Final Report

Robocop

Mark Kampfer

EEL 4665/5666

IMDL Fall 2011

Professors:

Dr. Schwartz

Dr. Arroyo

Teaching Assistants:

Tim Martin

Ryan Stevens

Contents

I. Abstract

Robocop is a robot that will search for and fire a ping pong ball at a person wearing a certain color of shirt. Once the robot is started up, it will patrol around searching for people to fire its ball at all the while using IR sensors to avoid any obstacles it may come across. If the robot doesn't come across a person for a period of fifteen seconds or if the IR sensors detect an object within fifteen cm, the robot will come to a halt and turn to the right and continue on in the new direction once again searching for that specific color of shirt. Once the robot finally detects the correct blob of color it will check to see if the target is within firing range and adjust by moving the ball launcher up or down. Since the robot is now in range of its target, it will fire the ping pong ball using a compressed air launcher that is controlled by a solenoid sprinkler valve. The compressed air for this launcher will be in the form of a 20 inch long, 1.5 inch diameter PVC tank attached to one end of the sprinkler solenoid and will be filled up via a bike pump or an air compressor.

II. Executive Summary

Robocop is an autonomous robotic agent that will patrol the area that it is placed in all the while searching for "criminals" that wear a certain color of shirt using an IP camera in conjunction with Xbee and OpenCV. Once the criminals are identified, Robocop will begin adjusting its position, by moving left or right, so that the target is in the center of the camera's field of vision. After that has been accomplished, the robot will than raise or lower the arm that holds the ball launcher so that it will hit the target dead center and a command will be sent to a relay switch circuit that will allow the necessary 27 volts to trigger the solenoid and cause the air to leave the chamber of the ball launcher and fire the ping pong ball at the target. Once the projectile has been launched, the arm will lower back to the resting position and resume its patrol, searching for the next criminal to apprehend.

Robocop begins by driving around using the two front mounted IR sensors to avoid obstacles until the IP camera has successfully synced with the router and by extension the computer that will be running Opencv and bob detection. Two bump sensors on the right and left sides of the robot will also be used as last ditch efforts to keep the wide turning Robocop from ramming into and object such as a chair or table leg or a human leg. Once the camera has connected, the IR obstacle avoidance will continue until the camera sees an image of the desired color that is large enough to be made into a blob. Once this blob is found, IR avoidance will cease and the only thing that will be moving the robot is commands sent through Xbee to the microcontroller telling it whether or not the object has been centered, these commands will tell the robot to move left, right, or that the object is centered. Once the object is centered, the robot will stop moving and raise the launcher arm from the default servo setting of ninety degrees to approximately seventy degrees, pause for a second, then send a command to PORTD to close the Relay switch which will send twenty-seven volts to the solenoid and as a result propelling the ping pong ball out of the chamber and at the target. Once the target has been fired at, the robot will resume patrolling.

The two main components of this robot are the actual physical robotic platform and the laptop and router combo that will run the Opencv blob tracking that will allow the camera on the robotic platform to see and direct the robot. The robotic platform and the router and laptop will communicate with each other through the use of the wireless Xbee device, with once being attached to the bottom of the microcontroller board and the other connected to a USB port on the laptop. The wireless IP camera will send the data on what it sees to the router which will be transferred to the laptop. Once the laptop has this information it will send the position data of the tracked blob back to the microcontroller via the Xbees.

III. Introduction

Background

The idea behind Robocop is to essentially build a robot that will protect its owner from harm by identifying and firing upon those that the owner deems to be enemies. While there are many robots around I could not find any on the internet that made use of a pneumatic ball launcher as it's' means of defense, the only ones that I could readily find were those that held a dart gun or used a catapult-esque contraption. This presents a unique twist to the otherwise very common sentry robot idea.

Scope and Objectives

Robocop will utilize two front mounted IR range finders, which will detect when an object is within 15cm of an object and avoid collisions by stopping it's movement and turning to the right and continuing on. During this time, Robocop will also actively search for passing people and analyze the colors of their shirts using an ip camera that wirelessly sends information to a PC running OpenCV. Once the robot detects the correct person, via their shirt color, it will adjust its position so that it is within range and launch a ping pong ball at the target.

References to the literature

The cv blobs lib and OpenCV were combined to create the blob tracking that was required to search for a specific color, center it, and keep out miscellaneous errors. The IP camera searches for a color and once it finds that color it creates a box aroud this blob and the IP camera then keeps the center of this box in the center of its vision with help from the movements of the robot. Tutorials on blob tracking can be found here:

Useful OpenCV blob tracking tutorials:

- http://www.technical-recipes.com/2011/tracking-coloured-objects-in-video-using-opencv/#more-1348
- http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/

