Date: 4/17/2011
Student Name: Justin Goll
TA: Devin Hughes
Tim Martin
Ryan Stevens
Josh Weaver
Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

“Cyclops”

University of Florida

EEL 4665: Intelligent Machines Design Laboratory

Formal Report

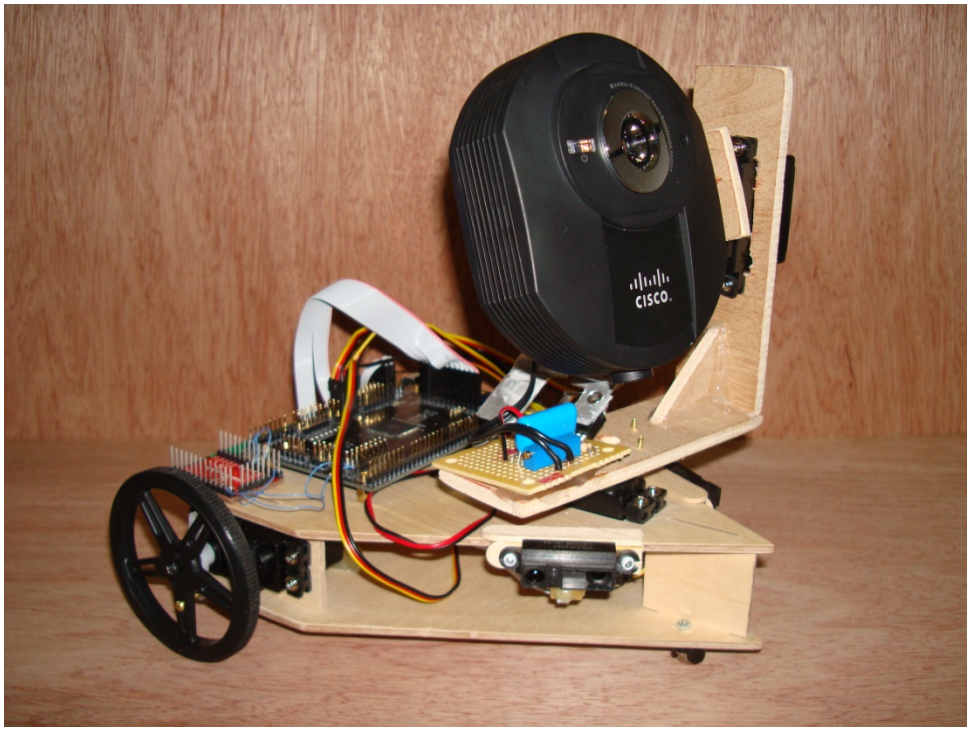


Table of Contents

Abstract

The purpose of my robot is to demonstrate the use of image processing in robotics to create a mobile platform that is capable of precisely targeting specific objects. For this robot I have chosen balloons as the target and a high powered laser as the device to be aimed at, and interact with, the target. An automated mobile aiming platform can be used in wide variety of applications from military drones to a face tracking mobile camera.

Executive Summary

Cyclops's purpose it to locate balloons and pop them with a laser. It does this with an IP webcam, two hacked servos to drive its wheels, a tilt/pan mechanism to rotate its camera, and two IR sensors to avoid obstacles. It also uses openCV to analyze images captured by the camera and Bluetooth to communicate between then computer running the openCV software and the robot. The robot will randomly search for balloons by driving around and scanning with its camera; it will avoid walls and obstacles while it searches. The balloons are painted with a symbol to allow the openCV software to more easily identify them. When the robot sees a balloon the openCV software tells the robot to stop. The software running on the computer then directs the robot to move its camera so the balloon is centered in the camera's view. The software then tells the robot to fire its laser and pop the balloon. After the balloon has been destroyed the robot continues to search for more balloons.

