Vision-Based Mobile Robot
Engineering 90
Senior Design Project

Abhay Manandhar
Advisor: Erik Cheever
May 08, 2009
Abstract

A vision-based mobile robot was designed by integrating an Acer Aspire One netbook and a Vex Robotics Kit. The robot used the single webcam on the netbook as its only sensor. The program to control the robot was written in C/C++ and Open CV Library. The robot was then programmed to navigate through artificial path created by two black tapes on whitish floor, and also to detect and follow red objects. A PID controller was implemented to control the robot for both the tasks. PID constant values were set using Zeigler-Nichols method and also by trial and error. Trial and error produced better constant values compared to ones produced by using the Zeigler-Nichols method.
## Table of Content

1. Introduction ................................................................................................................. 4
   1.1 Goals and Objectives ............................................................................................... 4
       1.1.1 Path Follow .......................................................................................................... 4
       1.1.2 Red Object Follow ............................................................................................... 4
   1.2 Web Camera ............................................................................................................... 5
   1.3 PID Controller .......................................................................................................... 5
       1.3.1 Zeigler-Nichols Method ..................................................................................... 6
2. Hardware Design .......................................................................................................... 6
   2.1 Acer Aspire One ......................................................................................................... 7
   2.2 Vex Robotics Kit ....................................................................................................... 7
   2.3 LynxMotion Servo Controller Board & Servo Motors .............................................. 7
   2.4 Body design ............................................................................................................... 8
3. Software Design ........................................................................................................... 9
   3.1 Computer software and library used ........................................................................ 9
       3.1.1 Visual Studio 2005: C/C++ .................................................................................. 9
       3.1.2 Open CV .............................................................................................................. 9
   3.2 Common Tasks .......................................................................................................... 10
       3.2.1 Serial Port Communication ............................................................................... 10
       3.2.2 Image Extraction ............................................................................................... 10
       3.2.3 PID Controller .................................................................................................... 10
       3.2.4 Debugging/Logging ............................................................................................ 10
   3.3 Path Follow ............................................................................................................... 11
   3.4 Red Object Follow .................................................................................................... 12
       3.4.1 Color Extraction ............................................................................................... 12
       3.4.2 Detection: Flood Fill Algorithm ......................................................................... 13
4. Results ........................................................................................................................... 14
   4.1 Path Following without a PID controller ................................................................. 14
   4.2 Path Following using Ziegler-Nichols Method to set PID constants ....................... 14
   4.3 PID Control using trial and error ............................................................................ 15
   4.4 Red Object Following .............................................................................................. 16
5. Conclusion ...................................................................................................................... 16
1 Introduction

A vision-based mobile robot was designed by integrating a Vex Robotics Kit, LynxMotion Servo Controller Board, and an Acer Aspire One netbook. The robot used the webcam on the netbook to get information from the environment. C/C++ programming language was used to write the program to control the robot, and Open CV library was used for vision processing. The robot was then programmed to navigate through an artificial path created by two black tapes on whitish floor, and to detect and follow red objects. A PID controller was implemented and tested to control the robot for both the tasks.

1.1 Goals and Objectives

The objective of the project was to learn C/C++ programming language, Computer Vision and Control System, and then apply the knowledge to create a vision-based mobile robot. The robot would be programmed to follow a path and to follow red objects. The two tasks are briefly described below.

1.1.1 Path Follow

An artificial path was created using two strips of black tape on white ground with some specks. The goal of the robot was to stay in the middle of the path and successfully follow it. The picture of the artificial path created is shown in the following picture.

![Figure 1. Artificial path created using two strips of black tape on whitish ground.](image)

1.1.2 Red Object Follow

The robot was programmed to detect red object with vision, and then to go towards the red object and stop before the object. If the object were to be moving, then the robot would follow the red object around. A red cup was used as a red object for this task, and a picture of the robot stopping before the red cup is shown below.

![Figure 2. Robot stopping before red cup.](image)
1.2 Web Camera

Laser or sonar sensors are the most common sensors used on robots to perceive depth in an environment as they return the distance objects are away at different angles to the robot, but they do not provide any other information. A web camera returns a lot of information about the environment but finding depth in a natural environment is a big challenge. Some robots make use of two calibrated cameras, and use the difference in the two images captured to calculate depth in the environment. The texture details and haziness of the objects in images captured by a single camera can be used to estimate depth, but are not very accurate. A simpler way to deal with depth perception with only one camera is to modify the environment and to read these modified features to map the environment. Black tapes were placed in the floor of the hallway to create an artificial path for the robot.

1.3 PID Controller

Proportional Integral and Derivative Controller is a general controller that uses the error between the measured value and the desired value to calculate the optimum output action to minimize the error. A block diagram of the feedback loop for a PID controller is given below in Figure 3.

![Figure 3. A block diagram of the PID feedback loop.](http://en.wikipedia.org/wiki/File:Pid-feedback-nct-int-correct.png)
The proportional term corrects the output by changing it proportionally with the error value. The integral term corrects the output proportionally with the time integral of the error value, and deals with steady state errors. The Derivative term corrects the output proportionally with the rate of change of error and helps to stabilize the system and decrease overshoots. The feedback loop equations for the Proportional, Integral and the Derivative terms are given below:

\[
e(t) = \text{desiredVal}(t) - \text{measuredVal}(t)
\]

\[
P_{out} = K_P \times e(t)
\]

\[
I_{out} = K_I \times \int_0^t e(\tau) d\tau
\]

\[
D_{out} = K_D \times \frac{de(t)}{dt}
\]

\[
Action = function\left( P_{out} + I_{out} + D_{out} \right)
\]

**Figure 4. PID controller feedback equations**

The constants for all the three terms can be changed individually to optimize the controller for any generic system. Zeigler-Nichols method was tested to set the PID constant values to control the robot, and the method is described briefly below.

