Balloon Buster

Justin Lu

EEL 4665C Fall 2010

Instructors: Dr. Antonio Arroyo & Dr. Eric Schwartz

TAs

Thomas Vermeer

Mike Pridgen

Tim Martin

Ryan Stevens

Devin Hughes

2. Table of Contents

3. Abstract

The Balloon Buster is an autonomous robot that looks for balloons and pops them. Pins are attached to the front of the robot so all it has to do is run into them to pop them. It is composed of two wooden circular layers and moves through two DC motors. IR sensors are used to avoid obstacles and a CMUCam1 is used to track colored objects. The entire system is controlled by an Atmel XMega 128 board. 8 NiMH batteries provide power for the entire system.

4. Executive Summary

The Balloon Buster's goal, as its name implies, is to locate balloons and pop them. It will avoid some balloons depending on the user. Which balloons it primarily targets are red balloons and balloons that have a red "target light" on them. Upon seeing a balloon it wants to pop, the Balloon Buster will move towards the balloon while attempting to center itself, then it pops the balloon through several pins located in front of the robot.

When it doesn't find a balloon worthy of popping, the Balloon Buster is in obstacle avoidance mode, where it moves forward it until finds an obstacle, whereupon it turns left or right, depending on which side of the robot the obstacle is closer.

The Balloon Buster consists of two circular pieces of wood that hold all functional hardware, which consists of a motor controller, two IR sensors, an LCD for debugging, a CMUCam for color tracking, and an XMega128 board. It moves through two DC gearhead motors, which is controlled through PWM signals sent to the motor controller.

The Balloon Buster uses two IR sensors for obstacle avoidance, and a CMUCam1 that allows for tracking colored blobs.

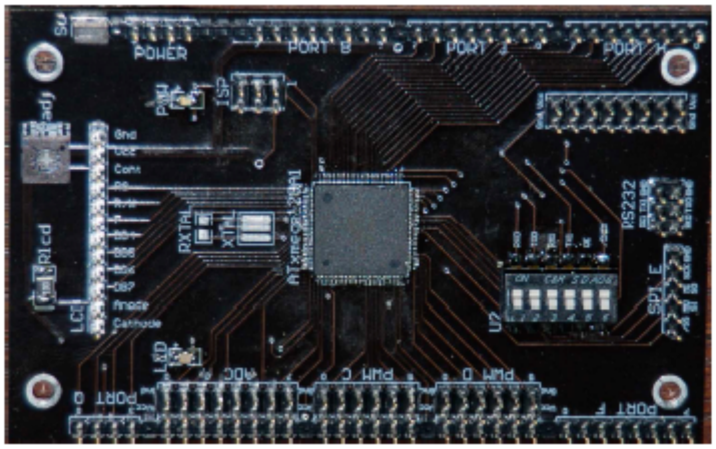
5. Introduction

Robots have always been an interesting subject for me, but I had no idea how to get started. Because this is my first foray into the realm of robotics, I decided to go with a project that didn't involve a lot of mechanical design, yet was complex enough to suit the needs of the IMDL class. It also needed to be fairly cheap as there are I have other classes in the future where I will have to worry about expenditures. After much contemplation, I decided to go with a search and destroy robot.

The robot would have to be entertaining since I would need to show it to others in the future. I eventually came up with the idea of popping balloons since everybody likes balloons and they're meant to be destroyed, while not causing any harm or monetary damages.

The fact that balloons come in varying colors presents some interesting possibilities in terms of color tracking. The idea was to use a camera for color tracking then proceed to pop the balloon once found. I had some different ideas about how the robot would pop the balloons, like using an arm to pick them up then another one to pop them with a pin, mostly to add some complexity to the project, but in the end simplicity won, and I ended up attaching several pins to the front of the robot, which meant that to pop them, it would simply have to run into them and they would be no more.

