

T-1001

AUTONOMOUS BATTLE ROBOT FINAL ROBOT REPORT

**FERNANDO HERNANDEZ
8/01/2005**

— TAs —

**WILLIAM DUBEL
STEVEN PICKLES**

— INSTRUCTORS —

**A. A. ARROYO
E. M. SCHWARTZ**

— UNIVERSITY OF FLORIDA —

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
EEL 5666
INTELLIGENT MACHINES DESIGN LABORATORY**

TABLE OF CONTENTS

Executive Summary	3
Abstract	4
Introduction	5
Integrated System	6
Mobile Platform	7
Actuation	8
Sensors	9
Behaviors	13
Conclusion	14
References / Sources	15
Code Appendix	16

EXECUTIVE SUMMARY

This project details the successful creation of the T-1001 autonomous battle robot, created during the Summer term of IMDL 2005 at the University of Florida in Gainesville, Florida. The T-1001 is an autonomous battle robot that detects enemies on a battlefield by their color. This robot then targets the enemies using a foam disc shooter and attempts to destroy them. If it runs out of ammunition, the robot converts into a mobile battering ram and attempts to run over the enemy.

The T-1001 uses four different types of sensors: SRF04 ultrasonic sensors, IRPD-01 modulated IR sensor, Hamamatsu photoreflector, and the CMUCam vision sensor. Using all of these in conjunction, it is able to interpret its surroundings with enough clarity to detect events and characteristics such as an obstacle in its path, the edge of the battle zone, an enemy unit, and friendly or civilian units on the battlefield.

The T-1001 is powered by 8 AA batteries and uses servos for actuation. Two modified Futaba servos are used to drive the main treads while a single servo is used to pull the trigger back and forth. The firing mechanism itself is a small children's foam disc shooter, which shoots small foam discs at high velocity about 10 – 15 feet.

The T-1001 uses an Atmega128 to do make all the decisions and has been programmed in C using the avr-gcc compiler. The entire code base that drives the robot can be found at the end of the report, and details about all other aspects can be found in the pages following this executive summary. In addition, a special sensor characterization report has been provided along with this report, detailing the workings and my experience with the CMUCam vision sensor.

ABSTRACT

The T-1001 is an autonomous “battle tank” robot that patrols a battle zone, fires on enemy units, and avoids friendly units. This tank distinguishes enemy from friend by color, firing on enemy units when encountered. In addition to simply firing, the T-1001 keeps track of the amount of ammunition that it has left, and when the ammo has run out the tank will switch into “battering ram” mode, in which it will ram the enemy instead of firing upon it. In addition to these two behaviors, the T-1001 will detect a black line on the floor marking the limit of the battle zone, and not wander beyond it (No retreat! No surrender!). The T-1001 uses sensors of the following four types: ultrasonic (Devantech SRF04), IR (Lynxmotion IRPD-01), photo reflector (Hamamatsu P5587), and vision (CMUCam 1) to perform its functions and to properly interpret the battle zone.

INTRODUCTION

Some say that war is in man's nature. Through this project, I plan to prove that it is also in machine's nature. There are many dangerous situations in battle fields where it would be great if a robot could be sent in, taking the place of a human. The benefits would be two fold: obviously the greatest benefit of this would be in taking humans out of harm's way, preventing unnecessary deaths on the battle field. The second advantage lies in a robot's superior abilities. Dozens of sensors (for example, IR and UV), near unlimited endurance, and perfect accuracy are some of the many qualities in which one would much prefer to have a robotic warrior on the battle field than a human. Humans can succumb to fatigue, and are sensitive to biological agents and many types of radiation.

The T-1001 robot will lay the foundations for the first generation of battle field robots. It will be capable of destroying enemy units and avoiding friendly units on the battlefield, all while avoiding obstacles in its path. This robot can be sent into hostile environments where the risk to human life is too great and when strategic (and conveniently color coded) targets need to be taken out. Its name is a derivative of the T-1000 liquid metal robot from the movie "Terminator 2," and since the T-1001 will be an intelligent machine, I decided such a name would be fitting.

INTEGRATED SYSTEM

At the heart of the T-1001 is the Maverick IIB board. This board has an Atmega128 processor and can be considered a robot builder's dream when it comes to controlling virtually any robot. A large program memory, tons of ram, I²C readiness, dozens of digital I/O pins, and its small size are just a few of the many reasons that I chose this board to be the basis for my robot. This board performs all the processing of sensor data, and is accompanied by a power board which takes care of voltage regulation and motor driving, as well as helps with sensor interfacing. A block diagram of the overall system is shown below:

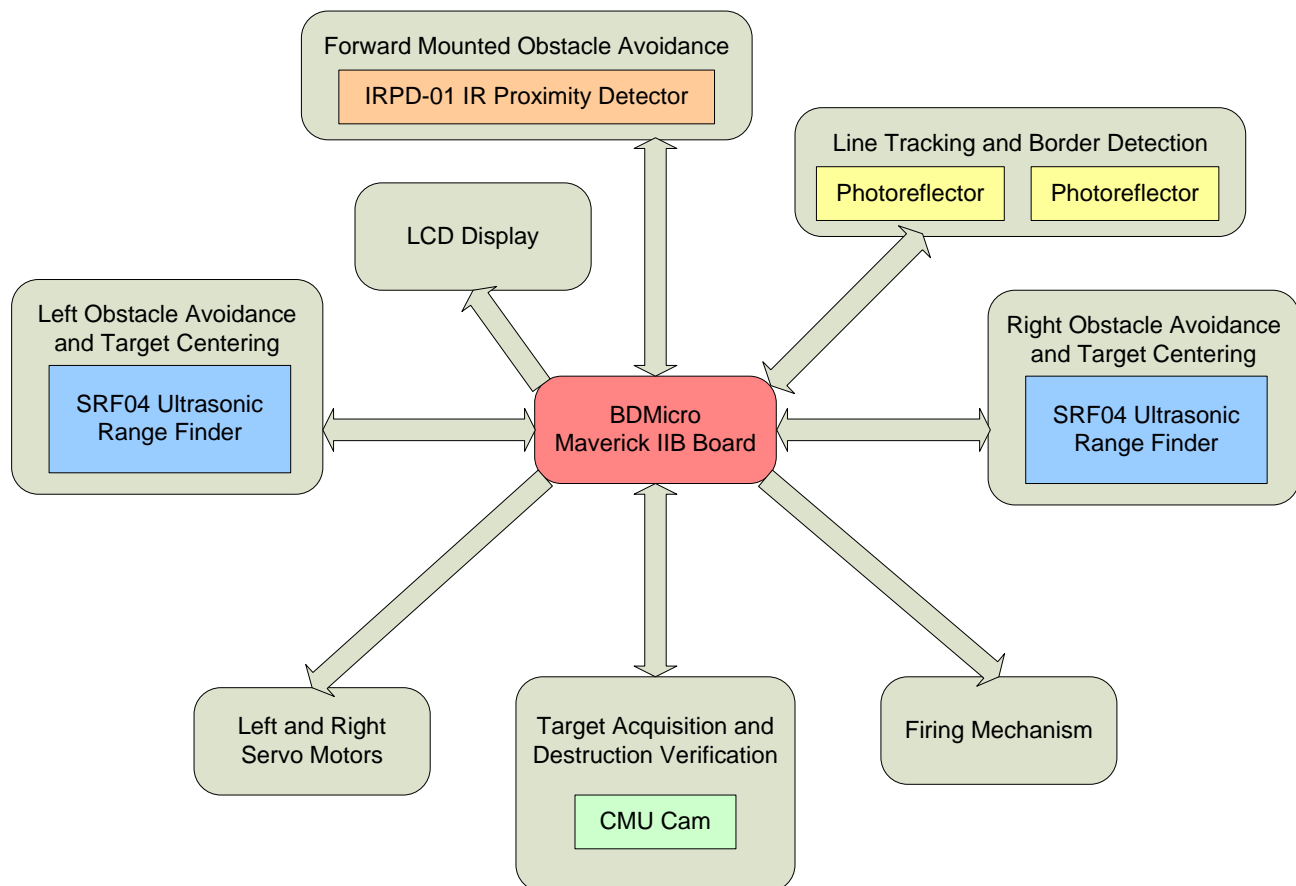


Fig 1. Block diagram of T-1001 control and processing system

MOBILE PLATFORM

Since T-1001 is an autonomous agent designed to work on the battlefield, my first instinct was to make it a tank, with treads for locomotion. This method of locomotion has proven successful in the real world and has also proven successful on my robot. The treads allow this robot to operate on surfaces ranging from a rough carpet to a smooth floor.

The platform itself is a dual-tiered platform made of a hard black plastic. This material is easy to drill into and is both lightweight and strong, and was ordered from Budget Robotics pre-cut into its current shape. Several holes were drilled to allow the mounting of sensors and the main board, and a large hole was cut into the top deck to allow the LCD cables to pass through.

On the first deck is where the ultrasonic sensors, the IR proximity detector, the Maverick board, the ON/OFF switch, and the power board are mounted. The second deck holds the LCD screen, the CMUCam, the firing servo, and the foam disc shooter. In order to accommodate an 8-AA battery pack, I decided to add a sub-deck, below the main deck and between the treads. After measuring the available space under the robot, I determined that the battery pack would fit snugly between the treads, and cut a rectangular piece of Lucite which served as the battery deck. It is attached to the main deck using two long 2" screws.

ACTUATION

The T-1001 is driven by two Futaba servos, each of which has been modified for continuous rotation. These servos drive the treads on which the base is mounted, and allow the robot to turn in place. The servo specifications are listed below:

Size:	1.55 x 1.40 x 0.79in.
Weight:	1.48oz.
Speed @ 6V:	.18 sec/60°
Torque @ 6V:	56 oz-in.

As can be seen, these servos are light weight and provide excellent torque, especially in this application in which overall weight of the system will be relatively low. In addition to using the two servos for locomotion, a third servo is used for operating the trigger mechanism which fires on the enemy. This firing servo is different however, because in order to pull the trigger back and forth precise positioning is required, so a non-modified hobby servo was used.

As stated in class, changing the motor speed abruptly may damage the servos, so I have developed a few actuation functions which will steadily change the servo speed from their current speed to the desired speed without any abrupt changes. The entire code can be found in the code appendix, but a sample can be found below:

```
// change speed until desired speed is reached (smoothly)
for(i=1;i<=speed;i++)
{
    L_MOTOR = L_MOTOR - 1;
    R_MOTOR = R_MOTOR + 1;
    ms_sleep(3);
}
```

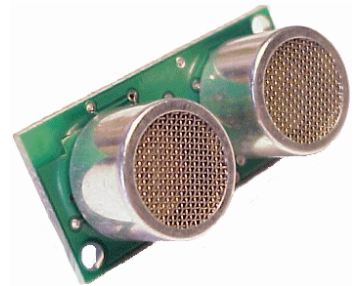
As it can be seen, this function will change the motor speed by 1 every 3 milliseconds. Having the robot wait a few milliseconds between changes, and making the changes small, result in a smooth acceleration or deceleration which does not harm the servos.

SENSORS

In order to detect the enemy and properly navigate on the battle field, the T-1001 implements various sensors, each of which is described on the following pages.

1) SRF04

The SRF04 is made by Devantech, and is an affordable ultrasonic ranging solution for any robot builder. This sensor sends out a sonic pulse above the range of human hearing and then listens for the return echo. The time between when the pulse is sent out and when the echo is received determines the distance to the object. Interfacing to a microcontroller is relatively simple, but care has to be taken to ensure that the time between when the pulse is sent out and when an echo is received is measure properly.



On the T-1001, there are two SRF04 sensors mounted approximately 30° to the left and 30° to the right of the center line. The pulses for each sensor have to be timed appropriately to ensure that the pulse from one sensor is not picked up as an echo by the other. To prevent this, they are simply fired in succession, with approximately 50ms in between firings to allow any echoes to dissipate.

2) IRPD-01 IR Proximity Sensor

The IRPD-01 is an all-in-one solution for IR proximity detection made by Lynxmotion. This sensor has two on board IR emitting diodes, and a single IR detector in the center. The diodes are modulated at 38 kHz by on board circuitry and the sensor picks



up any IR that happens to reflect off of an object in front of the detector. It has two on board potentiometers, one for adjusting the modulating frequency (not used for this project) and another for sensitivity (set to low sensitivity for this project).