### 1.3.1 Zeigler-Nichols Method

Zeigler-Nichols method is a method to set the PID constant values for a PID controller. Firstly, the Ki and the Kd values are set to 0, and the Ki value is increased to Kc to get critical oscillation. The time period, Pc of the critical oscillation is then measured. Then the Kc and the Pc values are used to set the Kp, Ki and Kd constants for the controller by using the equations shown in the following table.

**Table 1. Zeigler-Nichols equations to set the PID constant values.**

<table>
<thead>
<tr>
<th>Controller Type</th>
<th>Kp</th>
<th>Ki</th>
<th>Kd</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>0.50 Kc</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PI</td>
<td>0.45 Kc</td>
<td>1.2 Kp/Pc</td>
<td>0</td>
</tr>
<tr>
<td>PI</td>
<td>0.60 Kc</td>
<td>2 Kp/Pc</td>
<td>Kp Pc/8</td>
</tr>
</tbody>
</table>

2 Hardware Design

The components used to build the robot, and the robot body design are briefly discussed in this section. The connection of the different hardware components to form the robot is given in the following block diagram.
2.1 Acer Aspire One

Acer Aspire One netbook was chosen to be the brain of the robot. The netbook comes with a webcam and has enough processing power to smoothly run C/C++ code for image processing and controlling the robot. This model at 300 USD was one of the cheapest netbooks that had a web cam in the market.

2.2 Vex Robotics Kit

Vex Robotics Kit includes steel plates, screws, bolts, wheels, servo motors, gears, battery pack, remote control, etc. to construct and program mobile robots. The engineering department owned a Vex Robotics Kit, and the parts from the kit were used to design and build the body of the robot.

2.3 LynxMotion Servo Controller Board & Servo Motors

LynxMotion SSC-32 Servo Controller Board was used as a link between the Acer Aspire One and the servo motors. A USB to Serial converter was used to connect the netbook to the board as the netbook does not have with a serial port. The Board was powered by a 7.2 V battery from the Vex Robotics Kit which allowed the robot to be mobile.
Figure 7. LynxMotion Servo Controller Board used to control the servo.

The board takes inputs in the form of a string, and then outputs a pulse to up to 32 servos to move it. The structure of the string command to the board is to move the servo takes the following form which needs to be followed by a carriage return key which initiates the movement.

#<Channel number in decimal between 0 and 31> P<Pulse width in micro seconds>

The pulse width can be varied between the 0 and 3000 micro seconds, and it corresponds to different shaft positions in a servo. Three shaft positions for three different pulse widths are depicted in Figure 8. A shaft position sensor in a servo motor is hacked to always think that the shaft is the position corresponding to a pulse width of 1500 micro seconds. Therefore, any pulse width sent to the servo motor other than a 1500 micro seconds will result in the servo turning continuously to get to the corresponding position. Pulse width corresponding to different position of the shaft can be passed to turn the servo in the desired direction with different speeds.

![Figure 8. Figure showing the position of the servo shaft for 3 different pulse widths.](http://www.seattlerobotics.org/guide/servos.html)

2.4 Body design

The body of the robot was designed with two differential drive wheels on the back and a freely rotating spherical wheel on the front by using the Vex robotics kit. As the only sensor in the robot is a webcam, it can only sense the environment in one direction. The wheel configuration chosen allows the
robot to turn around in place, and deals with this constraint. The completed robot body is shown in the following figure, and Figure 10 shows the netbook on top of the robot body.

![Robot Body designed using parts from Vex Robotics Kit.](image)

**Figure 9. Robot Body designed using parts from Vex Robotics Kit.**

![Acer Aspire One netbook placed on the robot body to complete the robot.](image)

**Figure 10. Acer Aspire One netbook placed on the robot body to complete the robot.**

3 **Software Design**

The software used for programming, common modules for both the tasks, modules for the path follow task, and the modules for the red object follow task are described in this section.

3.1 **Computer software and library used**

3.1.1 **Visual Studio 2005: C/C++**

C/C++ language was used to program the robot because of its robustness, and Visual Studio 2005 was chosen for compiling the language. Visual Studio 2005 was chosen as the visual development environment is great for debugging code, and it is also readily compatible with the Open CV library.

3.1.2 **Open CV**

Open CV is an open source computer vision library developed by Intel, and is available for free for commercial and research applications. The library supports all major development environments, and includes libraries for matrix manipulation, image processing, GUI creation and machine learning. The library was configured for Visual Studio and used for image and video processing.
3.2 Common Tasks

The functions common to both the task are first described below.

3.2.1 Serial Port Communication

Windows.h library was used to communicate to the servo controller board via the serial port. The library can be used to create a handler for the serial port and then treat the serial port as a file. Information can then be sent or received through the serial port by reading and writing to the handler using the ReadFile and WriteFile functions respectively. An example C/C++ program to write to the serial port is given in Appendix A.