6. Integrated System

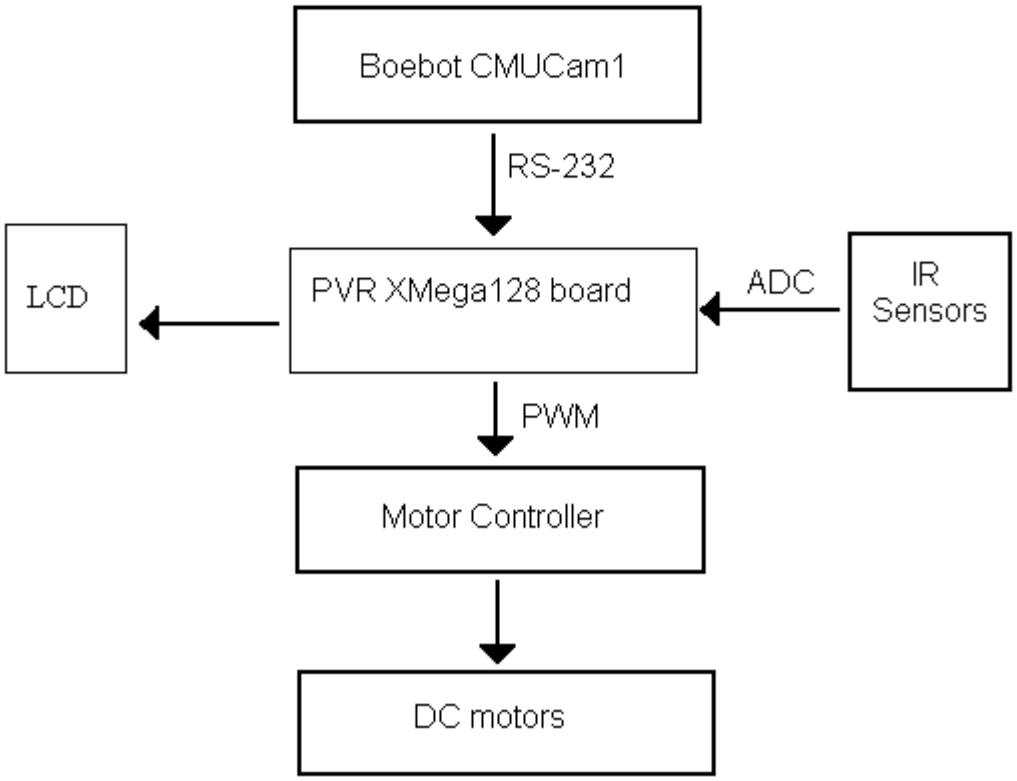


At the brains of the operation is the Pridgen-Vermeer Robotics (PVR) XMega128 board. This board uses an Atmel XMega 128 microprocessor and was designed with robotics in mind. It comes with ports that allow for pulse width modulation, serial communication using either TTL or RS232, A to D conversion, and interfacing of an LCD.

The IR sensors will be used for obstacle avoidance and are connected to the ADC ports on the PVR board. They collect data which is output to an LCD which the robot uses to make decisions.

The Boebot CMUCam1 is used to notify the robot if it sees a color within a range specified by the user. It is connected to either port F for TTL, or port E for RS232. RS232 allows for the ability to connect to a computer to view the output from the camera to a terminal or dump a frame to a computer. This data can also be output to an LCD for viewing when the robot is running.

