants
*/

//      Value Constants
#define LT0000 0
#define LT1000 128
#define LT0100 64
#define LT1100 192
#define LT0010 32
#define LT1010 160
#define LT0110 96
#define LT1110 224
#define LT0001 16
#define LT1001 144
#define LT0101 80
#define LT1101 208
#define LT0011 48
#define LT1011 176
#define LT0111 112
#define LT1111 240


/*
**      Motor Speed Constants
*/

//      Speed Constants
#define SLOW_FORWARD 1650;
#define MED_FORWARD 1725;
#define MED_FAST_FORWARD 1765;
#define FAST_FORWARD 1800;
```

```
#define FORWARD10 1810;
#define FORWARD9 1790;
#define FORWARD8 1770;
#define FORWARD7 1750;
#define FORWARD6 1730
#define FORWARD5 1710
#define FORWARD4 1690
#define FORWARD3 1670
#define FORWARD2 1650
#define FORWARD1 1630

#define REVERSE1 1370
#define REVERSE2 1350
#define REVERSE3 1330
#define REVERSE4 1310
#define REVERSE5 1290
#define REVERSE6 1270
#define REVERSE7 1250

#define STOP 1510;

#define SLOW_REVERSE 1350;
#define MED_REVERSE 1275;
#define MED_FAST_REVERSE 1235;
#define FAST_REVERSE 1200;



/*
**      Global/External Variables
*/

#define LINETRACK (PINE & 0xf0)

extern volatile uint16_t previousDirection;
extern volatile uint8_t lapCount;





/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
```

```
*/
SIGNAL(SIG_INTERRUPT4);



/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT5);


/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT6);



/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT7);



/*
**      Function:       initializeLineTracker
**      Purpose:        Sets the correct port E I/O pins to accept the external
**                              interrupts as driven by the line tracking device.
**      Parameters:     <none>
**      Variables:      <none>
**      Registers:
*/
void initializeLineTracker(void);


/*
**      Function:       trackLine
**      Purpose:        Tracks the line by adjusting the motor values.
**      Parameters:     <none>
```

```
**	Variables:	<none>
**	Registers:
*/
void trackLine(void);



#endif
```

```
/*	EEL 5666C - Intelligent Machine Design Laboratory
**	Department of Electrical and COmputer Engineering
**	University of Florida - Spring 20004
**
**	Professor:	Dr. A. A. Arroyo
**	TAs:		Louis Brandy
**				Max Koessick
**				William Dubel
**
**	Name:		Steven Pickles
**	Class:		5EG
**	UF-ID:		4129-2230
**
**	Project:	Forklift Robot
**	Robot Name:	<none yet>
**	Function:	Follows a line and reads "road signs" in the form of barcodes on
**				the warehouse floor.  The robot will either pick up or drop
off
**				a pallette from a specified place in the warehouse.
**
**	Filename:	delay.c
**	Purpose:	Implementation file for the line tracker and the external
**				interrupts that it generates.
**
**	Source Code References:
**		01)
*/



/*
**	Compiler Include Directives
*/
#include "tracker.h"
```

```
/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT4)
{
        trackLine();
}



/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT5)
{
        trackLine();
}


/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT6)
{
        trackLine();
}



/*
**      Interrupt Vector
**      Type:           External Interrupt Vector
**      Purpose:        ISR for when an external source generates an interrupt.
*/
SIGNAL(SIG_INTERRUPT7)
{
        trackLine();
}
```

```
/*
**      Function:       initializeLineTracker
**      Purpose:        Sets the correct port E I/O pins to accept the external
**                              interrupts as driven by the line tracking device.
**      Parameters:     <none>
**      Variables:      <none>
**      Registers:
*/
void initializeLineTracker(void)
{


        //      Set data direction for port E as inputs
        DDRE = 0x0f;

        //      No pull up resistor, set to tri-state input (high impedance)
        PORTE = 0x00;

        //      Setup the external interrup pins
        EICRA = 0x00;
        EICRB = 0x55;
        EIMSK = 0xf0;
        EIFR = 0xf0;
}




/*
**      Function:       trackLine
**      Purpose:        Tracks the line by adjusting the motor values.
**      Parameters:     <none>
**      Variables:      <none>
**      Registers:
*/
void trackLine(void)
{

        if(LINETRACK != previousDirection)
        {
                switch(LINETRACK)
                {
                        case LT0000:
                                        //neutral();
                                        fullForward();
```

```
          break;
case LT0001:
                    slightLeft();
                    lapCount++;
          break;
case LT0010:
                    straight();
          break;
case LT0011:
                    mediumLeft();
          break;
case LT0100:
                    straight();
          break;
case LT0101:
                    mediumLeft();
          break;
case LT0110:
                    straight();
          break;
case LT0111:
                    fullLeft();
          break;
case LT1000:
                    slightRight();
                    halfForward();
          break;
case LT1001:
                    straight();
          break;
case LT1010:
                    slightRight();
          break;
case LT1011:
                    slightLeft();
          break;
case LT1100:
                    mediumRight();
          break;
case LT1101:
                    slightRight();
          break;
case LT1110:
                    fullRight();
          break;
case LT1111:
```

```
                                if(previousDirection == LT0111 || previousDirection ==
LT1110)
                                {
                                        switch(previousDirection)
                                        {
                                                case LT0111:
                                                        straight();
                                                        break;
                                                case LT1110:
                                                        straight();
                                                        break;
                                        }
                                }



                                break;
                }
        }

        if((LINETRACK == LT1111) && (previousDirection != LT1111))
                previousDirection = LINETRACK;

}
```

This is the code used for counting the laps.