3.2.2 Image Extraction

The OpenCV library was linked and imported, and the functions from the library were used to extract images from the webcam. First, a capture object is created to read input from the webcam, and then an image is read from the capture object. This read image can then be processed as per requirement. Pseudo code to stream video from the webcam is given in Figure 11 below.

```
create captureObject from webcam
create an imagePointer
create a GUI window to display the stream
loop
    capture image from captureObject and store in imagePointer
    display it in the GUI window
    break if key is pressed
    call function to update GUI window
```

Figure 11. Pseudo code to stream video from a webcam

3.2.3 PID Controller

A PID controller was implemented in C/C++ to follow the artificial path and to follow detected red objects. The pseudo code used for the implementation is given below.

```
constants kp, ki, kd
pidcontroller(errorValue, timestep)
    static oldErrorValue, iterm
    pterm = kp * errorValue
    iterm = ki * (iterm + errorValue*timestep)
    dterm = kd * ((oldErrorValue-errorValue)/timestep)
    outputAction = pterm + iterm + dterm
    return outputAction
```

Figure 12. Pseudo code for a PID controller

3.2.4 Debugging/Logging Data

A text file was created for each run and all the data logged into the file for debugging purposes.
3.3 Path Follow

The path created using the black tapes was first detected and then the robot controlled using a PID controller to follow the path. To detect the path the image captured was first processed to detect the line strips created using black tape. The image was first converted to grayscale, then threshold, and then dilated to remove specks on the ground. The block diagram in the figure below depicts various steps used to process the image. After processing the image, the tape on the left and right were detected by looking at a horizontal line of pixel in the image. The horizontal black line drawn on the Figure 16 is the line of horizontal pixel used to detect the line. The number of pixels to the left and right from the center of the image to the lines were measured and this difference was sent to the PID controller to move the robot.

![Block Diagram]

Figure 13. Image processing to detect black lines in white grounds with speck

An image of the path as seen by the robot before processing and after image processing is shown in Figure 14 and Figure 15 respectively. The

![Image 14]

Figure 14. Image of the path captured by the robot

![Image 15]

Figure 15. Image of the path after processing it. The horizontal line in the image is the part of the tape line being sensed.
3.4 Red Object Follow

The red color was first extracted from a captured image, then the size and centroid of the red object was calculated and a PID controller was used to follow the detected object. The details of the process are given in the following sections.

3.4.1 Color Extraction

Red colored objects are rarer and therefore were chosen for this task and an algorithm written to extract red colored objects from captured images. The approach taken to detect red colored objects was to first separate the image to red, green and blue channels and threshold the red channel. White color has all three color components and therefore do not filter out of the red channel. To filter out the white color, the blue and green channels were used to detect white color and were subtracted. Two methods were implemented to filter out the white channel and are given below.

Method 1:

The red pixel values were normalized by looking at the corresponding pixel values at all the three different color channels. After normalizing, the pixel that were redder than the other colors were set as red and all other pixels were set as not-red. The method was good at separating out red objects from other objects but was inefficient as all the pixel in the three channels were begin read, and processed.

Method 2:

After extracting the three channels, a threshold was applied to all the three channels that converted pixels above a certain value to white and the pixels below the value to black. Then, for all the white pixels in the red channel, if it had a corresponding white pixel in the blue and green channel then it was filtered off as white color and converted to black. Then the image was eroded and then dilated to filter out noise from the image. A block diagram to visualize the processing steps is given in the following figure.

![Block diagram of step used to extract red color in an image.](image.png)
3.4.2 Detection: Flood Fill Algorithm

Flood fill algorithm calculates the area of all connected nodes in a multidimensional array. This algorithm is used by paint programs to implement the “Fill Color” tool, which changes the color of the chosen pixel and all the neighboring pixels of the same color to a different color. The flood fill algorithm was used to explore the processed red and not-red image to find all the connected red pixels in the image to measure the size of the red object and to calculate their centroid. As the robot only follows the biggest red object, the flood fill algorithm implemented only kept track of the largest white area and its centroid.

Implementation 1: Recursive Method

The flood fill algorithm was first implemented using a recursive function that modified a global area and centroid variable. The algorithm worked for small red objects, but was inefficiently slow and overflowed the stack when dealing with bigger red objects. It was also noted that the program crashed when the recursion depth exceeded about 4000 recursive calls. One solution was to increase the stack size for the program, but that would not improve the efficiency of the algorithm.

Implementation 2: Queue implementation

To deal with the issue of inefficiency and stack overflow, the flood fill algorithm was implemented by using a queue data structure to hold the nodes. The pseudo code for the implementation is given in the figure below and the C/C++ code is given in Appendix B.

```c
floodfill(node, targetcolor, newcolor)
    if pixel.color == targetcolor
        change node.color = newcolor
        initialize queue of nodes
        add node to queue
        while queue has nodes
            pop a node from the queue
            increase numberOfPixels by one
            for all neighboring nodes
                if node.color = targetcolor
                    node.color = newcolor
                    insert node in the queue

return numberOfPixels
```

Figure 17. Pseudo code for flood fill algorithm implemented with a queue.

Figure 18. Image of the red cup before image processing which was captured by the robot
4 Results

4.1 Path Following without a PID controller

The robot was first programmed with a non PID algorithm to follow the artificial path created by black tapes on the white floor. The goal of the robot was to stay in the middle of the path created by two black tapes, and so if it was closer to the tape on one side then it was programmed to turn in the other direction, if the tapes were equally far away on both the sides then it would move straight.

