Master Chief A

Final Report

Justin Dickinson

University of Florida

Department of Electrical and Computer Engineering

EEL 5666 – IMDL Spring 2009

Intelligent Machines Design Laboratory

Table of Contents

Abstract

Master Chief A is an autonomous robot built to seek and destroy another autonomous robot. Infrared sensors combined with a bump senor array, allows Master Chief to avoid walls while still seeking out the opposition. To locate the other robot, a CMU camera 1 is employed to track enemy robots based on color. Built in firmware of the CMU camera allows Master Chief A to not only track colors, but also use a function called middle mass mode. This allows for a connected servo to keep a tracked object in the center of the lenses view. Middle mass mode in conjunction with track color mode allows Master Chief A's turret to aim and engage the enemy.

Executive Summary

This project report illustrates all of the systems information about the Master Chief A autonomous robot created for the Intelligent Machines Design Laboratory (IMDL) class during the Spring 2009 semester. The professors in charge of IMDL are Dr. Arroyo and Dr. Schwartz. This semesters' TAs were Mike Pridgen and Thomas Vermeer.

Master Chief A is designed to be an autonomous sentry platform with the purpose of locating and firing upon another automated robot. The Master Chief A will patrol an area while using it's infrared, and bump sensors to perform obstacle avoidance. The CMU camera 1 will allow Master Chief A to locate the enemy robot and shoot at it with a fully custom built NERF dart launcher.

Once Master Chief A begins patrolling the target zone, if contact is made with the other enemy force, he will begin to aim at and track the target. If the target remains in the cross hairs for a period of time a burst of NERF darts will be fired. Both Master Chief A and it's enemies will have an onboard LED array for targeting purposes. If the enemy robot or Master Chief A are hit by a subsequent NERF dart, then the LED array will be shut off as a recognition of its defeat.

The main processing power of the Master Chief A robot is the PVR microcontroller board which was constructed by the TA's. This board uses an Atmel ATmega128 8-bit microcontroller programmable in C using the WinAVR GCC compiler and AVRStudio. All of the code for this project can be found in the appendix at the end of this report.

Introduction

Master Chief A was constructed for the real purpose of being battle-bot, built in conjunction with a fellow class mate. Initial designs included both paintball and air soft guns. As these are not allowed on campus, demoing such a robot would be exceptionally difficult. Instead, a NERF dart launching system was designed. Modeled after a football launcher, two counter rotating wheels provide plenty of force to propel the NERF dart a maximum of nearly 30 feet.

Another major design aspect of Master Chief A was the ability to act autonomously. I wanted the entire system to be a set it, and forget it vehicle. While patrolling for the enemy robot, Master Chief A needed to avoid obstacles so progress would not be hindered. This fell to a combination of IR and bump sensors. An infrared sensor array would allow the robot to view any obstacles on both sides, in front of, and behind the robot. If an object was not detected by the IR sensors, the bump sensor array will collide with the unseen object. This fail safe will randomly turn the robot upon collision.

The final subsystem of Master Chief A is the targeting and aiming system. This is firmly based on the CMU camera 1. Built in color tracking and middle mass modes allow the camera to communicate with the robot and adjust to aim at the enemy robot. A servo driven slip ring turret design allows for the turret to rotate in a 360 degree fashion. This allows the Master Chief A robot to truly act as an autonomous tank robot, much like automated weapons platforms in service today.

Integrated System

The integrated system of Master Chief A is based on a series of interrupts driven by the PVR microcontroller board that was built by TA's Mike and Thomas. The microcontroller acts as the brain of Master Chief by arbitrating behaviors based on the given data. This data is provided by an infrared sensor array, a bump sensor array, and a CMU 1 camera. These sensors are described in further detail later in the report. Retrieving this data allows the microcontroller's software to interpret and respond to the environment around Master Chief A.