Introduction

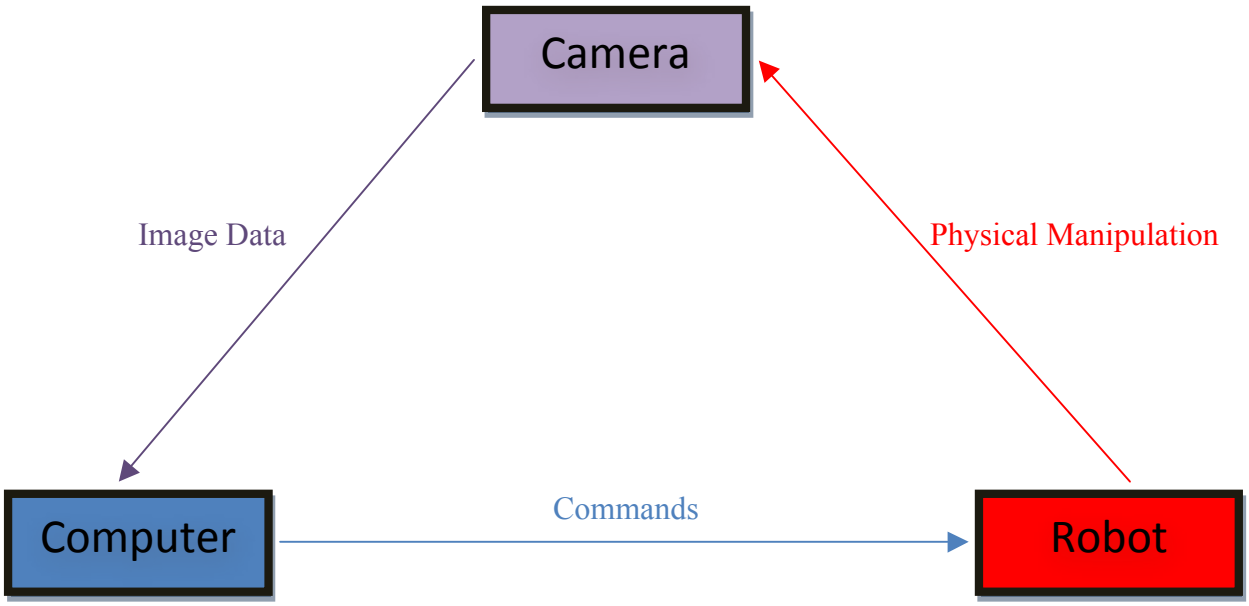
My robot Cyclops is an object finding and tracking robot. The primary goal of Cyclops is to demonstrate the use of image processing in a robot that is able to locate and track specific objects. Cyclops locates objects by driving around and scanning with its camera. The images captured by Cyclops's camera are sent to a computer where a program using openCV checks to see if the object is in the image. If an object is found Cyclops stops moving and only uses its turret mounted camera to track the object. After the object disappears, Cyclops continues to drive around and search for more objects.

The object Cyclops tracts is actually a symbol, the peace sign, which can be placed on any object. For demonstration purposes the symbol will be placed on balloons which Cyclops will pop after locating the symbol painted balloon. The reason Cyclops tracks a symbol instead of the actual balloon is mainly for convenience as the software used to train the image processing algorithm requires many thousands of images, the details of which are described in the Sensors section.

Integrated System

The system will consist of three nodes, an IP webcam, a computer, and the robot. The webcam will communicate via WIFI with the computer and the computer will communicate via Bluetooth with the robot. This communication is only one-way with the computer only receiving data from the camera and not transmitting data back to the camera and the robot only receiving data from the computer and not transmitting data back to the computer. Direct communication between the robot and the camera is not necessary as they interact via the software running on the computer.

The way in which the system will operate is that the robot will patrol the immediate area to allow the webcam to scan a larger area for the target object. The webcam which is mounted on the robot will send image data to the computer. The computer will analyze this data and determine if the target object is within the image. If the target is found to be within the image, the computer will send the robot a command to halt its movement and manipulate the orientation of the camera to center the target object in the camera's field of view. Upon verification that the target object is genuine and directly in the center of the cameras field of view, the computer will send a command to the robot to fire its laser. After visual conformation of the destruction of the target object the computer will instruct the robot to continue its patrol for more targets.



The robot itself consists of a PVR Xmega128A microcontroller board, a Rayson BTM-182 Bluetooth module, two Sharp GP2Y0A21YK Infrared sensors, four Hitec HS-485HB servos, two of which are hacked for continuous motion. It also has a basic 16x2 character LCD screen and a 5v regulator to power the IP webcam. The wiring diagram can be found in appendix A.

Mobile Platform

The purpose of the mobile platform is to allow Cyclops to search for balloons beyond what it can currently see with its turret mounted camera. This was achieved using a triangular shaped two wheeled platform driven by hacked servos and a steel caster for easy movement.

The mobile platform consists of a nearly identical triangular base and top, as well as three supports which connect them together. The two supports which are located on the side of the robot house the hacked servos and are 2.5 inches wide and 1.25 inches tall. The front support houses the sonar module and is 1.5 inches wide and 1.25 inches tall. The top houses the servo responsible for panning the camera as well as the microcontroller board and infrared sensors. The top and base are 10 inch equilateral triangles with two inches of the bottom two corners cut off to make room for the wheels which have a diameter of 70mm. The steel caster is attached to the bottom of the base at front of the robot, opposite the wheels at the back of the robot. The space between the base and the top is used to route wires and house the battery packs. The entire structure is wedged together as the pieces are so tight fitting. This allows for easy disassembly and access to the different components.

Hacked servos were chosen because they are relatively inexpensive and easy to use compared to motors. All the servos on Cyclops are HiTec HS-485HB Servos; these servos were chosen because they have gears made of a rugged plastic instead of Nylon and provide more torque (66.6 oz/in. at 4.8v) than other servos in its price range. The wheels were originally mounted to the hacked servos by driving a screw through the center of the wheel into the hole at the top of the servo. This proved to be an inadequate method of mounting the wheels as one of servos appeared to be slowing down over time; the robot would go around in circles when it was supposed to be going strait. It turned out that the screw was not properly grabbing one of the wheels and was slipping within the wheel. This was solved by screwing the wheel into the center hole and one of the arm attachments that came with the servo.

Actuation

One form of actuation that Cyclops has is the turret which pans and tilts the camera and laser. The turret consists of two servos (HS-485HB) one mounted vertically and the other mounted horizontally on top of the first servo. The camera and laser are then mounted to the second servo; this allows the camera to both pan side to side and also tilt up and down. The purpose of this is to allow the camera to look in all directions in front of the robot while searching for a balloon. These servos are also the only servos which will be directly controlled by both the computer and the onboard microcontroller. The microcontroller will control the actuation of the camera while searching for the balloon and the computer will control the actuation after a balloon has been spotted.