The robot successfully followed the path, but did not stay at the center of the path. The algorithm produced jerky motion. The servo motors were not calibrated, so instead of moving straight the robot actually drifted to one of the sides and this was one reason for the jerky motion.

4.2 Path Following using Ziegler-Nichols Method to set PID constants

The Proportional constant and time period at critical oscillation used to calculate the PID constants is given in the following table.

| Table 2. Proportion constant value and Time constant for Critical Oscillation |
|---------------------------------|---|---|
| Kc  | Pc  |
| 0.6 | 1.86 |

The following values for the Proportional, integral and the derivative term constants were calculated using the Zeigler-Nichols method. The PI and PID controllers did very poorly and deviated away from the path as the integral term set by this method was too large. The graph of the error term vs. time for the P controller is depicted in Figure 20 below.

<table>
<thead>
<tr>
<th>Table 3. PID constant values set using the Zeigler-Nichols method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant for</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>Proportional Term</td>
</tr>
<tr>
<td>Integral Term</td>
</tr>
<tr>
<td>Derivative Term</td>
</tr>
</tbody>
</table>
4.3 PID Control using trial and error

The Kp value calculated by the Zeigler-Nichols method was used and the Ki and the Kd values were tweaked to get the best performance. This resulted in better control and the graphs for two different experimental values of the PID constants are given in Figure 21 and Figure 22 below. The PID controller also corrected the drifting produced by the un-calibrated servo motors. The PID constant values by trial and error for two tests are given in the following table. The code for the path following program is given in Appendix C.

<table>
<thead>
<tr>
<th>Constant for</th>
<th>Experimental Values</th>
<th>Experimental Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportional Term</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Integral Term</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Derivative Term</td>
<td>0.12</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Figure 20. Graph of error term vs. time using the PID controller with Kp = 0.3, Ki = 0 and Kd = 0.

Figure 21. Graph of error term vs. time using the PID controller with Kp = 0.3, Ki = 0.05 and Kd = 0.12.
4.4 Red Object Following

Red objects were successfully detected and followed. The PID constant were chosen by trial and error and is given in the following table. The X component of the error will change more if the object is far away and less if it is near. So, an adaptable proportional component for the PID controller was implemented by calibrating it for different distances. As the constant was calibrated by using a stationary red object, it did not adapt well to moving red object. Optimal constant values can be measured for different speed at different distances by moving them around the robot. This can be then used to set the constant values adaptively and should produce much better result. The complete C/C++ code for the red object following task is given in Appendix D.

Table 5. PID constants for objects different distance away from the robot

<table>
<thead>
<tr>
<th>Constants</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPx</td>
<td>0.75</td>
</tr>
<tr>
<td>KPy</td>
<td>1.2</td>
</tr>
<tr>
<td>KIx</td>
<td>0</td>
</tr>
<tr>
<td>KIy</td>
<td>0</td>
</tr>
<tr>
<td>KDx</td>
<td>0</td>
</tr>
<tr>
<td>KDy</td>
<td>0</td>
</tr>
</tbody>
</table>

5 Conclusion

A PID controller was found to be a good controller for robotic systems, but the Zeigler-Nichols method is not an efficient way to set the PID constant values. Setting the PID constant values by trial and error is time consuming and will not produce an optimal solution. If a PID controller is to be used to control robots, an algorithm should be developed to set the PID constant values.
Appendix A: Serial Port Programming

```cpp
#include <windows.h>
#include <iostream>
using namespace std;

int main(int argc, char* argv[]){
    //open port for I/O
    HANDLE hSerial = CreateFile(L"COM3",
        GENERIC_READ|GENERIC_WRITE,
        NULL,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if(hSerial==INVALID_HANDLE_VALUE){
        cout<<"invalid handle."<<endl;
    } D
    DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

    if (!GetCommState(hSerial, &dcbSerialParams)) {
        cout<<"error occurred at getCommState"<<endl;
    }
    //setting parameters
    dcbSerialParams.BaudRate = CBR_115200;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;

    char buff1[] = ";#1P1000\r";
    int siz1 = sizeof(buff1);
    DWORD dwBytesRead = 0;
    while(true){
        WriteFile(hSerial, buff1, siz1, &dwBytesRead, NULL);
    }
    return 0;
}
```
Appendix B: Flood Fill

```c
void floodFill(IplImage* im, int X, int Y, areaStore& max, int target=-1, int changeto=-2)
{
    if(im->imageData[X+Y*im->widthStep]==target)
    {
        //do everything only if the passed pixel has target color
        queue<point> allPix;
        int x, y; //variables to store x and y values
        int step = im->widthStep; //widthstep of the image
        int numPixels=0; //total number of pixels
        long int totalX=0, totalY=0;
        int avgX=0, avgY=0;
        //insert only stuff with the target color into the queue
        allPix.push(point(X,Y));
        point tempPoint;
        im->imageData[X+Y*step] = changeto;
        while(!allPix.empty()){
            tempPoint = allPix.front();
            allPix.pop();
            x = tempPoint.x;
            y = tempPoint.y;
            numPixels++;
            totalX +=x;
            totalY +=y;
            for (int i = -1; i<2; i++)
            {
                for (int j = -1; j<2; j++)
                {
                    if (!((i==0)&&(j==0)) && im->imageData[x+i+(y+j)*step]==target){
                        im->imageData[x+i+(y+j)*step]=changeto;
                        allPix.push(point(x+i, y+j));
                    }
                }
            }
        }
        if(numPixels>1000 && numPixels>max.numPixels){
            avgX= totalX/numPixels;
            avgY= totalY/numPixels;
            calcHW(im, avgX, avgY, max, changeto);
            //update max area
            max.numPixels = numPixels;
            max.averageX = avgX;
            max.averageY = avgY;
        }
    }
    else{
        return;
    }
}
```
Appendix C: Line Follow

```cpp
#include <iostream>
#include <windows.h>
#include <time.h>
#include <cstring>
#include <string>
#include <fstream>
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
using namespace std;

bool DEBUG=false;
//global variables
HANDLE hSerial; //handler for the serial port
CvCapture* capture; //opencv image capture object
IplImage* src; //image object
ofstream ofile("output.txt"); //logfile
float kp, ki, kd;

//0 -> left servo
//1 -> right servo
void move(int diff){
    static int olddiff =0, integ = 0;
    static int oldolddiff=1000;
    int diffValue;

    //median filter
    if(diff<= olddiff && diff <= oldolddiff){
        if(olddiff<= oldolddiff){//median filter
            diffValue = olddiff;
        }
        else{
            diffValue = oldolddiff;
        }
    }
    else if( olddiff<= diff && olddiff<=oldolddiff){
        if(diff<= oldolddiff){//median filter
            diffValue = diff;//no change
        }
        else{
            diffValue = oldolddiff;
        }
    }
    else{
        if(diff<=olddiff){
            diffValue = diff;
        }
        else{
            diffValue = olddiff;
        }
    }
    DWORD a=0;
    int l = 1800, r =1220;
    //int l = 1700, r =1340;
    char lbuff[5], rbuff[5];//shouldnt need more than 4 spaces+
    string left="#0P", right="#1P"
    //P part
    r +=diffValue*kp;
    l +=diffValue*kp;
    //I part
    integ = integ + diffValue;
    //integx
    l+= integ * ki;
    r+= integ * ki;
```

