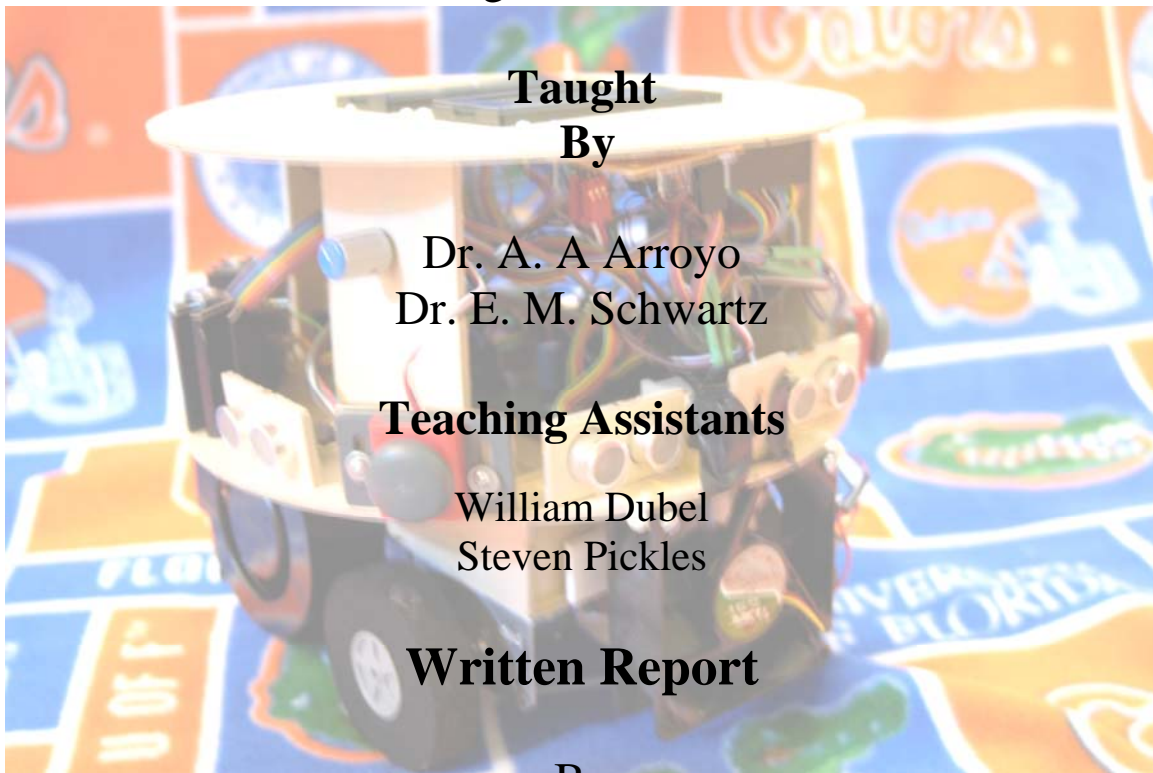


F . I . R . E

(Fire Intelligent Robot Extinguisher)

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory
August 1st, 2005



**Taught
By**

Dr. A. A Arroyo
Dr. E. M. Schwartz

Teaching Assistants

William Dubel
Steven Pickles

Written Report

By

Natthapol Prakongpan

Table of Contents

Abstract.....1

Introduction.....1

Integrated System.....2

Platform and Construction.....3

Actuators.....4

Sensors.....5

Behaviors.....7

Conclusion.....8

References.....9

Abstract

With the danger to human life of fighting and extinguishing a fire could be tremendous, The F.I.R.E (Fire Intelligent Robot Extinguisher) is created to aid firefighters fight fire. This report is a great start on building a live size F.I.R.E bot as it contains information of how this robot is designed and implemented.

Introduction

This report is a detailed description of the objectives, specifications, requirements, and system integration of the F.I.R.E. bot. The F.I.R.E bot is a robot that looks for a fire and then tries to extinguish it. The objective of the robot is to fight a fire successfully every time it sees a fire. The main sensor uses in this robot is a pyroelectric sensor which is used to detect a fire.

Integrated System

The F.I.R.E bot system consists of the following major components:

- Atmel ATmega 128 Microprocessor from Bdmicro.
- Hamamatsu UVTron Flame Detector
- Devantech Electronic Compass
- Four Ultrasonic Range Finders
- Two LCD Displays
- Three Bump Switches
- Two Motors
- Two Fans
- Battery Pack

The two most important sensors for this robot is the Hamamatsu UVTron Flame detector which is used to detect the fire and the electronic compass which is used to maintain heading to the fire after it has been detected.

Atmel ATmega 128 microprocessor is obtained from bdmicro.com at a cost of around \$100. It is the main processor of this robot and interfaces with all sensors and actuators.

Two battery packs are used in this robot to achieve the voltage requirement of the National Semiconductor, LM18201, H-Bridge motor drivers of 12V. A total of twelve rechargeable NiMH batteries are used on board this robot.

Two LCD displays are used to display the information about the system and the menu for which the robot modes (Detect Fire, Obstacles Avoid) can be chosen.

Here is a quick note on the LCDs' backlight power consumption: I have found that a 4x20 LCD screen with LED backlight draws about 300 – 500 mA of current. Although the voltage regulator, 7805, that I am using on this robot can provide a current up to 1.5A, it can only be done with adequate heat dissipation. Therefore a large heat sink has to be added to my regulator that powers two LCD displays otherwise the displays will keep resetting themselves as the supply voltage drops because of the regulator overheating.