IV. Integrated System

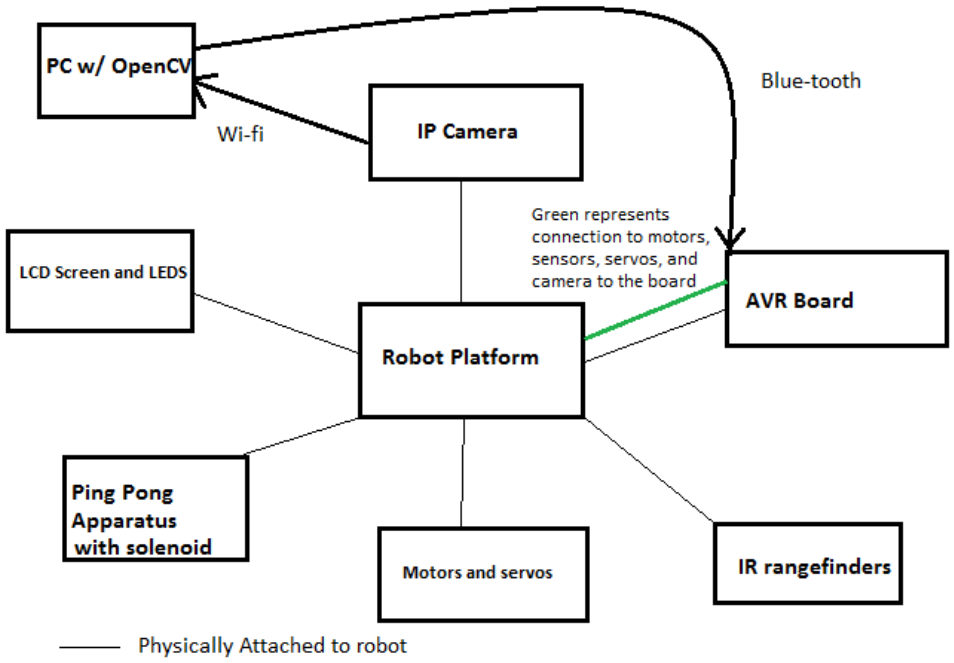


Figure 1: The Integrated System

The integrated System, as shown above, is made out of:

- The Epiphany DIY board which is used to control the motors, servos, IR sensors, IP camera, solenoid valve and the LCDs and LEDs.
- 3 9V batteries that run through a Relay to control the solenoid valve
- PVC pipe attached to a solenoid valve with the other end being connected to an air chamber that will be will via a bike pump.
- Two DC motors to power the rear wheels of the mobile platform.
- Two short range IR rangefinders to be used for obstacle avoidance.
- An IP camera to wirelessly send information from the camera to a PC equipped with OpenCV which is in turn sent via Bluetooth to the Epiphany DIY board.

The DC motors will propel the robot while it uses the IR sensors to detect objects for it to avoid. While the robot is patrolling, the ip camera will be sending information from the camera to an OpenCV running computer which will then scan the sent images for the correct colored blob and draw a rectangle around it indicating the target area. Once the target area is drawn, the computer will send the information back to the

Epiphany DIY board via a Bluetooth connection. From there, the robot will back up or move forward to get within a suitable range of the target. Once the target is within range, the Epiphany board will send a signal to a relay circuit which will then close and allow the voltage source of the solenoid valve to open and release air into the barrel of the ball launcher. After the ball has been launched the valve will close and the robot will resume its patrol.

V. Mobile Platform

The mobile platform will consist of wood parts that will be designed in SolidWorks and cut out and put together in the lab. The main platform will be shaped like a rectangle to keep things simple and allow maximum space. On this board there will be places to mount the ip camera, sensors, and the motors along with a holster for the ping pong ball launcher which is essentially a plank of would attached to a servo which will move the ping pong ball launcher ip and down. The platform will have 3 wheels two of which will be connected to motors and the other wheel will be free spinning to increase maneuverability. The ip camera will be placed at the front of the platform to one side to the launcher and will be in a fixed, forward facing postion.

VI. Actuation

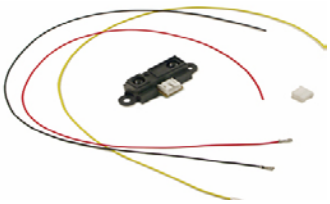
The Actuation for transportation of this robot will be 2 99:1 Metal Gearmotor 25Dx54L mm HP metal gearmotors attached to the rear wheels of the mobile platform with the caster wheel being place at the front. The motors will be powered at 6V which will result in 100 revolutions per minute and 450 mA free-run. The motors will be controlled using the PWM on the Epiphany DIY board.

A GWS S689-2BB/MG/JR Giant Servo will also be used to maneuver the launching apparatus from its resting position to an angle of approximately 25 degrees since this launcher has an incredible power, the standard 45 degrees will not be necessary for this machine to fire a significant distance. This particular servo has a speed of 0.157 sec/60degrees and a stall torque of 26 kg*cm at 4.8 V. This massive servo was chosen because ball launching device was very heavy and anything less would not have been able to move the launcher or would have caused it to stall or break.

Actuation will also be used to control the opening and closing of the solenoid sprinkler valve that is attached to the ping pong ball launcher. Attached to solenoid valve and the 3 9 volt batteries that will be powering it is a Relay and 2n2222 transistor that will act like a switch. When this switch is open, the valve will remain closed and no air will escape into the ping pong

ball chamber but when the switch closes, the valve will open and air will rush out and push the ball out of the tube with the Epiphany DIY board controlling when the switch opens and closes.