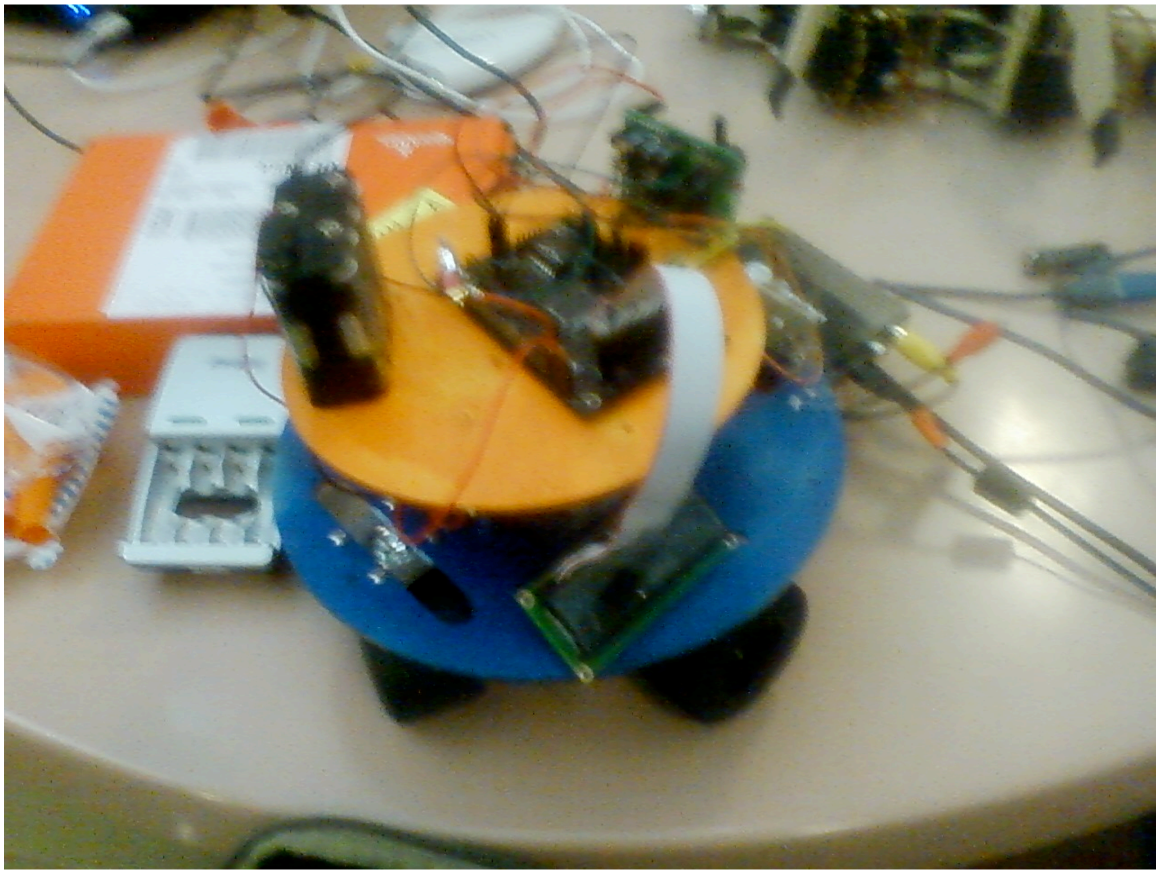
Finally, based on the input from the camera and IR sensors, the PVR board outputs PWM signals that are sent to a motor controller, which uses them to control two DC motors. These motors provide the necessary actuation to keep the robot moving.



7. Mobile Platform

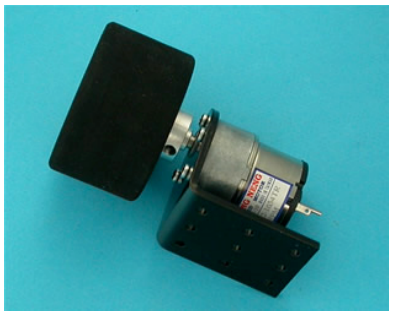
For simplicity's sake, I chose to do a dual circular platform. Two layers were chosen to make enough room for all the components. One layer is a 9 inch diameter circle and the other is 7 inch diameter circle. Both are constructed from balsa wood. Several holes were drilled into both platforms to allow wires to go through The two circular components were connected using several standoffs and epoxied to prevent them from separating. Finally, a rectangular 4 x 1.05 inch piece was used to mount two IR sensors, and was attached to the front of the bottom layer. Pins will also be glued to the front of the robot using epoxy.

The platform is spray painted orange and blue to show support for the university. This was done very late in the semester, and I managed to get some paint onto the motor controller in the process. Fortunately, it has not had any negative effects on the performance of Balloon Buster.