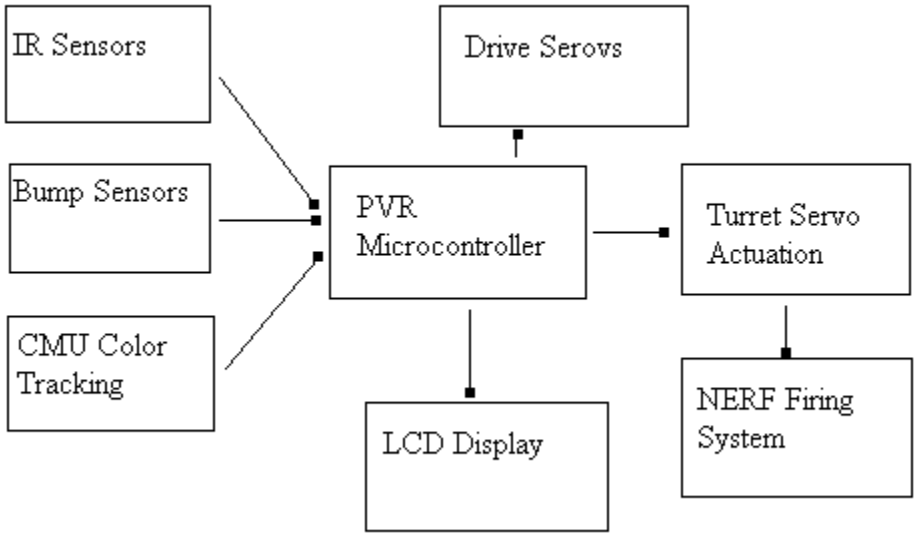

Figure 1: Subsystem Block Diagram

Mobile Platform

Master Chief A's platform was first modeled in Solid Works and then constructed with the 1/8th inch wood provided in lab. The IR sensor array includes 3 forward looking and 1 rear facing IR sensor. The bump switches are connected around the perimeter of the robot and connected to one another with fastened wood strut supports. The CMU camera 1 is mounted to the turret and allows for complete rotation for tracking enemy robots. Below is a Solid Works diagram of the preliminary platform design. Also shown below is a picture of the completed robot.