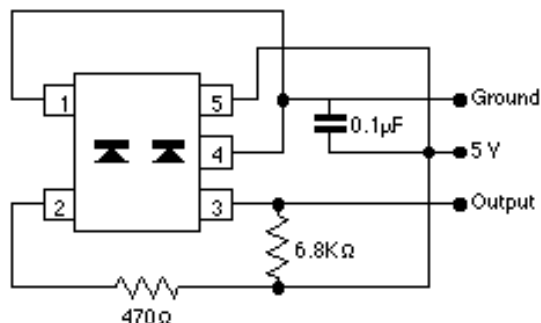
The sensor is mounted facing forward in order to pick up small objects that may pass through the small blind spot directly in front of the robot, in which the ultrasonic sensors cannot sense. Through trial and error, I have set the sensitivity to the point where it can detect most objects in front of it before contact is made, although a dark colored object may not reflect enough IR to set off the sensor, causing the robot to run into it. While I was tempted to simply set the sensitivity higher, this would mean that bright objects would be picked up from over 3 feet away, at which time they are not yet considered an obstacle.

3) Hamamatsu Photoreflector

The Hamamatsu photoreflector is a small 5 legged IC that incorporates an IR emitter and detector in one small package. The T-1001 uses two downwards facing photoreflectors in order to detect a black line which signifies the edge of the battle zone. Care has to be taken in mounting this



sensor, because the IR is not modulated and any type of IR can accidentally set it off. It is for this reason that I mounted the photoreflector inside of a special custom made plastic casing that shields it from as much external IR as possible. The circuit that I have used to mount it on the robot is pictured here:



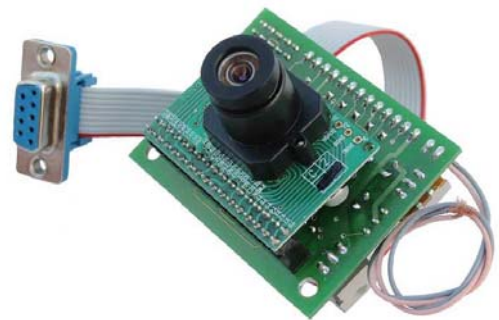
Since only a few components are needed, I decided to solder them directly to the legs of the photoreflector, instead of mounting them on a special board that would be needed otherwise. I was not sure if this would allow for correct operation of the photoreflector, but after building one and testing it, proper circuit operation was verified. The modified photoreflector can be seen at right



Unfortunately, the successful operating range from the photoreflector to the surface is only about one inch. Any larger distance and the photoreflector may report a false positive because some external IR may leak in. Also, the IR that is emitted from the photoreflector is not strong enough to reflect off a surface and be picked up by the receiver after about an inch. It is because of this that the photoreflector needs to be mounted fairly close to the ground, less than an inch and preferably within $\frac{1}{4}$ of an inch.

4) CMUCam Vision Sensor

In order to detect the enemy and properly navigate on the battle field, the T-1001 implements various sensors, of which the CMUCam is the most complex. The CMUCam vision sensor is made by a group of people at Carnegie Mellon University, hence its name. The CMUCam is a great vision sensor for applications where simple low level sensors like CDS cells do not provide enough data about the environment. This camera performs much of its processing on board, communicating statistics of the images that it sees to the microcontroller board attached to it at up to 17 frames per second. This allows a small



microcontroller (which would normally not be capable of dealing with vision) to use vision to interpret its environment. This means that now the realm of vision sensing is no longer out of reach of the casual robotics hobbyist.

The camera has a large serial command set, the entirety of which can be found in the CMUCam documentation. Of interest are commands such as “GM” which returns the average value of red, green, and blue on the image, as well as their standard deviations, the TC command which is used to track a color and return its bounding box and its center coordinates, and the SW command which is used to set the current view window, allowing one to track colors only in a certain portion of the screen.

The CMUCam is interfaced to the Maverick board using UART1, which is the second RS-232 port on the board, and I have selected 38.4k baud as the baud rate. Since the UART ports on the board are not the standard pin headers but instead the screw-down clamp type, I have them go first to a corner of the power board, to where they connect to a pin header, into which the camera’s pins are attached. This makes it easy to remove the camera, without needing a screwdriver to release the cables from the UART port.

The CMUCam is considered my “special sensor” for this project, and much more information about my experience with it can be found in my special sensor report.

BEHAVIORS

The T-1001 will has multiple modes of operation, depending on what it has sensed in the world around it. Its primary task is to search and destroy enemy units, which indentifies by color using its CMUCam. It centers itself on these units, fire upon them, and continues on when it has verified that the units have been destroyed, searching for more enemy units. The T-1001 keeps track of the amount of ammunition that it has left, and when it is out of ammo it converts to “battering ram” mode, in which it attempts to ram and destroy enemy units while pushing them out of the battle zone.

While searching for these enemy units, if the T-1001 detects that it is about to exit the battle zone (designated by a black strip running along the edge of the battle field) it will turn around and head back into the battle zone (No retreat! No surrender!). If at any point in its search it detects an obstacle in its path, the T-1001 will attempt to navigate around it, choosing either to go right or left. If the battle zone border is encountered while navigating around an obstacle, the T-1001 will stop, turn around, and head in the other direction. The main robot control algorithm is shown below:

- 1) Initialize the servos, the LCD screen and the CMUCam.
- 2) Rotate and perform an initial scan of the battle zone for five seconds (immediately go to step 3 if an enemy is detected)
- 3) Use CMUCam to attempt to find an enemy, if found, go to step 4, otherwise, go to step 5.
- 4) Center the robot with respect to the enemy and move forward until the robot is close enough to fire. When close enough, fire or ram the enemy depending on how much ammo remains, then go to step 3.

- 5) Check the photoreflectors for the edge of the battle zone. If detected, move backwards and turn accordingly, go to step 3.
- 6) Check the IR sensor for an obstacle in front of the robot. If an obstacle is found, move backwards and turn in a random direction, go to step 3.
- 7) Check both ultrasonic sensors for obstacles on the sides. If an obstacle is found, move in the opposite direction, go to step 3.
- 8) If this line is reached, then there are no obstacles or enemies in the field of vision, set motors to “forward” and go to step 3.

The code which this algorithm represents can be found in the code appendix at the end of the report.

CONCLUSION

Through the use of multiple sensors, the T-1001 is able to sense its environment with enough accuracy to avoid obstacles and destroy enemy units. It represents a step toward a fully autonomous fighting machine. The future of warfare will see devices and machines built with this idea in mind, allowing for surgical strikes and precision attacks with minimal loss of life. The T-1001 represents the future of warfare, a future that belongs to autonomous robotic agents.