// D part
int deriv = diffValue - olddiff;
L += deriv * kd;
R += deriv * kd;
if (L > 3000) { L = 3000; }
if (L < 0) { L = 0; }
if (R > 3000) { R = 3000; }
if (R < 0) { R = 0; }

// changing to string
itoa(L, lbuff, 10);
itoa(R, rbuff, 10);
left += lbuff;
right += rbuff;
cout << "diff/left/right: " << diffValue << " / " << left << " / " << right << endl;
left = " \r ";
right = " \r ";
WriteFile(hSerial, left.c_str(), 8, &a, NULL);
WriteFile(hSerial, right.c_str(), 8, &a, NULL);
cout << "Moving Forward" << endl;
ofile << diffValue << " \r ";
oldolddiff = olddiff;
olddiff = diff;
}

void stop()
{
    DWORD a = 0;
cout << "Stopping" << endl;
    WriteFile(hSerial, "#0P1500\r", 8, &a, NULL);
    WriteFile(hSerial, "#1P1500\r", 8, &a, NULL);
}

void init()
{
    // initialize serial handler
    // initialize opencv capture handler
    hSerial = CreateFile(L"COM3", GENERIC_READ|GENERIC_WRITE,
 NULL, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        cout << "Invalid handle." << endl;
    }
    DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    // get the current state of the serial handler
    if (!GetCommState(hSerial, &dcbSerialParams)) {
        cout << "Error getting CommState" << endl;
    }
    // modify the state
    dcbSerialParams.BaudRate = CBR_115200;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;
    // set the serial handler state
    if (!SetCommState(hSerial, &dcbSerialParams)) {
        cout << "Error setting the CommState" << endl;
    }
    capture = cvCaptureFromCAM(CV_CAP_ANY);
    if (capture == NULL) {
        cout << "Error setting up the CvCapture object" << endl;
    }
    // first few retrieved frames are blank
    // so just throwing a few frames
    for (int count = 0; count < 10; count++) {
        src = cvQueryFrame(capture);
    }
    ifstream pidconsts("pid_constants");
    pidconsts >> kp >> ki >> kd;
int pixelVal(IplImage* source, int height, int width){
    int ans = {uchar*{(source->imageData+source->widthStep*(height))}[width]};
    return ans;
}

int decide(IplImage* source){
    float fromTop = 0.6; // looking at the pixel line this far from top
    int height = (source->height)*fromTop;
    int center = (source->width)/2;
    int numLeft, numRight;
    // if the pixel in the middle is dark then stop
    if(pixelVal(source, height, center)==0){
        return 0;
    }
    // if there are more approachable(brighter)
    // pixels to the left then move left and vice versa
    // look at right to count pixels
    int count=0;
    int pix1, pix2;
    pix1 = pixelVal(source, height, center+count);
    pix2 = pixelVal(source, height, center+count+1);
    for(count=1; count<(center*2); count++){
        // stop if two pixels have the same value
        if(pix1==0&&pix2==0){
            break;
        }
        pix1=pix2;
        pix2=pixelVal(source, height, center+count+1);
    }
    numRight = count;
    // look at left to count pixels
    count=0;
    pix1 = pixelVal(source, height, center-count);
    pix2 = pixelVal(source, height, center-count-1);
    for(count=1; count<(center*2); count++){
        // stop if two pixels have the same value
        if(pix1==0&&pix2==0){
            break;
        }
        pix1=pix2;
        pix2=pixelVal(source, height, center-count-1);
    }
    numLeft = count;
    return numRight-numLeft;
}