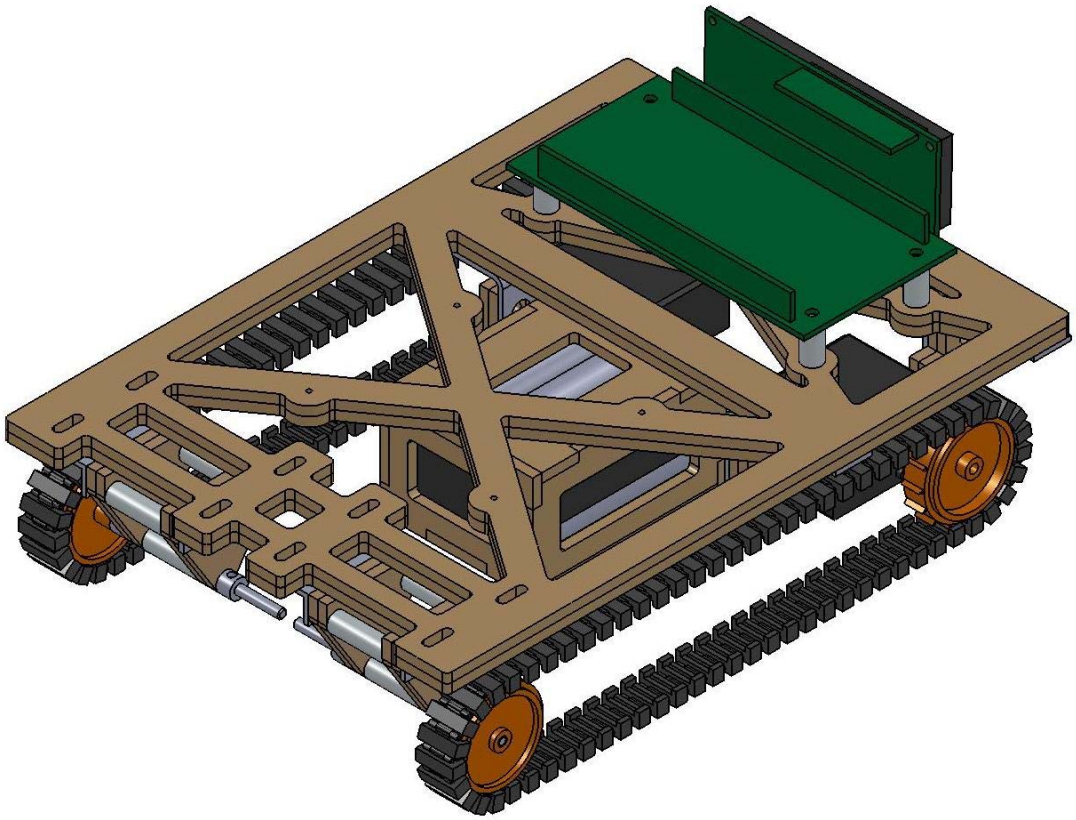

Figure 2: Solid Works Drawing of Preliminary Platform Design

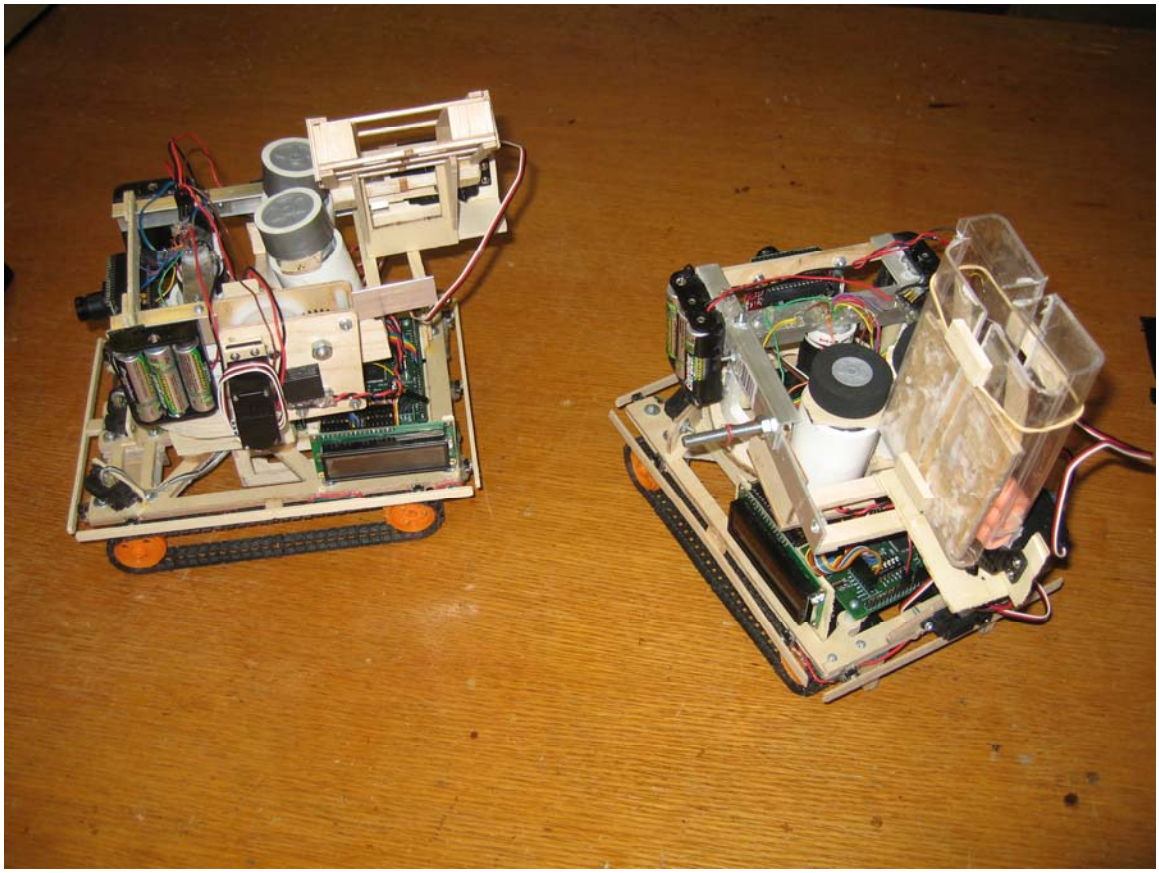

Figure 3: Completed Robot Platforms

Sensors

Bump Sensors

The bump switch mechanism on the robot will consist of a micro switch with an extended trigger mounted on the perimeter to aid in obstacle avoidance. The bump sensor will cause the robot to cut power to the servos and change direction in a random fashion. The goal for implementing these sensors will be to mount them and to program a stop command to the servo drive motors. This stop will include the random directional behavior and then allow for the obstacle avoidance program to continue as usual. These bump sensors will hopefully only be triggered as a last resort when the sharp IR sensors fail to detect an obstacle.