VII. Sensors



Sharp R146- GP2D120 IR range-finders

Robocop will utilize two of these close range IR sensors have a range of 4 cm to 30 cm and will be used exclusively for obstacle avoidance. The sensors will output a different voltage depending on how far they are from the object, if the object is 4 cm away than it will output 3.1V and if it is 30 cm away it will output 0.3 V. If the robot detects a object within15 cm it will stop and turn to the right and resume moving forward. The IR will both be mounted on the front of the robot at each corner and they will operate independently, if one is to clsoe to an object then the robot will start obstacle avoidance because of the one.

Bump Switch

Several bump switches will be used on the sides of the robot as failsafe obstacle avoidance and so that the robot doesn't attempt to ram into a wall when turning since it has a large turning radius becaue of its 20 inch length.

IP Camera

The IP camera will be a Foscam FI8918W Wireless/Wired Pan & Tilt IP/Network Camera with 8 Meter Night Vision and 3.6mm Lens (67° Viewing Angle) that will be used to send images of passing people to my computer. There, the image will be analyzed in search of a specific color of T-shirt. The IP camera will utilize OpenCV blob detection to search the relayed images for the correct shirt color. If the camera sees a blob of the correct color than it will draw a rectangle around the blob and the information that it is the correct shirt color will be sent back to the robot which will then commence firing.

XBee 1mW Chip Antenna - Series 1

Two of these Xbee chips were chosen in order for the DIY board to transmit and receive data from the computer that will be running OpenCV. This version was chosen because the range requirements were not extreme enough to warrant using the Xbee that has an antenna sticking out of it.

VIII. Behaviors

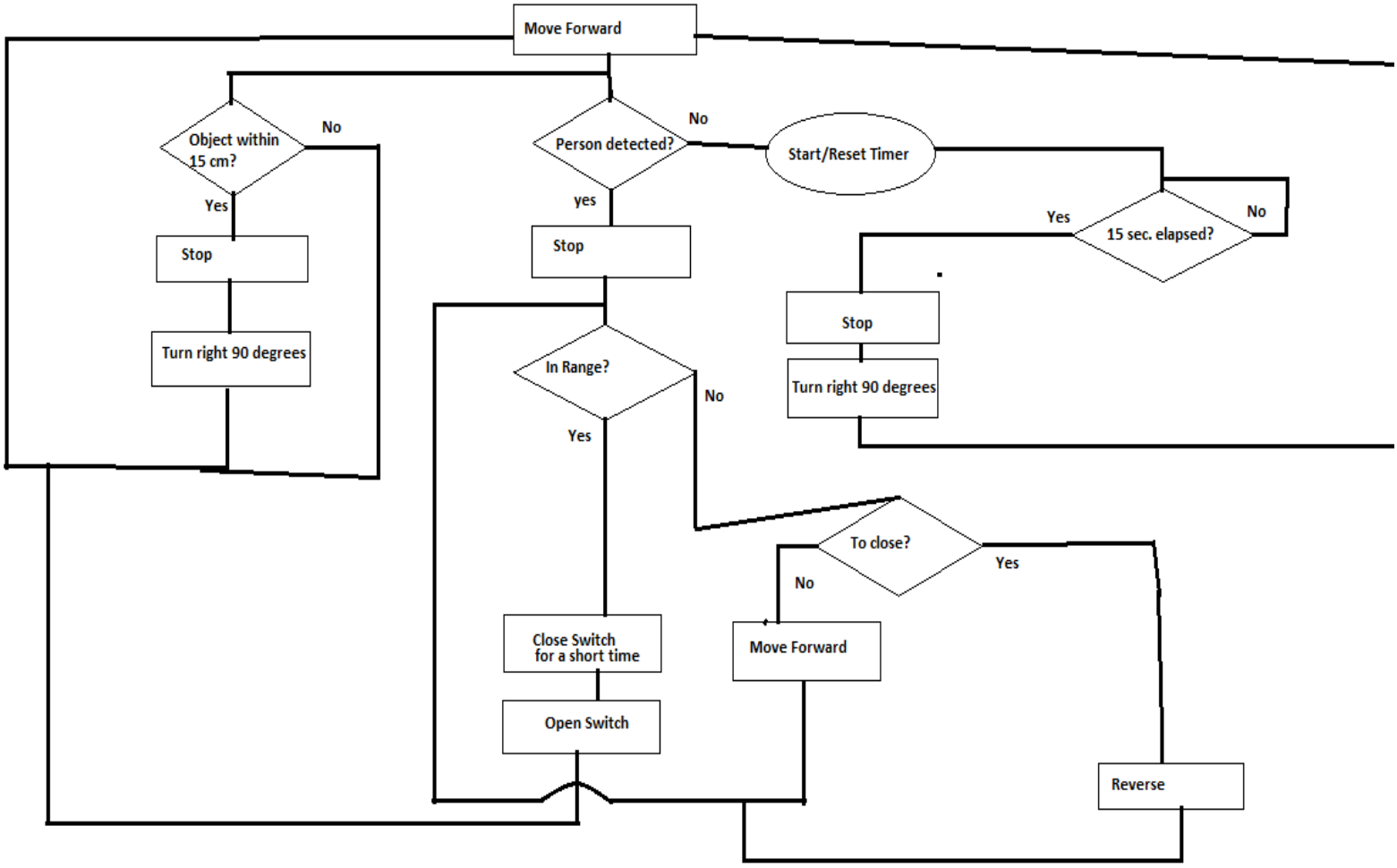


Figure 2: Robocop Behavioral Flow chart

The above flow chart has been designed to show how the robot will behave in certain situations. When the robot is activated it will begin moving forward at a predetermined speed using the IR sensors to detect objects and the ip camera to search for people. If the camera identifies a person then it stops checks to see if it is within range and adjusts accordingly. Next,