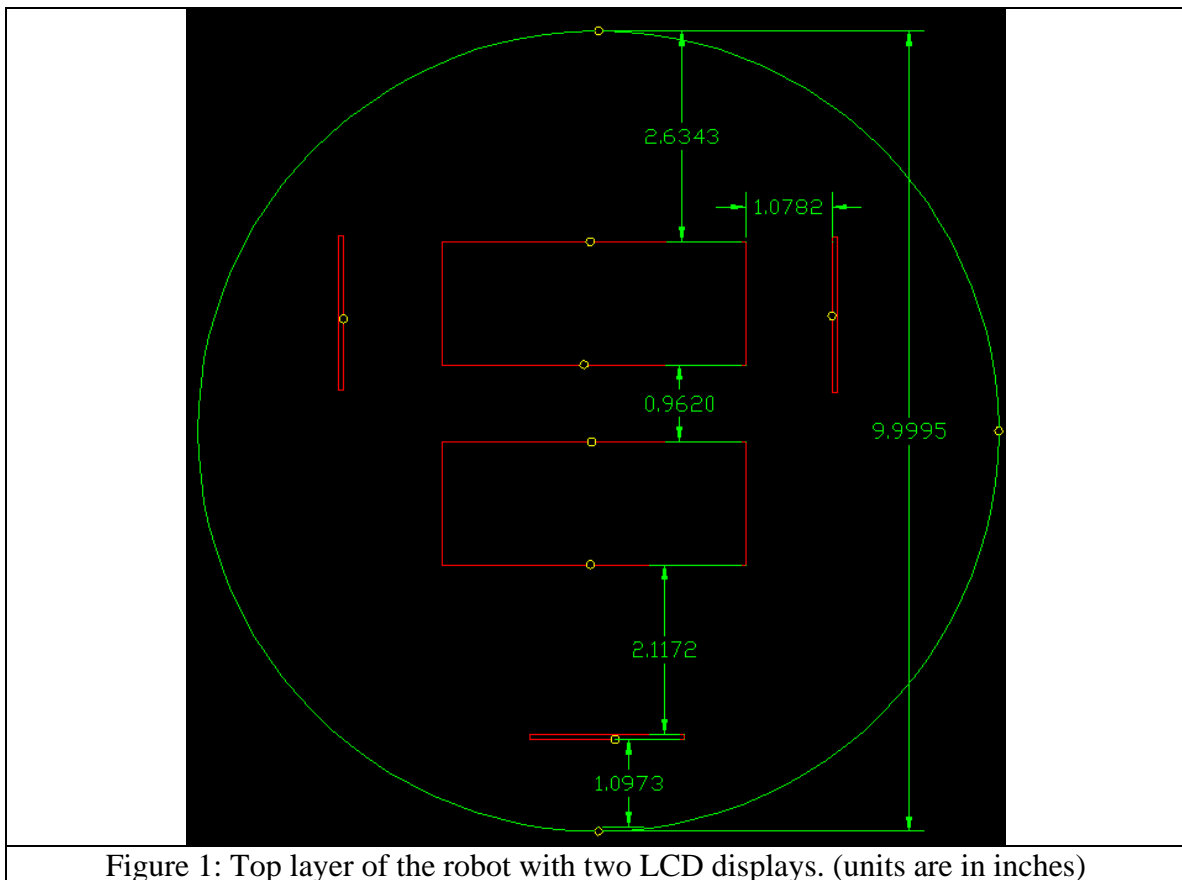
Four ultrasonic range finders are used to measure the distance between the robot and the flame. They are also used for the obstacles avoid mode of the robot.

Three bump switches are for the menu selection and obstacles avoid.

Platform and Construction

The platform of this robot was designed so that it can carry a large enough fan to adequately extinguish the fire. The body of the robot is made out of wood since it can easily be made by our in-house T-Tech machine. The secondary goal for the platform was the ability to access its' hardware and electronics since this is a prototype robot and changes were made to the system very often.

The final outcome of the platform is the two LCD displays on top of the robot with its' electronics and processor in the middle. Lastly, the motors and wheels go on the bottom layer of the platform.



Actuators

The actuation of the robot consists of two gearhead motors from Jameco Electronics with part number 155862. The specification of the motors is shown below:



Rated Voltage (VDC)	12
Operating Range (VDC)	4.5 – 12
Current @ Max. Efficiency (mA)	300
Speed @ Max. Efficiency (RPM)	70
Torque @ Max. Efficiency (g-cm)	1000
Gear Ratio	60:01:00
Gear Case Size (Diameter x Length) (inch)	1.3 x 0.9
Motor Size (Diameter x Length) (inch)	1.4 x 0.9
Shaft Size (Diameter x Length) (inch)	.23 x .90

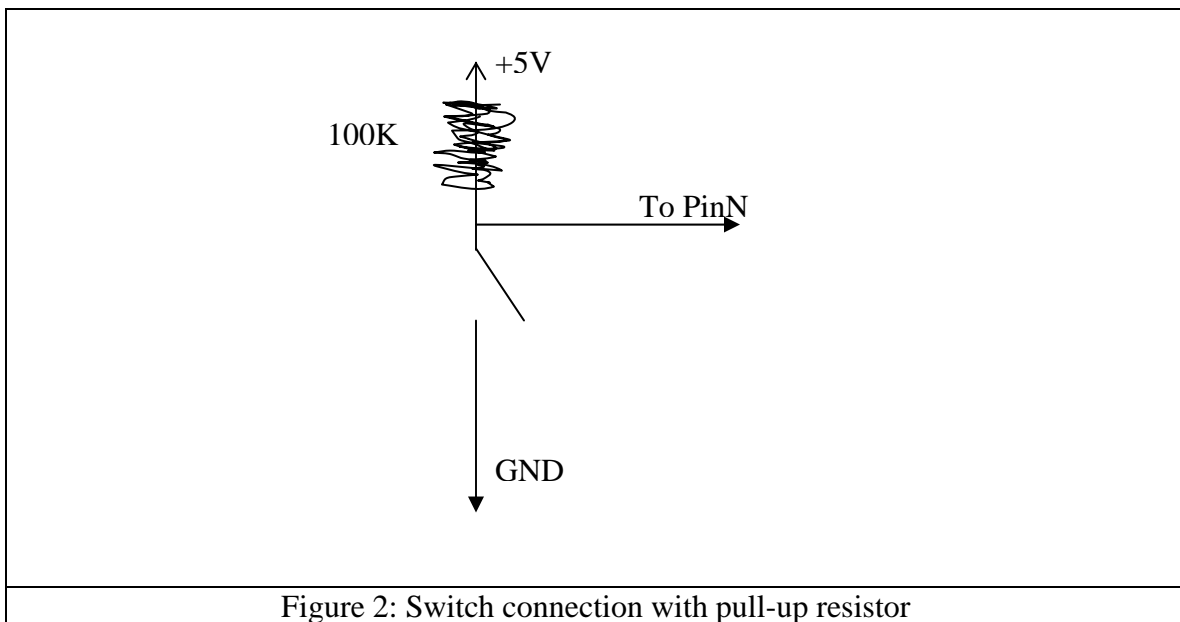
The motors are controlled by two LM 18201 H-Bridge motor drivers from National Semiconductor obtained as free samples. The PWM signals are generated from the microprocess with a rate of 1KHz to control the motors.

The wheels, Lite Flite Wheels, are also from Jameco Electronics with the part number Two wheels are attached to the motors' shaft directly by pushing it in really hard.

Sensors

There are four types of sensors on this robot, and I will explain these sensors from the most basic ones starting with the bump switches.

There are many ways to connect the bump switches to the microprocessor board. For example, we can use resistors network (voltage divider) to connect multiple bump switches to the microprocessor using only one A-to-D pin. However, since I will only be using three bump switches and there are many ports open for me to use. I connected my bump switches directly to digital pins on the board using pull-up resistor. The schematic for the bump switches is shown below in figure 2.