For the construction and setup of these sensors, eight total bump switches will be used (2 on every side of the rectangular platform). Each bump switch is also run through a unique resistor. When the switches are triggered for an entire side, a unique resistance will be read by the micro controller. This will allow for directionality to the reading while also only taking up one port of the micro controller. A certain voltage reading will be indicative to a front, rear, or either side collision. An example of the wiring setup is shown in figure 1 below.

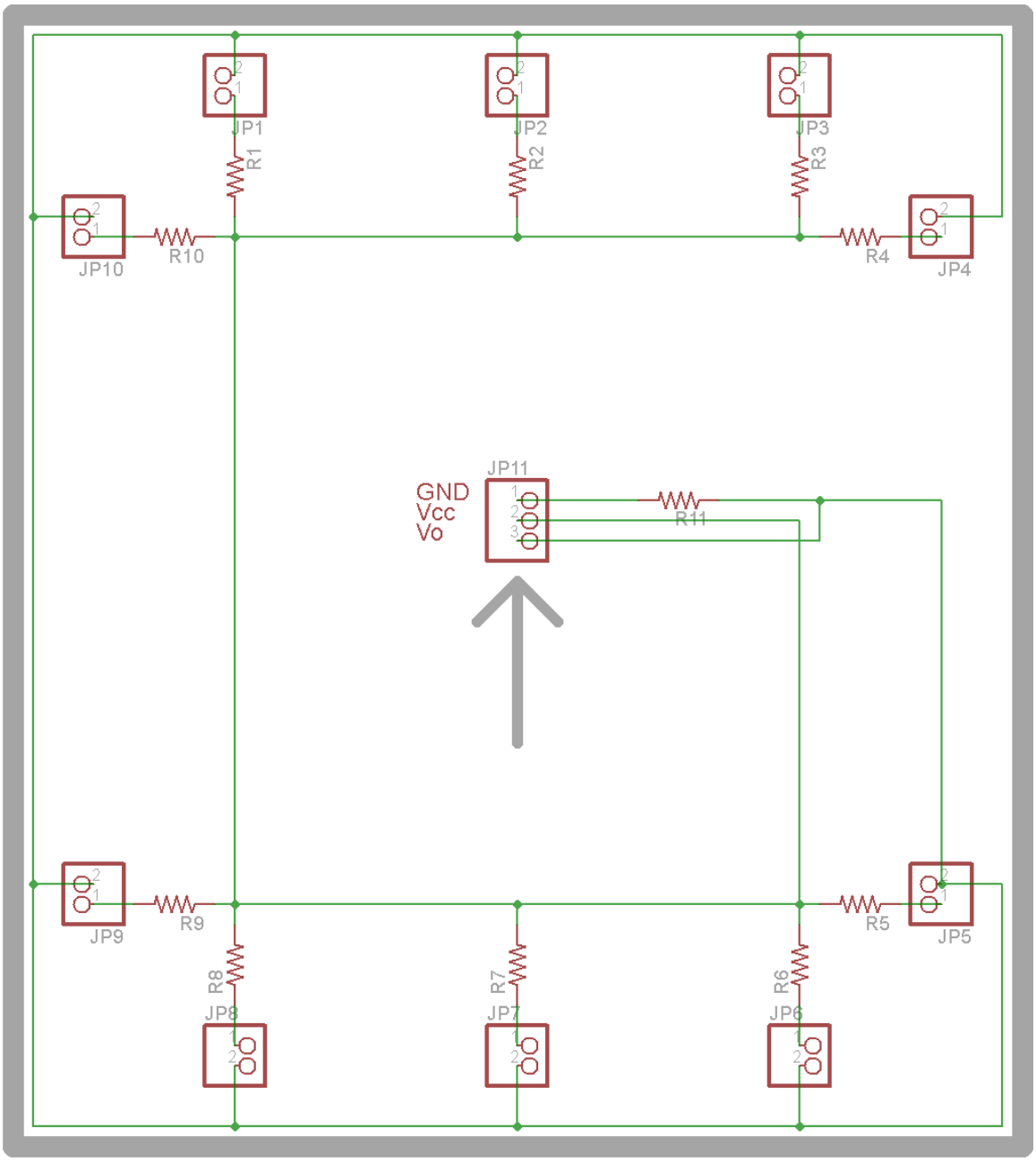

Figure 4: Wiring diagram for bump sensor array

IR Sensors

The Sharp IR sensor is a popular distance sensor that has a unique way of finding ranges with IR light. The sensor sends out a pulse of IR light into the environment. If it hits an object, the light reflects back to the sensor. When the light returns to the sensor, it arrives at an angle dependant of the distance it reflected back from. The sensor has an included integrated circuit which provides an analog value corresponding to the range it finds. The infrared proximity sensor made by Sharp, I chose the model GP2D120XJ00F, has an analog output that varies from 3.1V at 3cm to 0.3V at 40cm with a supply voltage between 4.5 and 5.5VDC. Corresponding voltages to detected distances are shown in table 1 and figure 2 below. The detected values compared to distance does vary linearly, but can be accurately predicted across all four sharp IR sensors used for this robot.\

CMU Camera

The CMU 1 Robot Vision Cam will be my special sensor used to find the enemy robot using vision processing. This device is designed to return high level vision capabilities to microprocessor boards that normally would not have enough processing power to execute such abilities. The camera to will operate through the microcontroller by sending serial commands to the CMU camera, and the preprogrammed firmware on the camera will then use the Track Color command and Middle Mass mode to find the target robot. The microcontroller will then send commands to the servos to drive the robot towards its colored target and then engage the firing mechanism. The Middle Mass function of the CMU camera will be vital in accurately targeting the enemy robot. By focusing on the