int main(int argc, char* argv[]){
    init(); // initialize
    if(DEBUG){cvNamedWindow("Image", 1);}
    cvNamedWindow("grayImage", 1);
    // time_t before, after;
    // before = time(NULL);
    // log file
    char keyPressed=0;
    IplImage* srcGray = cvCreateImage(cvGetSize(src), 8, 1);
    CvPoint pt1, pt2;
    while(true){
        // after = time(NULL);
        // if( after-before >1){
        src = cvQueryFrame(capture);// get frame
        // change to grayscale
        cvConvertImage(src, srcGray, 1);
//change to black and white
    cvAdaptiveThreshold(srcGray, srcGray, 255,
          CV_ADAPTIVE_THRESH_MEAN_C, CV_THRESH_BINARY, 1001, 10);
    cvDilate(srcGray, srcGray, 0,1);
    cvErode(srcGray, srcGray, 0,1);
    cvDilate(srcGray, srcGray, 0,1);
    int ab= decide(srcGray);//decide what to do
    move(ab);

    if(DEBUG){cvShowImage("Image", src);}
    pt1.x = 0;
    pt2.x = 640;
    pt1.y = 0.6* srcGray->height;
    pt2.y = 0.6* srcGray->height;
    cvLine( srcGray, pt1, pt2, CV_RGB(0,0,0), 2, CV_AA, 0 );
    cvShowImage("grayImage", srcGray);
    //before - after;
    keyPressed = cvWaitKey(1);
    if(keyPressed=='q'||keyCode=='Q'){
       stop();
       break;
    }
}
cvDestroyWindow("grayImage");
if(DEBUG){cvDestoryWindow("Image");}
return 0;
Appendix D: Red Object Follow

#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <queue>
#include <cmath>
#include <string>
#include <fstream>
using namespace std;

// struct to store points and to create points easily
// CvPoint doesn't have that constructor???
struct point{
    int x;
    int y;
    point(){
        x=0;
        y=0;
    }
    point(int a, int b){
        x=a;
        y=b;
    }

    point& operator=(const point& that){
        if(this!= &that){
            this->x=that.x;
            this->y=that.y;
        }
        return *this;
    }
};

// struct that will store info about the area
struct areaStore{
    int numPixels;
    int averageX;
    int averageY;
    int height;
    int width;

    areaStore(){
        numPixels = 0;
        averageX = 0;
        averageY = 0;
        height = 0;
        width = 0;
    }

    areaStore(int nPix, int avgx, int avgy, int h, int w){
        numPixels=nPix;
        averageX=avgx;
        averageY=avgy;
        height=h;
        width=w;
    }

    void clear(){
        numPixels = 0;
        averageX = 0;
        averageY = 0;
        height = 0;
        width = 0;
    }

    areaStore& operator=(const areaStore& that){
        if(this != &that){
            return *this;
        }
    }
};
this->numPixels = that.numPixels;
this->averageX = that.averageX;
this->averageY = that.averageY;
this->height = that.height;
this->width = that.width;
}
return *this;
}

};
//globals
//pid constants
float kpx, kpy, kdx, kdy, kix, kiy;
int WAITTIME= 50;
//areaStore oldValue;
//serial handler
HANDLE hSerial;
ofstream logFile("logfile.txt");
ofstream HWFile("distance.txt");

//areaStore is also passed
void calcHW(IplImage* im, int x, int y, areaStore& max, int pixVal=-1){
  //probably for rectangular objects
  //or atleasr for objects with centroid inside them
  int pUp=0, pDown=0, pLeft=0, pRight=0;
  int yPos = y*im->widthStep;
  if( (int)im->imageData[x + yPos]==pixVal){
    pUp = y+1;
pDown = y;
pLeft= x;
pRight= x+1;
    //count vertical pixels up
    while((int)im->imageData[x + pUp*im->widthStep]==pixVal){
      pUp++;
    }
    //count vertical pixels down
    while((int)im->imageData[x+pDown*im->widthStep]==pixVal){
      pDown--;
    }
    //count vertical pixels left
    while((int)im->imageData[pLeft+yPos]==pixVal){
      pLeft--;
    }
    //count vertical pixels right
    while((int)im->imageData[pRight+yPos]==pixVal){
      pRight++;
    }
  max.height=pUp-pDown;
  max.width =pRight-pLeft;
}
return;
}

//also passes areaStore reference to update the max area
void floodFill(IplImage* im, int X, int Y, areaStore& max, int target=-1, int changeto=-2){
  if(im->imageData[X+Y*im->widthStep]==target){
    //do everything only if the passed pixel has target color
    queue<point> allPix;
    int x, y;//Variables to store x and y values
    int step = im->widthStep;//widthstep of the image
    int numPixels=0; //total number of pixels
    long int totalX=0, totalY=0;
    int avgX=0, avgY=0;
    //insert only stuff with the target color into the queue
    allPix.push(point(X,Y));
    point tempPoint;
im->imageData[X*Y*step] = changeto;
    while(!allPix.empty()){ //while queue is not empty
      tempPoint = allPix.front();
      allPix.pop();
      allPix.push(tempPoint);
      //update the total values
      totalX += x;
totalY += y;
      numPixels++;
      avgX = totalX/numPixels;
      avgY = totalY/numPixels;
    }
    //update the max area
    max->area = numPixels;
    max->centerX = avgX;
    max->centerY = avgY;
    max->height = max->width = -1;
  }
}