REFERENCES / SOURCES

NOTE: The following papers/websites were used during the making of my robot and the creation of this paper. I kindly acknowledge their assistance.

CMUCam User's Manual v2.00

Anthony Rowe and Carnegie Mellon University, edited by Charles Rosenberg and Illah Nourbakhsh
< <http://www-2.cs.cmu.edu/~cmucam/Downloads/CMUcamManual.pdf> >

SRF04 Ultrasonic Ranger Technical Specification

< <http://info.hobbyengineering.com/specs/devantech-srf04-tech.pdf> >

Hamamatsu Photoreflector Data sheet

< <http://www.acroname.com/robotics/parts/R64-P5587.pdf> >

IRPD-01 User's Guide

< <http://www.lynxmotion.com/images/data/irpd-v7.pdf> >

MAVRIC-IIB Manual

< <http://www.bdmicro.com/images/mavric-iib.pdf> >

Atmega128 Complete Datasheet

< http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf >

I would like to give a special thanks to Julio Suarez for assistance with the initialization and communication code for the CMUCam.

CODE APPENDIX

Code Written using “Programmers Notepad 2” by William Steele

Code Compiled using avr-gcc

Board programmed using “Pony Prog 2000” v2.06f Beta by Claudio Lanconelli

```

/*****
Fernando Hernandez
Code designed to work with the Maverick IIB board from BDMicro
*****/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <stdlib.h>
#include <stdio.h>

// 20x4 LCD Screen =====
// LCD DATA
LCD Control
// Port Pin      0      1      2      3      4      5      6      7
// Port Pin      0      1      2
// LCD Pin      DB0    DB1 DB2 DB3 DB4 DB5 DB6 DB7          LCD Pin      RS
// RW          EN
#define LCD_DATA_PORTX      PORTA
#define LCD_CTRL_PORTX     PORTF
#define LCD_DATA_DDRX      DDRA
#define LCD_CTRL_DDRX      DDRF

// SRF04 Ultrasonic =====
// Port Pin      0      1      2      3      4      5      6      7
// LCD Pin      OUT TRG          OUT TRG
//          --#1---          --#2---
#define SRF_PORT      PORTD
#define SRF_DDRX      DDRD
#define SRF_PINX      PIND

// Photoreflectors =====
// Port Pin      0      1      2      3      4      5      6      7
// Ph.Ref.Pin    TRG GND PWR          TRG GND PWR
//          Left          Right
#define PHR_PORT      PORTC
#define PHR_DDRX      DDRC
#define PHR_PINX      PINC

// IRPD Infrared =====
// Port pin      2      1      0
// IR Pin      LED          LED          Output
#define IR_PORT      PORTG
#define IR_DDRX      DDRG
#define IR_PINX      PING

// Servo Motors =====
#define L_MOTOR      OCR1A
#define R_MOTOR      OCR1B
#define TRIGGER_MOTOR OCR1C
#define M_CENTER      18500
#define S_CENTER      18800

// Gun motor =====
// Port Pin      0      1      2      3      4      5      6      7
// Gun Pin      TRG
#define GUN_PORTX      PORTB
#define GUN_DDRX      DDRB

// Global vars used for delays
volatile uint16_t ms_count;
volatile uint16_t us_count;

// Delay for a specified number of milliseconds
// Note: will not work for more than about 3100ms because of interger overflow!
// 65536/21 = 3120ms max
void ms_sleep(uint16_t ms)
{
    TCNT0 = 0;
    ms_count = 0;
    while (ms_count != ms*21);
}

```

```

// Delay for specified number of microseconds * 48
void us_sleep(uint16_t us)
{
    TCNT0 = 0;
    us_count = 0;
    while (us_count != us);
}

// millisecond counter interrupt vector
SIGNAL(SIG_OUTPUT_COMPARE0)
{
    ms_count++;
    us_count++;
}

// initialize timer 0 to generate an interrupt every 48usec
void init_timer(void)
{
    TIFR |= _BV(OCIE0);
    TCCR0 = _BV(WGM01)|_BV(CS02)|_BV(CS00); // CTC, prescale = 128
    TCNT0 = 0;
    TIMSK |= _BV(OCIE0); // enable output compare interrupt
    OCR0 = 6; // match in 48usec
}

// =====
// LCD Functions =====
// =====

// Initializes the display
void lcd_initialize(void)
{
    LCD_CTRL_DDRX = 0x07; // enable lower 3 bits of control port
    LCD_DATA_DDRX = 0xFF; // enable entire data port

    LCD_DATA_PORTX = 0x38; // set for 8 bit mode, 1/16 duty cycle
    LCD_CTRL_PORTX = 0x04; // enable high
    ms_sleep(1); // wait
    LCD_CTRL_PORTX = 0x00; // enable low
    ms_sleep(1); // wait

    LCD_DATA_PORTX = 0x0C; // turn display on, cursor off, and set no blink
    LCD_CTRL_PORTX = 0x04;
    ms_sleep(1);
    LCD_CTRL_PORTX = 0x00;
    ms_sleep(1);

    LCD_DATA_PORTX = 0x06; // set auto increment (to write forwards)
    LCD_CTRL_PORTX = 0x04;
    ms_sleep(1);
    LCD_CTRL_PORTX = 0x00;
    ms_sleep(1);
}

// Turns the cursor on - cursor("on") and off - cursor ("off")
void lcd_cursor(char command[])
{
    if(command[1]==116 || command[1]==110){
        LCD_DATA_PORTX = 0x0E;
        LCD_CTRL_PORTX = 0x04;
        ms_sleep(1);
        LCD_CTRL_PORTX = 0x00;
        ms_sleep(1);}
    else{
        LCD_DATA_PORTX = 0x0C;
        LCD_CTRL_PORTX = 0x04;
        ms_sleep(1);
        LCD_CTRL_PORTX = 0x00;
        ms_sleep(1);}
}

// Goes to the beginning of the line specified by its argument (valid ranges = 1 to 4)