The next sensor is the ultrasonic range finder or sonar. The sonar sensor sends out a sound pulse and wait for the echo to come back and hit its' receiver. The time between the initial pulse and the echo can be translated into a distance because a sound wave travels at a constant speed. For the sonar, I used polling technique to measure the time of the pulses.



One of the two important sensors for this robot is the flame detector. The flame detector detects the UV ray radiate from the light source. The UV ray is usually high in concentration in the fire, and therefore, can be distinguish from other sources of the UV ray. The sensor has a detection range of about five feet. When it detects a fire, the output pin of the sensor goes high for 10ms. I then use this pin to generate an interrupt on the processor board.

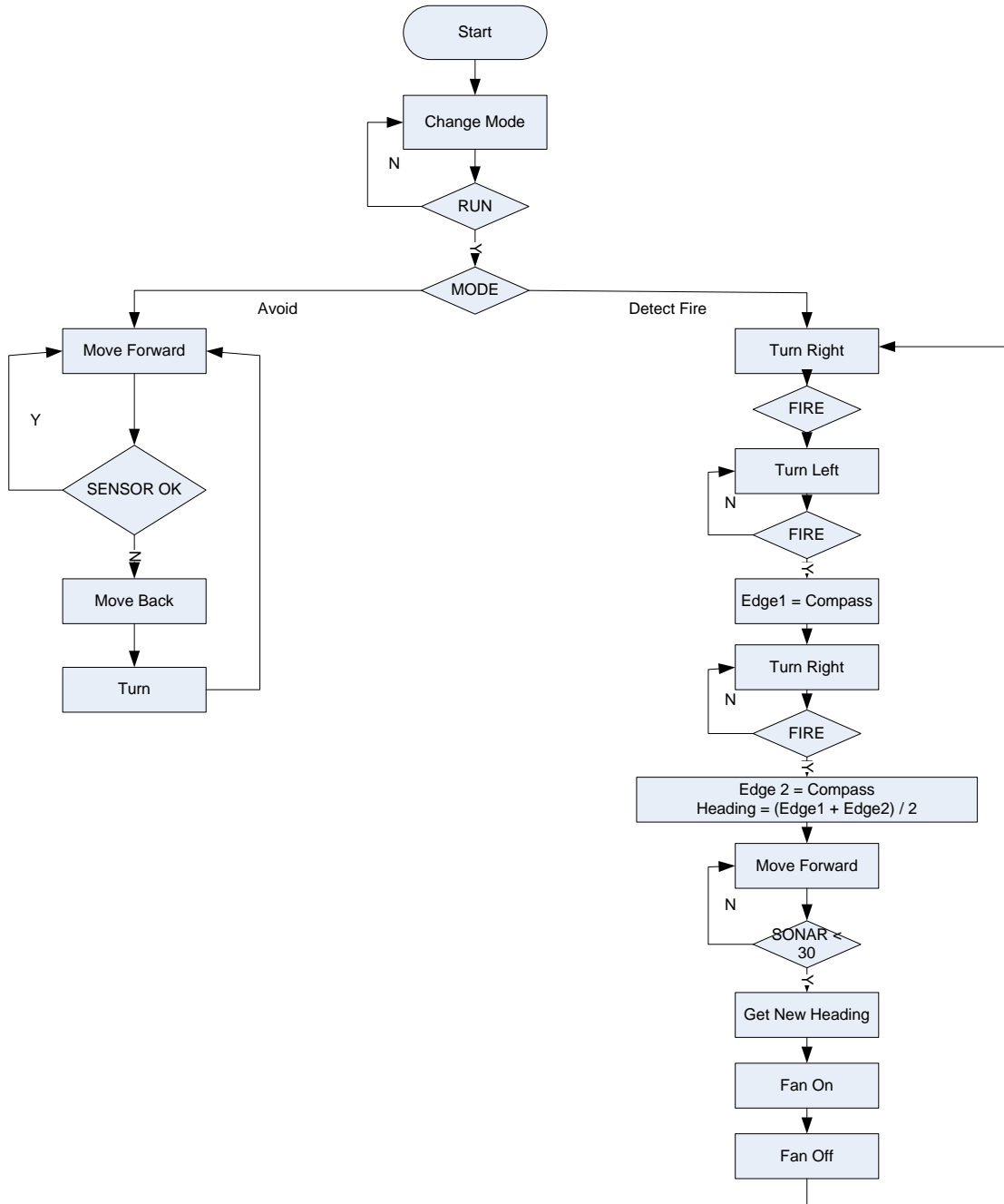


The last sensor for this robot is the electronic compass. The communication between compass and the microprocessor board is done via the I2C standard. It was really easy to use. I can read the current heading directly from the compass registers. If there is no I2C interface available on for your microcontroller, you can also use the PWM output from the module. The compass came calibrated, so I don't have to re calibrated the unit.

All of the sensors except for the bump switches were obtained from Acroname.com.

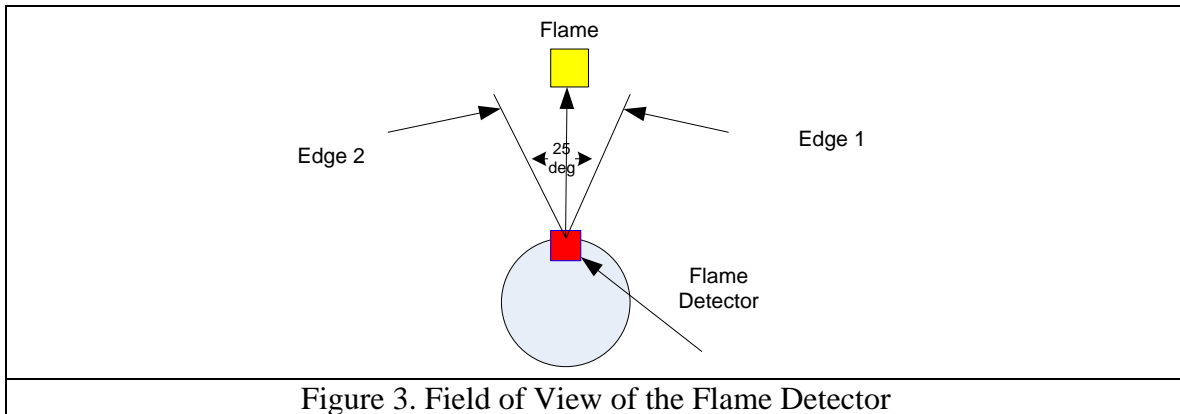
Behaviors

This robot can be used in two modes. One of the modes is to have to walk around without hitting any objects. Another mode is of course a fire fighting mode. The flow char below shows how the robot thinks:



The system starts looking for a fire after it switches on by keep turning around in circle. Once the flame detector detects a fire, the level of the output pin of the detector

goes high for 10ms. The output pin of the detector, which connects to the External Interrupt pin on the bdmicro board, then generates an interrupt in the processor. When the interrupt is fired, a flag is set for the software to see that the flame has been detected. Because the field of view of the sensor is wide, the robot has to detect the edges of the field of view in order to calculate the exact heading (the middle point between two edges) from the compass. Figure 3 shows the field of view of the flame detector and how the robot detects the position of the flame relative to the earth magnetic field.