//also passes areaStore reference to update the max area
void updateMaxArea(IplImage* im, int x, int y, areaStore& max){
  //update the area
  if(x >= 0 && y >= 0 && x < im->width && y < im->height){
    int absX = x;
    int absY = y;
    int step = im->widthStep;
    int numPixels=0; //total number of pixels
    long int totalX=0, totalY=0;
    int avgX=0, avgY=0;
    //insert only stuff with the target color into the queue
    allPix.push(point(X,Y));
    point tempPoint;
im->imageData[X*Y*step] = changeto;
    while(!allPix.empty()){ //while queue is not empty
      tempPoint = allPix.front();
      allPix.pop();
      allPix.push(tempPoint);
      //update the total values
      totalX += x;
totalY += y;
      numPixels++;
      avgX = totalX/numPixels;
      avgY = totalY/numPixels;
    }
    //update the max area
    max->area = numPixels;
    max->centerX = avgX;
    max->centerY = avgY;
    max->height = max->width = -1;
  }
}
x = tempPoint.x;
y = tempPoint.y;
numPixels++;
totalX += x;
totalY += y;
for (int i = -1; i<2; i++){
    for (int j = -1; j<2; j++){
        if (!((i==0) && (j==0)) && (x+i)>=0 && (x+i)<640 && (y+j)>=0 && (y+j)<480) {
            if (im->imageData[(x+i)+(y+j)*step]==target) {
                im->imageData[(x+i)+(y+j)*step]=changeto;
                allPix.push(point(x+i, y+j));
            }
        }
    }
}
if(numPixels>800 & numPixels>max.numPixels) {
    avgX = totalX/numPixels;
    avgY = totalY/numPixels;
    calcHW(im, avgX, avgY, max, changeto);
    //update max area
    max.numPixels = numPixels;
    max.averageX = avgX;
    max.averageY = avgY;
} else {
    return;
}
void mouseHandler(int type, int x, int y, int flag, void* param){
    switch(type) {
        case CV_EVENT_LBUTTONDOWN:
            IplImage* img = cvCloneImage((IplImage*) param);
            y = img->height - y;
            floodFill((IplImage*)param, x, y);
            break;
    }
}
void mouseRGB(int type, int x, int y, int flag, void* param){
    switch(type) {
        case CV_EVENT_LBUTTONDOWN:
            IplImage* im = (IplImage*) param;
            y = im->height - y;
            cout<<"x/y/index: "<<x<<"/"<<y<<"/"<<3*x+y*im->widthStep<<endl;
            cout<<"Blue: "<<im->imageData[3*x+y*im->widthStep] <<endl;
            CvPoint a;
            a.x = x;
            a.y = y;
            cvCircle(im, a, 3, CV_RGB(255,255,255), 2, 8);
            break;
    }
}
void searchArea(IplImage* im, areaStore& max, int target = -1, int changeto = -2){
    //searches all pixels of the passed image to find white areas
    //prints out the total number of pixels and the centroid of
    //each of the found area.
    for(int x=10; x< im->width; x+=10) {
        for( int y=10; y<im->height; y+=10) {
            floodFill(im, x, y, max);
        }
    }
    cout<<"numPix, avgx, avgy, h, w: "<< max.numPixels<<" , "<<max.averageX<<" , "<<max.averageY<<" , "<<max.height<<" , "<<max.width<<endl;
void stop(){
    unsigned long a=0;
    WriteFile(hSerial, "#1P1500\r", 8, &a, NULL);
    WriteFile(hSerial, "#0P1500\r", 8, &a, NULL);
}

void move(areaStore& realPos){
    static int oldxdiff=0, oldydiff=0;
    static int integx=0, integy=0;
    if(realPos.averageX==0&&realPos.averageY==0){
        stop();
    } else{
        DWORD a=0;
        int l=1500, r=1500;//start at 0 speed
        int dummy=0;
        char lbuff[5], rbuff[5]; //shouldnt need more than 4 spaces+\0
        //these variables dont work for values less than 1000 as that uses only 3 chars
        areaStore targetPos(0, 320, 125, 0, 0);
        //P part
        int xdiff = realPos.averageX - targetPos.averageX;
        int ydiff = realPos.averageY - targetPos.averageY;
        //kpx = -0.00025 * pow((float)realPos.height,2)+ 0.049* realPos.height-1.6;
        //kpx=0.8;
        //xdiff * kpx;
        l+=xdiff * kpx;
        r+=xdiff * kpx;
        //ydiff * kpy;
        l+=ydiff * kpy;
        r-=ydiff * kpy;
        //I part
        integx = integx + xdiff * kix;
        integy = integy + ydiff * kiy;
        //integx
        l-= integx;
        r+= integx;
        //integy
        l+= integy;
        r-= integy;
        //D part
        int derivx = (xdiff-oldxdiff)*kdx;
        int derivy = (ydiff-oldydiff)*kdy;
        //derivx
        l+=derivx;
        r-=derivx;
        //derivy
        l-=derivy;
        r+=derivy;///???????????
        if(l>3000){
            l=3000;
        } else{
            l=0;
        }
        if(r>3000){
            r=3000;
        } else{
            r=0;
        }
        //send the signal to the servo controller board
itoa(l, lbuff, 10);
itoa(r, rbuff, 10);
string left="#0P", right="#1P";
left += lbuff;
right += rbuff;