```

```

void lcd_goto_line(int line)
{
    switch(line)
    {
        case(1):
        {
            LCD_DATA_PORTX = 0x80; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(2):
        {
            LCD_DATA_PORTX = 0xC0; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(3):
        {
            LCD_DATA_PORTX = 0x94; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(4):
        {
            LCD_DATA_PORTX = 0xD4; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
    }
}
// Goes to a line specified by the argument (valid ranges = 1 to 4) and
// then a position on that line (valid ranges = 1 to 20)
void lcd_goto_pos(int line, int pos)
{
    switch(line)
    {
        case(1):
        {
            LCD_DATA_PORTX = 0x80+pos-1; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(2):
        {
            LCD_DATA_PORTX = 0xC0+pos-1; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(3):
        {
            LCD_DATA_PORTX = 0x94+pos-1; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
        case(4):
        {
            LCD_DATA_PORTX = 0xD4+pos-1; LCD_CTRL_PORTX = 0x04; ms_sleep(1);
            LCD_CTRL_PORTX = 0x00; ms_sleep(1);
            break;
        }
    }
}

// Clears the entire LCD screen
void lcd_clear_display(void)
{
    LCD_CTRL_PORTX = 0x04;
    LCD_DATA_PORTX = 0x01;
    ms_sleep(1);
    LCD_CTRL_PORTX = 0x00;
    ms_sleep(1);
}

```

```

// Outputs a single character to the LCD
void lcd_put_char(char c)
{
    LCD_CTRL_PORTX = 0x05;
    LCD_DATA_PORTX = c;
    ms_sleep(1);
    LCD_CTRL_PORTX = 1;
    ms_sleep(1);
}

// Outputs a string message (character array) to the screen
void lcd_put_string(char message[])
{
    int i;
    for(i=0;i<21;i++)
    {
        if(message[i]=='\0' || message[i]=='\n') break;
        else lcd_put_char(message[i]);
    }
}

// Flashes a text message on a specified line, every specified
// number of milliseconds, a specified number of times.
void lcd_flash_message(int line, int delay, int num_of_times, char message[] )
{
    int flag = 0;
    num_of_times = num_of_times * 2;
    for(;num_of_times>0;num_of_times--)
    {
        if(flag==0)
            {ms_sleep(delay); lcd_goto_line(3); lcd_put_string(message); flag=1;}
        else
            {ms_sleep(delay); lcd_goto_line(3); lcd_put_string("                ");
flag=0;}
    }
}

// Displays the intro text
void lcd_show_intro(void)
{
    int delay = 100;
    lcd_clear_display();
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("1");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("01");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string(" T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("  T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("   T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("    T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("     T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("      T-1001");
    ms_sleep(delay); lcd_goto_line(2); lcd_put_string("       T-1001");

    ms_sleep(delay); lcd_flash_message(3,250,3,"  Initializing... ");
}

// =====
// SRF04 Functions =====
// =====

// Function to get distance data from an SRF04, first
// argument specifies which one. ( 1 = left, 2 = right )
int SRF04(int sensor_num, int times)
{
    int i,counter = 0;

    for (i=0; i<times ;i++)
    {
        ms_sleep(50); // wait 50msec between pulses for echo to settle

```

```

// Trigger a ping on coresponding SRF04 and wait a few usec
if(sensor_num == 1){
    SRF_DDRX = 0x02;           // set output pin
    SRF_PORT = 0x02;          // trigger
    us_sleep(8);              // wait a few usec
    SRF_PORT = 0x00;          // end trigger
}else {
    SRF_DDRX = 0x80;
    SRF_PORT = 0x80;
    us_sleep(8);
    SRF_PORT = 0x00;}

// wait a few usec for echo circuit initialization
us_sleep(8);

while(1)
{
    if(sensor_num == 1){
        if(SRF_PINX & 0x01){ // check if echo has been received
            counter++;       // increment counter
            us_sleep(4);     // wait a few usec
        }else break;        // break if echo was received
    }else{
        if(SRF_PINX & 0x40) {
            counter++;
            us_sleep(4);}
        else break;}
    }
}
return ((counter/times)*1.8); // return average (calibrated for inches)
}

// =====
// Motor Drive Functions =====
// =====

void motors_initialize(void)
{
    // count up to 20,000 at a rate 8x slower than the 16MHz clock
    // yielding 10ms to count up, and 10ms to count down = 20 ms period
    ICR1 = 20000;

    // Enables OC1 and OC3 on all channels (A, B, and C)
    // OC bits will set on upcount and clear on downcount, and the counters
    // take their TOP value from the ICRx register. (waveform generation mode #8)
    //      7           6           |           5           4           |           3
    //      2           |           1           0           |           |           |
    //      Compare Mode |           Compare Mode |           Compare Mode |           Waveform
Generation
//      Channel A           |           Channel B           |           Channel C           |
Bits 1 and 0
TCCR1A = 0xFC;

    // Set prescaler to 8x slower than chip clock, set other half of
    // waveform generation mode
    //      7           6           |           5           |           4           3
    //      |           2           1           0           |           |           |
    //      ICNC   ICES   |           Reserved|           Wave Generation           |           Clock Select Bits
    //      |           |           |           |           |           |           |
    //      Bits 3 and 2   |
TCCR1B = 0x12;

    // Set timer to zero (16 bit timers)
    TCNT1 = 0x0000;

    // Sets ports to outputs.(OC1A ,B, C)
    // Pin outputs for A, B, C are B5, B6, B7, respectively
    DDRB = 0xE1;

    // M_CENTER both servos
    L_MOTOR = M_CENTER;
}

```

```

        R_MOTOR = M_CENTER;
    }

// Stops the motors
void motors_stop(void)
{
    int l_incrementor, r_incrementor;

    // if motors are already centered, just return
    if(L_MOTOR == M_CENTER && R_MOTOR == M_CENTER) return;

    // if the left motor is currently going backwards, set L incrementor to positive, else
    negative
    if(L_MOTOR < M_CENTER) l_incrementor = 1;
    else l_incrementor = -1;

    // if the right motor is currently going backwards, set L incrementor to positive, else
    negative
    if(R_MOTOR < M_CENTER) r_incrementor = 1;
    else r_incrementor = -1;

    // change their speed until desired speed is reached (smoothly)
    while(L_MOTOR != M_CENTER)
    {
        L_MOTOR = L_MOTOR + l_incrementor;
        R_MOTOR = R_MOTOR + r_incrementor;
        ms_sleep(3);
    }
}