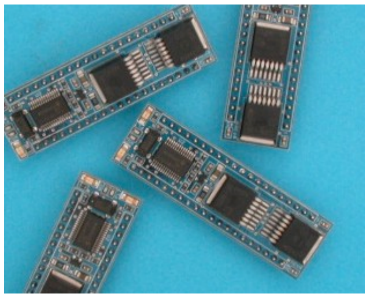
The final product

8. Actuation

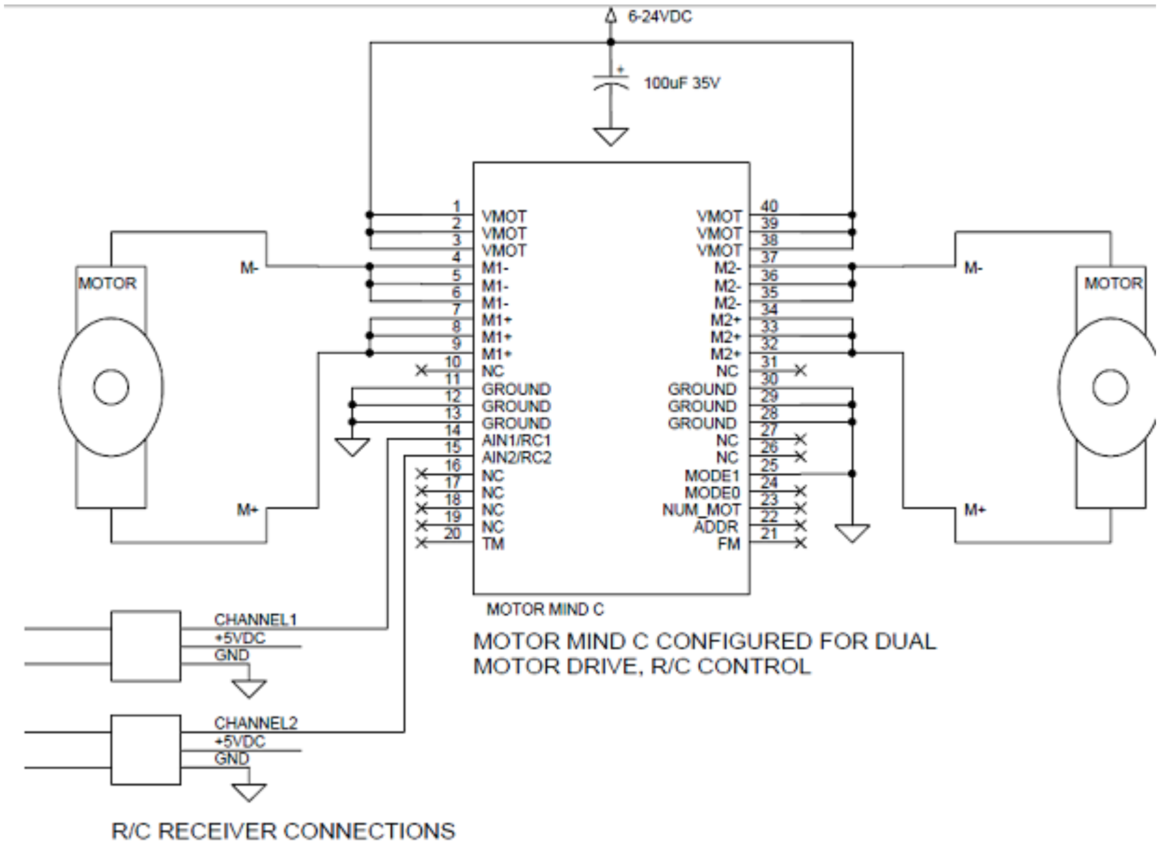


The robot moves through two wheels each driven by a DC gearhead motor. A motor controller is used to control both motors through PWM. Two additional caster wheels were added to support the robot since it had trouble maintaining balance without them.

The motors have a 3.6 kg cm torque rating and 200rpm and have a gear ratio of 30:1. They operate at a good range of speeds which allows for some flexibility when deciding an appropriate speed. They also run smoothly and do not produce much noise when running.



The motor controller used is the Motor Mind C. It was chosen because it was made by the same people whom I got the wheel and motor set from, so I figured it would be the best choice for controlling the motors. The Motor Mind C has 3 different modes of operation. R/C mode was chosen because it uses PWM to control the motors, which is easy to do.



Setup for R/C mode for motor controller and motors. Channel 1 and 2 correspond to the PWM pins on the PVR board.

9. Sensors

- IR sensors

Two short range Sharp GP2Y0A21YK IR sensors are used to detect obstacles in the robot's path.

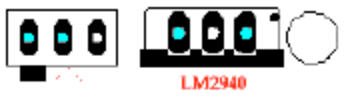
These sensors are rated for 5V so to get an accurate reading Vcc must go to 5V (on the PWM part of the PVR board), GND goes to GND, and Vo can go to any of the ADC ports. A jumper wire connector will also need to be purchased in order to attach the sensor to the PVR board. From testing two of them they appear to be able to detect objects from up to 20 inches.