The ultrasonic range finders are then used to measure how far the robot is relative to the flame. The robot is then moved toward the flame until the distance between the robot and the flame is roughly five inches. The robot is now stop moving and turn on the fans. After the fan is on for about ten seconds, the robot starts turning around just in case that it was not position itself right in front of the flame. The robot then returns to fire detection routine again.

Conclusion

In conclusion, this robot is a good learning platform for many navigating robot because of the use of the compass. The heading can be set my many kinds of sensor such as the Flame detector. Users can also program the heading into the robot for navigation purposes. The motor control code and circuit give users a quick start guide actuator problems. So far, this robot has been a great learning platform for me.

References

- C Code for robot, <http://www.bdmicro.com/>
- Datasheet for sensors, <http://www.acroname.com/>

```
# Hey Emacs, this is a -*- makefile -*-
#-----
# WinAVR Makefile Template written by Eric B. Weddington, Jörg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O'Flynn
# Reiner Patommel
# Markus Pfaff
# Sander Pool
# Frederik Rouleau
#
#-----
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                 Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                 bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#-----

# MCU name
MCU = atmega128

# Processor frequency.
#   This will define a symbol, F_CPU, in all source code files equal to
#   the processor frequency. You can then use this symbol in your source code
#   to calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#   automatically to create a 32-bit value in your source code.
F_CPU = 16000000

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

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

```

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

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

# Debugging format.
#   Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#   AVR Studio 4.10 requires dwarf-2.
#   AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2

# List any extra directories to look for include files here.
#   Each directory must be seperated by a space.
#   Use forward slashes for directory separators.
#   For a directory that has spaces, enclose it in quotes.
EXTRA_INCLUDE_DIRS =

# Compiler flag to set the C Standard level.
#   c89 = "ANSI" C
#   gnu89 = c89 plus GCC extensions
#   c99 = ISO C99 standard (not yet fully implemented)
#   gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99

# Place -D or -U options here
CDEFS = -DF_CPU=$(F_CPU)UL

# Place -I options here
CINCS =

#----- Compiler Options -----
# -g*:      generate debugging information
# -O*:      optimization level
# -f...:    tuning, see GCC manual and avr-libc documentation
# -Wall...: warning level
# -Wa,...:  tell GCC to pass this to the assembler.
# -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS) $(CINCS)

```

```
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums
CFLAGS += -Wall -Wstrict-prototypes
CFLAGS += -Wa, -adhlns=$(<:.c=.lst)
CFLAGS += $(patsubst %, -I%, $(EXTRA_INCLUDES))
CFLAGS += $(CSTANDARD)
```

```
#----- Assembler Options -----
# -Wa,...: tell GCC to pass this to the assembler.
# -ahlns: 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
```

```
#----- Library Options -----
```

```
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl, -u, vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl, -u, vfprintf -lprintf_float
```

```
# If this is left blank, then it will use the Standard printf version.
#PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
PRINTF_LIB = $(PRINTF_LIB_FLOAT)
```

```
# Minimalistic scanf version
SCANF_LIB_MIN = -Wl, -u, vscanf -lscanf_min
```

```
# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl, -u, vscanf -lscanf_float
```

```
# If this is left blank, then it will use the Standard scanf version.
#SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)
```

```
MATH_LIB = -lm
```

```
#----- External Memory Options -----
```

```
# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl, -Tdata=0x801100, --defsym=__heap_end=0x80ffff
```

```
# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl, --defsym=__heap_start=0x801100, --defsym=__heap_end=0x80ffff
```

```
EXTMEMOPTS =
```

```
#----- Linker Options -----
```

```

# -Wl,...:      tell GCC to pass this to linker.
#   -Map:      create map file
#   --cref:    add cross reference to map file
LD_FLAGS += -Wl, -Map=$(TARGET).map, --cref
LD_FLAGS += $(EXTMEMOPTS)
LD_FLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)

#----- Programming Options (avrdude) -----

# 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 = stk500

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

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

# 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_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_NO_VERIFY = -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_VERBOSE = -v -v

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)

#----- Debugging Options -----

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

```

```
# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr,
normally
#   just set to localhost unless doing some sort of crazy debugging when
#   avarice is running on a different computer.
DEBUG_HOST = localhost
```

```
#=====
```

```
# Define programs and commands.
```

```
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
COPY = cp
WINSHELL = cmd
```

```
# 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 = $(SRC:.c=.lst) $(ASRC:.S=.lst)
```

```
# Compiler flags to generate dependency files.
```

```
GENDEPFLAGS = -MD -MP -MF .dep/$(@F).d
```

```

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

# Default target.
all: begin gccversion sizebefore build sizeafter end

build: elf hex eep lss sym

elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym

# 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)

end:
    @echo $(MSG_END)
    @echo

# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf
AVRMEM = avr-mem.sh $(TARGET).elf $(MCU)

sizebefore:
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE);
$(ELFSIZE); \
    $(AVRMEM) 2>/dev/null; echo; fi

sizeafter:
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE)
; \
    $(AVRMEM) 2>/dev/null; echo; fi

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

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

```

```

# Generate avr-gdb config/init file which does the following:
#   define the reset signal, load the target file, connect to target, and
set
#   a breakpoint at main().
gdb-config:
    @$(REMOVE) $(GDBINIT_FILE)
    @echo define reset >> $(GDBINIT_FILE)
    @echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
    @echo end >> $(GDBINIT_FILE)
    @echo file $(TARGET).elf >> $(GDBINIT_FILE)
    @echo target remote $(DEBUG_HOST):$(DEBUG_PORT) >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND), simulavr)
    @echo load >> $(GDBINIT_FILE)
endif
    @echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
    @echo Starting AVaRICE - Press enter when "waiting to connect" message
displays.
    @$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program
--file \
    $(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
    @$(WINSHELL) /c pause
else
    @$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
    $(DEBUG_MFREQ) --port $(DEBUG_PORT)
endif
    @$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)

# 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

# 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) $@
$(NM) -n $< > $@

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

# Create preprocessed source for use in sending a bug report.
%.i : %.c
$(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@

# Target: clean project.
clean: begin clean_list end

clean_list :
@echo
@echo $(MSG_CLEANING)
$(REMOVE) $(TARGET).hex
$(REMOVE) $(TARGET).eep

```

```
$(REMOVE) $(TARGET).cof
$(REMOVE) $(TARGET).elf
$(REMOVE) $(TARGET).map
$(REMOVE) $(TARGET).sym
$(REMOVE) $(TARGET).lss
$(REMOVE) $(OBJ)
$(REMOVE) $(LST)
$(REMOVE) $(SRC:.c=.s)
$(REMOVE) $(SRC:.c=.d)
$(REMOVE) .dep/*
```

Include the dependency files.

