University of Florida
College of Engineering
ECE Dept.

Course: **EEL 5666C**

Instructors: **Dr. Schwartz**
Dr. Arroyo

TA's: **Mike Pridgen**
Thomas Vermeer

Sensor Report Brian Long's FanBot

Dec 8, 2009

Table of Contents

1. Abstract

FanBot is an alligator shaped robot made out of balsa wood. Naturally, it is a fan of the Gators. As such, it is not a fan of Alabama. If it sees an Alabama fan, it will go up to them and attack by biting. If it sees a fellow Gator, it will go up to them and play the Gator Fight song.

It uses 7 IR sensors for obstacle avoidance. It also has a SONAR sensor for ranging and a CMU Cam for color blob detection. By using these sensors, it can distinguish between targets and navigate towards them.

FanBot also has an Arduino board with an Adafruit Waveshield for making sound. It has 3 servos to actuate the head, tail, and jaw. Finally, Fanbot has 4 motors for propulsion.

Taken together, these components have made an entertaining and successful robot.

2. Executive Summary

FanBot seeks out targets and identify which SEC team they support. After navigating towards the target, it will play various sounds depending on the team allegiance identified. It will then seek out a new target.

FanBot utilizes the PVR board developed by Mike Pridgen and Thomas Vermeer. This board features an Atmel Xmega 128 microcontroller. One advantage of the PVR board is that it has an onboard voltage regulator for the Xmega and two 5V regulators for servos and A/D conversion. The other advantage is the well thought out pinout of the PVR board and the software library provided by Mike and Thomas. There are more than enough available ports to support serial communication, A/D conversion, Digital I/O, and PWM. This gives the student a lot of freedom in design and development of the robot.

FanBot uses a CMU Cam for color blob recognition. The two colors FanBot will look for is green for a Gator fan and red for a Crimson Tide fan. The CMU cam is used to distinguish targets and to line up on the target while it is approaching it. The PVR board communicates with the CMU Cam using RS-232 serial communication.

The other sensors Fanbot has includes seven IR sensors and one SONAR sensor. There is one IR sensor on each side and three in the front on the snout. There is also an IR sensor to detect low lying targets and one sensor on the tail used when backing up. All seven IR sensors are used for obstacle avoidance. The two side IR sensors will wake up FanBot from the wait state. The front three IR sensors also let FanBot know that it has reached the target. The SONAR is used for ranging.

FanBot has three servos. Two high torque servos actuate the tail and the head. A third servo actuates the jaw. The other actuators include four propulsion motors, two on each side. These motors are extremely high torque and are run at 40 – 50% of top speed. This allows FanBot to find more readily find targets and avoid obstacles.

FanBot uses an Arduino board with an Adafruit Wave Shield for sound. The Wave Shield will uses an SD card with .wav files to produce sound. The PVR board communicates with the Arduino board using parallel communication. The bit sequence detected by the Arduino is used to select which file to play. Two battery powered Sony speakers are used to play the sound.

3. Introduction

FanBot was a project to create a fun robot that shows off Gator pride. Creating a nearly life size alligator was challenging, but was the best was the best way I knew of to integrate engineering skills with being a Gator fan.

I was determined to make FanBot as lifelike as possible. Having the jaw, head, and tail actuate using servos was a key factor in accomplishing this.

This report will detail the various systems in FanBot including sensors, actuators, and platform. It will also discuss the behaviors. Finally, it will detail the lessons learned so that other students can learn from my mistakes and observations.

4. Integrated System

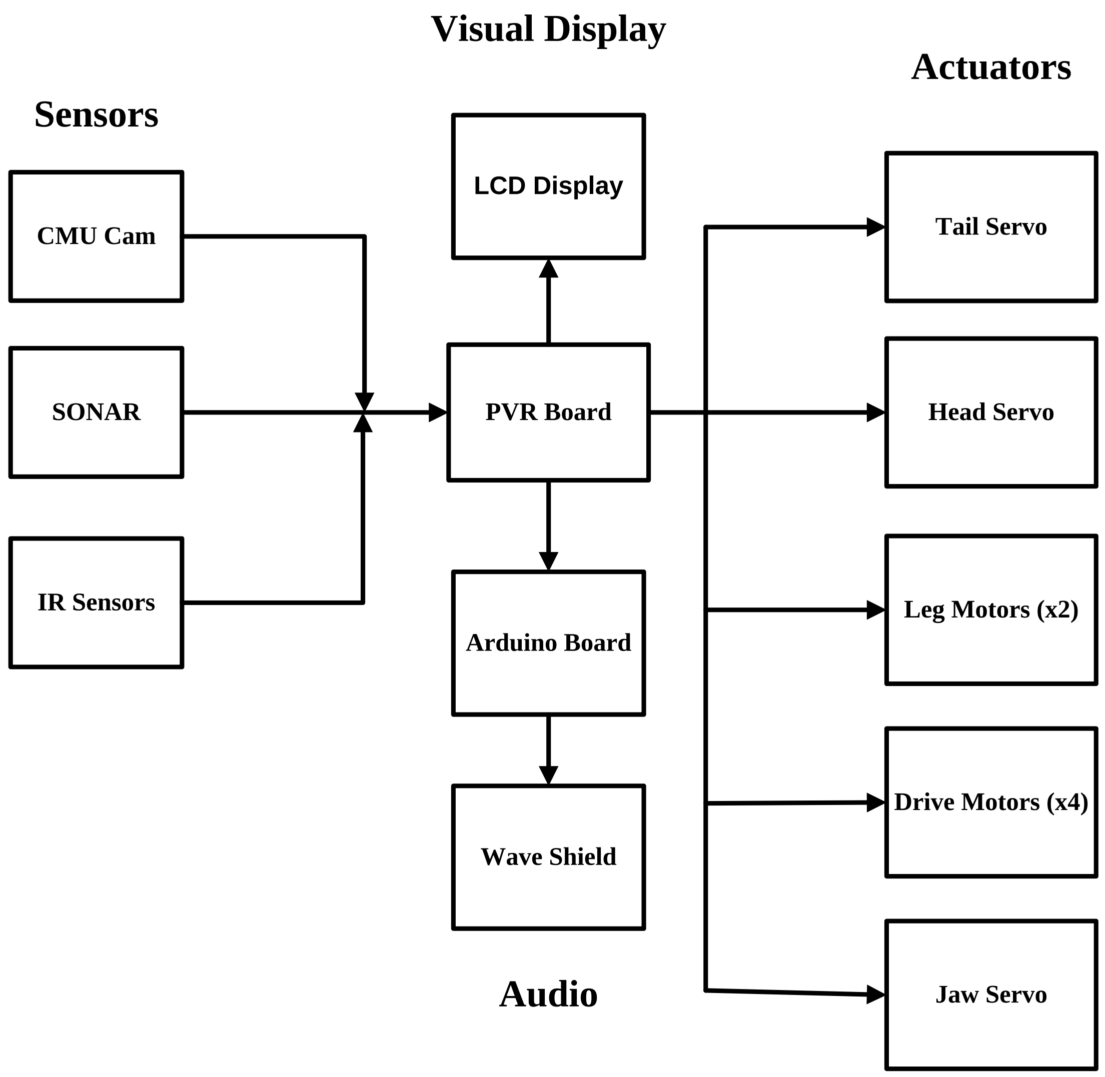

Figure 1 – Integrated Block Diagram

The microcontroller board used for FanBot is the Atmel XMega based PVR board designed by Mike Pridgen and Thomas Vermeer. This is an excellent and well thought out board for any future student to use. The pinout of the board gives enough serial communication ports, Digital I/O pins, A/D connections, and PWM connections to support a wide variety of possible projects.

The sensors guides FanBot toward a target, distinguishes a Gator fans from other ones, and for obstacle avoidance. They consist of the following:

1. Two Sharp GP2Y0A21YK IR sensor are mounted on the each side of FanBot for obstacle avoidance. These have a maximum range of about 18 inches and an effective range of about 9 – 10 inches.
2. One Sharp GPY0A21YK IR sensor is mounted at the bottom of the throat to detect small obstacles in front of FanBot.
3. Three Sharp GP2Y0A02YK0F sensors on the snout for detecting obstacles in the front of FanBot. They have a maximum range of 36 inches and an effective range about 24 inches.
4. One EV-LZ0 SONAR sensor is mounted on the front of FanBot for ranging and target detecting. This sensors has an effective range of about 12 feet.
5. The CMU camera is mounted on the head for target detection and distinguishing between the teams.

The LCD display is used to troubleshooting and for functionality checks.

The Arduino board drives the Wave Shield for sound production. A battery powered speaker produces the sound.

The jaw servo actuates through coordination of the PVR board and the Arduino board using parallel communication. Proper timing is done through using a series of delay statements in the code.

The propulsion system consists of two DC planetary gear motors per side. The left side motors are driven by one motor driver and the right side motors are driven by a second motor driver. This allows the control of each side to be synchronized for propulsion and turning.

Two high torque standard size servos will be used to move the head and tail from side to side to give a natural motion. They drive a turntable assembly for torque transfer.

Two motors drive offset cams on the rear and on the front of FanBot. These motors were supposed to actuate four “legs”. This was to be for appearance only and not for propulsion. The motors could not provide enough torque to actuate the legs. Mechanically the design works. If I had more time, I would have implemented the design using four servos instead of the two motors.

5. Mobile Platform

The platform is made of balsa wood to reduce the weight. The platform consists of four sections: the body, the connector pieces, the head, and the tail. The goal was to make a lifelike alligator.

The body is a box. It is approximately 14” long, 12 inches wide, and 4 inches tall. The top is hinged to allow access to components. The body houses the propulsion system, the leg system, and all major electronics. A removable tray mounts the electronics.

The head is 14 inches long and consists of a neck that extends out from the body and the head itself which will be slightly raised up. The CMU is mounted on top of the head and the SONAR sensor is mounted on the snout. The three GP2Y0A02YK0F IR sensors are mounted on the snout. The GPY0A21YK is mounted on the throat. The throat will also house the Wave Shield and the speakers. A caster is on the bottom of the neck for support and to reduce the torque on the servo.

The tail will be about 14 inches long and consists of two parts. The back part will connect to the front part with via two hinges. This allows a more realistic motion. Each section has casters underneath for support. The tail also houses the rear IR sensor.

I did not the parts into Mike early enough. As a result, I was severely hampered in building and integrating behaviors. I would recommend that future students get any T-Tech requests in as early as possible.

6. Actuation

The propulsion system layout can be seen in Figure 2 below.