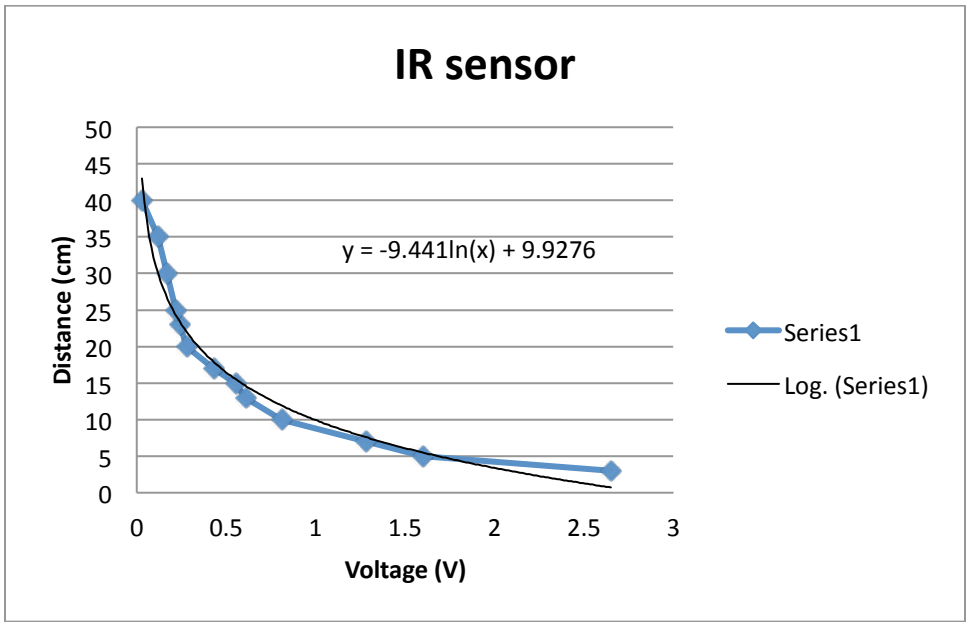
Relay circuit will be closed and power will be sent to the solenoid valve which will release air into the chamber of the ball launcher. After a very short amount of time has elapsed the switch will be reopened to prevent a waste of the compressed air that is propelling the balls and the robot will return to moving forward. Also, if Robocop detects any objects using the IR sensors that are within 15 cm then it will stop, turn right 90 degrees, and resume moving forward. If the robot does not detect a human than an internal timer will start and once 15 seconds of not detecting any human presence, the robot will stop, turn right 90 degrees and continue straight on in the new direction. This was done in order to prevent Robocop from going a significant amount of time without seeing any humans.

IX. Experimental Layout and Results

Most of the experimentation on this robot took place by setting it on top of my tool box and placing either my hand in front of the IR sensors to test the obstacle avoidance or placing a blue colored object in front of the camera to test the centering capabilities of the IP camera and robot motor combination with the solenoid detached from the launcher and the launcher not present on the robot. This was done mainly due to the fear that the epoxy sealing the bike valve to the air chamber would rupture and force a reconstruction of the entire chamber, which, is what happened on demo day. Unfortunately, this means that the true maximum range of this launcher is unknown but even with few tests that were conducted it can be seen that at 70 psi this launcher is capable of shooting well beyond the range of the ip camera.

Obstacle avoidance:

The obstacle avoidance capabilities of the sharp IR sensors were tested by first setting the robot on the ground and turning it on. The two IR sensors were then set so that if the values read from the ADC were less than 450 the robot would assume there is nothing in front of it and continue moving forward until a change occurred. If one of the IR's registered a value above 450, then there was an object in front of that side of the robot and it then turne din the opposite direction so as to avoid the obstacle. After turning, the robot will continue going straight until another obstacle is met or the IP camera picks up something. Obstacle avoidance was also achieved through bump sensors which, if pressed caused the robot to immediately turn away from the pressed switch because the robot had hit something. However, there was a pretty decent delay between when the switch was pressed and recognition by the robot which was determined to be fine since the robot never had to use the bump sensors anyway and the robot doesn't move fast enough for it to be terribly hurt by hitting a wall with its' side. If at any point during this obstacle avoidance the IP camera picked up on the tracked color than it would enter into the OpenCV blob tracking routines as described below. A graph of the distance vs. voltage values for the IR sensor can be seen below.



IP camera experiment:

The IP camera experiment was performed by first hooking the Xbees up to the PC and to the microcontroller. Next, the IP camera was powered on and connected to the same router that the pc was connected to. Visual Studio 10 was then launched and the OpenCV blob tracking program was run. What popped up were the three windows seen below, one show the posX data, and the others show what the camera is seeing in the threshold and normal modes. Once a blob of sufficient size is seen by the camera, a box is drawn around it and tracked via cv blobs lib tracking methods. Once the blob is formed values are sent from the pc to the microcontroller that tells it whether the blob is centered or not. If the blob isn't centered than the robot will turn either left or right until it is which then triggers the Relay switch and launches a ping pong ball at the center of the blob. For these camera experiments, it was set so that if the posX data was greater than 215, the object was on the right side of the robot and if the object was less than 190 it was to the left side of the screen, as seen in the code in the appendices. A range between 190 and 215 was chosen as the center because the robot isn't the most precise of machines and it would have taken far too long for the target to be centered if the range was any less especially since there is still a small amount of lag between object movement and it being registered on the camera.