```
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)
```

Listing of phony targets.

```
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```

```

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

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
/*
#include "lcd.h"
#include "servos.h"
#include "sonar.h"
#include "adc.h"
#include "i2c.h"
#include "cmps03.h"
*/

#define I2C_START      (_BV(TWINT) | _BV(TWSTA) | _BV(TWEN))
#define I2C_MASTER_TX (_BV(TWINT) | _BV(TWEN))
#define I2C_TIMEOUT   50000 //1000
#define I2C_ACK       1
#define I2C_NACK      0

#define SERV01      OCR1A
#define SERV02      OCR1B
#define SERV03      OCR1C

#define SERVO_MIN   1000
#define SERVO_MIN_D 1500
#define SERVO_MAX   2000

#define FF 0
#define LT 1
#define RT 2

#define SPEED_MIN 0
#define SPEED_MIN_D 600
#define SPEED_MAX 1300

void ms_sleep(uint16_t);
void init_timer(void);

void lcd_command(uint8_t, uint8_t);
int lcd_write(char);
void lcd_line(uint8_t);
void lcd_init(void);
void set_lineenum(uint8_t ln);

void sonar(void);

volatile uint16_t SONAR1;
volatile uint16_t SONAR2;
volatile uint16_t SONAR3;
volatile uint16_t SONAR4;

```

```

int8_t i2c_stop(void);
void i2c_error(const char * message, uint8_t cr, uint8_t status);
int8_t i2c_start(uint8_t expected_status, uint8_t verbose);
int8_t i2c_sla_rw(uint8_t device, uint8_t op, uint8_t expected_status,
uint8_t verbose);
int8_t i2c_data_tx(uint8_t data, uint8_t expected_status, uint8_t verbose);
int8_t i2c_data_rx(uint8_t * data, uint8_t ack, uint8_t expected_status,
uint8_t verbose);

const char s_i2c_start_error[]    PROGMEM = "I2C START CONDITION ERROR";
const char s_i2c_sla_w_error[]    PROGMEM = "I2C SLAVE ADDRESS ERROR";
const char s_i2c_data_tx_error[]  PROGMEM = "I2C DATA TX ERROR";
const char s_i2c_data_rx_error[]  PROGMEM = "I2C DATA RX ERROR";
const char s_i2c_timeout[]        PROGMEM = "I2C TIMEOUT";
const char s_i2c_error[]          PROGMEM = "I2C ERROR\n";
const char s_fmt_i2c_error[]      PROGMEM = " TWCR=%02x STATUS=%02x\n";

int8_t cmpr03_get_byte(uint8_t device, uint8_t addr, uint8_t * value);
int8_t cmpr03_get_word(uint8_t device, uint8_t addr, uint16_t * value);
int16_t bearing16(void);
uint8_t bearing8(void);

void init_servos(void);
void motor_forward(void);
void motor_backward(void);
void motor_left(void);
void motor_right(void);
void motor_stop(void);
void motor_ff(void);
void motor_fl(void);
void motor_fr(void);
void motor_flb(void);
void motor_frb(void);
void motor_turn(void);
uint8_t get_mode(void);

volatile uint8_t MODE;

uint8_t bumper(void);
uint8_t bumperL(void);
uint8_t bumperR(void);
void detectFire(void);
void getBearing(void);

void avoid(void);

volatile uint16_t ms_count;
volatile uint16_t sonar_c;
volatile uint16_t motor_count;
volatile uint8_t PYRO;
volatile uint8_t RUN_MODE;
volatile uint16_t CMP;
volatile uint8_t lcd_counter = 0;
volatile uint8_t lcd_linenum = 1;