centroid of the object, aiming is done almost entirely by the built in software of the CMU camera.

Behaviors

The programmable behaviors for the Master Chief Robot platform are a complex network work of individual behaviors working together as one. These are broken up into obstacle avoidance, target acquisition and firing sequence.

Obstacle avoidance is the most basic of these behaviors. By taking in data from IR sensors, if the value reads too high, Master Chief will move in the opposite direction of the obstacle. This is coupled with the bump sensor array. Any obstacle not picked up by the IR sensors will trigger a bump sensor when a collision is detected. Depending on which bump switch is triggered a unique voltage will be passed based on the resistor attached to the switch. This gives directionality to the collision and allows Master Chief to back away from and turn away from it.

The target acquisition is done through the CMU camera through a middle mass mode and color tracker. The color tracking mode picks up a desired color based on it red, green, blue make up. The middle mass mode keeps this tracked color in the center of the lens of the camera. By sweeping the area in front of the robot the camera will eventually pick up the desired color on the opposing robot. When this is done the firing sequence is triggered.

When a positive reading from the CMU camera is read as true, the firing sequence spools up the two counter rotating DC motors and cocks back the loading servo. After

giving enough time for the DC motors to get up to speed, the loading servo moves forward and back three times to send a burst of darts at the enemy target.

Experimental Layout and Results

Values read from the infrared sensors have the most need for experimental data to be drawn from. When the IR sensor sends out an infrared light, if an object is detected, the infrared signal is sent back to the sensor. How strong the signal is returned to the sensor hub is directly proportional to the distance the object is away from the robot. A set of values was needed to be experimentally determined to come up with a threshold for the IR sensors. The data below represents the voltage received based on distance of the object detected. By raising the threshold of the sensors the robot will be considered more aggressive, while setting the threshold lower will make the robot more conservative.