```

```

//=====
// Move in a specified direction, at a specified speed, for a certain
// amount of time * .1 sec. (Ex. time=10 corresponds to 1 second.
//=====
// Valid ranges for arguments are:
// Direction:  F, B, L, R           Forward, backward, left, right
// Speed:      0 - 100              Full Stop through full speed
// Time:       0 - 32k              No delay through 4.5 hours
//=====

```

```

void motors_move(char direction[], int speed)
{
    int i;

    // Forwards, decrement left motor, increment right motor
    if(direction[0]==70 || direction[0]==102)
    {
        if(L_MOTOR == M_CENTER - speed && R_MOTOR == M_CENTER + speed) return;
        else motors_stop();

        // change their speed until desired speed is reached (smoothly)
        for(i=1;i<=speed;i++)
        {
            L_MOTOR = L_MOTOR - 1;
            R_MOTOR = R_MOTOR + 1;
            ms_sleep(3);
        }
    }

    // Backwards, increment left motor, decrement right motor
    if(direction[0]==66 || direction[0]==98)
    {
        if(L_MOTOR == M_CENTER + speed && R_MOTOR == M_CENTER - speed) return;
        else motors_stop();

        // change their speed until desired speed is reached (smoothly)
        for(i=1;i<=speed;i++)
        {
            L_MOTOR = L_MOTOR + 1;
            R_MOTOR = R_MOTOR - 1;
            ms_sleep(3);
        }
    }
}

```

```

    }
}

// Left turn, increment both motors
if(direction[0]==76 || direction[0]==108)
{
    if(L_MOTOR == M_CENTER + speed && R_MOTOR == M_CENTER + speed) return;
    else motors_stop();

    // change their speed until desired speed is reached (smoothly)
    for(i=1;i<=speed;i++)
    {
        L_MOTOR = L_MOTOR + 1;
        R_MOTOR = R_MOTOR + 1;
        ms_sleep(3);
    }
}

// Right turn, decrement both motors
if(direction[0]==82 || direction[0]==114)
{
    if(L_MOTOR == M_CENTER - speed && R_MOTOR == M_CENTER - speed) return;
    else motors_stop();

    // change their speed until desired speed is reached (smoothly)
    for(i=1;i<=speed;i++)
    {
        L_MOTOR = L_MOTOR - 1;
        R_MOTOR = R_MOTOR - 1;
        ms_sleep(3);
    }
}
}

// =====
// IR Sensor Functions =====
// =====

// check the IR sensors a specified number of times
// returns a number between 0 and .5
// if it returns a value > .5, its likely an object was detected
float ir_sense(int times)
{
    int value = 0, repeat_counter = times;

    // Mask LED's and activate them
    IR_DDRX = 0x06;
    IR_PORT = 0x06;
    ms_sleep(1);

    while(repeat_counter > 0)
    {
        if(IR_PINX & 0x01) value = value;
        else value = value + 1;
        repeat_counter--;
        ms_sleep(1);
    }

    return(value/times);
}

// =====
// CMUCam Functions =====
// =====

volatile int MAX_MSG_SIZE = 30;
volatile unsigned char CMUResponseBuffer[15];

// initialize UART1 to 38.4k baud rate

```

```

void UART1_init(void)
{
    UBRR1H = 0x00;
    UBRR1L = 0x33;
    UCSR1A |= 0x02;
    UCSR1C = 0x06;
    UCSR1B = 0x18;
}

// transmit a message (char array) over UART1
void USART1_Transmit(char data[MAX_MSG_SIZE])
{
    int t = 0;
    while ((t < (MAX_MSG_SIZE + 1)) & (data[t] != 0x00))
    {
        // wait for empty transmit buffer
        while ((UCSR1A & _BV(UDRE1)) == 0);
        UDR1 = data[t];
        t++;
    }
}

// wait for data to be received and then returns it
// NOTE: this is a BLOCKING receive!
unsigned char USART1_Receive( void )
{
    while ( !(UCSR1A & (1<<RXC1)) );
    return UDR1;
}

void CMU_init(void)
{
    USART1_Transmit("RS\r"); // reset
    ms_sleep(20);
    USART1_Transmit("PM 1\r"); // poll mode
    ms_sleep(20);
    USART1_Transmit("RM 3\r"); // raw output
    ms_sleep(20);
    USART1_Transmit("MM 1\r"); // middle mass on
    ms_sleep(20);
    USART1_Transmit("SW 1 80 80 143 \r"); // use bottom half for tracking
    ms_sleep(20);
}

// queries the CMU cam to get the mean values of R, G, B
// stores them in the global "CMUResponseBuffer"
void CMU_GetMean(void)
{
    int i = 0;
    char tempChar;

    USART1_Transmit("GM\r");

    // initial 255 framing byte is read in, discarded
    tempChar = USART1_Receive();

    // this command returns a "type S" packet, 7 bytes long
    // we read in those 7 bytes
    for(i=0;i<7;i++)
    {
        CMUResponseBuffer[i] = USART1_Receive();
    }

    // last 255 framing byte is read in, discarded
    while(tempChar != ':')
    {
        tempChar = USART1_Receive();
    }

    CMUResponseBuffer[i] = '\0';
}

```

```

// tells the CMUCam to track a color
void CMU_TrackColor(int Rmin, int Rmax, int Gmin, int Gmax, int Bmin, int Bmax)
{
    int i = 0;
    char tempChar, tempMessage[30];

    sprintf(tempMessage,"TC %i %i %i %i %i %i\r",Rmin, Rmax, Gmin, Gmax, Bmin, Bmax);

    USART1_Transmit(tempMessage);

    // initial 255 framing byte is read in, discarded
    tempChar = USART1_Receive();

    // this command returns a "type M" packet, 9 bytes long
    // we read in those 9 bytes
    for(i=0;i<9;i++)
    {
        CMUResponseBuffer[i] = USART1_Receive();
    }

    // last 255 framing byte is read in, discarded
    while(tempChar != ':')
    {
        tempChar = USART1_Receive();
    }

    CMUResponseBuffer[i] = '\0';
}

// used to convert the raw data returned by the CMUCam to an integer
int binary2int(unsigned char binary_num)
{
    int result = 0;
    if(binary_num & 1) result +=0;
    if(binary_num & 2) result +=2;
    if(binary_num & 3) result +=4;
    if(binary_num & 8) result +=8;
    if(binary_num & 16) result +=16;
    if(binary_num & 32) result +=32;
    if(binary_num & 64) result +=64;
    if(binary_num & 128) result +=128;
    return result;
}

// =====
// Photoreflector Functions =====
// =====

// check a photoreflector a certain number of times
int photoreflector(char which[], int times)
{
    // Mask photoreflectors ports, set triggers to input, and others to output
    PHR_DDRX = 0b11011110;
    int counter = times, value = 0;

    while(counter>0)
    {
        //if right sensor is the one desired
        if(which[0]==82 || which[0]==114)
        {
            // Turn on the right photoreflector
            PHR_PORT = 0b00000100;
            ms_sleep(5);
            // If pin was high, then white was detected
            if(PHR_PINX & 0b00000001) value += 0;
            // If pin was low, then black was detected
            else value += 1;
        }
        // if left sensor is the one desired
        if(which[0]==76 || which[0]==108)