- CMUCam 1

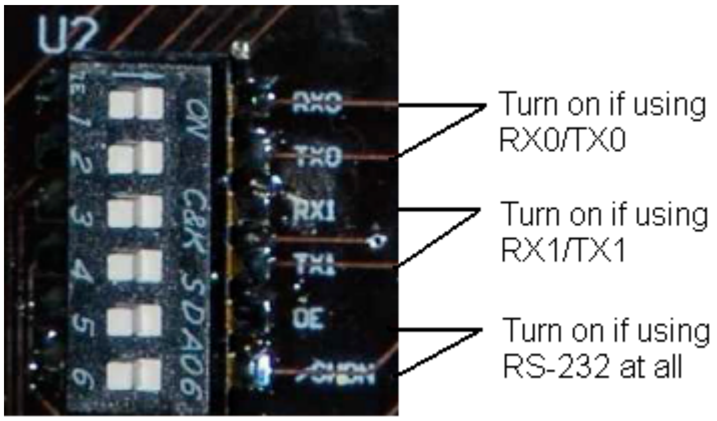
The CMUCam1 will be used to track a color. Due to the main source of CMUcams discontinuing the product's sale, I resorted to ordering a Boebot CMUCam from parallax.com, which had a setup made specifically to tailor to one of their products. In order to make it usable, some other parts had to be ordered to match the diagram below, the main part being the MAX232 chip that allows the camera to communicate with the processor through RS-232. Getting a MAX232 chip is not necessary to get the camera working, since the TTL configuration for the camera works fine on the PVR board, but if you wish to dump frames to a computer to see what the camera sees, then a MAX232 chip is needed.

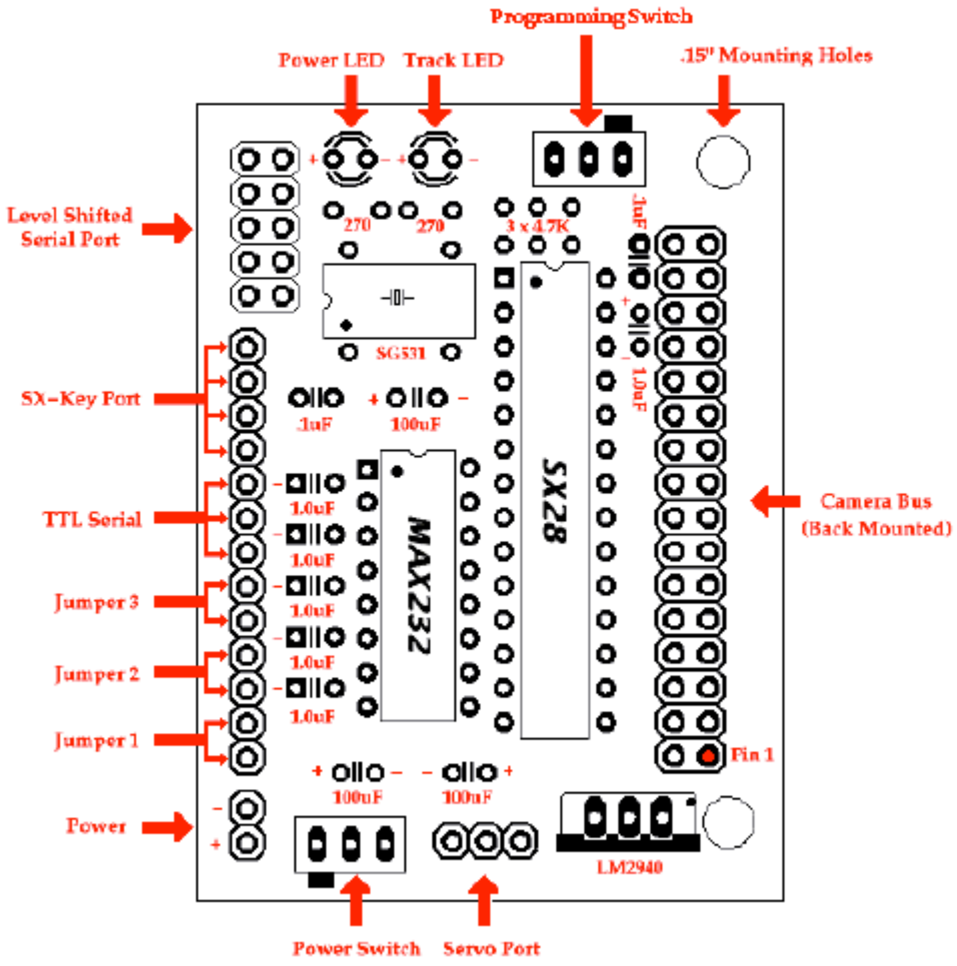
The TTL configuration just needs the SX-Key Port (for power) and the TTL Serial port (port F) on the PVR board. This is recommended if one does not wish to do any extra soldering or setup to connect to a computer.