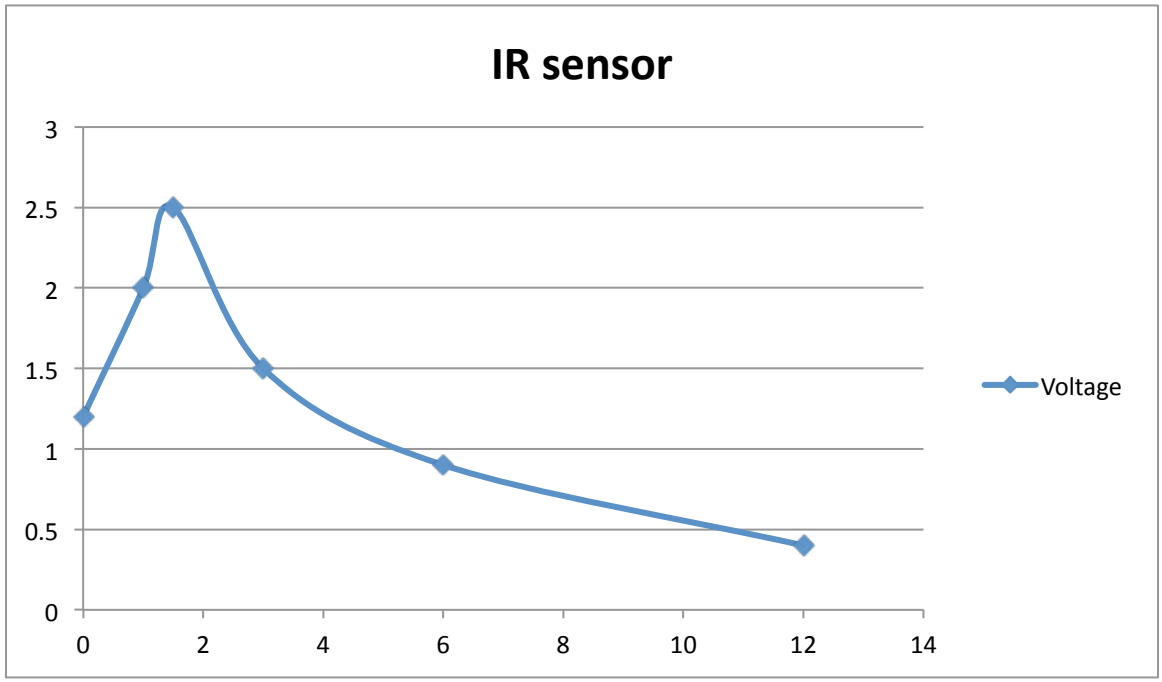
Mobile actuation was described in the previous section.

Sensors

Cyclops makes use of many different sensors in its pursuit of balloons these include, an IP webcam, IR sensors, sonar, and bump switches.