Table 1: Output voltages compared to detected distances

Distance (in)	Value
0	158
1	140
2	119
3	86
4	65
5	56
6	45
7	40
8	34
9	29
10	27
11	25
12	23

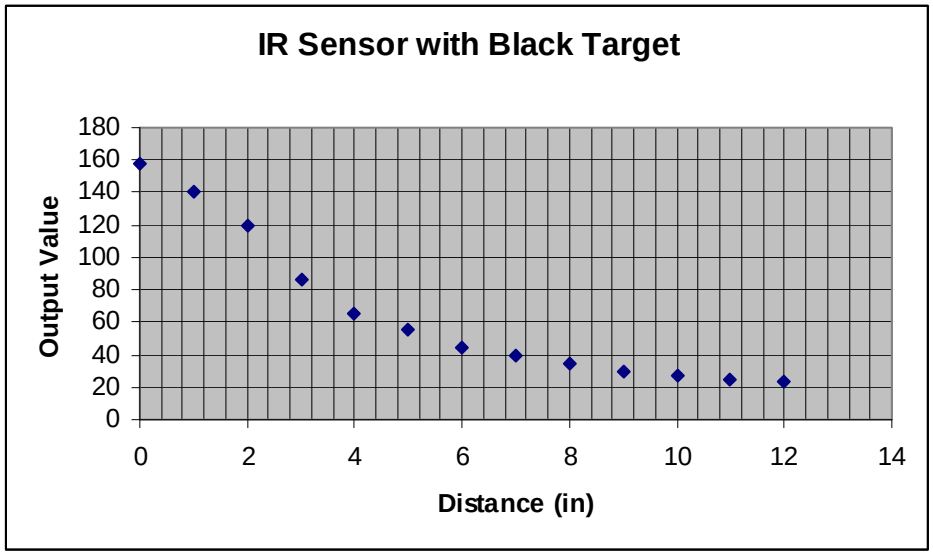

Figure 5: Output voltages corresponding to detected distances

Conclusion

In conclusion, the Master Chief A has been a great success. I have accomplished all of my goals of achieving obstacle avoidance, target acquisition and a custom made NERF firing launcher. I have learned so much through this class in the field of robotics and electronics that I never thought would be possible. I honestly could not have done it without my teammate and competition, Taylor Lusk. Both TA's Mike and Thomas were so incredibly helpful and understanding it still baffles me. Without their weekly input, critical design changes would not have been made. And last, but certainly not least, Dr. Schwartz and Dr. Arroyo have inspired me like no other professor has. They cultivated a new passion and hobby for me in the field of electronics and engineering. Going from a simple mechanical engineer who had never even read a wiring diagram, to someone who completed an autonomous vehicle with success, is unbelievable to me. Though times did get tough, and I was frustrated beyond belief at points, the payoff was completely and utterly worth all of the hard work. I would now like to say thanks you to all the previously mentioned people. I really could not have done it without you.