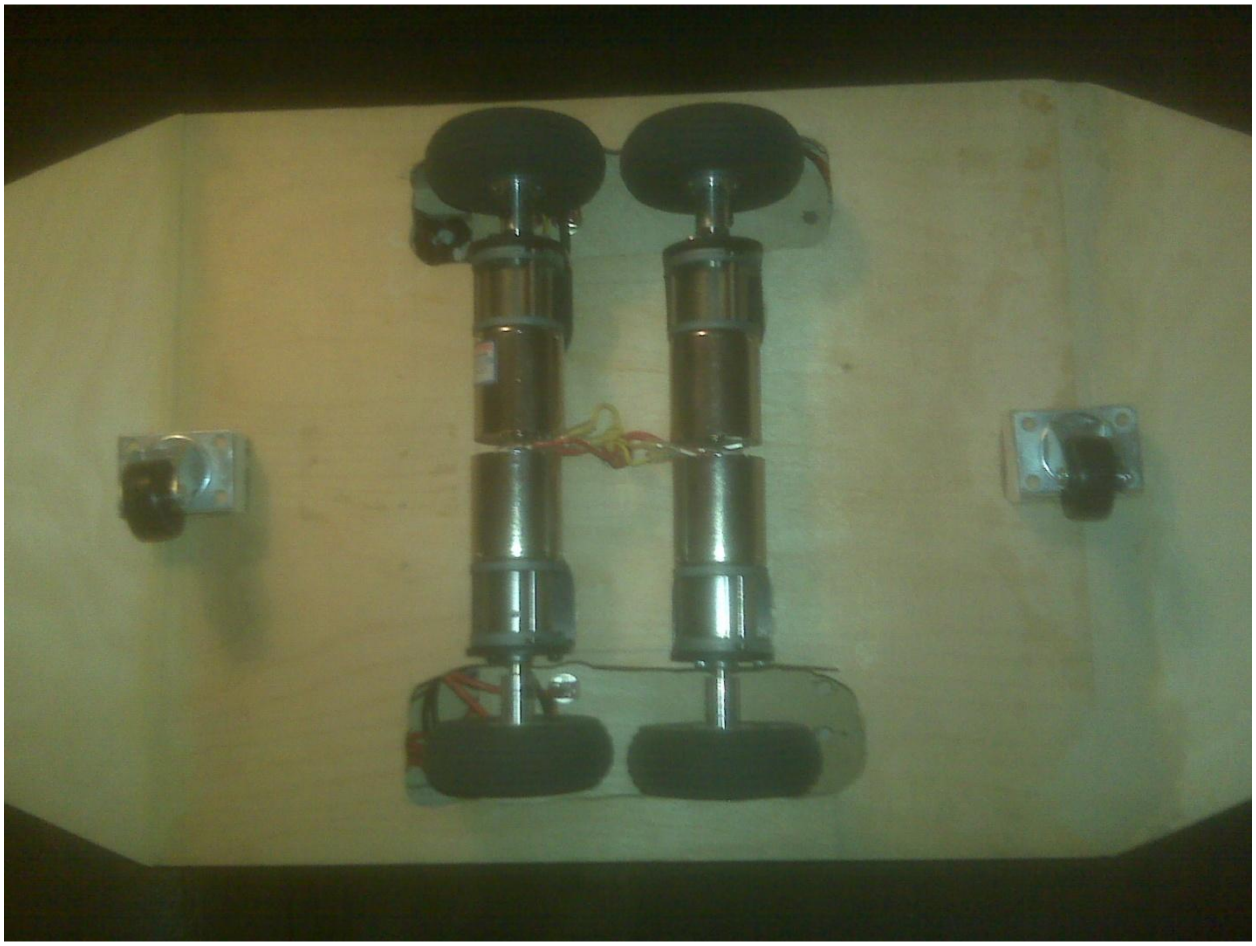

Figure 2 – Bottom of Body Section

Each motor powers one of the wheels. Each side is synchronized via a motor driver to control turning, direction, and speed of FanBot.

The leg system will consist of two parts. There are two motors that drive offset cam systems. One assembly drives the rear legs and one the front. The legs will be balsa wood cutouts and will almost touch the ground. They are for appearance and not for propulsion.

The head and tail servos will turn each from side to side while FanBot is looking for a target.

The Wave Shield and Arduino Board are both from Adafruit.com. Timothy Martin used them in his project last year and offered to provide me with any technical assistance needed. The Wave Shield can play mono sounds and outputs a signal through a stereo mini jack to battery powered speakers. The sounds are stored on an SD card. The Arduino board interfaces with the PVR board via an RS-232 connection. The Arduino drives the wave shield. The jaw servo actuates when the Wave Shield plays sound.

7. Sensors

IR Sensors:

A total of three GP2Y0A21YK IR sensors and associated pigtails were ordered from www.sparkfun.com . Two are mounted on each side and one in the low in the front for obstacle avoidance. The experimental data can be seen in Part 9.

Figure 3 – GP2Y0A21YK [1]

A total of three GP2Y0A02YK0F IR sensors and associated pigtails were also ordered from www.sparkfun.com . These are mounted on the snout for obstacle avoidance. The experimental data can be seen in Part 9.

Figure 4 – GP2Y0A02YK0F [1]

SONAR Sensors:

Two Maxbotix LV-EZ0 sensors were ordered from www.sparkfun.com. The LV-EZ0's will be used on the front and on the rear of FanBot. The front sensor is used for ranging and target detection, the rear one is used for obstacle avoidance when backing up. Originally, a LV-EZ4 was going to be used, but it was shown to too sensitive. SONAR's were also shown to be a little slow when it comes to obstacle avoidance. I made the decision to limit their use to searching for a target and for ranging. Experimental data is in Section 9.

Figure 5 – LV-EZ0 SONAR [1]

CMUcam:

A CMUcam is used for color blob detection. It uses the mean color value in its window to differentiate targets. It is connected to the PVR board via RS-232 connections. Experimental data is seen in Section 9.
It is important for future students wanting to use the CMU Cam to get started early. This was the hardest part to integrate and to code for properly. Also it is important to realize that reds, pinks, and oranges are virtually indistinguishable from each other. Do not use similar colors for any targets.

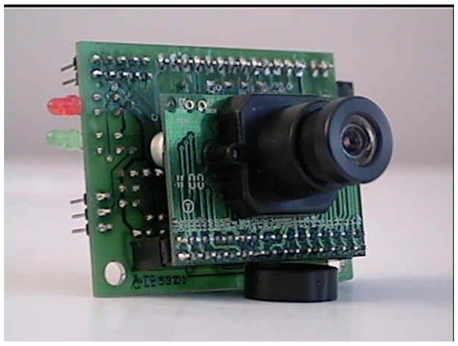
Figure 6 – CMUcam 1 [2]

8. Behaviors

FanBot will initialize and then say, "All Systems Go." It will then be in a wait state. While in the wait state, it will growl if anyone walks in front of it. It will leave the wait state when the side sensors are covered. When the side sensors are covered, the head will move from side to side and the jaw will move up and down. It will also play the Tickle Me Elmo laugh.

FanBot then goes into a search mode looking for an Alabama target. Once the target is located using the CMU Cam, it will play the Jaws theme. It will then approach the target and use the CMU Cam to make course corrections. When it gets to the target, it uses the IR sensors to determine when it is close to the target. It will then stop and the jaws will actuate with chomping sounds. It then says, "That's how I roll."

After locating the Alabama target, it will then search for a Gator target. When the target is located it will say, "Hail to the King!" It will approach the target, as above, and when it gets close it will stop and play the Gator fight song. It will then look for another Alabama target.

The code for these behaviors can be seen in Appendix B.

9. Experimental Results

a) IR Sensors

The code used to test the IR Sensors can be seen in Appendix A, Part 1. Each IR sensor was connected to the same port and tested. I took a measuring tape and recorded the analog values I got from a white and then a black target. I did that for 3 lighting conditions for each sensor including fluorescent lighting, incandescent lighting, and sunlight. I then averaged the white and black target results for each lighting condition and used curve fitting to get a best fit curve. A fourth order polynomial seemed to work best. I used this best fit curve to compare the results are as follows:

1. For the Right IR Sensor, a Sharp GP2Y0A21YK0F:

Distance (Inches)	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
2	3197	3257	3157	3183	3257	3257
4	3197	3257	3257	3257	3257	3257
6	3512	3512	3257	3257	3257	3257
8	3325	3368	3357	3097	3567	3569
10	3082	3155	3097	2544	3321	3057
12	2237	2175	2297	2137	2390	2233
14	2043	1945	2053	1963	2053	2066
16	1625	1869	1931	1625	1936	1866
18	921	1625	1803	1621	1625	1625
20	921	921	1621	921	921	921
22	921	921	921	921	921	921
24	921	921	921	921	921	921
26	921	921	921	921	921	921
28	921	921	921	921	921	921
30	921	921	921	921	921	921

Figure 7 – Right IR Sensor Results

Resulting curves after curve fitting:

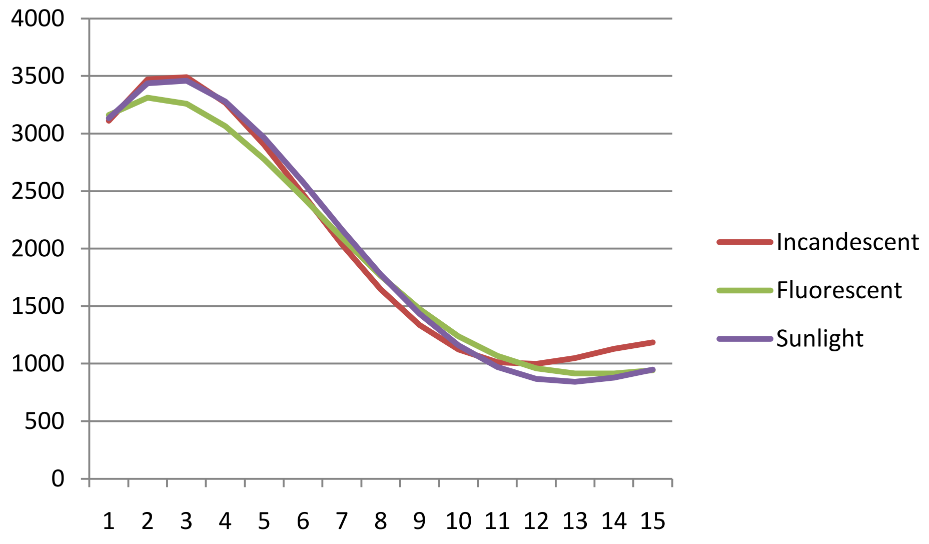

Figure 8 – Right IR Sensor Analog Value vs. Distance in Inches

2. For the Left IR Sensor, a Sharp GP2Y0A21YK0F:

Distance (Inches)	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
2	3197	3197	3197	3178	3167	3167
4	3197	3197	3197	3178	3167	3233
6	3567	3569	3578	3578	3512	3233
8	3327	3257	3369	3571	3326	3568
10	3082	3083	3165	3167	3083	3155
12	2175	2160	2458	2571	2233	2295
14	2137	1949	2161	2237	2045	2169
16	1866	1866	2160	2233	1968	2168
18	1625	1625	1625	1625	1625	1945
20	921	921	921	921	921	1621
22	921	921	921	921	921	921
24	921	921	921	921	921	921
26	921	921	921	921	921	921
28	921	921	921	921	921	921
30	921	921	921	921	921	921