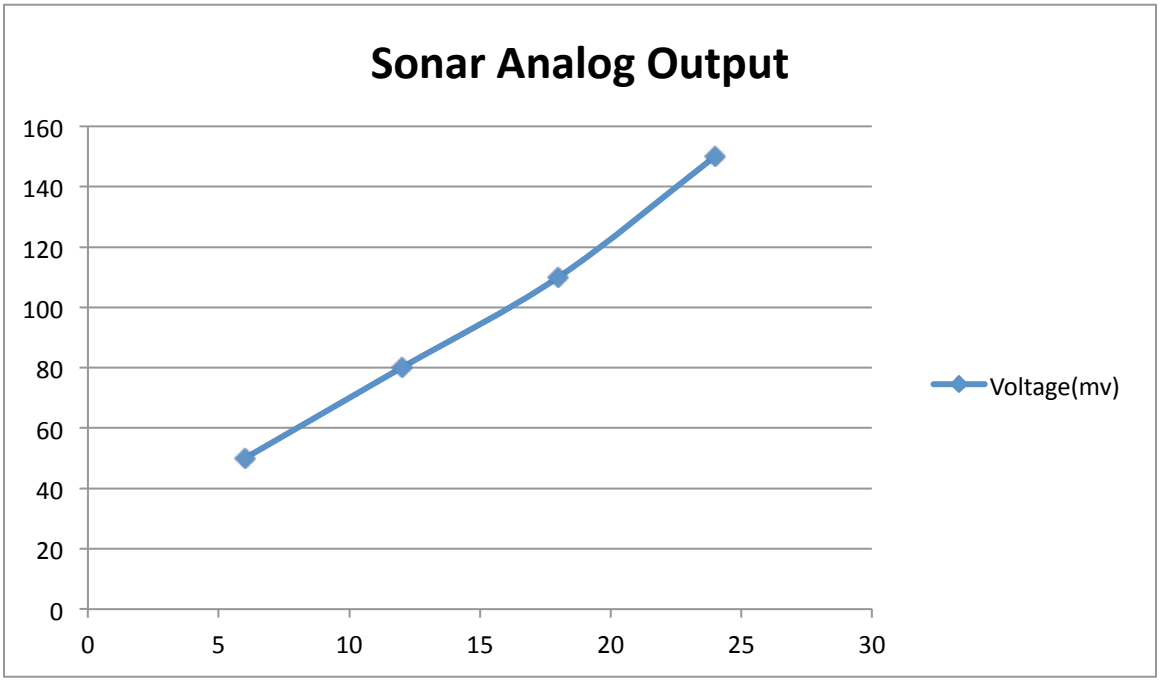
Infrared

Cyclops uses two IR sensors facing forward-left and forward-right to detect obstacles in its path. The IR sensors used are Sharp GP2D120XJ00F short range proximity sensors. Based on my own experimental data these sensors have a maximum range of 12in. when supplied with 5v from the ADC port on the PVR board. They output a voltage between .4v at 12in. to 2.5v at 1.5 inches. The digital value received after passing through the ADC from these sensors can be any value between 0 and 4095 however values in the range of 2000 are usually output even when objects are not detected due to the infrared in the ambient light. To prevent the robot from becoming blinded by bright lights and allow it to operate the same in different lighting conditions; the program used to read from the IR sensors keeps a running average of values taken from the sensors. It then uses that average as the baseline representing the ambient light and values are compared to that number instead of zero or some arbitrarily chosen number. This allows the robot to operate in almost all lighting conditions (except when sensor is saturated) and changing lighting conditions.



Sonar

Cyclops uses one sonar sensor facing forward to detect small or thin objects directly in front of the robot such as table and chair legs. The Sonar sensor used is the Maxbotix LV-EZ1 ultrasonic range finder. According to the datasheet this sonar has a range of 0 to 255 inches, It has an RS232 and an analog output. The analog output was experimentally measured and found to be quite week and noisy (50mv at 6in. and 150mv at 24in.). An amplifier was made to try and interface the sonar to the robot using the ADC, that however ended in failure as the op-amp used required a voltage that exceeded the voltage available on the robot. Instead the sensor was interfaced using the RS232 interface.



IP Webcam

Cyclops uses a single camera to look for and identify balloons. The camera used is the Linksys Wireless-N+RJ45 IP Camera. This camera was chosen because of its ability to wirelessly transmit its images over a network to a computer. It has a maximum resolution of 640 x 480 pixels and a max frame rate of 30 frames per second. The images captured by this camera are transmitted to a compute to be analyzed by the vision software and determine if a balloon has been seen.

The vision software used is openCV haartraining which divides the image up into rectangles and looks for patters in those rectangles. A profile for the desired patter is created using thousands of images which contain the desired object as well as a text file which tells the trainer the location of the object in each one of those images. After a profile is created images can be quickly checked to see if they contain the desired object. As mentioned in the introduction I have trained the software to search for a symbol rather than the balloons themselves; this is for two main reasons. One, balloons do not have very many distinguishing features when divided up

into rectangles. This would make the profile created less robust as the more complex an object the more accurate the resulting profile will be. Second, the haartrainer has a built in sample image generator which can use a few pictures of the desired object, and thousands of pictures that do not contain the object to create thousands of images which do contain the object. The reason why I did not use a picture of a balloon with this generator is because the picture of the balloon it would use would have the same background in every picture. So the profile created would interpret the thing it is trying to find as a balloon surrounded by a square of that given background. A symbol can have a static background as the background is the balloon which is a plain background.

Behaviors

The robot will feature three different behavior modes, patrol mode, turret mode, and mixed mode. The reason for creating the three different behavior modes is to retain the ability to demonstrate the robot's features in a controlled and safe manner.

In patrol mode the robot will be completely independent of both the camera and the computer. The purpose of this mode is to demonstrate the robots mobility and ability to avoid collisions with obstacles. In this mode the robot will move around semi-randomly, if it detects an object in its path it will change its direction to avoid the object. The robot will use its onboard sensors to detect objects at a distance in front of it and also to detect objects that make direct contact with the back of it. The robot will never fire its laser in this mode as that can only be done with a command sent by the computer. The robot will also not manipulate the camera during this mode as it is unnecessary.

In turret mode the robot will be not move with its wheels but will instead sit still and pan and tilt its camera. The purpose of this mode is to demonstrate the robots ability to recognize target objects in space and destroy them with its laser. In this mode the robot is scanning the immediate area in front of it for the target object with the camera. The camera is sending image data to the computer and the computer can send commands to the robot to pan and tilt the camera as well as fire the laser. The robot will never move during this mode and will not be using its onboard sensors to detect objects around it.

In mixed mode the robot will be performing the functions of patrol mode and turret mode at the same time. The purpose of this mode is to demonstrate the complete integration of the robots mobile and vision systems. In this mode the robot will patrol the immediate area for target objects by moving with its wheels and manipulating the camera in a sweeping scanning motion. If a target object is seen the robot will stop patrolling and destroy the target object with its laser. After destruction of a target object the robot will continue patrolling for more targets.