```

```

        {
            // Turn on the left photoreflector
            PHR_PORT = 0b10000000;
            ms_sleep(5);
            // If pin was high, then white was detected
            if(PHR_PINX & 0b00100000) value += 0;
            // If pin was low, then black was detected
            else value += 1;
        }
        counter--;
    }
    return value/times;
}

// =====
// Trigger Functions =====
// =====

// Turns the trigger forward (1), backwards (-1), or stop the trigger (0)
void trigger(int command)
{
    if(command == 1)
        TRIGGER_MOTOR = M_CENTER + 10;
    else if(command == -1)
        TRIGGER_MOTOR = M_CENTER - 18;
    else
        TRIGGER_MOTOR = M_CENTER;
}

// =====
// Main Gun Functions =====
// =====

// Fires the gun! FIRE IN THE HOLE!
void main_gun_on(void)
{
    GUN_PORTX = 0x01;
}

void main_gun_off(void)
{
    GUN_PORTX = 0x00;
}

// =====
// =====
// =====

int main(void)
{
    int TOLERANCE = 5, left_dist, right_dist;
    int tracked_object_position = -1;
    int right_PR_turn = 0;
    int left_PR_turn = 0;
    int ammo_remaining = 3;
    int target_approach_counter = 15;
    int confidenceValue;
    char message[10];
    int left_edge_hit = 0, right_edge_hit = 0;

    // Intialization stuff
    =====

    init_timer();           // initialize timer
    sei();                  // enable interrupts
    UART1_init();          // enable UART1
    lcd_initialize();       // initialize LCD screen
    lcd_show_intro();      // show spiffy intro!
    CMU_init();             // initialize CMUCam

```

```

lcd_clear_display(); // clear the LCD screen
motors_initialize(); // initialize servos

// Trigger reset, because it jerks a little when system is started up=====

TRIGGER_MOTOR = S_CENTER - 525;

// =====
// before we enter the main while loop, we want to scan the area for possible targets.....

// spin in a random direction
if(TCNT1L & 0x01) {motors_move("L",20);}
else {motors_move("R",20);}

// teack a color once, because first command always returns a false positive.... *shrug*
CMU_TrackColor(100, 255, 0, 30, 0, 20);
ms_sleep(50);

// display the initial message
lcd_clear_display();
lcd_goto_line(1); lcd_put_string("Running initial");
lcd_goto_line(2); lcd_put_string(" scan of battle");
lcd_goto_line(3); lcd_put_string(" zone....");

// search for a target as you spin, to find next target
int k;
for(k = 40 ; k > 0; k--)
{
    lcd_goto_pos(4,19); sprintf(message,"%i",k/8);
    lcd_put_string(message);
    ms_sleep(125);
    CMU_TrackColor(100, 255, 0, 30, 0, 20);
    if(binary2int(CMUResponseBuffer[1]) != 0) goto start;
}
// =====

while(1)
{
    start:
    ms_sleep(50);

    // track the color and see if it is found
    CMU_TrackColor(100, 255, 0, 30, 0, 20);
    if(binary2int(CMUResponseBuffer[1]) == 0)
    {
        // if not found, then display normal message, and follow through
        lcd_goto_line(1);lcd_put_string("Scanning for an");
        lcd_goto_line(2);lcd_put_string(" enemy to target...");
        lcd_goto_line(4);lcd_put_string("Ammo left: ");
        sprintf(message,"%i",ammo_remaining);
        lcd_put_string(message);
        tracked_object_position = -1;
        goto do_sensors;
    }else{
        // if a target is found, figure out a few stats, which are used below to
determine what to do
        confidenceValue = binary2int(CMUResponseBuffer[8]);
        tracked_object_position = binary2int(CMUResponseBuffer[1]);

        // do not go after target if the confidence isnt high enough
        if(confidenceValue < 150) goto do_sensors;

        // if we are sure its a target, go for it!
        lcd_clear_display();
        lcd_put_string("Enemy detected!!");

        lcd_goto_line(2);
        sprintf(message,"%i",confidenceValue);
        lcd_put_string("Confidence: "); lcd_put_string(message);lcd_put_string("
");

        lcd_goto_line(4); lcd_put_string("Ammo left: ");
        sprintf(message,"%i",ammo_remaining);

```

```

        lcd_put_string(message);

        // get a couple of more readings, to average it out
        CMU_TrackColor(100, 255, 0, 30, 0, 20);ms_sleep(50);
        tracked_object_position += binary2int(CMUResponseBuffer[1]);

        CMU_TrackColor(100, 255, 0, 30, 0, 20);ms_sleep(50);
        tracked_object_position += binary2int(CMUResponseBuffer[1]);

        tracked_object_position /= 3;
    }

    // if the object is on the left, move that way
    if(tracked_object_position > 0 && tracked_object_position < 38)
    {
        motors_move("L",15);
        continue;
    }

    // if the object is on the right, move that way      (adjusted for faster right turns
than left)
    else if(tracked_object_position > 48)
    {
        motors_move("R",10);
        continue;
    }

    // if the object is pretty much centered, move towards it
    else if(tracked_object_position <= 48 && tracked_object_position >= 38)
    {
        // shot the target when the coutner reaches zero (kinda sorta randomizes
distance)
        if(target_approach_counter == 0) goto kill_target;
        target_approach_counter--;

        // if we are directly in front of the object
        if(ir_sense(5) > .5)
        {
            kill_target:
            motors_stop();
            // if we are directly in front of the object, and have enough ammo to
shoot
            target_approach_counter = 10;
            if(ammo_remaining > 0)
            {
                // move backwards a little before we shoot in the case that we
are directly in front of the target
                if(ir_sense(5) > .5)
                {
                    motors_move("B",15);
                    ms_sleep(1000);
                    motors_stop();
                }

                // shooting procedure!=====
                main_gun_on();
                ms_sleep(2500);
                // pull trigger (shoot!)
                TRIGGER_MOTOR = S_CENTER + 500;
                ms_sleep(750);
                main_gun_off();
                // push trigger forward (no shoot!)
                TRIGGER_MOTOR = S_CENTER - 525;
                // =====
                ms_sleep(3000);
                ammo_remaining--;

                // spin in a random direction
                if(TCNT1L & 0x01) {motors_move("L",20);}
                else {motors_move("R",20);}

                // display the scanning message