Figure 9 – Left IR Sensor Results

Resulting curves after curve fitting:

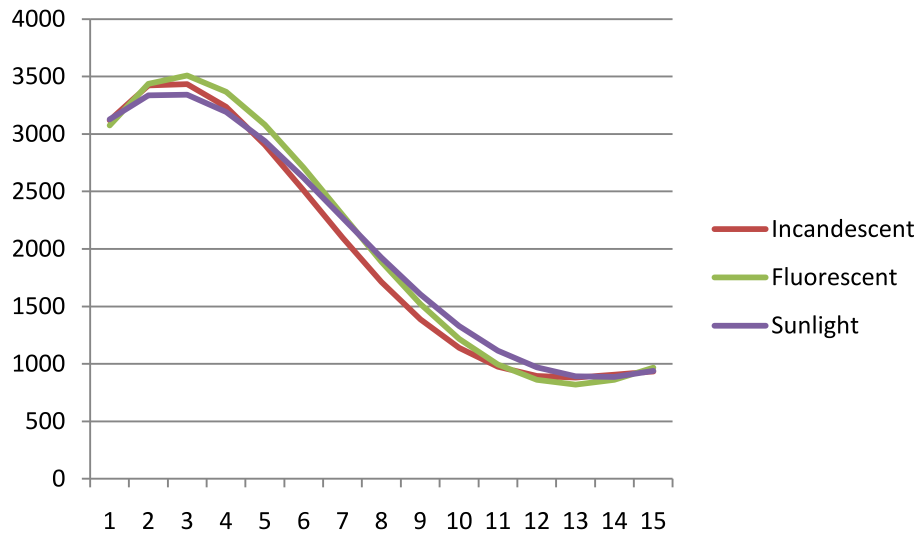

Figure 10 – Left IR Sensor Analog Value vs. Distance in Inches

3. For the Lower, Front IR Sensor, a Sharp GP2Y0A21YK0F:

Distance (Inches)	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
2	3197	3257	3157	3178	3257	3257
4	3197	3257	3257	3257	3257	3257
6	3545	3545	3257	3257	3385	3257
8	3325	3368	3357	3097	3567	3569
10	3082	3155	3097	2544	3321	3057
12	2238	2175	2297	2237	2390	2233
14	2137	1945	2053	1961	2053	2066
16	1625	1869	1931	1625	1936	1866
18	921	1625	1803	1621	1625	1625
20	921	921	1621	921	921	921
22	921	921	921	921	921	921
24	921	921	921	921	921	921
26	921	921	921	921	921	921
28	921	921	921	921	921	921
30	921	921	921	921	921	921

Figure 11 – Lower, Front IR Sensor Results

Resulting curves after curve fitting:

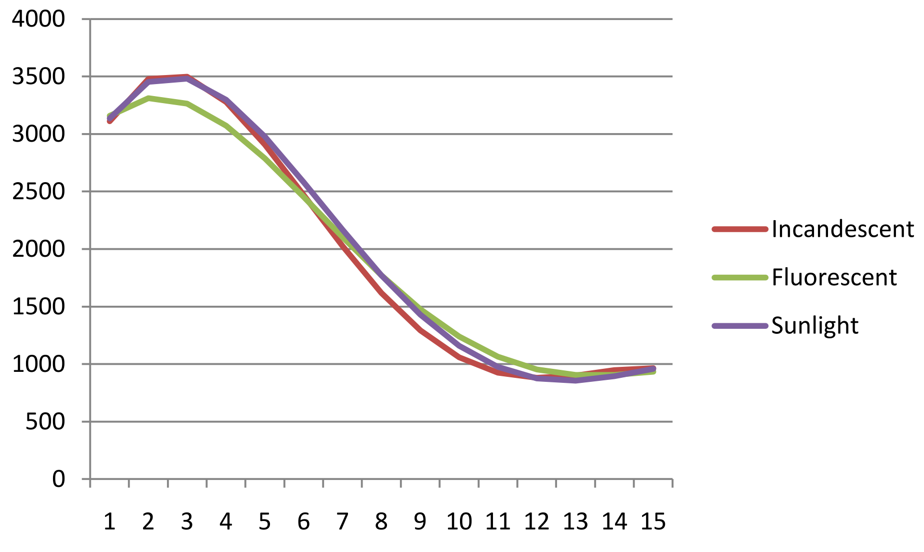

Figure 12 – Lower, Front IR Sensor Analog Value vs. Distance in Inches

4. For the Left Snout IR Sensor, a Sharp GP2Y0A02YK0F:

Distance (Inches)	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
12	3187	3197	3187	3197	3187	3197
14	3512	3512	3521	3512	3569	3512
16	3587	3587	3577	3567	3569	3567
18	3577	3581	3577	3517	3361	3391
20	3261	3192	3261	3233	3165	3263
22	3167	3097	3183	3197	3157	3097
24	3167	2544	3097	3161	2513	2537
26	2499	2458	3488	3037	2391	2335
28	2233	2369	2401	2497	2161	2153
30	2161	2237	2233	2197	2132	2047
32	2137	1943	2107	2143	1945	1941
34	1945	1866	2002	2042	1861	1625
36	1869	1930	1940	1920	1625	921
38	1731	1934	1803	1968	921	921
40	1625	921	1625	1867	921	921
42	921	921	921	921	921	921

Figure 13 – Left Snout IR Sensor Results

Resulting curves after curve fitting:

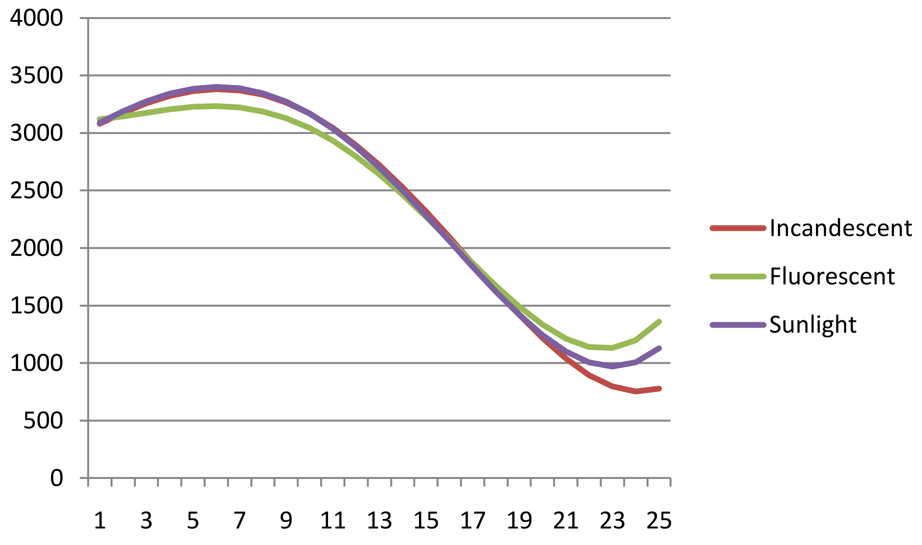

Figure 14 – Left Snout IR Sensor Analog Value vs. Distance in Inches

5. For the Center Snout IR Sensor, a Sharp GP2Y0A02YK0F:

Distance (Inches)	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
12	3187	3197	3167	3197	3187	3197
14	3512	3568	3521	3567	3569	3512
16	3587	3587	3577	3567	3569	3567
18	3577	3581	3577	3517	3361	3391
20	3261	3192	3261	3233	3165	3263
22	3167	3097	3183	3197	3157	3097
24	3161	2544	3097	3161	2513	2537
26	2538	2458	3488	3037	2391	2335
28	2233	2369	2312	2497	2161	2153
30	2161	2237	2233	2197	2132	2047
32	2137	1943	2037	2143	1945	1941
34	1945	1866	1949	2042	1861	1625
36	1869	1930	1940	1920	1625	921
38	1731	1934	1803	1968	921	921
40	1625	921	1625	1867	921	921
42	921	921	921	921	921	921

Figure 15 – Center Snout IR Sensor Results

Resulting curves after curve fitting:

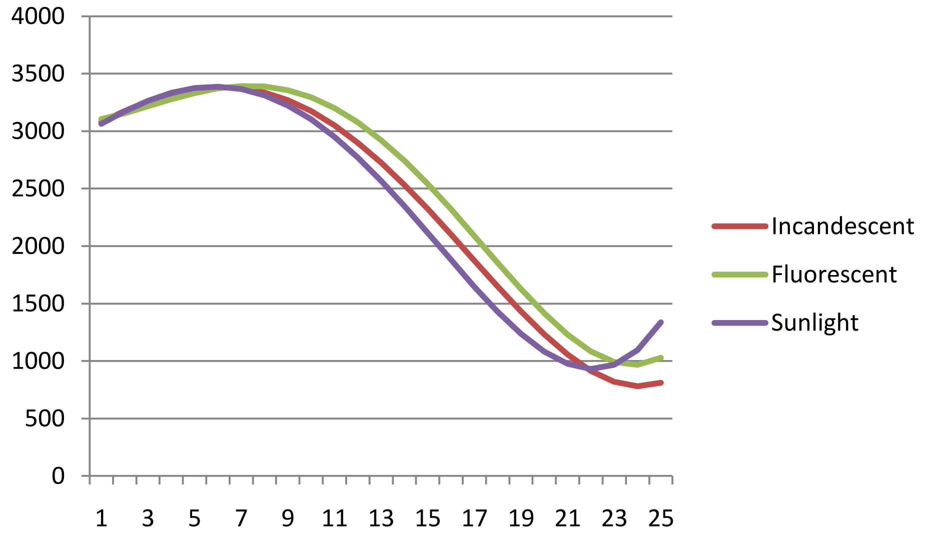

Figure 16 – Center Snout IR Sensor Analog Value vs. Distance in Inches

6. For the Right Snout IR Sensor, a Sharp GP2Y0A02YK0F:

Distance	Incandescent		Fluorescent		Sunlight	
	White	Black	White	Black	White	Black
12	3187	3197	3167	3197	3187	3197
14	3512	3568	3521	3567	3569	3512
16	3587	3587	3577	3567	3569	3567
18	3577	3581	3577	3517	3361	3391
20	3261	3192	3261	3233	3165	3263
22	3167	3097	3183	3197	3157	3097
24	3161	2544	3097	3161	2513	2537
26	2538	2458	3488	3037	2391	2335
28	2233	2369	2312	2497	2161	2153
30	2161	2237	2233	2197	2132	2047
32	2137	1943	2037	2143	1945	1941
34	1945	1866	1949	2042	1861	1625
36	1869	1930	1940	1920	1625	921
38	1731	1934	1803	1968	921	921
40	1625	921	1625	1867	921	921
42	921	921	921	921	921	921