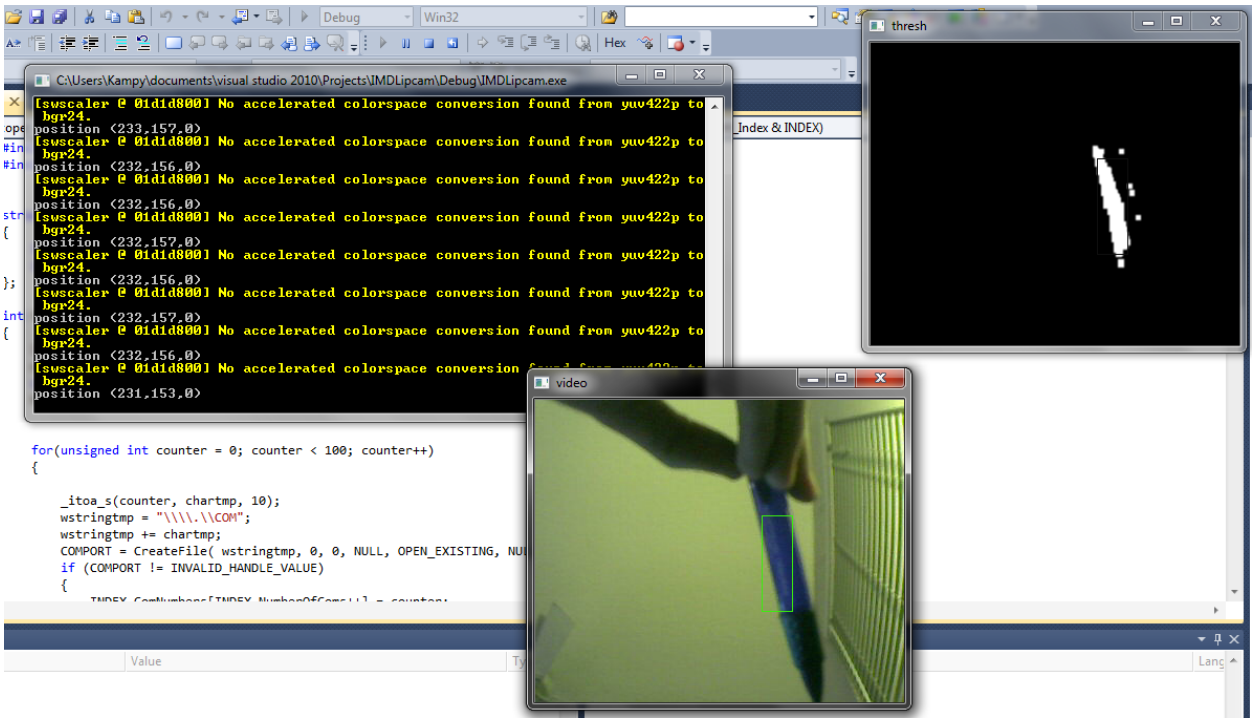


Figure 3: OpenCV output windows

X. Conclusion

Work Accomplished:

One of the many things that were accomplished by this robot was that the IR obstacle avoidance worked perfectly, the construction of a working robotic platform to house the motors and other components as well the use of Xbee devices to communicate between the microcontroller board and the PC running OpenCV. Another accomplishment was the successful implementation of a behavior into the robot that caused it to patrol the area searching for a specific color using the IP camera. Once the color was found the robot successfully stop, centered the object, and then adjusted the ball launchers position and fired on the target fairly accurately. Thanks to Chris Dobson for helping me understand how to properly set the comports to work with Opencv and providing me with some code to help me on my way as well as Tim Martin for providing much of the motor, servo, LCD and ADC code which made it possible to complete this robot in such a short amount of time.

Limitations of Work:

While Robocop works well for the amount of money that was available to spend on it, many improvements could have easily been made. First off, the IR sensors were very cheap and sometimes there will be spikes in that data that cause Robocop to believe there is an object in front of one of the sensors when there is not. Furthermore, the IP camera used for blob detection has a significant amount of image lag along with being very difficult to properly integrate with the rest of the robot. Furthermore, while the color detection works quite well, the difference in

results varies greatly with the lighting which may not be a problem if a more expensive camera was purchased.

Another limitation is that once the power is switched on, it takes a very long time for the camera to sync up with the router that I was using, the cheapest one I could find, and could have been eliminated by either turning my laptop into a router or by buying a higher quality one.

The PVC pneumatic launcher works quite well if it is perfectly assembled and the epoxy and PVC cement is expertly applied to every single crack in the device which was not done for this project. For the first demo day, the epoxy keeping the bike valve in the air chamber cracked and began to leak air and the same may happen for media day. In hindsight, silicone would have been a much better choice to seal the air inside this device, which would have meant the air chamber could have easily held up to 100 psi of compressed air instead of the measly 65 psi that was used in experimentation.