```

```
#include "main.h"
```

```
int main(void)
```

```
{
    init_timer();
    init_servos();
    PYRO = 0;
    EICRA |= _BV(ISC21) | _BV(ISC20);
    EIMSK |= _BV(INT2);

    TWSR &= ~0x03;
    TWBR = 28;
    TWCR |= _BV(TWEN);
    sei();

    DDRC = 0xff; //LCD
    DDRB = 0x00; //EMPTY
    DDRD = 0x90; //PYRO: BUMP: FAN 0x90
    DDRE = 0xff; //MOTOR
    DDRA = 0x55; //SONAR
    PORTB = 0x00;

    lcd_init();

    fdevopen(&lcd_write, NULL, 0);

    while (!bumperL())
    {
        if (bumper() && bumperR())
            RUN_MODE = 1;
        else if (bumper())
            RUN_MODE = 2;

        lcd_line(1);
        if (RUN_MODE == 1)
            printf("MODE DETECT FIRE");
        else
            printf("MODE OBS AVOID ");

        lcd_line(2);
        printf("HIT L. BUMPER TO RUN");
    }
    lcd_command(0x0f, 1);
    lcd_command(0x01, 1);

    while(1)
    {
        sonar();
        lcd_line(2);
        printf("CMP: %d ", bearing8());
        lcd_line(3);
        printf("BMP (B,L,R): %d %d %d", bumper(), bumperL(), bumperR());
        if (RUN_MODE == 1)
        {
```

```
        motor_turn();
        detectFire();
    } else
        avoid();
    //PYRO = 0;
}

} //end main

/** LCD
*****
*****/
/** LCD
*****
*****/
void lcd_init(void)
{
    ms_sleep(10000);

    PORTC = 0x03; //0x03
    PORTC = 0xC3; //43
    PORTC = 0x03;
    ms_sleep(105);
    PORTC = 0x03;
    PORTC = 0xC3; //43
    PORTC = 0x03;
    ms_sleep(5);
    PORTC = 0x03;
    PORTC = 0xC3;
    PORTC = 0x03;
    ms_sleep(105);
    PORTC = 0x02;
    PORTC = 0xC2; //42
    PORTC = 0x02;
    ms_sleep(5);
    lcd_command(0x28, 0);
    lcd_command(0x0f, 0);
    lcd_command(0x01, 0);

    lcd_command(0x28, 1);
    lcd_command(0x0f, 1);
    lcd_command(0x01, 1);
}

void lcd_command(uint8_t input, uint8_t display)
{
    uint8_t high;
    uint8_t low;
    uint8_t ENABLE;

    if (display == 0)
        ENABLE = 0xc0; //0x40
    else if (display == 1)
        ENABLE = 0xc0;
}
```

```
    high = (input >> 4);
    low = input & 0x0F;

    PORTC = high | 0x00; //0x00
    PORTC = high | ENABLE;
    PORTC = high | 0x00;
    ms_sleep(1);
    PORTC = low | 0x00;
    PORTC = low | ENABLE;
    PORTC = low | 0x00;
    ms_sleep(30);
}

int lcd_write(char strOut)
{
    uint8_t high;
    uint8_t low;
    uint8_t ENABLE;

    high = ((uint8_t)(strOut) >> 4);
    low = (uint8_t)(strOut) & 0x0F;
    //lcd_counter++;

    if (lcd_linenum < 5) //5
        ENABLE = 0xd0; //0x50
    else
        ENABLE = 0xd0;

    PORTC = high | 0x00; // 0x00s
    PORTC = high | ENABLE;
    PORTC = high | 0x00;
    ms_sleep(1); //1
    PORTC = low | 0x00;
    PORTC = low | ENABLE;
    PORTC = low | 0x00;
    ms_sleep(30); //30

    return strOut;
}

void lcd_line(uint8_t line)
{
    lcd_linenum = line;

    if (line == 1)
        lcd_command(0x80, 0);
    else if (line == 2)
        lcd_command(0xC0, 0);
    else if (line == 3)
        lcd_command(0x94, 0);
    else if (line == 4)
        lcd_command(0xd4, 0);

    else if (line == 5)
        lcd_command(0x80, 1);
    else if (line == 6)
        lcd_command(0xC0, 1);
}
```

```
else if (line == 7)
    lcd_command(0x94, 1);
else if (line == 8)
    lcd_command(0xd4, 1);
else
    { /*do nothing*/ }
}

/** LCD END
*****
*****/
/** LCD END
*****
*****/

/** SONAR
*****
*****/
/** SONAR
*****
*****/
void sonar(void)
{
    PORTA = 0x00;
    ms_sl eep(1);
    PORTA = 0x10;
    ms_sl eep(1);
    sonar_c = 0;
    PORTA = 0x00;
    while ((PI NA&0x20) != 0x20)
    ;
    while ((PI NA&0x20) != 0)
    ;
    //i f ((SONAR3 - sonar_c) < 20)
        SONAR2 = (SONAR2 + sonar_c) >> 1;
    ms_sl eep(20);

    PORTA = 0x00;
    ms_sl eep(1);
    PORTA = 0x01;
    ms_sl eep(1);
    sonar_c = 0;
    PORTA = 0x00;
    while ((PI NA&0x02) != 2)
    ;
    while ((PI NA&0x02) != 0)
    ;
    //i f ((SONAR1 - sonar_c) < 20)
        SONAR1 = (SONAR1 + sonar_c) >> 1;
    ms_sl eep(20);

    PORTA = 0x00;
    ms_sl eep(1);
    PORTA = 0x40;
    ms_sl eep(1);
```

```
sonar_c = 0;
PORTA = 0x00;
while ((PINA&0x80) != 0x80)
;
while ((PINA&0x80) != 0)
;
//if ((SONAR4 - sonar_c) < 20)
SONAR3 = (SONAR3 + sonar_c) >> 1;
ms_sleep(20);

PORTA = 0x00;
ms_sleep(1);
PORTA = 0x04;
ms_sleep(1);
sonar_c = 0;
PORTA = 0x00;
while ((PINA&0x08) != 8)
;
while ((PINA&0x08) != 0)
;
//if ((SONAR2 - sonar_c) < 20)
SONAR4 = (SONAR4 + sonar_c) >> 1;
ms_sleep(20);

lcd_line(4);
printf("SR: %d %d %d %d  ", SONAR1, SONAR2, SONAR3, SONAR4);
}

/** SONAR END
*****
*****/
/** SONAR END
*****
*****/

/** I2C
*****
*****/
/** I2C
*****
*****/

/*
* signal the end of an I2C bus transfer
*/
int8_t i2c_stop(void)
{
TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWSTO);
while (TWCR & _BV(TWSTO))
;
return 0;
}

/*
* display the I2C status and error message and release the I2C bus
*/
```

```
void i2c_error(const char * message, uint8_t cr, uint8_t status)
{
    i2c_stop();
    printf_P(message);
    printf_P(s_fmt_i2c_error, cr, status);
}

/*
 * signal an I2C start condition in preparation for an I2C bus
 * transfer sequence (polled)
 */
int8_t i2c_start(uint8_t expected_status, uint8_t verbose)
{
    uint8_t status;

    ms_count = 0;

    /* send start condition to take control of the bus */
    TWCR = I2C_START;
    while (!(TWCR & _BV(TWINT)) && (ms_count < I2C_TIMEOUT))
        ;

    if (ms_count >= I2C_TIMEOUT) {
        if (verbose) {
            i2c_error(s_i2c_start_error, TWCR, TWSR);
            i2c_error(s_i2c_timeout, TWCR, TWSR);
        }
        return -1;
    }

    /* verify start condition */
    status = TWSR;
    if (status != expected_status) {
        if (verbose) {
            i2c_error(s_i2c_start_error, TWCR, status);
        }
        return -1;
    }

    return 0;
}

/*
 * initiate a slave read or write I2C operation (polled)
 */
int8_t i2c_sla_rw(uint8_t device, uint8_t op, uint8_t expected_status,
                uint8_t verbose)
{
    uint8_t sla_w;
    uint8_t status;

    ms_count = 0;

    /* slave address + read/write operation */
    sla_w = (device << 1) | op;
```

```
TWDR = sla_w;
TWCR = I2C_MASTER_TX;
while (!(TWCR & _BV(TWINT)) && (ms_count < I2C_TIMEOUT))
;

if (ms_count >= I2C_TIMEOUT) {
    if (verbose) {
        i2c_error(s_i2c_sla_w_error, TWCR, TWSR);
        i2c_error(s_i2c_timeout, TWCR, TWSR);
    }
    return -1;
}

status = TWSR;
if ((status & 0xf8) != expected_status) {
    if (verbose) {
        i2c_error(s_i2c_sla_w_error, TWCR, status);
    }
    return -1;
}

return 0;
}

/*
 * transmit a data byte onto the I2C bus (polled)
 */
int8_t i2c_data_tx(uint8_t data, uint8_t expected_status, uint8_t verbose)
{
    uint8_t status;

    ms_count = 0;

    /* send data byte */
    TWDR = data;
    TWCR = I2C_MASTER_TX;
    while (!(TWCR & _BV(TWINT)) && (ms_count < I2C_TIMEOUT))
        ;

    if (ms_count >= I2C_TIMEOUT) {
        if (verbose) {
            i2c_error(s_i2c_data_tx_error, TWCR, TWSR);
            i2c_error(s_i2c_timeout, TWCR, TWSR);
        }
        return -1;
    }

    status = TWSR;
    if ((status & 0xf8) != expected_status) {
        if (verbose) {
            i2c_error(s_i2c_data_tx_error, TWCR, status);
        }
        return -1;
    }

    return 0;
}
```

```

}

/*
 * receive a data byte from the I2C bus (polled)
 */
int8_t i2c_data_rx(uint8_t * data, uint8_t ack, uint8_t expected_status,
                  uint8_t verbose)
{
    uint8_t status;
    uint8_t b;

    ms_count = 0;

    if (ack) {
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA);
    }
    else {
        TWCR = _BV(TWINT) | _BV(TWEN);
    }
    while (!(TWCR & _BV(TWINT)) && (ms_count < I2C_TIMEOUT))
        ;

    if (ms_count >= I2C_TIMEOUT) {
        if (verbose) {
            i2c_error(s_i2c_data_rx_error, TWCR, TWSR);
            i2c_error(s_i2c_timeout, TWCR, TWSR);
        }
        return -1;
    }

    status = TWSR;
    if ((status & 0xf8) != expected_status) {
        if (verbose) {
            i2c_error(s_i2c_data_rx_error, TWCR, status);
        }
        return -1;
    }

    b = TWDR;

    *data = b;

    return 0;
}
/** I2C END
*****
*****/
/** I2C END
*****
*****/

/** COMPASS
*****
*****/
/** COMPASS
*****
*****/