Figure 17 – Right Snout IR Sensor Results

Resulting curves after curve fitting:

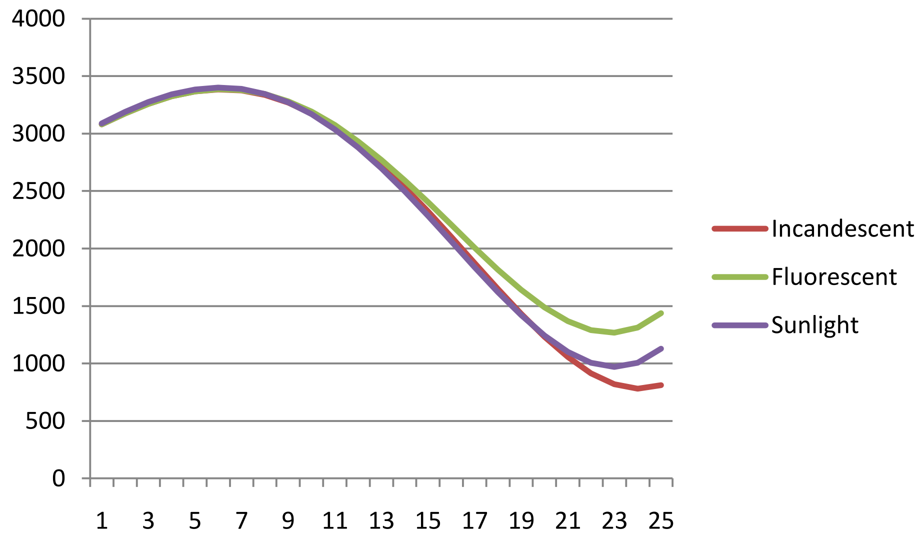

Figure 18 – Right Snout IR Sensor Analog Value vs. Distance in Inches

b) SONAR

The LV-EZ0 SONAR sensors are almost identical in their performance. I recorded their output as I moved a target progressively farther away. 13 feet seemed to be the cutoff point. I took the data, plotted it, and then did curve fitting. A third degree polynomial gave the best results. The delay required between analog readings and the possible interference that can occur between SONAR sensors placed close to each other resulted in my decision to use IR for obstacle avoidance and SONAR for target detection. I also decided not to use the LV-EZ4 because the narrow beam made it extremely sensitive. The code used to test the SONAR is in Appendix A, part 2.

The results are as follows (all distances are in feet):

Feet	FRONT	REAR
1	471.27	473.9744
2	394.64	397.2932
3	402.51	407.6748
4	482.46	492.1076
5	622.07	637.58
6	808.92	831.0804
7	1030.59	1059.597
8	1274.66	1310.119
9	1528.71	1569.634
10	1780.32	1825.13
11	2017.07	2063.596
12	2226.54	2272.021
13	2396.31	2437.393
14	2513.96	2546.7
15	2567.07	2586.93
16	2543.22	2545.072

Figure 19 – Front and Rear SONAR Results

After curve fitting:

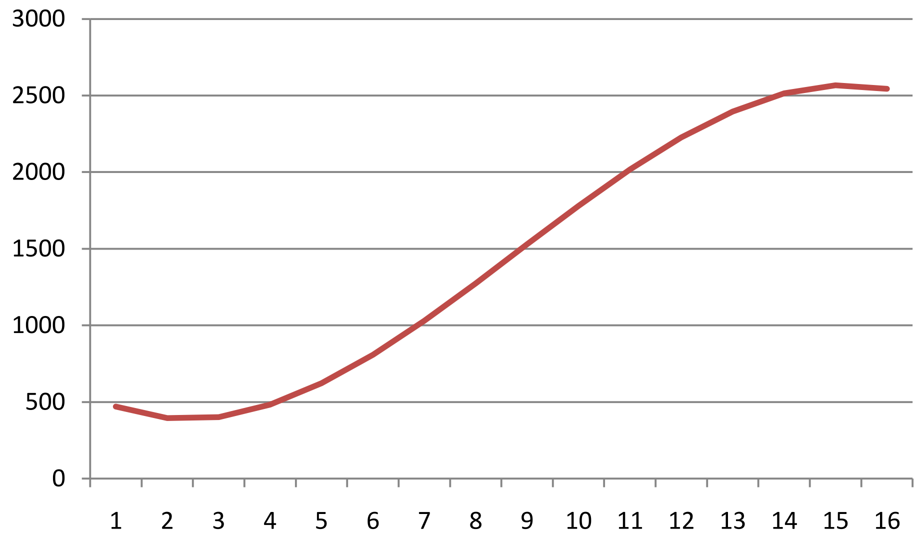

Figure 20 – Front SONAR Analog Value vs. Distance in Feet

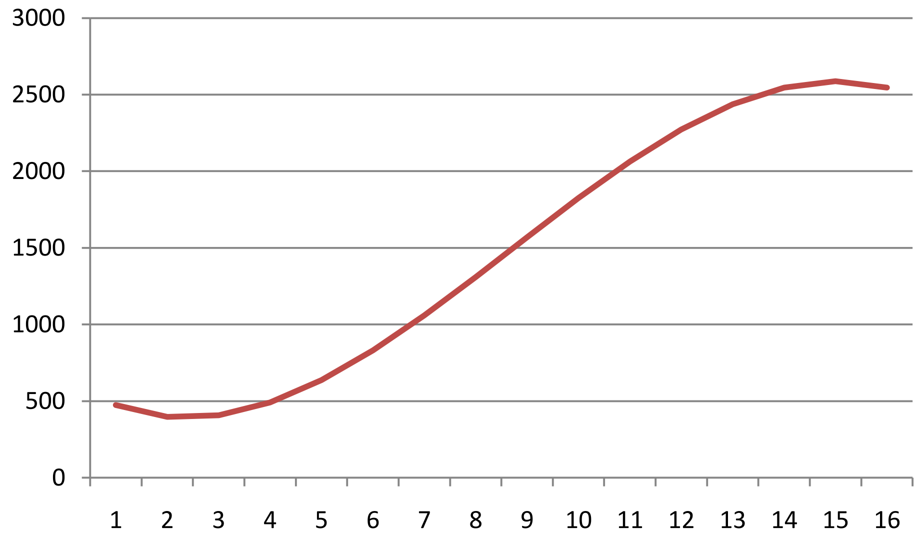

Figure 21 – Rear SONAR Analog Value vs. Distance in Feet

c) CMUcam

The CMUcam was the hardest device to integrate. A lot of problems came from initializing the XMEGA's USARTs since it is different than the more familiar ATmega. Joshua Phillip's wrote some initial code that I heavily modified to get the interface between the PVR Board the CMUcam to work. By setting up the camera in polling mode, 9600 baud, and using raw data mode I was able to get usable data from the camera. The Java based interface is essential to determining color values and visualizing what the camera is detecting.

The connections are made as follows:

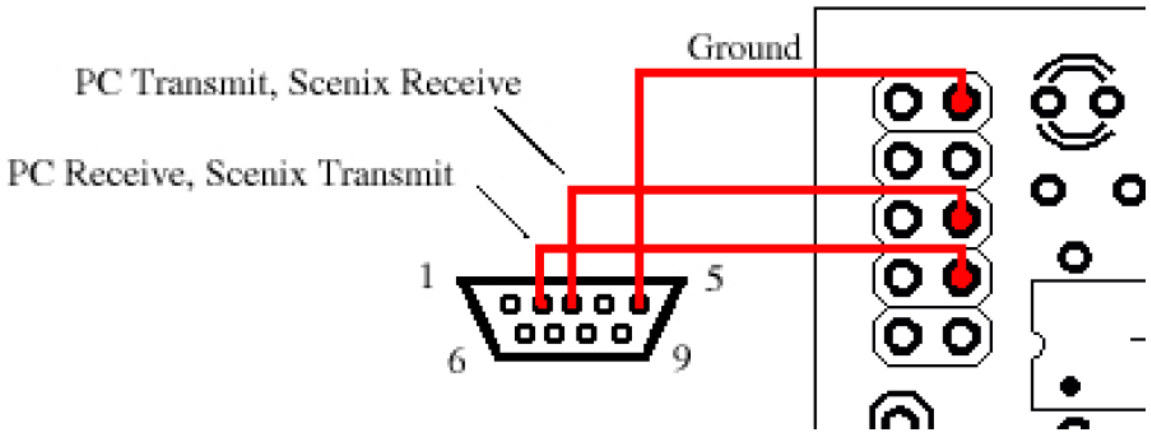

Figure 22 – Connections between PVR board and CMUcam [3]

The first thing done was to test the interface between the camera and the motherboard. The easiest way to do this is to turn the tracking light on and off. The code for this can be seen in Appendix A, Part 5. Appendix A, Part 3 and 4 are the modified uart.c and uart.h files actually used. This was successful.

The next thing done was getting usable data from the camera. The code for this is in Appendix A, Part 6. Data is continuously sent from the camera and displayed on FanBot's LCD.