In order to get the camera working through RS-232, a few parts need to be added. Several capacitors will need to be soldered on the camera board (See below figure, some parts may already be on there). Secondly, a connection will need to be made between some pins in the power switch and voltage regulator ports. For the power switch a jumper will work, while for the regulator soldering a jumper wire will do. Finally, the DIP switch on the PVR board will need to be turned on, depending on which RX/TX connection is used on Port E, that corresponding switch will need to be turned on, as well as the OE and /SHDN switches on the PVR board. Once all that is done, the hardware configuration portion of the camera is complete.



Connections for power switch and regulator port that need to be made (required pins to be connected in light blue)





Setup for CMUCam1. This figure shows the capacitor values necessary for RS232

For the software portion, I looked at previous semesters of people who used a CMUCam and found that most people had similar, if not the same code for initializing the camera and the USART, so I used that code and it worked for an RS232 connection. The same code works for TTL but any references to port E will need to be changed to port F. (i.e. PORTE.DIRSET becomes PORTF.DIRSET, USARTE0_STATUS becomes USARTF0_STATUS, etc)

The main command used for the camera is the TC (Trackcolor) command. By specifying a range of colors in RGB 24 bit format, the camera will track a color in that range. Once it detects such a color, the green tracking LED will light up as a notification. A packet of data will also be outputted, and can be written to an LCD to view. From this packet, the number of interest is MX, which outputs an integer between 1 and 80. If MX is around 80, then the color is to the left, otherwise it is to the right. This is useful because it tells the robot where the color is in relation to the camera, and so necessary adjustments can be made.

10. Behaviors

Using sewing needles attached to the front, the robot will pop balloons by running into them. There are at least two pins on 3 different heights, so popping a balloon should not be an issue.

A red light will be shone onto a balloon, as it is one of the easier colors for the camera to pick up. Depending on the lighting conditions, the camera can pick up a red balloon without any extra help.

The robot will move forward until it either sees red, or it runs into an obstacle. If it sees red, it will attempt to center onto it (using the MX data to calibrate itself) then move forward until the red disappears (meaning the balloon has popped). If it runs into an obstacle, the robot will turn left or right until the object is out of sight. Whether it turns left or right depends on the proximity of the object, and is determined by comparing the two IR sensors and seeing which one holds a bigger value.

11. Experimental Layout and Results

When testing the IR sensors, I had to determine what readings were appropriate for the robot to start turning and obstacle avoiding. The values varied depending on what speed the robot was running at. In the end after running the robot and looking at the IR readings I decided that with the speed I was using, an average reading of 2000 on both sensors, which is about 7 inches away from the sensor, would suffice for seeing an obstacle.

The first thing to do with the CMUCam before using it is focus it. I managed to do this through the help of a fellow classmate who was able to dump frames to their computer. I later ended up readjusting the camera to see of I could get better readings.

Since I didn't have the cable needed to dump frames to a computer, I used the TC command to send output to an LCD while having a colored object in front of the camera and adjusted it based on the readings on the LCD.

The CMUCam's performance relies heavily on the surrounding lighting conditions. At first I wanted the camera to find balloons without any help. I tested it with the primary colors in the lab room during the day, where white light is everywhere, and found the camera responded best to red. At nighttime the camera could not track colors at all unless they were really bright like LEDs, so I went to Walmart and bought a Coleman Multi-color LED headlamp to use for testing during the night. This device shines white light of varying intensities, and can also emit red and blue light. The camera responded well when white light was shone on a red balloon, and the same can be said for other colors. Since most of the demos were during the day I decided to test the headlamp during the day. Shining red light onto a non-red balloon caused the camera to pick it up, depending on the proximity of the lamp, which is about 4-5 inches away from a balloon. This allows the robot to pop balloons that are not red provided a red light is shone on them at an appropriate distance and angle.