Technical caveats:

1. NEVER connect power to ground, I destroyed far too many parts by being careless and doing this.
2. Don't waste time wire wrapping, learn how to crimp and start immediately.
3. Pay very close attention to the various provided OpenCV installation tutorials; it can get very confusing if you don't pay attention.
4. If using an inductive load, DO NOT forget a flyback diode otherwise you will be spending more money on parts.

Future Work:

If I were to start this project over I would buy a better ip camera that more easily interfaces with OpenCV in order to save myself some much need time. Also, I would buy a much stronger servo right off the bat and do some calculations to determine the amount of torque I needed to lift up the massive pneumatic launcher. Furthermore, I would actually design my robot's platform in Solidworks and take more care to cut it so that it doesn't look as shabby as this one does.

One enhancement I would make is that I would add a speaker system so that while my robot it driving around it would play the Robocop theme song to add to the theme. Another enhancement would have been to include an onboard air compressor and ball reloader so that the robot could fire off more than one shot without having to manually refill the air and replace the ping pong ball.

XI. Documentation

[1] Technical-Recipes (2011, October) Tracking Coloured Objects in video using OpenCV Available: December 2011
http://www.technical-recipes.com/2011/tracking-coloured-objects-in-video-using-opencv/#more-1348

[2] AI Shack (2010, July) Tracking color objects in OpenCV Available: December 2011

http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/

[3] Pololu (2011, August) Sharp GP2D120 IR sensor Available: December 2011
http://www.pololu.com/file/0J157/GP2D120-DATA-SHEET.pdf

[4] Dsynflo (2010, February) cvblobslib with OpenCV: Installation Available: December 2011
http://dsynflo.blogspot.com/2010/02/cvblobskib-with-opencv-installation.html

XII. Appendices

A. Main AVR Studio Code (header files not included, see Tim Martin's code)

The header and source files can be found at https://sites.google.com/site/epiphanydiy/home

```
#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>
#include "motor.h"
#include "lcd.h"
#include "uart.h"
#include "RTC.h"
#include "picServo.h"
//#include "ATtinyServo.h"
#include "ADC.h"
#include "sonar.h"

#define DbLedOn()           (PORTR.OUTCLR = 0x02)             //Turns the debug led on.
The led is connected with inverted logic
#define DbLedOff()          (PORTR.OUTSET = 0x02)             //Turns the debug led off.
The led is connected with inverted logic
#define DbLedToggle()       (PORTR.OUTTGL = 0x02)             //Toggles the debug led
off.  The led is connected with inverted logic


volatile uint8_t uarte1IncomingData = 0;
int main (void)
{
       board_init(); /*This function originates in the file init.c, and is used to
initialize the Epiphany DIY
                                    motorInit() is declared within because by default you
the user should define what your
                                    motor setup is to prevent hurting the Epiphany.  You
can do this by
                                  */
       RTC_DelayInit();//initializes the Real time clock this seems to actually take an
appreciable amount of time
       DbLedOn();    //I like to do this by default to show the board is no longer
suspended in the bootloader.
       servoControlInit();//initializes the servo module ***including enabling global
interrupts*** required for the servo control module
```

```
	uartInit(&USARTC0,57600);//as can be seen in the schematic.  This uart is
connected to the USB port.  This function initializes this uart
	uartInit(&USARTE1,9600);//as can be seen in the schematic.  This uart is connected
to the USB port.  This function initializes this uart