```

```
*****/
/*
 * read the data byte at the specified address from the specified device
 */
int8_t cmps03_get_byte(uint8_t device, uint8_t addr, uint8_t * value)
{
    uint8_t v;

    /* start condition */
    if (i2c_start(0x08, 1))
        return -1;

    /* address slave device, write */
    if (i2c_sla_rw(device, 0, TW_MT_SLA_ACK, 1))
        return -2;

    /* write register address */
    if (i2c_data_tx(addr, TW_MT_DATA_ACK, 1))
        return -3;

    /* repeated start condition */
    if (i2c_start(0x10, 1))
        return -4;

    /* address slave device, read */
    if (i2c_sla_rw(device, 1, TW_MR_SLA_ACK, 1))
        return -5;

    /* read data byte */
    if (i2c_data_rx(&v, I2C_NACK, TW_MR_DATA_NACK, 1))
        return -6;

    if (i2c_stop())
        return -7;

    *value = v;

    return 0;
}

/*
 * read the data byte at the specified address from the specified device
 */
int8_t cmps03_get_word(uint8_t device, uint8_t addr, uint16_t * value)
{
    uint8_t v1, v2;

    /* start condition */
    if (i2c_start(0x08, 1))
        return -1;

    /* address slave device, write */
    if (i2c_sla_rw(device, 0, TW_MT_SLA_ACK, 1))
        return -2;
```

```

/* write register address */
if (i2c_data_tx(addr, TW_MT_DATA_ACK, 1))
    return -3;

/* repeated start condition */
if (i2c_start(0x10, 1))
    return -4;

/* address slave device, read */
if (i2c_slave_rw(device, 1, TW_MR_SLA_ACK, 1))
    return -5;

/* read data byte 1 */
if (i2c_data_rx(&v1, I2C_ACK, TW_MR_DATA_ACK, 1))
    return -6;

/* read data byte 2 */
if (i2c_data_rx(&v2, I2C_NACK, TW_MR_DATA_NACK, 1))
    return -7;

if (i2c_stop())
    return -8;

*value = (v1 << 8) | v2 ;

return 0;
}

int16_t bearing16(void)
{
    uint16_t d;
    int8_t rc;

    rc = cmps03_get_word(0x60, 2, &d);
    if (rc < 0) {
        return -1;
    }

    return d;
}

uint8_t bearing8(void)
{
    uint8_t d;

    cmps03_get_byte(0x60, 1, &d);

    return d;
}

/** COMPASS END
*****
*****/

/** COMPASS END
*****
*****/

```

```
/** SERVO
*****
*****/
/** SERVO
*****
*****/
void init_servos(void)
{
    TCCR1A = 0xA0; //Clear on compare match (PORT B); PE5 = 0C1A; PE6 = 0C1B
    TCCR3A = 0xA0; //Clear on compare match (PORT E); PE3 = 0C3A; PE4 = 0C3B
    TCCR1B = 0x12; // CLK/8
    TCCR3B = 0x12;
    ICR1 = 20000; //50hz
    ICR3 = 2000; //1000hz

    OCR1A = 1500;
    OCR1B = 1500;

    OCR3A = 0;
    OCR3B = 0;
    PORTE = 0x24; //HIGH = RIGHT : LOW = LEFT
}

void motor_forward(void)
{
    //MODE = FF;
    PORTE = 0x24;
    OCR3A = SPEED_MAX;
    OCR3B = SPEED_MAX;
}

void motor_backward(void)
{
    PORTE = 0x00;
    if (bumper() == 0)
    {
        OCR3A = SPEED_MAX;
        OCR3B = SPEED_MAX;
    } else {
        OCR3A = SPEED_MIN;
        OCR3B = SPEED_MIN;
    }
}

void motor_left(void)
{
    MODE = LT;
    PORTE = 0x24;
    OCR3A = SPEED_MIN;
    OCR3B = SPEED_MAX;
}

void motor_right(void)
{
    MODE = RT;
    PORTE = 0x24;
}
```

```
    OCR3A = SPEED_MAX;
    OCR3B = SPEED_MIN;
}

void motor_stop(void)
{
    uint8_t tempA = OCR3A;
    uint8_t tempB = OCR3B;

    motor_count = 0;
    while (tempA != 0 || tempB != 0)
    {
        if (motor_count > 100 )
        {
            if (tempA > 0)
                tempA--;

            if (tempB > 0)
                tempB--;

            OCR3A = tempA;
            OCR3B = tempB;
            motor_count=0;
        }
    }
}

void motor_ff(void)
{
    PORTE = 0x24;
    OCR3A = SPEED_MID;
    OCR3B = SPEED_MID;
}

void motor_fl(void)
{
    PORTE = 0x20;
    OCR3A = SPEED_MID-150;
    OCR3B = SPEED_MID-150;
}

void motor_fr(void)
{
    PORTE = 0x04;
    OCR3A = SPEED_MID-150;
    OCR3B = SPEED_MID-150;
}

void motor_flb(void)
{
    PORTE = 0x24;
    OCR3A = SPEED_MIN;
    OCR3B = SPEED_MID;
}

void motor_frb(void)
{
```

```

    PORTE = 0x24;
    OCR3A = SPEED_MID;
    OCR3B = SPEED_MIN;
}

void motor_turn(void)
{
    PORTE = 0x04;
    OCR3A = SPEED_MID;
    OCR3B = SPEED_MID;
}

uint8_t get_mode(void)
{
    return MODE;
}
/** SERVO END
*****
*****/
/** SERVO END
*****
*****/

/** BUMPER
*****
*****/
/** BUMPER
*****
*****/
uint8_t bumper(void)
{
    uint8_t tempBump;
    tempBump = PIND;
    tempBump = tempBump & 0x08;
    if (tempBump != 0x08)
        return 1;
    else
        return 0;
}

uint8_t bumperL(void)
{
    uint8_t tempBump;
    tempBump = PIND;
    tempBump = tempBump & 0x20;
    if (tempBump != 0x20)
        return 1;
    else
        return 0;
}

uint8_t bumperR(void)
{
    uint8_t tempBump;
    tempBump = PIND;
    tempBump = tempBump & 0x40;
    if (tempBump != 0x40)