```
#ifndef __lapcounter_h__
#define __lapcounter_h__

#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <stdio.h>
#include "lcd.h"
#include "motor.h"
#include "tracker.h"
#include "delay.h"

extern volatile uint8_t lapCount;

void lapcounting(void);
#endif
```

```
#include "lapcounter.h"
```

```
void lapcounting(void){
lcd_send_command(0x01);
        if(lapCount == 1){
                lcd_send_str("Lap One");
                wait(2000);
                lcd_send_command(0x01);
                }
        else if(lapCount == 2){
                lcd_send_str("Lap Two");
                wait(2000);
                lcd_send_command(0x01);
                }
        else if(lapCount == 3){
                lcd_send_str("RACE OVER");
                neutral();
                wait(50000);
                lcd_send_command(0x01);
                }

}
```

This is the code for the motor using a PWM.

```
#ifndef __motor_h__
#define __motor_h__

#include <avr/pgmspace.h>
#include <avr/signal.h>




//      Initialize timer 1 for motor PWM function
void initializeMotorTimer(void);

void updateMotorSpeed(int motorNumber, int newMotorSpeed);
int main(void);

void halfForward(void);
void fullForward(void);
void neutral(void);
void reverse(void);

void fullLeft(void);
void fullRight(void);
void straight(void);
void mediumRight(void);
void slightRight(void);
void mediumLeft(void);
void slightLeft(void);
void slowReverse(void);
void slowForward(void);
#endif
```

```
#include "motor.h"
#include "delay.h"
#include "lcd.h"


//      Initialize the motor timer
void initializeMotorTimer(void)
{
        //      Timer/Counter Control Register A - TCCR1A
        //              Set to non-inverting for channels A and B,
        //              Phase and Frequency Correct PWM Mode
        TCCR1A = 0xA0;

        //      Timer/Counter Control Register B - TCCR1B
        //              Set to Phase and Frequency Correct PWM Mode
        //              and a prescaler of clock/8
        TCCR1B = 0x12;


        //      Input Capture Compare Register 1 H/L
        //              Set to get the correct frequency
        ICR1 = 0x4332;
}



//void updateMotorSpeed(int motorNumber, int newMotorSpeed)
//{

        //int k = 10;
        //int time = 0;

        //servo values max_left=1300
        //servo values max_right=1880
        //servo values straight=1600
        //lcd_send_str("Motor 1520");
        //OCR1B = 1520;
```

```
		//wait(8000);
		//OCR1A = 1880;
		//wait(2000);
		//OCR1A = 1300;
		//wait(2000);
		//OCR1A = 1600;
		//OCR1B = 1800;
//}

//Set motor speeds
void slowForward(void){
		OCR1B=1600;
		}
void halfForward(void){
		OCR1B = 1700;

}

void fullForward(void){
		OCR1B=1800;

		}

void neutral(void){
		OCR1B=1495;


}

void reverse(void){
		OCR1B=1100;


}
void slowReverse(void){
   OCR1B=1350;
}

//Set servo positions

void fullLeft(void){
		OCR1A=1300;

}
void mediumLeft(void){
		OCR1A=1420;
```

```
}
void slightLeft(void){
        OCR1A=1550;
}
void fullRight(void){
        OCR1A=1930;

}
void mediumRight(void){
        OCR1A=1810;
}
void slightRight(void){
         OCR1A=1690;
 }
void straight(void){
        OCR1A=1630;


}
```

And this is the MAIN program.

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


#include "delay.h"
#include "bumped.h"
#include "motor.h"
#include "lcd.h"
#include "adc.h"
#include "avoid.h"
#include "tracker.h"
#include "lapcounter.h"

#define CPU_FREQ 16000000L  /* set to clock frequency in Hz */

#define BAUD_RR ((CPU_FREQ/(16L*9600L) - 1))



//      Millisecond Counter
volatile uint16_t msCount;
volatile uint16_t previousDirection;
volatile uint8_t lapCount;


int main(void){

  initializeMotorTimer();
  lcd_set_ddr();
  lcd_init();
  initializeWaitTimer();
  DDRB = 0x60;  /* enable PORTB 1 as an output */
  //initializeLineTracker();

        // Some variables
        //uint16_t avg_distance;
        //uint16_t threshold = 50;
        //uint16_t readValue2;
        //uint8_t channel = 0;
        adc_init();

        //readValue2= adc_readn(channel, 10);
        //lcd_send_str("IR reading: " + readValue2);

        //reverse();


  /* enable interrupts */
  sei();
        slowForward();
        wait(2500);


 while(1){
        avoid_obst();
        hit();
        lapcounting();
        }

        return 0;
}
```