The Java interface was used to get mean and deviations for incandescent and fluorescent lighting. I tested fluorescent lighting first with white balance on and off. The white balance on seemed to give the best results. I then tested in incandescent lighting with the white balance on. Fluorescent lighting gave the best results. Even though there is a red shift under fluorescent lighting, it is more pronounced under incandescent lighting. Data is as follows:

1. Fluorescent Lighting:

White Balance Off						
Color	**Red Mean**	**Green Mean**	**Blue Mean**	**Red Dev**	**Green Dev**	**Blue Dev**
Green	122	120	62	20	26	8
Orange	134	38	16	28	6	0
Blue	127	132	137	12	12	11
Red	150	17	18	16	2	2
Yellow	138	98	16	19	12	1

White Balance On						
Color	**Red Mean**	**Green Mean**	**Blue Mean**	**Red Dev**	**Green Dev**	**Blue Dev**
Green	130	127	67	21	27	8
Orange	142	40	16	27	6	0
Blue	136	135	135	13	14	12
Red	149	17	18	16	2	2
Yellow	138	94	17	19	13	1

Figure 23 – CMUcam Results Under Fluorescent Lighting

2. Incandescent Lighting

Color	Red Mean	Green Mean	Blue Mean	Red Dev	Green Dev	Blue Dev
Green	144	99	99	15	12	9
Orange	145	49	47	31	26	6
Blue	136	99	110	12	11	10
Red	157	44	28	21	5	5
Yellow	141	65	30	21	7	5

Figure 24 – CMU Cam Results Under Incandescent Lighting

10. Conclusion

I successfully integrated almost everything I set out to do. The camera and all sensors worked properly. I had problems with CMU Cam, but eventually got it working. I had some silly mistakes along the way such as leaving the lens cap on. The CMU Cam works best at extremely low baud rates. There is no need to operate at 115, 000 baud. Also, leaving the auto gain and white balance on interferes with the ability to track targets. I recommend that future students fully read the CMU Cam Manual to understand the impact of the various settings.

Initially, FanBot moved far too quickly. This interfered with the ability to find targets and avoid obstacles. This was corrected by limiting the speed. The motors were probably over designed, but this is better in the long run. The planetary gear motors worked really well.

Fully testing all sensors paid off when integrating everything. It was a matter of small tweaking instead of major coding. Experimentation will save time. To quote Dr. Schwartz, "Be the tortoise and not the hare."

My biggest failure on the project was the feet. If I had to do over again, I would use servos and not motors. It would be far easier to accomplish what I wanted.

Finally, I want to thank the TA's Mike and Thomas for the outstanding support along the way. I highly recommend their PVR board for future students. I also would like to thank Dr. Schwartz and Dr. Arroyo. This was the best and most challenging course I have ever taken. Thanks also to Tim Martin for his advice and support on using the Wave Shield. Lastly, I want to thank all my fellow students for the suggestions and recommendations.

11. Documentation

[1] IR Sensor and Sonar photos taken from www.sparkfun.com.

[2] CMUcam 1 photo taken from www.seattlerobotics.com

[3] CMUcam 1 Manual v. 2.0, Carnegie Mellon University, 2003

12. Appendices

A. Appendix A – Sensor Testing Code

1. Code used to test IR Sensors:

```
#include <avr/io.h>
#include "PVR.h"

void main(void)
{
        xmegaInit();                    //setup XMega
        delayInit();                    //setup delay functions
        ServoCInit();                   //setup PORTC Servos
        ServoDInit();                   //setup PORTD Servos
        ADCAInit();                     //setup PORTA analog readings
        lcdInit();                  //setup LCD on PORTK
        lcdString("IR Value   ");       //display "IR Value" on top line (Line 0) of LCD
        int IR;
        while(1)
        {
                IR = ADCA0();           //call IR value
                lcdGoto(1,0);           //move LCD cursor to Line 1 of LCD
                lcdInt(IR);             //display IR Value on second line
                lcdString("  ");        //ensures value displays correctly
        }
}
```

2. Code use to test SONAR:

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

```
#include "PVR.h"

void main(void)
{
	xmegaInit();					//setup XMega
	delayInit();					//setup delay functions
	ServoCInit();					//setup PORTC Servos
	ServoDInit();					//setup PORTD Servos
	ADCAInit();					//setup PORTA analong readings
	lcdInit();				//setup LCD on PORTK
	lcdString("SONAR Value");			//display "SONAR Value" on top line (Line 0) of LCD
	int SONAR;					//SONAR value
	delay_ms(350);					//Delay needed for SONAR to startup and self-calibrate
	while(1)
	{
		SONAR = ADCA0();			//call Left SONAR value
		lcdGoto(1,0);				//move LCD cursor to the second line (Line 1) of LCD
		lcdInt(SONAR);			//display  SONAR Value on second line
		lcdString("  ");			//spaces to ensure value displays correctly
		delay_ms(200);			//delay 200ms needed between value calls
	}
}
```

3. Modified uart.h header file, original provided by Joshua Phillips

```
#ifndef  __uart_h__
#define  __uart_h__

#include <avr/io.h>
#include "uart.h"

void uart_init0(void);						//Initiates the E0 USART
unsigned char uart_getchar0(void);			//All functions ending in 0 correspond to E0
unsigned int uart_getint0(void);
void uart_sendchar0(unsigned char tx_char);
void uart_sendstring0(unsigned char tx_string[]);
unsigned char uart_getstring0(void);

void uart_init1(void);						//Initiates the E1 USART
unsigned char uart_getchar1(void);			//All functions ending in 1 correspond to E1
unsigned int uart_getint1(void);
void uart_sendchar1(unsigned char tx_char);
void uart_sendstring1(unsigned char tx_string[]);
unsigned char uart_getstring1(void);

#endif
```

4. Modified uart.c source file, original provided by Joshua Phillips

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

```
#include "uart.h"

void uart_init0(void)                                       //E0 will be used to connect to the CMUcam
{
        PORTE_DIR   = PIN3_bm;                              //Pin 3 is used to transmit.  This is part of
        PORTE_OUT   = PIN3_bm;                              //the required setup in the XMEGA datasheet.
        PORTE.DIRCLR   = PIN2_bm;                           //Pin 2 is used to receive data.

        USARTE0_CTRLC = 0x03;                               // set 8N1 asynchronous serial tx/rx
        USARTE0_BAUDCTRLA = 0xF5;                           // The spreadsheet provided gives the values
                                                            // of BSEL and BSCALE to establish various baud
                                                            //rates and the associated errors.  The values written
        USARTE0_BAUDCTRLB = 0xCC;                           // in this case establishes 9600 baud and < 0.01%
                                                            //error.
        USARTE0_CTRLB |= 0x08;                              // turn on TX system
        USARTE0_CTRLB |= 0x10;                              // turn on RX system
}




unsigned char uart_getchar0(void)
{
        unsigned char rx_char;                                      // initialize received character buffer
        while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));    // wait for RXCIF to be set
        rx_char = USARTE0_DATA;                                     // rx_char set to equal E0 DATA value
        return rx_char;                                             //returns character.
}

unsigned int uart_getint0(void)
{
        unsigned int rx_int;                                        // initialize received int buffer
        while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));    // wait for RXCIF to be set
        rx_int = USARTE0_DATA;                                      // rx_int set to equal E0 DATA value
        return rx_int;                                              //rx_int is returned
}

void uart_sendchar0(unsigned char tx_char)
{
        while (!(USARTE0_STATUS & (1<<USART_DREIF_bp)));    // check if data register is empty
        USARTE0_DATA = tx_char;                                     // store data in tx_char
        while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp)));    // wait for TXCIF to be set (data sent)
}

void uart_sendstring0(unsigned char tx_string[])
{
        int i = 0;
                while (tx_string[i] != '\0')                //while not at the null character (string terminator)
                                                            // continue sending serial string
        {
                uart_sendchar0(tx_string[i++]);             //call to uart character sending function
        }                                                   //loop through sequence of i until finished
}

unsigned char uart_getstring0(void)
```

```
{
	static char *rx_string;				//define pointer to array rx_string of unknown length
	int i = 0;

	do
	{
		rx_string[i] = uart_getchar0();		//continue receiving string from peripheral till
							//it returns the line return sequence
	} while (rx_string[i++] != '\r');		//eliminate the carriage return sent by CMUcam
return rx_string;					//return the character array

}
```

```
void uart_init1(void)					//All E1 functions are identical to the E0 functions
{							//Except that they use the E1 USART.
	PORTE_DIR   = PIN7_bm;				//The E1 USART will be used to communicate with
	PORTE_OUT   = PIN7_bm;				//the Arduino and Wave Shield
	PORTE.DIRCLR   = PIN6_bm;

	USARTE1_CTRLC = 0x03;
	USARTE1_BAUDCTRLA = 0xF5;
	USARTE1_BAUDCTRLB = 0xCC;
	USARTE1_CTRLB |= 0x08;
	USARTE1_CTRLB |= 0x10;
}

unsigned char uart_getchar1(void)
{
	unsigned char rx_char;
	while (!(USARTE1_STATUS & (1<<USART_RXCIF_bp)));
	rx_char = USARTE1_DATA;
	return rx_char;
}

unsigned int uart_getint1(void)
{
	unsigned int rx_int;
	while (!(USARTE1_STATUS & (1<<USART_RXCIF_bp)));
	rx_int = USARTE1_DATA;
	return rx_int;
}

void uart_sendchar1(unsigned char tx_char)
{
	while (!(USARTE1_STATUS & (1<<USART_DREIF_bp)));
	USARTE1_DATA = tx_char;
	while (!(USARTE1_STATUS & (1<<USART_TXCIF_bp)));
}

void uart_sendstring1(unsigned char tx_string[])
{
        int i = 0;
        while (tx_string[i] != '\0')
        {
                uart_sendchar1(tx_string[i++]);
        }
}

unsigned char uart_getstring1(void)
{
        static char *rx_string;
        int i = 0;

        do
        {
        rx_string[i] = uart_getchar1();
        } while (rx_string[i++] != '\0');
return rx_string;
}
```

5. Code to turn the CMUcam track light on and off

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "uart.h"
#include "PVR.h"
int main(void)
{
        xmegaInit();                                    //setup XMega
        delayInit();                                    //setup delay functions
        ServoCInit();                                   //setup PORTC Servos
        ServoDInit();                                   //setup PORTD Servos
        ADCAInit();                                     //setup PORTA analong readings
        lcdInit();                                      //setup LCD on PORTK
        uart_init0();                                   //Initalizes E0 USART

        uint8_t receivedData;                           //Establishes1byte variable

  while (1)
                {
                uart_sendstring0("L1 1\r");             //Sends code to turn light on

                do {                                    //Gets and displays ACK
                receivedData = uart_getchar0();         //returned from CMUcam
                lcdChar(receivedData);                  //and the ‘:’ sent when cam
                } while (receivedData != ':');          //is ready for the next command

                lcdInt(1);                              //Displays 1 when command to
                                                        //turn the light on is sent.
                lcdGoto (0,0);
```