Appendix

Code in its entirety

```
include <avr/io.h>
#include <avr/interrupt.h>
#include "PVR_Servos.h"
#include "sleep.h"
#include "LCD.h"
#include "ADC.h"
#include "uart.h"

#ifndef F_CPU
#define F_CPU 16000000UL          // Atmega128 runs at 16mHz
#endif
#define UART_BAUD_RATE        19200  // Baud rate

void receive(void);
void shoot(void);
void track(void);

int data[8], pix, con, shotCnt = 0, increment = 0, found = 0, lock = 0, count = 0, TC = 0;

int main(void)
{
        int i, j = 0, bump = 0, IR[4], p = 0b00000000, up = 1, found = 0;
        int loc[4] = {0, 0, 0, 0};
        char c = '\n';

        // Initialization functions
        initSleep();                        // Sleep
        lcdInit();                                // LCD display
        initServo();                        // Servos
        initADC();                                // A/D channels
        DDRA = 0b00001111;                        // Port A (for A/D Mux) - First four
bits to ouput
        ms_sleep(100);
        uart_init( UART_BAUD_SELECT( UART_BAUD_RATE, F_CPU ) );
        sei();
        lcdString("Justin The Conqueror");

        lcdClear();
        uart_puts("PM 1 \r");
        ms_sleep(500);
        do {
                c = uart_getc();
```

```
} while (c != ':');
ms_sleep(1000);

lcdClear();
uart_puts("SW 1 1 80 143 \r");
ms_sleep(500);
do {
        c = uart_getc();
} while (c != ':');
ms_sleep(1000);

lcdClear();
uart_puts("MM 1 \r");
ms_sleep(500);
do {
        c = uart_getc();
} while (c != ':');
ms_sleep(1000);
// End initialization

moveServo4(-70);

while(shotCnt < 10)
{
        moveServo1(-100);
        moveServo2(100);
        moveServo3(increment);
        ms_sleep(50);
        for (i = 0; i < 4; i++)
        {
                PORTA = p;
                p = p + 2;
                IR[i] = adcOne();
                if (IR[i] > 80) loc[i] = 1;
        }
        if (IR[0] > 80 || IR[1] > 80 || IR[2] > 80 || IR[3] > 80)
        {
                moveServo1(0);
                moveServo2(0);
                moveServo3(0);
                ms_sleep(25);
                IRAvoid(loc, IR);
                loc[0] = 0;
                loc[1] = 0;
                loc[2] = 0;
                loc[3] = 0;
```

```
                }
                bump = adcZero();
                if (bump > 20)
                {
                        moveServo1(0);
                        moveServo2(0);
                        moveServo3(0);
                        ms_sleep(100);
                        bumpAvoid(bump);
                        lcdClear();
                }
                if (!TC) uart_puts("TC 155 255 0 32 0 32 \r");
                ms_sleep(50);

                receive();
                if (data[0] > 0)
                {
                        found = 1;
                        moveServo1(0);
                        moveServo2(0);
                        lcdClear();
                        lcdString("Found");
                        ms_sleep(100);
                        while (found && shotCnt < 5)
                        {
        //                      track();
                                shoot();
                        }
                }
                if (increment < 100 && up == 1 && !found) increment = increment + 10;
                if (increment > -100 && up == 0 && !found) increment = increment - 10;
                if (increment > 99) up = 0;
                if (increment < -99) up = 1;
        }
        return 0;
}

void receive(void)
{
        int i, x = 1;
        char temp[3], c = '\n';
        if (!TC)
        {
                while (c != 'A' && c != 'N') c = uart_getc();
                if (c == 'N') {
                        while (c != ':') c = uart_getc();
```

```
                        lcdString("Command failed");
                        ms_sleep(1000);
                        lcdClear();
                }
                TC = 1;
        }
        do {
                c = uart_getc();
                lcdChar(c);
        } while (c != 'M');
        c = uart_getc();
        i = 0;
        temp[0] = uart_getc();
        temp[1] = uart_getc();
        if ((temp[1] >= 0x30) && (temp[1] <= 0x39))
        {
                x = 2;
                temp[2] = uart_getc();
                if (temp[2] >= 0x30 && temp[2] <= 0x39)
                {
                        data[0] = 100 * (temp[0] - 0x30) + 10 * (temp[1] - 0x30) +
(temp[2] - 0x30);
                }
                else data[0] = 10 * (temp[0] - 0x30) + (temp[1] - 0x30);
        }
        else data[0] = temp[0] - 0x30;
        lcdInt(data[0]);
        ms_sleep(250);
        lcdClear();
}

/*void track(void)
{
        if (data[0] > 32 && data[0] < 48) count++;   // Center turret on x = 40
        else count = 0;
        if (count == 5) lock = 1;         // If held on target for ~4 seconds lock = true
        if (!lock)
        {
                if (data[0] < 36) increment = increment + 5;
                if (data[0] > 44) increment = increment - 5;
                moveServo3(increment);
                ms_sleep(50);
                receive();
//              if ((data[0] < 10) || (data[0] > 70)) found = 0;// If color lost return to search
loop
                if (data[0] == 0) found = 0;
```

```
	}
	if (lock && shotCnt < 10)
	{
		lcdClear();
		lcdString("Lock");
		shoot();
		receive();
		if (data[0] < 10 || data[0] > 70)
		{
			found = 0;
			lock = 0;
		}
		if (data[0] < 33 || data[0] > 47) lock = 0;
		if (data[0] < 36) increment = increment + 2.5;
		if (data[0] > 44) increment = increment - 2.5;
		moveServo3(increment);
		ms_sleep(50);
	}

}*/

void shoot(void)
{
	DDRE = 0b11111111;
	shotCnt++;
	PORTE = 0b11100000;
	moveServo4(80);
	ms_sleep(1000);
	PORTE = 0b00000000;
	moveServo4(-70);
	ms_sleep(1000);
	DDRE = 0b00100000;
}
```