//for debugging purposes
cout<<"xdiff/ydiff/left/right: ";
cout<<xdiff<<"/";cout<<ydiff<<"/";cout<<left<<"/";cout<<right<<"\n";

left +="r";
right +="r";
WriteFile(hSerial, left.c_str(), 8, &a , NULL);
WriteFile(hSerial, right.c_str(), 8, &a , NULL);

//store values
olddiff = xdiff;
olddiff = ydiff;
//cout<<"Moving"<<\n;
}

void init(){
//initialize serial handler
//initialize opencv capture handler
hSerial = CreateFile(L"COM3", GENERIC_READ|GENERIC_WRITE,
NULL, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
if(hSerial==INVALID_HANDLE_VALUE){
cout<<"Invalid handle."<<\n;
}
DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
//get the current state of the serial handler
if (!GetCommState(hSerial,
&dcbSerialParams)) {
cout<<"Error getting CommState"<<\n;
}
//modify the state
dcbSerialParams.BaudRate=CBR_115200;
dcbSerialParams.ByteSize=8;
dcbSerialParams.StopBits=ONESTOPBIT;
dcbSerialParams.Parity=NOPARITY;
//set the serial handler state
if(!SetCommState(hSerial, &dcbSerialParams)){
cout<<"Error setting the CommState"<<\n;
}
fstream pidconsts("pid_constants");
pidconsts>>kpx>>kix>>kdx>>kpy>>kiy>>kdy;
}

int main() {
//initialize serial port
init();
//declarations
cvNamedWindow( "all", 1 );
cvNamedWindow( "onlyred", 1 );

/*
kix = 0.08;
kdx = 0.1;
kpy = 1.2;
kiy = 0.0;
kdy = 0;
*/
cout<<kpx<<kpy<<kix<<kiy<<kdx<<kdy<<\n;
cvNamedWindow( "b", 1);
cvNamedWindow( "test1", 1);
cvNamedWindow( "test2", 1);
*
int tBarVal = 100;
char keyPressed = 0;
areaStore max;
cvCreateTrackbar("Thresh", "onlyred", &tBarVal, 200, NULL);
CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
if ( !capture ) {
    cout<< "ERROR: capture is NULL 
"
    return -1;
}
IplImage* frame;
// first few frames are blank so throwing them away
for( int i = 0; i<10; i++ ){
    frame = cvQueryFrame( capture );
}
IplImage* r = cvCreateImage(cvGetSize(frame), 8, 1);
r->origin = 1; // else its 0 and upside down
IplImage* onlyr = cvCreateImage(cvGetSize(frame), 8, 1);
onlyr->origin = 1; // else its 0 and upside down
IplImage* g = cvCreateImage(cvGetSize(frame), 8, 1);
g->origin = 1; // else its 0 and upside down
IplImage* b = cvCreateImage(cvGetSize(frame), 8, 1);
b->origin = 1; // else its 0 and upside down
IplImage* test1 = cvCreateImage(cvGetSize(frame), 8, 1);
test1->origin = 1; // else its 0 and upside down
IplImage* test2 = cvCreateImage(cvGetSize(frame), 8, 1);
test2->origin = 1; // else its 0 and upside down

cvSetMouseCallback("onlyred", mouseHandler, onlyr);
cvSetMouseCallback("all", mouseRGB, frame);
CvPoint p1, p2, p3, p4;
p1.y = 0;
p2.y = 480;
p3.x = 0;
p4.x = 640;

while (true){
    frame = cvQueryFrame( capture );
    if ( !frame ) {
        cout<<"ERROR: frame is null...
"
    }
    cvCvtColor( frame, b, g, r, 0);
    // threshold them
    cvThreshold(b, onlyr, tBarVal, 255, CV_THRESH_BINARY);
    cvThreshold(g, g, tBarVal, 255, CV_THRESH_BINARY);
    cvThreshold(b, b, tBarVal, 255, CV_THRESH_BINARY);
    for( int i = 0; i<onlyr->imageSize; i++ ){
        if(onlyr->imageData[i] == -1) {
            if(b->imageData[i] == -1 || g->imageData[i] == -1) {
                onlyr->imageData[i] = 0;
            }
        }
    }
    cvDilate(onlyr, test1, 0, 3);
    cvDilate(onlyr, onlyr, 0, 1);
    cvErode(onlyr, onlyr, 0, 1);
    cvErode(onlyr, onlyr, 0, 2);
    cvErode(onlyr, onlyr, 0, 2);
    max.clear();
    searchArea(onlyr, max);
    p1.x = max.averageX;
    p2.x = max.averageX;
}
p3.y = max.averageY;
p4.y = max.averageY;

cvLine(onlyr, p1, p2, CV_RGB(255,255,255), 2);
cvLine(onlyr, p3, p4, CV_RGB(255,255,255), 2);

cvShowImage( "all", frame);
cvShowImage( "onlyred", onlyr);
move(max);

/*/  
cvShowImage( "g", g);
cvShowImage( "b", b);
cvShowImage( "test1", test1);
cvShowImage( "test2", test2);
*/
keyPressed = cvWaitKey(1);
if(keyPressed == 'q' || keyPressed =='Q'){
    stop();
    break;
}
//cvReleaseImage(&frame);
//cvReleaseImage(&onlyr);
//cvReleaseImage(&r);
//cvReleaseImage(&g);
//cvReleaseImage(&b);

cvReleaseCapture(&capture);
logFile.close();
return 0;