12. Conclusion

By media day, I was able to get the robot to dodge obstacles without injuring anybody and have it popping balloons that I specified without me touching the robot except to turn it on and off. I had different colored balloons each representing a different college and was able to get the robot to pop all of them save for the one representing UF. Overall, I felt this project was a success.

The robot still needs some assistance in order to pop colors that are not in a specific color range, which limits its effectiveness. Getting the robot to rotate in place also proved troublesome since one of the motors would move at full speed when the other one was moving slowly, even though the PWM arguments did not indicate the motor would move as such, which meant I had to improvise the behaviors to get it to do acceptable obstacle avoidance.

If I were to start over, I would have planned for several weeks before buying all the parts I needed. Several times during the semester I realized I needed to get part x or part y too many times which cost me too many trips to Radioshack, Walmart or Lowes than was necessary. I must have done this about 15 times for just this semester alone, which could easily have been reduced to 2 or 3 with careful planning. Switching to hacked servos would be a cheaper alternative because while the DC motors have their advantages, the job could be done just as well as if I had used hacked servos. Some possible enhancements could include adding a nerf gun on the robot and be able to control it, and having it target other things besides balloons. Maybe a better sensor with a better detection range than the CMUCam could be used to find objects. Finally, making more use of Solidworks' extra functions could have saved me some trouble in figuring out what kind of caster wheels I needed or even how the wheels could be positioned so that I wouldn't need caster wheels at all.

Overall, this was a satisfying experience. I would definitely recommend this class to people who aren't fond of sitting in a classroom and listening to professors lecture, and are looking for an interesting engineering endeavor. It provides first hand experience and insight into starting a real engineering project, and shows how important planning in advance really is and the time it can save you.

Finally, some advice to future students

- Plan ahead. Figure out what your robot will do as early as possible, look at past semester projects to determine what you'll need to order, be it sensors, motors, servos, or other circuits, and order them as soon as possible. You will want to order extra components in case something fails, which saves you from having to order another set of parts and getting delayed another week, not to mention those shipping costs can add up quickly!

-Familiarize yourself with the tools commonly used. You may want to buy these and work on your robot at home instead of working only in lab. Some useful tools/things to buy include a crimper, wire stripper, screwdriver, soldering iron, rechargeable batteries, a battery charger, epoxy/tape/other adhesives, nuts and bolts, and an extra battery pack. You may not use them all, but you may in the future, and it never hurts to have them in advance.

-Everyone else is going to tell you this, but don't procrastinate. Get the basis of your robot done weeks ahead of time, so you can spend the days before demo day just polishing things up. The end of the semester gets really stressful especially for engineers, when every other class is giving projects that are due the last few weeks of the semester.

-If you choose to cut parts out through T-tech, make sure you know the dimensions of the wood that you are allotted, or you will have to redo your design. The biggest wood piece was around 10 by 14 inches. It may change in the future, so check to make sure your designs are within the dimensions of the biggest piece of wood allotted.

-When handling epoxy or other adhesives, use gloves. It takes a while to get rid of that epoxy smell if you get it on you.

-Talking to people helps. Chances are they are having similar problems as you, and putting together more heads can help overcome it.

-If your program does not run after programming your board, sometimes resetting the robot will fix it.

-Try to have all your connections have a common ground. Some components may not work otherwise.

-Sometimes, the simplest solution is the best.

13. Documentation

http://www.sparkfun.com/ - IR sensors and LCD

http://www.solutions-cubed.com/Solutions%20Cubed/index.htm - Motors and motor controller

http://sites.google.com/site/projectcrawlbot/design - source for USART code (usart.c, global.c, global.h, and usart.h)

http://www.digikey.com/?curr=USD – MAX232 chip

http://www.parallax.com/StoreSearchResults/tabid/768/txtSearch/cmucam/List/0/SortField/4/ProductID/29/Default.aspx - Boebot CMUCam