Experimental Layout and Results

Bluetooth

The Bluetooth module turned out to be one of the most troublesome parts of my project. I originally tried to interface my Bluetooth module though SPI on port E. That failed as I was unable to send or receive data using SPI. I talked to other students in the class that were using Bluetooth and they suggested using the UART interface. Matt Morgan also provided me with the code he used to interface his bluesmirf Bluetooth module. I modified Matt's code for my Bluetooth module changing its baud rate and com port. Using that code I was able to send data to my computer and have it show up on a terminal and in a C++ program I wrote. I was however unable to transmit data to the robot from the computer. I believe that I most likely have a defective Bluetooth module, as the code follows the steps outlined in the Atmel documentation and should work to enable any form of UART communication regardless of the peripheral.

openCV

The openCV haartraining required the most research to get working correctly. At first I simply modified the openCV facedetect software to access the IP webcam. This however resulted in strange buffering problems which would make the video lag behind what is actually happening. I then tried to find a way to get single jpg images from the IP webcam which proved difficult as it only output mjpg video. I found a perl command that allows you to download files from web addresses and a command for IP webcams which forces them to send back a jpg image (lwp-download http://10.0.0.3/img/snapshot.cgi?size=3). Using this I changed the facedetect software again to download an image from the camera, analyze it for the desired object, then delete the image so the next one can be loaded. This eliminated the lag issue.

I then used the openCV createsamples program to generate three thousand positive sample images using seven images that contain the peace sign symbol and four thousand negative images that did not contain the peace sign. Using the negative and positive sample images I ran the haartraining program to generate the xml file with contains the profile used to do object detection.

I then further modified the facedetect software to use the new xml file and to no longer display the image with the target circled but to print the coordinates of the object to the terminal.

Conclusion

I am mostly happy with my experience making this robot as I was able to successfully get it to search, scan for and recognize objects. It is however disappointing that something as simple as a defective Bluetooth module can ruin a project like this and make it mostly useless. The lack of communication from the computer to the robot makes it impossible for the robot to react to the objects it sees with the camera.

If I were to redo this project I would probably use a cheaper Arduino microcontroller board as I only had a need for a fraction of the ports provided by the PRV board and there is a larger community of developers for the Arduino platform. I would then use the money saved to buy a more expensive Bluetooth module and two more IR sensors to better detect obstacles directly in front of the robot and behind the robot. I would also make verifying the wireless connections of my devices a higher priority to be completed before the special sensor rather than after.

My robot has provided me with valuable experience in visual computer, networking, and robotics that I was seeking when I signed up for IMDL. I was very pleased by the performance of the openCV object detection code and the IP webcam. In the future I plan to replace the Bluetooth module with a bluesmirf. Then repurpose the robot as a mobile webcam that will use face detection to follow me around my room.

Refrences

Microcontroller Board:

http://www.atmel.com/dyn/resources/prod_documents/doc8067.pdf

http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

Bluetooth Module:

http://www.sparkfun.com/datasheets/Wireless/Bluetooth/BTM182.pdf

http://www.sparkfun.com/datasheets/Wireless/Bluetooth/SPP%20AT%20command%20set.pdf

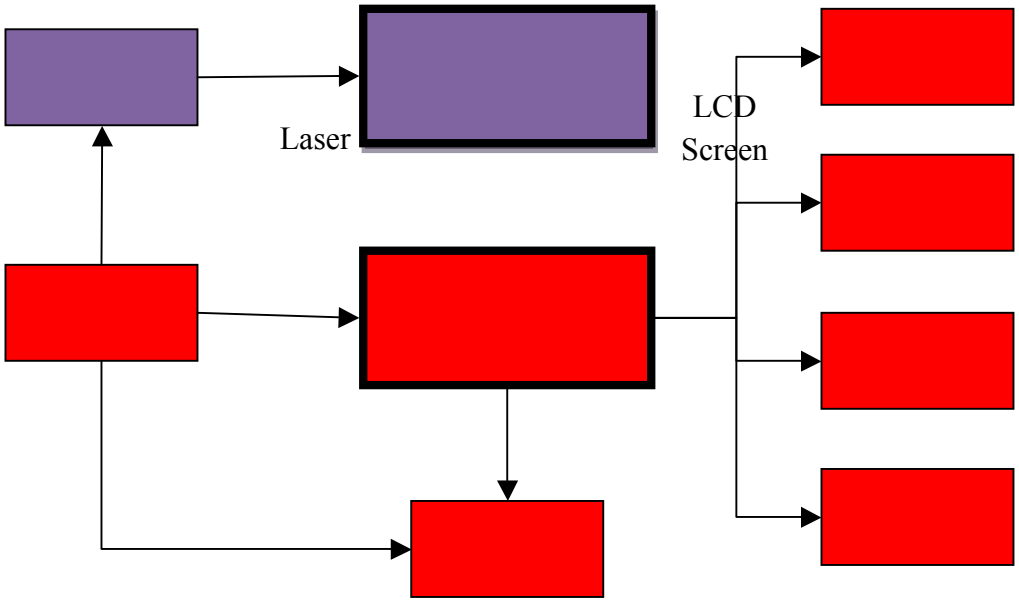
openCV:

http://note.sonots.com/SciSoftware/haartraining.html

http://opencv.willowgarage.com/wiki/FaceDetection

Appendices

A. Wiring Diagram



B. Robot Code

```
//Cyclops
//Justin Goll

#include <avr/io.h>
#include "PVR.h"
#include "bluetooth.h"                  //original code provided by Matt Morgan
#define LCDclear lcdData(0x01);         //clear LCD

void MoveForward(void){
ServoC0(-100);
ServoC1(100);
bluetooth_transmit_char('F'); // tell computer robot going forward
}

void MoveBackward(void) {
ServoC0(100);
ServoC1(-100);
bluetooth_transmit_char('B'); // tell computer robot going backward
}

void TurnLeft(void) {
ServoC0(-10);
ServoC1(100);
bluetooth_transmit_char('l'); // tell computer robot turning left
}

void TurnLeftHard(void) {
ServoC0(100);
ServoC1(100);
bluetooth_transmit_char('L'); // tell computer robot turning left hard
}

void TurnRight(void){
ServoC0(-100);
ServoC1(10);
bluetooth_transmit_char('r'); // tell computer robot turning right
}

void TurnRightHard(void) {
ServoC0(-100);
ServoC1(-100);
bluetooth_transmit_char('R'); // tell computer robot turning right hard
}

void main(void) {
```

```
	xmegaInit();				//setup XMega
	delayInit();				//setup delay functions
	ADCAInit();				//setup PORTA analong readings
	ServoCInit();				//setup PORTC Servos
	ServoC0(0);
	ServoC1(0);
	ServoC2(0);
	ServoC3(0);
	lcdInit();				//setup LCD on PORTK
	bluetooth_init();			//setup bluetooth on port F
	PORTA_DIR &= 0xFC;//sets portA pin0 & pin1 to input and leaves other pins the same
	PORTQ_DIR |= 0x01;		//set Q0 (LED) as output
	lcdString("O Avoid");
	lcdGoto(1,0);
	lcdString("Justin Goll");
	delay_ms(1000);
	int Average = 0, count = 1,leftSensor = 0, rightSensor = 0,i = 1,Remainder = 0,pan = 0,
tilt = 0,panDir = 0, tiltDir = 0;
	long Total = 0;
	char bluetooth_Rx;
	while(1){
		if(i == 1){
		i = 0;
		}
		else {
		i = 1;
		}
	PORTQ_OUT = i;							//blink LED
	leftSensor = ADCA0();		//gets value between 0 and 4096 from left IR sensor
	rightSensor = ADCA1();		//gets value between 0 and 4096 from right IR sensor
	Total = Total + ((leftSensor + rightSensor)/2);
	Average = Total/count++;
	Remainder = 4096 - Average;
	leftSensor -= Average;
	rightSensor -= Average;
	if(leftSensor > (Remainder * 0.3) && rightSensor > (Remainder * 0.3)) {
	//drive backwards then turn around
	MoveBackward();
	delay_ms(1000);
	TurnRightHard();
	delay_ms(1000);
	}
	else if(leftSensor > (Remainder * 0.5)){
	//turn right hard
	TurnRightHard();
	}
```

```
else if(rightSensor > (Remainder * 0.5)){
//turn left hard
TurnLeftHard();
}
else if(leftSensor > (Remainder * 0.3)){
// turn right
TurnRight();
}
else if(rightSensor > (Remainder * 0.3)){
// turn left
TurnLeft();
}
else{
//move forward
MoveForward();
}
//camera part of code
if(pan == 90) {
panDir = 1; //make camera pan in other direction
}
if(pan == -90) {
panDir = 0; //Make camera pan in other direction
}
if(tilt == 80) {
tiltDir = 0; // make camera tilt in other direction
}
if(tilt == -80) {
tiltDir = 1; // make camera tilt in other direction
}
if(panDir == 1) {
pan = pan - 5;
}
if(panDir == 0) {
pan = pan + 5;
}
if(tiltDir == 1 && (pan == 90 || pan == -90)){
tilt = tilt + 10;
}
if(tiltDir == 0 && (pan == 90 || pan == -90)){
tilt = tilt - 10;
}
ServoC2(pan);
ServoC3(tilt);
//see if computer sent message to robot
if((USARTF1.STATUS >> 7) & 0x1){
bluetooth_Rx = BluetoothReadChar();
```

```
        LCDclear
        lcdChar(bluetooth_Rx);      //would normally enter different mode now
                                    //stoping servos and letting computer
                                    //send commands to move the camera
                                    //bluetooth not working however
        }
        delay_ms(100);
        }
}
```