	ADCsInits();//this function initializes the ADCs inside the Xmega
	LCDInit();
	//you should add any further initializations here
	sei();
	fprintf_P(&lcd_str,PSTR("Out of the Box\nEpiphany DIY"));
	fprintf_P(&USB_str,PSTR("Out of the Box:\rElectronics and Robotics\rPresents
the\rEpiphany DIY\rSoftware Version %s\r%s"),sVersion,date);

	PORTD_DIR |= 0x02;                              // make port d0 input and d1 an
output
	PORTC_DIR = 0;
	PORTD_PIN0CTRL = 00011111;
	//PORTD_PIN1CTRL = 00011111;             // set the pullup for the bump switches
--- may need to switch to 1 and siwtch 2
	PORTD_PIN2CTRL = 00011111;
	PORTC_PIN0CTRL = 00011111;


	//you should add any further initializations here


	if(dataInBufE1())
	{
		fscanf(&uartE1_str,"%c",&uarte1IncomingData);
	}

	while (1)
	{
		//request values for first IR
		ADCA_request(0, ExternalChannel0);
		while(!(ADCA_CH0_ConvComplete));
		unsigned int x = ADCA_getVal(0);
		printf("Dist: %u", x);//   %u", x, y);   (Vin/5) * 4095 = ADC val
		printf("posX= %x", USARTE1.DATA);
		//request values for 2nd IR
		ADCA_request(1, ExternalChannel1);
		while(!(ADCA_CH1_ConvComplete));
		unsigned int y = ADCA_getVal(1);
		printf("\n%u", y);

		_delay_ms(425);

		/*setServoAngle(75, 24);
		_delay_ms(200);*/

		//you may fire when ready
		//PORTD_OUT = 0x02;
		//PORTD_OUTSET = 0x2;
		//_delay_ms(250);

		//PORTD_OUTCLR = 0x2;

		printf("\r");
		//setServoAngle(60, 24);
```

```
		//right side of screen turn left
		if((uarte1IncomingData) == 1)
		{
			setMotorDuty(4, 600, MOTOR_DIR_BRAKE_gc);
			setMotorDuty(3, 600, MOTOR_DIR_FORWARD_gc);
			//_delay_ms(250);
		}
		else if(uarte1IncomingData == 2)
		{

			//left side of the screen, turn right
			setMotorDuty(4, 600, MOTOR_DIR_BACKWARD_gc);
			setMotorDuty(3, 600, MOTOR_DIR_BRAKE_gc);
			//_delay_ms(250);
		}
		else if(uarte1IncomingData == 3)
		{

			//object is centered... add in the servo adjustment and firing
control code
			setMotorDuty(4, 600, MOTOR_DIR_BRAKE_gc);
			setMotorDuty(3, 600, MOTOR_DIR_BRAKE_gc);

			_delay_ms(500);
			setServoAngle(70, 24);
			_delay_ms(1500);

			//you may fire when ready
			PORTD_OUT = 0x02;
			PORTD_OUTSET = 0x2;

			_delay_ms(250);

			PORTD_OUTCLR = 0x2;
			setServoAngle(90, 24);

		}
		else
		{
			asm("nop");
			// bumper 1 hit something turn right
			if((PORTD.IN & 1))
			{
				setMotorDuty(3, 600, MOTOR_DIR_FORWARD_gc);
				setMotorDuty(4, 600, MOTOR_DIR_BRAKE_gc);


			}
			// bumper 2 hit something, turn left
			else if((PORTC.IN & 1))
			{

				setMotorDuty(3, 600, MOTOR_DIR_BRAKE_gc);
				setMotorDuty(4, 600, MOTOR_DIR_BACKWARD_gc);
```

```
                        }
                        // smaller means farther away, so if it is less than 400 go forward
                        else if(x < 450 && y < 450)
                        {

                                setMotorDuty(3, 600, MOTOR_DIR_BACKWARD_gc); //was motor 2
                                setMotorDuty(4, 600, MOTOR_DIR_FORWARD_gc);

                        }
                        //x = 500, y == 400 -- to close to x IR so turn away  - only used
for when IR left is busted
                        else if((x > 450) &  (x > y))
                        {

                                setMotorDuty(3, 600, MOTOR_DIR_FORWARD_gc);

                                setMotorDuty(4, 600, MOTOR_DIR_BRAKE_gc);


                        }
                        // x = 400, y = 600 -- y IR is to close turn away
                        else if((y > 450) & (y > x))
                        {

                                setMotorDuty(4, 600, MOTOR_DIR_BACKWARD_gc);
                                setMotorDuty(3, 600, MOTOR_DIR_BRAKE_gc);


                        }
                }
        if(dataInBufE1())
        {
                fscanf(&uartE1_str,"%c",&uarte1IncomingData);
        }

        }
}
```