```
                    PVRdelay_ms(1000);                                    //1 second delay

                    uart_sendstring0("L1 0\r");                           //Send command to turn light off

                    do {                                                  //Used to get ACK as above
                    receivedData = uart_getchar0();
                    lcdChar(receivedData);
                    } while (receivedData != ':');

                    lcdInt(2);                                            //Displays 2 when command to
                                                                          //turn light off is sent.
                    lcdGoto (0,0);

                    PVRdelay_ms(1000);
                    }
return 0;
}
```

6. Code used to get S packet from CMUcam

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "uart.h"
#include "PVR.h"

int main(void)
{
          xmegaInit();                              //setup XMega
          delayInit();                              //setup delay functions
          ServoCInit();                             //setup PORTC Servos
          ServoDInit();                             //setup PORTD Servos
          ADCAInit();                               //setup PORTA analong readings
          lcdInit();                                //setup LCD on PORTK
          uart_init0();                             //Initializes usart E0 at 9600 baud
                                                    //8 bits, 1 stop bit, no parity

          uint8_t receivedData;                     //Used to clear received data

          int data[7];                              //Int array to get data

/******************************Initialize Cam***********************************/


          lcdString("Initializing Cam");            //LCD display while initializing cam settings

          uart_sendstring0("RS \r");                //Resets the cam

          do {                                      //Clears out the receive buffer
                    receivedData = uart_getchar0();
```

```
			} while (receivedData != ':');
		lcdInt(1);

		uart_sendstring0("SW 1 1 40 143\r");		//Uses the entire window for color mean analysis

		do {						//Clears out the receive buffer
			receivedData = uart_getchar0();
			} while (receivedData != ':');
		lcdInt(0);

		uart_sendstring0("CR 18 44\r");			//Adjusts for incandescent lighting/sunlight

		/*uart_sendstring0("CR 45 7 18 44\r");*/		//Adjusts for fluorescent lighting

		do {						//Clears out receive buffer
			receivedData = uart_getchar0();
			} while (receivedData != ':');
		lcdInt(1);

		PVRdelay_ms(2000);				//Delay to allow lighting adjustment

		uart_sendstring0("PM 1\r");			//Establishes Polling Mode

		do {
			receivedData = uart_getchar0();		//Clears out receive buffer
			} while (receivedData != ':');
		lcdInt(2);

		uart_sendstring0("RM 3\r");			//Raw Mode - gets rid of ACK returns,
								//Camera sends data in raw mode
								//Instead of ASCII characters, data is sent in integers
								//The data start byte will be 255, followed by packet
								// type, and then color values and deviations
		do {
			receivedData = uart_getchar0();		//Clears out receive buffer
			} while (receivedData != ':');
		lcdInt(3);

		PVRdelay_ms(1000);				//Delay to allow initialization to complete

		lcdGoto (0,0);					//Clears first line
		lcdString("                ");

		int i;


/****************************Data Loop*****************************************/

		while(1)
		{
		uart_sendstring0("GM\r");			//Requests S Packet

		i=0;

		while(i <8)					//Read in data
			{
```

```
            data[i] = uart_getint0();
            i++;
            }

lcdGoto(0,0);                                     //Displays packet type, 83 = "S"
lcdString("Packet Type");
if(data[1]==83)
            {
            lcdGoto(1, 0);
            lcdString("S Type");
            }
else                                              //In case of transmission error
            {
            lcdGoto(1, 0);
            lcdString("Unknown");
            }

PVRdelay_ms(700);





lcdGoto(0,0);                                     //Displaye Red Mean
lcdString("Red Mean     ");
 lcdGoto(1,0);
lcdInt(data[2]);
lcdString("      ");

PVRdelay_ms(700);

lcdGoto(0,0);                                     //Displays Blue Mean
lcdString("Blue Mean    ");
 lcdGoto(1,0);
lcdInt(data[3]);
lcdString("      ");

PVRdelay_ms(700);

lcdGoto(0,0);                                     //Displays Green Mean
lcdString("Green Mean     ");
 lcdGoto(1,0);
lcdInt(data[4]);
lcdString("      ");

PVRdelay_ms(700);
lcdGoto(0,0);                                     //Displays Red Deviation
lcdString("Red Dev     ");
 lcdGoto(1,0);
lcdInt(data[5]);
lcdString("      ");

PVRdelay_ms(700);

lcdGoto(0,0);                                     //Displays Blue Deviation
```

```
        lcdString("Blue Dev    ");
         lcdGoto(1,0);
        lcdInt(data[6]);
        lcdString("      ");

        PVRdelay_ms(500);

        lcdGoto(0,0);
        lcdString("Green Dev     ");                    //Displays Green Deviation
         lcdGoto(1,0);
        lcdInt(data[7]);
        lcdString("      ");

        PVRdelay_ms(700);

        uart_getchar0();                                //Clears the ':' sent after the data packet

        }
return 0;
}
```

B. Appendix B – The Final, Integrated Code:

```
/**********************************************
                                              *
*                FanBot Code                  *
*                                             *
*                by Brian Long                *
*                                             *
*   (UART originally written by Joshua        *
*    Phillip's and modified for use in        *
*    FanBot.)                                 *
*                                             *
**********************************************/



/************   Includes   ********************/


#include <stdlib.h>

#include <stdbool.h>

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

```
#include <avr/interrupt.h>

#include <avr/pgmspace.h>

#include "uart.h"

#include "PVR.h"



/**************   Defines   **********************/


/*  Start and stop .wav files */


#define Chomp_Start                    0X70;

#define Chomp_End                      0XF0;

#define Fight_Start                    0X71;

#define Fight_End                      0XF1;

#define Hail_Start                     0X72;

#define Hail_End                       0XF2;

#define WantSome_Start                 0X73;

#define WantSome_End                   0XF3;

#define Roll_Start                     0X74;

#define Roll_End                       0XF4;

#define Systems_Start                  0X75;

#define Systems_End                    0XF5;

#define Elmo_Start                     0X76;

#define Elmo_End                       0XF6;

#define Gator_Start                    0X77;

#define Gator_End                      0XF7;

#define Jaws_Start                     0X78;

#define Jaws_End                       0XF8;

#define Sniff_Start                    0X79;

#define Sniff_End                      0XF9;
```

```
#define Territorial_Start           0X7A;

#define Territorial_End             0XFA;

#define Stupidity_Start             0X7B;

#define Stupidity_End               0XFB;


/* Motor Control */


#define All_Off                     0X00;

#define Right_Fwd                   0X01;

#define Right_Rev                   0X02;

#define Left_Fwd                    0X04;

#define Left_Rev                    0X08;

#define Legs_Fwd                    0X10;

#define Legs_Rev                    0X20;


#define Fwd                         0X05;

#define Rev                         0X0A;

#define     Hard_Right              0X06;

#define     Hard_Left               0X09;


#define     Fwd_With_Legs           0X15;

#define     Rev_With_Legs           0X2A;

#define     Hard_Left_Legs_Fwd      0X16;

#define     Hard_Left_Legs_Rev      0X26;

#define     Hard_Right_Legs_Fwd     0X19;

#define Hard_Right_Legs_Rev         0X29;


/***********   Function Prototypes   *************/

void Initialize (void);

void JawUp(void);
void JawDown(void);

void HeadLeft(void);

void HeadRight(void);

void HeadCenter(void);

void Head(int);

void TailLeft(void);

void TailRight(void);

void TailCenter(void);

void LeftMotors(int);

void LegMotors(int);

void RightMotors(int);



void LCD_Clear(void);

void Cam_Initialize(void);

float SonarRead(void);

float LeftIR_Read(void);

float LeftFwdIR_Read(void);

float CenterIR_Read(void);

float UnderIR_Read(void);

float RightFwdIR_Read(void);

float RightIR_Read(void);

float RearIR_Read(void);

void RightTurnFwd(void);

void LeftTurnFwd(void);

void SmallLeftTurnFwd(void);

void HardRightTurnFwd(void);

void HardLeftTurnFwd(void);

void GoFwd(void);

void GoRev(void);
void RightTurnRev(void);

void LeftTurnRev(void);

void StopMoving(void);

void GatorSearch(void);

void BamaSearch(void);

void BamaTargetCheck(void);

void GatorTargetCheck(void);

void Obstacle(void);

void BamaApproach(void);

void GatorApproach(void);


/******************   Global Variables **********************/



uint8_t middle_x, middle_y, lcx, lcy, rcx, rcy, pix, conf;


bool GotTarget = false;                    //Used to determine if object detected


float LeftIR;                              //Left IR value

float RightIR;                                     //Right IR value

float LeftFwdIR;                           //Left Fwd IR value

float RightFwdIR;                          //Right Fwd IR value

float CenterIR;                                    //Center IR value

float UnderIR;                                     //Under IR value

float RearIR;                              //Rear IR value


int Side_greater_avg;              //Figures out whether

int Fwd_greater_avg;               //left or side averages are larger


/******************   Main   *********************/
```

```
int main ()
{
/* Variables */
bool going = false;                     //A bool to ensure the robot does not move until both
                                        //IR sernsors are covered.
float Sonar;                            //Sonar value
uint8_t receivedData;           //Used to clear received data
int data[10];                           //Int array to get data
int GoCheck;
/* Inititialize */
Initialize();
PORTH_OUT = Systems_Start;                      //All Systems Go played
PVRdelay_ms(500);
PORTH_OUT = Systems_End;


/* Holding Loop */



while(!going)                                   //Until both IR sensors are covered,
                                        //it stays in this loop and waitis.
    {
    LeftIR = LeftIR_Read();
    RightIR = RightIR_Read();
    LeftFwdIR = LeftFwdIR_Read();
    RightFwdIR = RightFwdIR_Read();
    CenterIR = CenterIR_Read();
    UnderIR = UnderIR_Read();
    RearIR = RearIR_Read();
    if((RightFwdIR >=3300) || (LeftFwdIR >=3300) || (CenterIR >=3300))
            {
```

```
			PORTH_OUT = Gator_Start;			//Territorial played

			PVRdelay_ms(500);

			PORTH_OUT = Gator_End;

			}

	if ((RightIR >= 3400) && (LeftIR >=3400)) //Until the sensors are covered (IR values >= 3500), do 
nothing.