C. Computer Code

```
// facedetect2.cpp : Defines the entry point for the console application.
//
//cyclops
//Justin Goll
//Com port code original code provided by Matt Morgan
#include "stdafx.h"
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <windows.h>
#include <tchar.h>
#include <stdio.h>

using namespace std;
using namespace cv;

void help()
{
   cout << "\nThis program demonstrates the haar cascade recognizer\n"
                "this classifier can recognize many ~rigid objects, it's most known use is for
faces.\n"
        "Usage:\n"
        "./facedetect [--cascade=<cascade_path> this is the primary trained classifier such as
frontal face]\n"
      "   [--nested-cascade[=nested_cascade_path this an optional secondary classifier such as
eyes]]\n"
      "   [--scale=<image scale greater or equal to 1, try 1.3 for example>\n"
      "   [filename|camera_index]\n\n"
                "see facedetect.cmd for one call:\n"
                "./facedetect --cascade=\"../../data/haarcascades/haarcascade_frontalface_alt.xml\"
--nested-cascade=\"../../data/haarcascades/haarcascade_eye.xml\" --scale=1.3 \n"
        "Hit any key to quit.\n"
                "Using OpenCV version %s\n" << CV_VERSION << "\n"
```

```
                << endl;
}

void PrintCommState(DCB dcb)
{
   //  Print some of the DCB structure values
   _tprintf( TEXT("\nBaudRate = %d, ByteSize = %d, Parity = %d, StopBits = %d\n"),
            dcb.BaudRate,
            dcb.ByteSize,
            dcb.Parity,
            dcb.StopBits );
}

HANDLE create_com()
{
          DCB dcb;
          HANDLE hCom;
          BOOL fSuccess;
   TCHAR *pcCommPort = TEXT("COM6"); //  Most systems have a COM1 port

  //  Open a handle to the specified com port.
  hCom = CreateFile( pcCommPort,
                GENERIC_READ | GENERIC_WRITE,
                0,      //  must be opened with exclusive-access
                NULL,   //  default security attributes
                OPEN_EXISTING, //  must use OPEN_EXISTING
                0,      //  not overlapped I/O
                NULL ); //  hTemplate must be NULL for comm devices

  if (hCom == INVALID_HANDLE_VALUE)
  {
     //  Handle the error.
     printf ("CreateFile failed with error %d.\n", GetLastError());
  }

  //  Initialize the DCB structure.
  SecureZeroMemory(&dcb, sizeof(DCB));
  dcb.DCBlength = sizeof(DCB);

  //  Build on the current configuration by first retrieving all current
  //  settings.
  fSuccess = GetCommState(hCom, &dcb);

  if (!fSuccess)
  {
    //  Handle the error.
```

```
    printf ("GetCommState failed with error %d.\n", GetLastError());
  }

  PrintCommState(dcb);       //  Output to console

  //  Fill in some DCB values and set the com state:
  //  115200 bps, 8 data bits, no parity, and 1 stop bit.
  dcb.BaudRate = CBR_19200;     //  baud rate
  dcb.ByteSize = 8;             //  data size, xmit and rcv
  dcb.Parity   = NOPARITY;      //  parity bit
  dcb.StopBits = ONESTOPBIT;    //  stop bit

  fSuccess = SetCommState(hCom, &dcb);

  if (!fSuccess)
  {
    //  Handle the error.
    printf ("SetCommState failed with error %d.\n", GetLastError());
  }

  //  Get the comm config again.
  fSuccess = GetCommState(hCom, &dcb);

  if (!fSuccess)
  {
    //  Handle the error.
    printf ("GetCommState failed with error %d.\n", GetLastError());
  }

  PrintCommState(dcb);       //  Output to console

  _tprintf (TEXT("Serial port %s successfully reconfigured.\n"), pcCommPort);
  //printf("Serial port COM4 successfully reconfigured.\n");

  return hCom;
}

void write_char(HANDLE hCom, BYTE message)
{
        DWORD /*dwError,*/ dwNumBytesWritten;
   BOOL fSuccess;

        fSuccess = WriteFile (hCom,            // Port handle
                              &message,        // Pointer to the data to write
                              1,               // Number of bytes to write
                              &dwNumBytesWritten, // Pointer to the number of bytes
```

```
                                                                    // written
                        NULL            // Must be NULL for Windows Embedded CE
                                        );
        if (!fSuccess)
  {
    //  Handle the error.
    printf ("WriteFile failed with error %d.\n", GetLastError());
  }

        while (!EV_TXEMPTY);

        return;
}

BYTE read_char(HANDLE hCom)
{
        BYTE Byte = 0;
        DWORD dwBytesTransferred;
        unsigned long dwCommModemStatus;

        //while(!EV_RXFLAG);
        SetCommMask (hCom, EV_RXCHAR);
        // Wait for an event to occur for the port.
        WaitCommEvent (hCom, &dwCommModemStatus, 0);
        if (dwCommModemStatus & EV_RXCHAR)
        {


        ReadFile (hCom,         // Port handle
                        &Byte,          // Pointer to data to read
                        1,              // Number of bytes to read
                        &dwBytesTransferred,  // Pointer to number of bytes
                                                        // read
                        NULL            // Must be NULL for Windows Embeddded CE
                );

        //printf("%d\n", Byte);
        }

        return Byte;
}

void close_serial(HANDLE hCom)
{
        BOOL fSuccess = CloseHandle(hCom);
```

```
        if (!fSuccess)
                printf ("CloseHandle failed with error %d.\n", GetLastError());

        return;
}



void detectAndDraw( Mat& img,
                CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
                double scale);

String cascadeName =
"C:/OpenCV2.2/data/haarcascades/positives.xml";
String nestedCascadeName =
"../../data/haarcascades/haarcascade_eye_tree_eyeglasses.xml";

int main( int argc, const char** argv )
{
   CvCapture* capture = 0;
   Mat frame, frameCopy, image;
   const String scaleOpt = "--scale=";
   size_t scaleOptLen = scaleOpt.length();
   const String cascadeOpt = "--cascade=";
   size_t cascadeOptLen = cascadeOpt.length();
   const String nestedCascadeOpt = "--nested-cascade";
   size_t nestedCascadeOptLen = nestedCascadeOpt.length();
   String inputName;

   help();
   CascadeClassifier cascade, nestedCascade;
   double scale = 1;

   for( int i = 1; i < argc; i++ )
   {
        cout << "Processing " << i << " " <<  argv[i] << endl;
      if( cascadeOpt.compare( 0, cascadeOptLen, argv[i], cascadeOptLen ) == 0 )
      {
         cascadeName.assign( argv[i] + cascadeOptLen );
         cout << "  from which we have cascadeName= " << cascadeName << endl;
      }
      else if( nestedCascadeOpt.compare( 0, nestedCascadeOptLen, argv[i],
nestedCascadeOptLen ) == 0 )
      {
         if( argv[i][nestedCascadeOpt.length()] == '=' )
```

```
            nestedCascadeName.assign( argv[i] + nestedCascadeOpt.length() + 1 );
        if( !nestedCascade.load( nestedCascadeName ) )
            cerr << "WARNING: Could not load classifier cascade for nested objects" << endl;
    }
    else if( scaleOpt.compare( 0, scaleOptLen, argv[i], scaleOptLen ) == 0 )
    {
        if( !sscanf( argv[i] + scaleOpt.length(), "%lf", &scale ) || scale < 1 )
            scale = 1;
        cout << " from which we read scale = " << scale << endl;
    }
    else if( argv[i][0] == '-' )
    {
        cerr << "WARNING: Unknown option %s" << argv[i] << endl;
    }
    else
        inputName.assign( argv[i] );
}

 if( !cascade.load( cascadeName ) )
 {
    cerr << "ERROR: Could not load classifier cascade" << endl;
    cerr << "Usage: facedetect [--cascade=<cascade_path>]\n"
        "   [--nested-cascade[=nested_cascade_path]]\n"
        "   [--scale[=<image scale>\n"
        "   [filename|camera_index]\n" << endl ;
    return -1;
 }
        HANDLE com_6 = create_com();
        while(1){
                system("del snapshot.cgi.jpeg");
                system("lwp-download http://10.0.0.3/img/snapshot.cgi?size=3");
    image = imread("snapshot.cgi.jpeg", 1 );
    if( image.empty() )
    {
        capture = cvCaptureFromAVI( inputName.c_str() );
        if(!capture) cout << "Capture from AVI didn't work" << endl;
    }



// cvNamedWindow( "result", 1 );
          if( !image.empty() )
    {
        detectAndDraw( image, cascade, nestedCascade, scale );
        waitKey(0);
    }
```

```
                printf("%c\n",(char)read_char(com_6));
  // cvDestroyWindow("result");
        }
        close_serial(com_6);
  return 0;
}

void detectAndDraw( Mat& img,
            CascadeClassifier& cascade, CascadeClassifier& nestedCascade,
            double scale)
{
  int i = 0;
  double t = 0;
  vector<Rect> faces;
  const static Scalar colors[] =  { CV_RGB(0,0,255),
    CV_RGB(0,128,255),
    CV_RGB(0,255,255),
    CV_RGB(0,255,0),
    CV_RGB(255,128,0),
    CV_RGB(255,255,0),
    CV_RGB(255,0,0),
    CV_RGB(255,0,255)} ;
  Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale), CV_8UC1 );

  cvtColor( img, gray, CV_BGR2GRAY );
  resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
  equalizeHist( smallImg, smallImg );

  t = (double)cvGetTickCount();
  cascade.detectMultiScale( smallImg, faces,
    1.1, 1, 0
    //|CV_HAAR_FIND_BIGGEST_OBJECT
    //|CV_HAAR_DO_ROUGH_SEARCH
    |CV_HAAR_SCALE_IMAGE
    ,
    Size(20, 20) );
  t = (double)cvGetTickCount() - t;
  //printf( "detection time = %g ms\n", t/((double)cvGetTickFrequency()*1000.) );
  for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end(); r++, i++ )
  {
    Mat smallImgROI;
    vector<Rect> nestedObjects;
    Point center;
    Scalar color = colors[i%8];
    int radius;
    center.x = cvRound((r->x + r->width*0.5)*scale);
```

```
        center.y = cvRound((r->y + r->height*0.5)*scale);
                printf("X = %d Y = %d\n",center.x,center.y);
                if(center.x <= 200 && center.y <= 175){
                        printf("Top Left\n");
                }
                else if(center.x <= 400 && center.y <= 175){
                        printf("Top Center\n");
                }
                else if(center.x <= 600 && center.y <= 175){
                        printf("Top Right\n");
                }
                else if(center.x <= 200 && center.y <= 350){
                        printf("Center Left\n");
                }
                else if(center.x <= 400 && center.y <= 350){
                        printf("Center\n");
                }
                else if(center.x <= 600 && center.y <= 350){
                        printf("Center Right\n");
                }
                else if(center.x <= 200 && center.y <= 500){
                        printf("Bottom Left\n");
                }
                else if(center.x <= 400 && center.y <= 500){
                        printf("Bottom Center\n");
                }
                else if(center.x <= 600 && center.y <= 500){
                        printf("Bottom Right\n");
                }

    }

}
```