14. Appendices

Code

```
#include <avr/io.h>
#include "PVR.h"
#include "usart.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("Balloon Demo");                  //display "Board Demo" on
second line
        delay_ms(5000);
        int sensorsum, avg, count;
        count=0;
        sensorsum=0;
        avg=0;
```

```
	int time=0;
	USARTInit();
	while(1)
	{
		//ADC0 is left IR sensor, ADC2 is right
		/*//IR sensor test
		while(1)
		{
			lcdInit();
			lcdInt(ADCA2());
			lcdString(" ");
			lcdInt(ADCA0());
		}
		*/
		/*//Timer test code, used to determine values of loop in relation to real
time
		while(1)
		{
			lcdInit();
			time=time+1;
			lcdInt(time);
		}
		*/


		/*/Wheel/Motor test code, check direction wheels turn and at what speed
		while(1)
		{
		for(int d=700; d<800; d++)
		{
			lcdInit();
			lcdInt(d);
			ServoC0(-15);//Left wheel
			ServoD0(d);//Right wheel
			delay_ms(400);
		}
		}
		*/

		//debugging code, test all useful functions
		/*
		while(1)
		{
			RotateLeft();
			delay_ms(200);//about 180 degrees
			stop();
		delay_ms(5000);
		TurnRight();
		delay_ms(1000);
		TurnLeft();
		delay_ms(1000);
		Movebackward();
		delay_ms(2000);
		Moveforward();
		delay_ms(1000);
		stop();
		delay_ms(2000);
	}
	*/
	//end debugging code




	Colortrack();

	//find color
	while(buffer[2]>=1 && buffer[2]<80)
	{

		Colortrack();
		delay_ms(100);
		if(buffer[2]>=36&&buffer[2]<=46)
		{Moveforward();
		delay_ms(800);}
		else if(buffer[2]>=1&&buffer[2]<36)
		{TurnRightS();
		delay_ms(100);}
		else if(buffer[2]>46&&buffer[2]<80)
		{TurnLeftS();
		delay_ms(100);}


	}
	//Obstacle avoidance
	while(buffer[2]<1) //don't see target color
	{
		int r=random();
		Colortrack();
		Moveforward();
		lcdString(" ");
		lcdInt(r);
```

```
                        while(  ADCA2()+ADCA0()>4000) //obstacle found
                        {
                                delay_ms(100);
                                if(ADCA2()<ADCA0())//object is closer to the left
                                {TurnRightS();
                                delay_ms(2000);
                                }
                                else //on the left
                                {TurnLeft();
                                delay_ms(2000);
                                }
                                time=0;
                        }

                        sensorsum=0;

                }




        }
}

void Moveforward()
{
        ServoD0(700);
        ServoC0(-20);
}

void Movebackward()
{
        ServoD0(850);
        ServoC0(40);

}

void RotateLeft()
{

        ServoD0(1000);
        delay_ms(100);
        ServoC0(90);
        delay_ms(100);
```

```
}

void RotateRight()
{

        ServoD0(400);
        delay_ms(100);
        ServoC0(-100);
        delay_ms(100);


}
void Slowrotate()
{

        ServoD0(0);
        ServoC0(15);


}
void TurnRight()
{ServoC0(-20);
ServoD0(0);
}
void TurnLeft()
{

        ServoD0(700);
        ServoC0(0);
        ;
}
void TurnRightS()
{ServoC0(-15);
ServoD0(0);
}
void TurnLeftS()
{

        ServoD0(715);
        ServoC0(0);

}
void stop()
{
ServoD0(0);
delay_ms(100);
```

```
	ServoC0(0);
	delay_ms(100);
}

void Colortrack()
{
		lcdInit();
		TrackColor("TC 100 255 0 31 0 31\r"); //track red
		//TrackColor("TC 0 31 80 255 200 255\r"); //track green
		//TrackColor("TC 0 31 130 255 130 255\r"); //track yellow
		lcdInt(buffer[2]);
		lcdString(" ");
		lcdInt(ADCA2());
		lcdString(" ");
		lcdInt(ADCA0());
		lcdString("   ");
		int g=ADCA2()+ADCA0();
		lcdGoto(1,0);
		lcdInt(buffer[6]);
		lcdString(" ");
		lcdInt(buffer[7]);
		lcdString(" ");
		lcdInt(buffer[8]);
		lcdString(" ");
		lcdInt(buffer[9]);
		lcdString("   ");
}

//PVR.c, PVR.h, global.h, global.c, usart.c, and usart.h will be left out as they are other people's
//code and can be found in Seon Kim's website linked in the documentation section. I did not
make any edits to these header files except change one function in PVR.c to

//void ServoD0(int value)
//{
//	TCD0_CCA = (value);			//Generate PWM.
//}

//This was to test one of the motors and I saw no need to change it back
```