			{

			lcdGoto(0,0);

			lcdString("Wakey Wakey");

			going = true;					//Sets going bool to true to break out of the loop

			}

	else going = false;

	}


/*   Clears LCD    */


PVRdelay_ms(1000);

lcdGoto(0,0);

LCD_Clear();


/*  Play Elmo    */

PORTH_OUT = Elmo_Start;

PVRdelay_ms(200);

PORTH_OUT = Elmo_End;

PORTJ_OUT = All_Off;

PVRdelay_ms(800);

	do

	{

			for(int i =0; i<6; i++)

			{

			HeadLeft();
```

```
                JawDown();
                PVRdelay_ms(400);
                HeadCenter();
                JawUp();
                PVRdelay_ms(400);
                HeadRight();
                JawDown();
                HeadCenter();
                TailCenter();
                JawUp();
                }
        }while(PORTF_IN == 0X6F);

while(going)
        {
        BamaSearch();
        BamaTargetCheck();
        while (GotTarget == true)
                {
                BamaApproach();
                }

        Obstacle();
        GatorSearch();
        GatorTargetCheck();
        while (GotTarget == true)
                {
                GatorApproach();
                }
        Obstacle();
```

```
	}

return 0;
}

/*************   Functions   ********************/

/*   Initialize   */
void Initialize ()			//Initializes PVR board
{

xmegaInit();				//setup XMega
delayInit();				//setup delay functions
ServoCInit();				//setup PORTC Servos
ServoDInit();				//setup PORTD Servos
ADCAInit();					//setup PORTA analog readings
lcdInit();					//setup LCD on PORTK
lcdString("Initializing");
PORTJ_DIR |= 0xFF;			//Sets up Port J as an output port
							//to control motor direction.
PORTJ_OUT = 0XFF;				//All motors are off intially.
```

```
PORTH_DIR |= 0xFF;                  //Sets up Port H as an output port
PORTH_OUT = 0XFF;                   //to allow comms with the Arduino board.
PORTF_DIR |= 0X6F;                  //Sets up Port F as an input port
                                            //for comms with Arduino board
PORTF_OUT = 0X6F;
uart_init0();               //Initializes USART E0 at 9600 baud
                                            //8 bits, 1 stop bit, no parity
Cam_Initialize();           //Initailizes CMU Cam
lcdGoto(0,0);                       //Clears LCD.
LCD_Clear();
}




/*    Servos    */



#define Tail_Left                   27;
#define Tail_Right                  -5;
#define Tail_Center                 9;

#define Head_Left1                  33;
#define Head_Right                  -33;
#define Head_Center                 2;
void JawUp (void)
    {
    ServoD2(90);
    }
void JawDown (void)
    {
    ServoD2(-100);
    }
void TailLeft (void)
    {
    ServoC5(60);
    }
void TailCenter (void)
    {
    ServoC5(28);
    }
void TailRight (void)
    {
    ServoC5(0);
    }



void HeadLeft (void)
    {
    ServoD1(-33);
    }
void HeadCenter (void)
    {
    ServoD1(0);
    }
void HeadRight (void)
    {
    ServoD1(33);
    }

void Head (int value)
```

```
    {
    ServoD1(value);
    }

/*  Motors   */

void LeftMotors(int value)
{
    if ((value > 0 && value < 30) || (value < 0)) value = 30;
    value *= 100;
    TCC0_CCA = (value);                          //Generate PWM.
}



void RightMotors(int value)
{
    if ((value > 0 && value < 30) || (value < 0)) value = 30;
    value *= 100;
    TCC0_CCB = (value);                          //Generate PWM.
}

void LegMotors(int value)
{
    if ((value > 0 && value < 30) || (value < 0)) value = 30;
    value *= 100;
    TCC0_CCC = (value);                          //Generate PWM.
}

/*  Cam Initialize  */
```

```
void Cam_Initialize()
{
    uint8_t receivedData;                                   //Used to clear received data
    lcdGoto(0,0);                                                //Clears LCD.
    lcdString("Initializing Cam");                    //LCD display while initializing cam settings
    uart_sendstring0("RS \r");                        //Resets the cam
    do {                                              //Clears out the receive buffer
            receivedData = uart_getchar0();
            } while (receivedData != ':');
    lcdInt(1);

    uart_sendstring0("SW 1 1 40 143\r");      //Uses the entire window for color mean analysis

    do {                                      //Clears out the receive buffer
            receivedData = uart_getchar0();
            } while (receivedData != ':');
    lcdInt(0);

    uart_sendstring0("CR 17 10 18 44\r");     //Turns on white balance.  Changes clock speed.

    do {                                      //Clears out receive buffer
            receivedData = uart_getchar0();
            } while (receivedData != ':');



    for(int j=0; j<10; j++) {         //It takes 5 seconds to adjust to lighting conditions.  Leaving the auto
    PVRdelay_ms(500);                 //gain and auto white balance on could impact the ability to track.
    }                                 //See CMU Cam manual for explanation.

    uart_sendstring0("CR 18 40 19 32\r");     //Turn off auto gain and auto white balance
```

```
do {                                                    //Clears out receive buffer
            receivedData = uart_getchar0();
            } while (receivedData != ':');


lcdInt(1);


PVRdelay_ms(2000);                  //Delay to allow lighting adjustment


uart_sendstring0("PM 1\r");         //Establishes Polling Mode


do {
            receivedData = uart_getchar0();                 //Clears out receive buffer
            } while (receivedData != ':');
lcdInt(2);


uart_sendstring0("MM 1\r");                     //Establishes Middle Mass Mode


do {
            receivedData = uart_getchar0();                 //Clears out receive buffer
            } while (receivedData != ':');
lcdInt(2);



uart_sendstring0("RM 3\r");         //Raw Mode - gets rid of ACK returns, sends data in raw mode
                                    //Instead of ASCII characters, data is sent in integers
                                    //The data start byte will be 255, followed by packet type, and data
do {
            receivedData = uart_getchar0();                 //Clears out receive buffer
            } while (receivedData != ':');
    lcdInt(3);

    PVRdelay_ms(1000);              //Delay to allow initialization to complete

}

/* LCD Clear */

void LCD_Clear()
{

lcdString("                ");

}
/* SONAR */
float  SonarRead()
{

float SonarValue;
float SonarSum =0;
float SonarAvg;

for (int i = 0; i<20; i++)

    {
    SonarValue = ADCA3();   //Call SONAR value
    SonarSum += SonarValue; //Adds readings for average;
    PVRdelay_ms(50);                //50 msec delay required between
    }                                                   //readings.
```

```
SonarAvg = SonarSum/20;

return SonarAvg;
}



/* IR Read */

float LeftIR_Read(void)
{

float value;

float avg;

float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA0();

sum = sum + value;

}

avg = sum/16;
```

```
return avg;
}




float LeftFwdIR_Read(void)
{

float value;

float avg;

float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA1();

sum = sum + value;

}

avg = sum/16;

return avg;
}
```

```
float CenterIR_Read(void)

{

float value;

float avg;

float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA2();

sum = sum + value;

}

avg = sum/16;

return avg;
}
```

```
float UnderIR_Read(void)
{

float value;

float avg;

float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA4();

sum = sum + value;

}

avg = sum/16;

return avg;
}

float RearIR_Read(void)

{

float value;
float avg;

float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA5();

sum = sum + value;

}

avg = sum/16;

return avg;
}



float RightFwdIR_Read(void)

{

float value;

float avg;
float sum =0;

int i;

for (i=0; i< 16; i++)
{

value = ADCA6();

sum = sum + value;

}
avg = sum/16;

return avg;

}



float RightIR_Read(void)

{

float value;

float avg;

float sum =0;

int i;
```

```
for (i=0; i< 16; i++)
{

value = ADCA7();

sum = sum + value;

}

avg = sum/16;

return avg;
}



/******************* Movement ****************************/



void RightTurnFwd(void)

{

lcdGoto(0,0);
lcdString("Right Turn!     ");
PORTJ_OUT = Fwd;
HeadRight();
TailRight();
LeftMotors(70);
```

```
RightMotors(0);
PVRdelay_ms(500);
RightMotors(50);
}

void HardRightTurnFwd(void)

{
lcdGoto(0,0);
lcdString("Hard Right Turn!     ");

LeftMotors(0);
RightMotors(0);

PVRdelay_ms(200);

PORTJ_OUT = Hard_Right;
HeadRight();
TailRight();
LeftMotors(40);
RightMotors(40);
PVRdelay_ms(300);
LeftMotors(0);
RightMotors(0);
PVRdelay_ms(200);
}


void SmallLeftTurnFwd(void)
```

```
{

lcdGoto(0,0);

lcdString("Left Turn!     ");

PORTJ_OUT = Fwd;


HeadLeft();

TailLeft();

RightMotors(80);

LeftMotors(0);

PVRdelay_ms(325);

RightMotors(40);

LeftMotors(40);

}




void LeftTurnFwd(void)


{

lcdGoto(0,0);

lcdString("Left Turn!     ");

PORTJ_OUT = Fwd;


HeadLeft();

TailLeft();

RightMotors(85);

LeftMotors(0);

PVRdelay_ms(620);

RightMotors(50);

LeftMotors(50);
```

```
}



void HardLeftTurnFwd(void)


{

lcdGoto(0,0);

lcdString("Hard Left Turn!     ");


LeftMotors(0);

RightMotors(0);


PVRdelay_ms(200);




PORTJ_OUT = Hard_Left;

HeadLeft();

TailLeft();

LeftMotors(40);

RightMotors(40);

PVRdelay_ms(300);

LeftMotors(0);

RightMotors(0);

PVRdelay_ms(200);

}



void GoFwd(void)

{

lcdGoto(0,0);
```

```
lcdString("Time to go!     ");


PORTJ_OUT = Fwd;

HeadCenter();

TailCenter();

LeftMotors(50);

RightMotors(50);

}


void GoRev(void)

{


lcdGoto(0,0);

lcdString("Back Up!     ");




HeadCenter();

TailCenter();


LeftMotors(0);

RightMotors(0);


PORTJ_OUT = Rev;


PVRdelay_ms(200);


LeftMotors(40);

RightMotors(40);


PVRdelay_ms(400);
```

```
LeftMotors(0);
RightMotors(0);

PVRdelay_ms(200);
}



void RightTurnRev(void)
{

lcdGoto(0,0);
lcdString("Back Right!     ");

HeadRight();
TailRight();

LeftMotors(0);
RightMotors(0);

PVRdelay_ms(200);

PORTJ_OUT = Hard_Left;

LeftMotors(40);
RightMotors(40);