B. OpenCV

1. IMDLipcam code, main function:

```
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
#include "BlobResult.h"
//#include "cvblob.h"
#include "stdafx.h"
#include "COMfunct.h"
IplImage* GetThresholdedImage(IplImage* img)
{

    // Convert the image into an HSV image
    IplImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV, CV_BGR2HSV);
```

```
       IplImage* imgThreshed = cvCreateImage(cvGetSize(img), 8, 1);

       cvInRangeS(imgHSV, cvScalar(100, 100, 100), cvScalar(125, 255, 255), imgThreshed);

       cvReleaseImage(&imgHSV);
    return imgThreshed;
}
int main()
{
       CBlobResult blobs;
    CBlob *currentBlob;
    CvPoint pt1, pt2;
    CvRect cvRect;
       HANDLE comport;
       char dataout;

       comport = OpenCom(6);
       // if failes, -1

    int key = 0;
        // Initialize capturing live feed from the camera
    //CvCapture* capture = 0;
       IplImage* frame;
    //capture =
cvCaptureFromCAM("http://192.168.2.4/videostream.cgi?user=admin&pwd=&resolution=32&rate=5
0&a=.mjpg");    //may need to change to the url thing
       //CvCapture* capture =
cvCreateFileCapture("http://192.168.2.4/videostream.cgi?user=admin&pwd=&resolution=32&rat
e=50&a=.mjpg");
        CvCapture* capture =
cvCreateFileCapture("http://192.168.2.4/videostream.cgi?user=admin&pwd=&resolution=16&rat
e=50&a=.mjpg");
    // Couldn't get a device? Throw an error and quit
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
        return -1;
    }

       // The two windows we'll be using
    cvNamedWindow("video", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("thresh", CV_WINDOW_AUTOSIZE);

       // This image holds the "scribble" data...
    // the tracked positions of the ball
    IplImage* imgScribble = NULL;

        // An infinite loop
    while(true)
    {
        // Will hold a frame captured from the camera
       // IplImage* frame;// = 0;
        frame = cvQueryFrame(capture);


              // If we couldn't grab a frame... quit  this is the thing that makes it so
if it doesnt see blue the program quits.. kinda dumb
        if(!frame)
```

```
            break;

             // If this is the first frame, we need to initialize it
        if(imgScribble == NULL)
        {
            imgScribble = cvCreateImage(cvGetSize(frame), 8, 3);
        }

             // Holds the yellow thresholded image (yellow = white, rest = black)
        IplImage* imgThresh = GetThresholdedImage(frame);
             blobs = CBlobResult(imgThresh, NULL, 0);

             //Exclude white blobs smaller than the given value (10)
        // The bigger the last parameter, the bigger the blobs need
        // to be for inclusion
        blobs.Filter( blobs,  B_EXCLUDE, CBlobGetArea(), B_LESS, 75 );

             // Attach a bounding rectangle for each blob discovered
        int num_blobs = blobs.GetNumBlobs();

             cvDilate(imgThresh, imgThresh, NULL, 2);

             //smooth
             cvSmooth(imgThresh, imgThresh, CV_GAUSSIAN, 3, 0, 0, 0);
             for ( int i = 0; i < num_blobs; i++ )
        {
            currentBlob = blobs.GetBlob( i );
            cvRect = currentBlob->GetBoundingBox();

            pt1.x = cvRect.x;
            pt1.y = cvRect.y;
            pt2.x = cvRect.x + cvRect.width;
            pt2.y = cvRect.y + cvRect.height;

                       // Attach bounding rect to blob in orginal video input
                     cvRectangle( frame, pt1, pt2, cvScalar(15, 236, 60, 0), 1, 8, 0);
                     cvRectangle( imgThresh, pt1, pt2, cvScalar(15, 236, 60, 0), 1, 8,
0);
             }


              // Calculate the moments to estimate the position of the ball
        CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgThresh, moments, 1);

        // The actual moment values
        double moment10 = cvGetSpatialMoment(moments, 1, 0);
        double moment01 = cvGetSpatialMoment(moments, 0, 1);
        double area = cvGetCentralMoment(moments, 0, 0);

              // Holding the last and current ball positions
        static int posX = 0;
        static int posY = 0;

        int lastX = posX;
        int lastY = posY;

        posX = moment10/area;
```

```
        posY = moment01/area;

               // Print it out for debugging purposes
        printf("position (%d,%d,%d)\n", posX, posY, area);
              //dataout = posX/4;
              if(posX > 215)
              {
                     //object is to the right, so turn left
                     dataout = 1;
              }
              else if((posX < 190) && (posX >=0))
              {
                     //send out 2 so the robot turns right cuz object is to the left of
center
                     dataout = 2;
              }
              else if((posX > 190) && (posX < 215))
              {
                     //object is centered
                     dataout = 3;
              }
              else
              {
                     //no blue so go back to regular old obstacle avoiding
                     dataout = 0;
              }

              WriteCom(comport, &dataout, 1);
              // We want to draw a line only if its a valid position
 /*       if(lastX>0 && lastY>0 && posX>0 && posY>0)
        {
            // Draw a yellow line from the previous point to the current point
            cvLine(imgScribble, cvPoint(posX, posY), cvPoint(lastX, lastY),
cvScalar(0,255,255), 5);
        }*/

              // Add the scribbling image and the frame...

        cvShowImage("thresh", imgThresh);
        cvShowImage("video", frame);

               // Wait for a keypress
        int c = cvWaitKey(5);
        if(c!=-1)
        {
            // If pressed, break out of the loop
            break;
        }

              // Release the thresholded image+moments... we need no memory leaks..
please
        cvReleaseImage(&imgThresh);
        delete moments;
    }
       // We're done using the camera. Other applications can now use it
   // cvReleaseCapture(&capture);
   return 0;
}
```

2. IMDLipcam code, COMFUNCT.h (thanks to Chris Dobson for providing this code)

```
#include "wstring_variable.h"
#include "windows.h"


struct COM_Index
{
	int NumberOfComs;
	int ComNumbers[256];
};

int EnumerateComs(COM_Index& INDEX)
{
	INDEX.NumberOfComs= 0;
	wstring_variable wstringtmp;
	char chartmp[5];
	HANDLE COMPORT;


	for(unsigned int counter = 0; counter < 100; counter++)
	{

		_itoa_s(counter, chartmp, 10);
		wstringtmp = "\\\\.\\COM";
		wstringtmp += chartmp;
		COMPORT = CreateFile( wstringtmp, 0, 0, NULL, OPEN_EXISTING, NULL, NULL);
		if (COMPORT != INVALID_HANDLE_VALUE)
		{
			INDEX.ComNumbers[INDEX.NumberOfComs++] = counter;
			CloseHandle(COMPORT);
		}



	}



	return INDEX.NumberOfComs;
}

HANDLE OpenCom(int ComNumber)
{
	wstring_variable wstringtmp;
	char chartmp[5];

	_itoa_s(ComNumber, chartmp, 10);
	wstringtmp = "\\\\.\\COM";
	wstringtmp += chartmp;

	return CreateFile( wstringtmp, (GENERIC_READ | GENERIC_WRITE), 0, NULL,
OPEN_EXISTING, NULL, NULL);
}


int WriteCom(HANDLE ComHandle, char* DataBuffer, int BytesToWrite)
{
```

```
        DWORD  byteswritten = 0;
        WriteFile( ComHandle, DataBuffer, BytesToWrite, &byteswritten, NULL);
        return byteswritten;
}
```