```

```
        return 1;
    else
        return 0;
}
/** BUMPER END
*****
*****/
/** BUMPER END
*****
*****/
SIGNAL(SIG_INTERRUPT2)
{
    PYRO = 1;
}

void detectFire(void)
{
    if (PYRO == 1)
    {
        getBearing();

        uint8_t PD = 0x90; //0x90
        //PORTD = PD;

        sonar();
        motor_stop();
        int16_t compass_diff = 0;
        uint8_t count = 0;
        while (1)
        {
            sonar();
            if (SONAR2 < 32 || SONAR3 < 32)
                count++;

            if (count > 5)
                break;

            compass_diff = CMP - bearing8();
            lcd_line(4);
            printf("CMP DIFF: %d  ", compass_diff);
            if (compass_diff < -2)
            {
                motor_count = 0;
                while (motor_count < 250)
                    motor_flb();
            } else if (compass_diff > 2) {
                motor_count = 0;
                while (motor_count < 250)
                    motor_frb();
            } else
            {
                motor_ff();
            } //end if
        } //end while

        getBearing();
    }
}
```

```
compass_diff = 0;
while(1)
{
    compass_diff = CMP - bearing8();
    lcd_line(4);
    printf("CMP DIFF: %d ", compass_diff);
    if (compass_diff < -1)
    {
        motor_count = 0;
        while (motor_count < 50)
            motor_flb();
    } else if (compass_diff > 1) {
        motor_count = 0;
        while (motor_count < 50)
            motor_frb();
    } else
        break;
    //end if
} //end while

lcd_line(4);
printf("FAN ON ");
motor_stop();
ms_sleep(1000);
for (uint16_t aa = 0; aa < 22000; aa++)
{
    PORTD = PD;
    if (PD == 0x90)
        PD = 0x10;
    else
        PD = 0x90;

    if (aa < 6000)
        ;
    else if (aa < 11000)
        motor_fl();
    else
        motor_fr();
    ms_sleep(20);
} //end for

PORTD = 0x00;
lcd_line(4);
printf("FAN OFF");

motor_stop();
motor_count = 0;
while (motor_count < 10000)
    motor_backward();
}
PYRO = 0;
}

void getBearing(void)
{
    motor_stop();
```

```

        motor_count = 0;
        while (motor_count < 10000)
            motor_fl ();
        PYRO = 0;
        while (PYRO != 1)
            motor_fr ();
        CMP = bearing8 ();
        lcd_line(1);
        printf("1st Lock: %d  ", CMP);
        motor_count = 0;
        while (motor_count < 20000)
            motor_fr ();
        PYRO = 0;
        while (PYRO != 1)
            motor_fl ();
        lcd_line(2);
        printf("2st Lock: %d  ", bearing8());
        CMP = CMP + bearing8();
        CMP = CMP >> 1;
        lcd_line(3);
        printf("CMP LOCK: %d  ", CMP);
    }

    /** TIMER
    *****/
    /** TIMER
    *****/
    /*
    * ms_sleep() - delay for specified number of milliseconds
    */
    void ms_sleep(uint16_t ms)
    {
        TCNT0 = 0;
        ms_count = 0;
        while (ms_count < ms)
            ;
    }

    /*
    * millisecond counter interrupt vector
    */
    SIGNAL(SIG_OUTPUT_COMPARE0)
    {
        ms_count++;
        sonar_c++;
        motor_count++;
    }

    /*
    * initialize timer 0 to generate an interrupt every millisecond.
    */
    void init_timer(void)
    {
        /*
        * Initialize timer0 to generate an output compare interrupt, and

```

```

    * set the output compare register so that we get that interrupt
    * every millisecond.
    */
    TIFR |= _BV(OIE0);
    TCCR0 = _BV(WGM01) | _BV(CS02) | _BV(CS00); /* CTC, prescale = 128 */
    TCNT0 = 0;
    TIMSK |= _BV(OIE0); /* enable output compare interrupt */
    OCRO = 5; /*(was 125) match in 1 ms (5 - this makes 40us) */
}

/** TIMER END
*****
*****/
/** TIMER END
*****
*****/

void avoid(void)
{
    uint16_t MOTOR_LENGTH = 25000;
    if (bumperL() == 1 || bumperR() == 1)
    {
        motor_stop();
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH)
            motor_backward();
    } else

    if (SONAR1 < 20)
    {
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH)
            motor_right();
    } else

    if (SONAR4 < 20)
    {
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH)
            motor_left();
    } else

    if (((SONAR2+SONAR3)>>1) > 25 && SONAR2 > 25 && SONAR3 > 25 && bumper()
!= 1)
        motor_forward();
    else
    {
        motor_stop();
        sonar();

        while (bumper() == 1)
            motor_left();

        if (SONAR2 > SONAR3)
        {
            motor_stop();
            motor_count = 0;
            while (motor_count < MOTOR_LENGTH+10000)

```

```
        motor_backward();

        motor_stop();
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH)
            motor_left();

    } else {
        motor_stop();
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH+10000)
            motor_backward();

        motor_stop();
        motor_count = 0;
        while (motor_count < MOTOR_LENGTH)
            motor_right();
        //ms_sleep(1000);
    }
}
}
```