PVRdelay_ms(300);

LeftMotors(0);
```

```
RightMotors(0);


PVRdelay_ms(200);
}


void LeftTurnRev(void)
{


lcdGoto(0,0);
lcdString("Back Left!     ");




HeadLeft();
TailLeft();


LeftMotors(0);
RightMotors(0);


PVRdelay_ms(200);


PORTJ_OUT = Hard_Right;


LeftMotors(40);
RightMotors(40);


PVRdelay_ms(300);


LeftMotors(0);
RightMotors(0);
```

```
PVRdelay_ms(200);
}



void StopMoving(void)
{



LeftMotors(0);
RightMotors(0);
}

/************** Searches ******************/

void BamaSearch(void)

{
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Color - Tide?");



PORTH_OUT = Sniff_Start;
PVRdelay_ms(200);
PORTH_OUT = Sniff_End;

middle_x = 0;
middle_y = 0;
pix = 0;
```

```
conf= 0;

int data[10];

int i =0;

uart_sendstring0("TC 135 255 0 80 0 80\r");

while(i <10)
	{
	data[i] = uart_getint0();
	i++;
	}

middle_x = data[2];
middle_y = data[3];
pix = data[8];
conf= data[9];

uart_getchar0();

}



void GatorSearch(void)

{
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
```

```
lcdString("Color -  Gator?");



PORTH_OUT = Sniff_Start;

PVRdelay_ms(200);

PORTH_OUT = Sniff_End;


middle_x = 0;

middle_y = 0;

pix = 0;

conf= 0;


int data[10];


int i =0;


uart_sendstring0("TC 0 110 60 200 0 80\r");


while(i <10)

        {

        data[i] = uart_getint0();

        i++;

        }


middle_x = data[2];

middle_y = data[3];

pix = data[8];

conf= data[9];


uart_getchar0();
```

```
}



void BamaTargetCheck(void)
{
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Tide?   ");




PVRdelay_ms(400);

int count = 0;

if (conf < 35)
        {
        lcdGoto(0,0);
        LCD_Clear();
        lcdGoto(0,0);
        lcdString("No Tide");
        PVRdelay_ms(800);
        GotTarget = false;
        }

else if (conf >= 35)
        {
        lcdGoto(0,0);
        LCD_Clear();
        lcdGoto(0,0);
        lcdString("Got Tide!   ");
        PVRdelay_ms(800);
        GotTarget = true;
        PORTH_OUT = Jaws_Start;
        StopMoving();
        PVRdelay_ms(200);
        PORTH_OUT = Jaws_End;
        PVRdelay_ms(5000);
        }
GoFwd();
PVRdelay_ms(500);
while(GotTarget == false)
        {

        BamaSearch();
        StopMoving();
        PVRdelay_ms(1000);
        if (conf < 35)
                {
                lcdGoto(0,0);
                LCD_Clear();
                lcdGoto(0,0);
                lcdString("No Tide");
                PVRdelay_ms(800);
                GotTarget = false;
                SmallLeftTurnFwd();
                HeadCenter();
                TailCenter();
                }

        if (conf >= 35)
                {
                lcdGoto(0,0);
                LCD_Clear();
                lcdGoto(0,0);
                lcdString("Got Tide!   ");
                PVRdelay_ms(800);
                GotTarget = true;
                PORTH_OUT = Jaws_Start;
                StopMoving();
                PVRdelay_ms(200);
                PORTH_OUT = Jaws_End;
                PVRdelay_ms(8000);
                HeadCenter();
                TailCenter();
                break;
                }

        }
}



void GatorTargetCheck(void)
{
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Gator?   ");
```

```
PVRdelay_ms(400);

int count = 0;

if (conf < 35)
    {
    lcdGoto(0,0);
    LCD_Clear();
    lcdGoto(0,0);
    lcdString("No Gator");
    PVRdelay_ms(800);
    GotTarget = false;
    }

else if (conf >= 35)
    {
    lcdGoto(0,0);
    LCD_Clear();
    lcdGoto(0,0);
    lcdString("Got Gator!   ");
    PVRdelay_ms(800);
    GotTarget = true;
    PORTH_OUT = Hail_Start;
    StopMoving();
    PVRdelay_ms(200);
    PORTH_OUT = Hail_End;
    PVRdelay_ms(5000);
    }
```

```
GoFwd();
PVRdelay_ms(500);
while(GotTarget == false)
    {

    GatorSearch();
    StopMoving();
    PVRdelay_ms(1000);
    if (conf < 35)
            {
            lcdGoto(0,0);
            LCD_Clear();
            lcdGoto(0,0);
            lcdString("No Gator");
            PVRdelay_ms(800);
            GotTarget = false;
            SmallLeftTurnFwd();
            HeadCenter();
            TailCenter();
            }

    if (conf >= 35)
            {
            lcdGoto(0,0);
            LCD_Clear();
            lcdGoto(0,0);
            lcdString("Got Gator!   ");
            PVRdelay_ms(800);
            GotTarget = true;
            PORTH_OUT = Hail_Start;
            StopMoving();
            PVRdelay_ms(200);
            PORTH_OUT = Hail_End;
            PVRdelay_ms(8000);
            HeadCenter();
            TailCenter();
            break;
            }


    }
}
void Obstacle(void)
{

int count = 0;

PORTJ_OUT = Fwd;

LeftMotors(40);
RightMotors(40);

while (count < 2)
    {
    LeftIR = LeftIR_Read();
    RightIR = RightIR_Read();
    LeftFwdIR = LeftFwdIR_Read();
    RightFwdIR = RightFwdIR_Read();
    CenterIR = CenterIR_Read();
    RearIR = RearIR_Read();
```

```
        if(LeftIR >= RightIR)                                //Checks to see which reading is greater, in
case a decsion has to be made.

                {

                Side_greater_avg = 2;

                }

        else Side_greater_avg = 1;


        if(LeftFwdIR >= RightFwdIR)                                //Checks to see which reading is
greater, in case a decsion has to be made.

                {

                Fwd_greater_avg = 2;

                }

        else Fwd_greater_avg = 1;



        if (CenterIR >= 3150)

                {

                StopMoving();



                PVRdelay_ms(200);


                if(Fwd_greater_avg == 1)

                        {

                        GoRev();

                        RightTurnRev();

                        }


                else

                        {
```

```
                    GoRev();

                    LeftTurnRev();

                    }

          }


else if (RearIR >= 3000)

          {

          StopMoving();

          }


else if ((LeftFwdIR >=3100) && (Fwd_greater_avg == 2))

  {

  HardRightTurnFwd();

  }


else if ((RightFwdIR >= 3100) && (Fwd_greater_avg == 1))

  {

  HardLeftTurnFwd();

  }


else if ((LeftFwdIR >= 2800) && (Fwd_greater_avg == 2))

  {

  RightTurnFwd();

  }


else if ((RightFwdIR >=2800) && (Fwd_greater_avg == 1))

  {

  LeftTurnFwd();

  }
```

```
        else if ((LeftIR >=3100) && (Side_greater_avg == 2))
          {
          RightTurnFwd();
          }

        else if ((RightIR >=3000) && (Side_greater_avg == 1))
          {
          LeftTurnFwd();
          }

        else
                {
                GoFwd();
                }

        count ++;
        }

}

void BamaApproach(void)
{
float IR1, IR2, IR3;
GoFwd();
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Approaching");
PVRdelay_ms(2000);
```

```
while(1)
    {
            PVRdelay_ms(200);
            BamaSearch();

            IR1 = CenterIR_Read();
            IR2 = RightFwdIR_Read();
            IR3 = LeftFwdIR_Read();



            if(IR1 < 3000 && IR2 < 3200 && IR3 < 3200)

            {
                    while (IR1 < 3000 && IR2 < 3200 && IR3 < 3200)
                    {
                            BamaSearch();

                            if (middle_x > 55)
                            {
                                    RightTurnFwd();
                                    HeadCenter();
                                    TailCenter();

                            }

                            else if (middle_x < 25)
                            {
                                    LeftTurnFwd();
                                    HeadCenter();
                                    TailCenter();
```

```
                }
                else
                {
                GoFwd();
                PVRdelay_ms(100);
                }
        IR1 = CenterIR_Read();
        IR2 = RightFwdIR_Read();
        IR3 = LeftFwdIR_Read();
        }
}

else
{
StopMoving();
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Attack!!!");
PVRdelay_ms(4000);
for(int q =0; q< 4; q++)
        {
        PORTH_OUT = Chomp_Start;
        PVRdelay_ms(300);
        PORTH_OUT = Chomp_End;
        do
                {
                JawDown();
                PVRdelay_ms(200);
```

```
                                JawUp();
                                }while(PORTF_IN == 0X6F);

                        PVRdelay_ms(1000);
                        }

                PVRdelay_ms(2000);
                PORTH_OUT = Roll_Start;
                PVRdelay_ms(200);
                PORTH_OUT = Roll_End;
                PVRdelay_ms(2000);
                GotTarget = false;
                break;
                }
        }
}

void GatorApproach(void)
{
float IR1, IR2, IR3;
GoFwd();
lcdGoto(0,0);
LCD_Clear();
lcdGoto(0,0);
lcdString("Approaching");
PVRdelay_ms(2000);
while(1)
        {
                PVRdelay_ms(200);
                GatorSearch();
```

```
IR1 = CenterIR_Read();
IR2 = RightFwdIR_Read();
IR3 = LeftFwdIR_Read();
if(IR1 < 3000 && IR2 < 3200 && IR3 < 3200)
{
        while (IR1 < 3000 && IR2 < 3200 && IR3 < 3200)
        {
                GatorSearch();

                if (middle_x > 55)
                {
                RightTurnFwd();
                HeadCenter();
                TailCenter();
                }

                else if (middle_x < 25)
                {
                LeftTurnFwd();
                HeadCenter();
                TailCenter();
                }
                else
                {
                GoFwd();
                PVRdelay_ms(100);
                }

        IR1 = CenterIR_Read();
        IR2 = RightFwdIR_Read();
```

```
                IR3 = LeftFwdIR_Read();
                }
        }


        else
        {
        StopMoving();
        lcdGoto(0,0);
        LCD_Clear();
        lcdGoto(0,0);
        lcdString("Go Gators!!!");
        PVRdelay_ms(4000);
        PORTH_OUT = Fight_Start;
        PVRdelay_ms(200);
        PORTH_OUT = Fight_End;
        PVRdelay_ms(2000);
        GotTarget = false;
        break;
        }
    }
}
```