```

```

        lcd_clear_display();
        lcd_goto_line(1); lcd_put_string("Scanning battlezone");
        lcd_goto_line(2); lcd_put_string(" for additional");
        lcd_goto_line(3); lcd_put_string(" targets...");

        // search for a target as you spin, to find next target
        int k;
        for(k = 30 ; k > 0; k--)
        {
            lcd_goto_pos(4,19); sprintf(message,"%i",k/8);
            lcd_put_string(message);
            ms_sleep(125);
            CMU_TrackColor(100, 255, 0, 30, 0, 20);
            if(binary2int(CMUResponseBuffer[1]) != 0) goto start;
        }

        // if we are directly in front of the object, and don't have enough
ammo to shoot
    }else{
        lcd_clear_display();
        lcd_goto_line(1);lcd_put_string("Ramming speed!");
        motors_move("F",50);
        int z = 0;
        for(; z<100 ; z++)
        {
            // if we have detected the edge, we go back,
            if(photoreflector("R",5)==1) right_edge_hit =
and continue normal behavior
            if(photoreflector("L",5)==1) left_edge_hit = 1;
            if(right_edge_hit && left_edge_hit)
            {
                ms_sleep(300);
                motors_move("B",30);
                ms_sleep(2000);

                // move in a random direction
                if(TCNT1L & 0x01) {motors_move("L",30);
ms_sleep(1500);}
                else
                {motors_move("R",30); ms_sleep(1500);}

                right_edge_hit = left_edge_hit = 0;
                goto start;
            }
        }

        // if we came out of the while loop and didnt detect an
edge, then we back up and
        // center ourselves with the target again before we
        continue pushing
        motors_move("B",40);
        ms_sleep(1500);
        continue;
    }

    // if we are not close enough to shoot the object
    }
    else
    {
        motors_move("F",30);
        continue;
    }
}

do_sensors:
// *****
// Line detection and display code (Photoreflectors) *****
// *****

// Check right photoreflector
lcd_goto_line(1);

```

```

if(photoreflector("R",5)==1)
{
    motors_move("B",30); ms_sleep(1750);
    motors_move("L",30);
    while(photoreflector("R",5)==1);
        if(right_PR_turn > 0)
        {
            // search for a target as you spin, to find next target
            int k;
            for(k = 15 ; k > 0; k--)
            {
                ms_sleep(125);
                CMU_TrackColor(100, 255, 0, 30, 0, 20);
                if(binary2int(CMUResponseBuffer[1]) != 0){right_PR_turn = 8;
motors_stop(); goto start;}
            }
        }
    else
    {
        // search for a target as you spin, to find next target
        int k;
        for(k = 6 ; k > 0; k--)
        {
            ms_sleep(125);
            CMU_TrackColor(100, 255, 0, 30, 0, 20);
            if(binary2int(CMUResponseBuffer[1]) != 0){right_PR_turn = 8;
motors_stop(); goto start;}
        }
    }
}

// Check left photoreflector
lcd_goto_line(1);
if(photoreflector("L",5)==1)
{
    motors_move("B",30); ms_sleep(1750);
    motors_move("R",30);
    while(photoreflector("L",5)==1);
    if(left_PR_turn > 0)
    {
        // search for a target as you spin, to find next target
        int k;
        for(k = 15 ; k > 0; k--)
        {
            ms_sleep(125);
            CMU_TrackColor(100, 255, 0, 30, 0, 20);
            if(binary2int(CMUResponseBuffer[1]) != 0){left_PR_turn = 8;
motors_stop(); goto start;}
        }
    }
    else
    {
        // search for a target as you spin, to find next target
        int k;
        for(k = 6 ; k > 0; k--)
        {
            ms_sleep(125);
            CMU_TrackColor(100, 255, 0, 30, 0, 20);
            if(binary2int(CMUResponseBuffer[1]) != 0){left_PR_turn = 8;
motors_stop(); goto start;}
        }
    }
}

// code to prevent gettin stuck in corners
if(right_PR_turn > 0) right_PR_turn--;
if(left_PR_turn > 0) left_PR_turn--;

// *****
// Forward obstacle detection (IR Proximity sensor) *****
// *****

```

```

// If the IR sensors in front go off, there is an object directly ahead, so move
backwards and turn
if(ir_sense(5) > .5)
{
    motors_move("B",30);
    while(ir_sense(5) > .5);
    ms_sleep(200);

    // move in a random direction
    if(TCNT1L & 0x01) {motors_move("L",30); ms_sleep(1500);}
    else                {motors_move("R",30); ms_sleep(1500);}

    continue;
}
else{lcd_goto_pos(3,1); lcd_put_string("                ");}

// *****
// Side obstacle detection (SRF04 Ultrasonic) *****
// *****

// get SRF data
left_dist = SRF04(1,1);
right_dist = SRF04(2,1);

// If we find ourselves in a tight space, we turn in a random direction until we can
move again
if(left_dist<TOLERANCE && right_dist<TOLERANCE)
{
    if(TCNT1L & 0x01) {motors_move("L",30); ms_sleep(1500);}
    else                {motors_move("R",30); ms_sleep(1500);}
    motors_move("L",35);
    while(SRF04(1,1)<TOLERANCE && SRF04(2,1)<TOLERANCE);
    continue;
}

// If there is an object on the left, we turn right until we clear it
if(left_dist<TOLERANCE)
{
    motors_move("R",30);
    while(SRF04(1,1)<TOLERANCE);
    continue;
}

// If there is an object on our right, we turn left until we clear it
if(right_dist<TOLERANCE)
{
    motors_move("L",30);
    while(SRF04(2,1)<TOLERANCE);
    continue;
}

motors_move("F",50);

}
return 0;
}

```