Comparing Different Visual Motion Detection Algorithms

New Mexico Supercomputing Challenge Final Report

April 1, 2014

Team: 143

School of Dreams Academy

Team Members:

Albert Reed

Zack Daniels

Chloe Grubb

Teacher/ Project Mentor:

Creighton Edington

Contents

Executive Summary

Visual motion detection is a field that can be used in a variety of different applications, ranging from security to navigation. One of the largest drawbacks of optical motion detection is how computationally heavy it can be. The goal of this project is to compare the computational time and effectiveness of three different motion detection algorithms, specifically on a microcontroller.

This project is intended to be applied to a security device for law enforcement. The specific desired outcome is an effective motion detection algorithm that runs on a microcontroller well enough to fit desired specifications. Said specifications include but are not limited to the ability to notice movement, the ability to ignore repetitive and insignificant movement, and the ability to iterate through the algorithm quickly enough to indicate detected movement at a steady rate. Testing included ratings on accurate motion detection, occurrences of false positives, and time taken for iterations of the algorithms to completion of test.

Three different algorithms were tested: frame subtraction, background subtraction, and a modified version of the frame subtraction method. Frame subtraction, although very simple, produced many false positives. Background subtraction gave the best performance in motion detection and tracking, but the results show that the modified frame subtraction algorithm is the most effective in regards to the security device, as it is computationally lighter than background subtraction and more accurate than the standard frame subtraction method.

Introduction

Background:

The problem of developing a motion tracking algorithm first arising with another project called Police ALERT (Awareness for Law Enforcement Reinforced with Technology). The Police A.L.E.R.T is a safety device that utilizes visual motion detection to assist law enforcement while they are in their vehicle and their attention is not on their surroundings. The Police A.L.E.R.T uses a microcontroller and a camera in order to process things seen around a police vehicle. This project was conducted to test three different visual motion detection algorithms in order to find one that will be the most applicable to the Police A.L.E.R.T. The algorithms tested include frame subtraction, background subtraction, and a modified version of the frame subtraction method.

Algorithms:

Frame subtraction is a very simple method of motion detection. First, a frame is taken and saved as a matrix with a value for every pixel in the image. Then another frame is taken and saved as a second matrix. In order to find motion, the matrices are subtracted. Differences in pixel values are seen as motion. One issue with this method is it is very sensitive and prone to false positives. In order to adjust for that, motion was marked with a bounding box that would only appear if the motion was large enough for the area of the bounding box to meet the established threshold.

Background subtraction, the second method tested, is similar yet much more complex than frame subtraction. The difference is background subtraction utilizes Gaussian mixtures and a confidence interval in order to ignore repetitive movement, such as a flag waving in the background. Each pixel has its own mean. Like frame subtraction, frames are stored as matrices.

Motion is detected by comparing frames to past frames. If pixel values fall within the confidence interval of the mean of pixels from past frames, they are ignored. This allows for the excluding of repetitive motion because as that motion is seen over and over again, the mean and confidence interval of the pixels changes to include the values that are seen.

The final method tested was a modified frame subtraction. The difference is an implementation that allows it to ignore repetitive motion using bounding boxes. If motion is detected, a bounding box is placed around it and the x and y coordinates of the top left and bottom right corners of that box are recorded. If motion is detected again and the coordinates of the box fit within the x and y values taken from the first box, a warning is given but the motion will be marked. If the previous step is repeated, the warning is abolished and the motion is ignored.

Purpose and Goals:

We plan to test the three different methods for accuracy of detection and time taken to iterate through the algorithm once. They were also evaluated to see if they meet standards set for implementation with the Police A.L.E.R.T device. The standards are as follows:

- The algorithm must be able to detect an occurrence of motion accurately
- The algorithm must be capable of ignoring insignificant and repetitive motion
- The algorithm must be able to update detected motion at a suitable speed
- The algorithm must be implementable on a microcontroller
- The algorithm must not have a large number of occurrences of false positives

Insignificant movement is considered to be things such as flickering lights and movement that is very small, such as a leaf blowing in wind. Because the algorithms are to be implemented into a security device, the detection of a human was the focus.

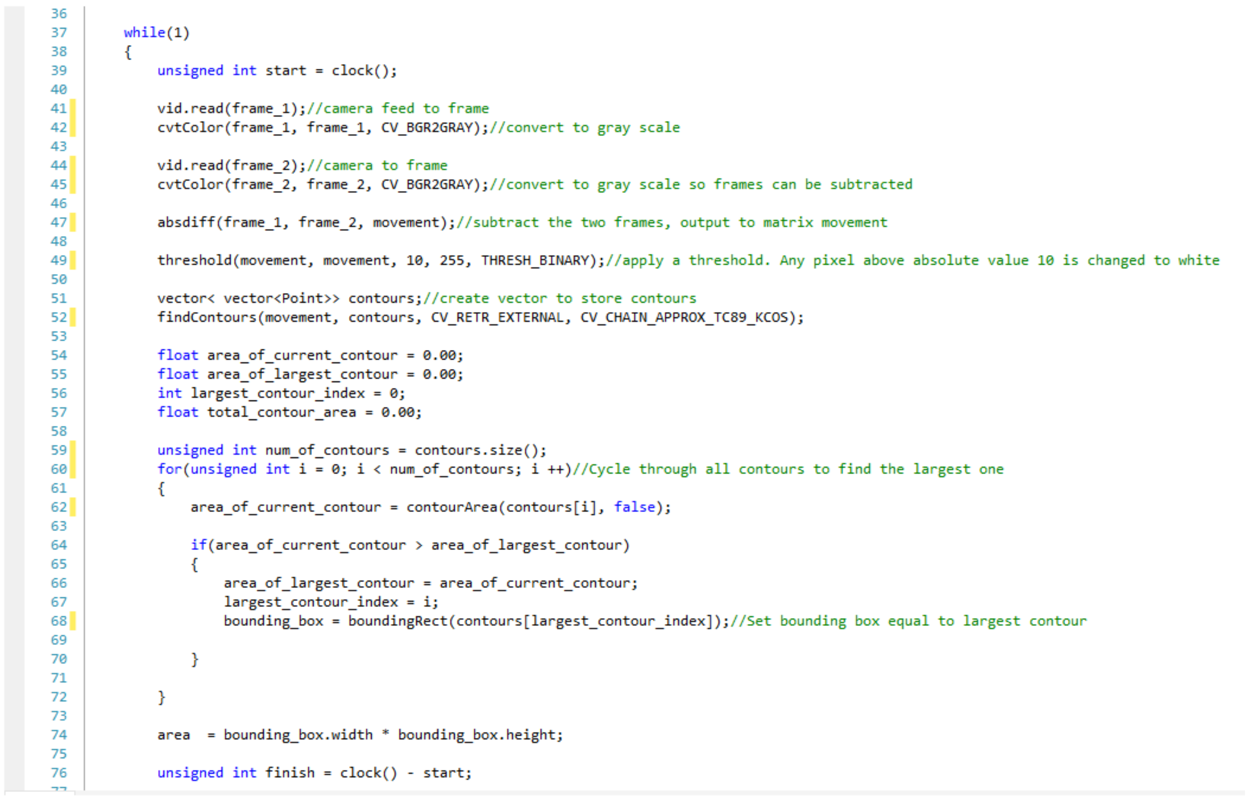
Methods and Materials

This project was completed using a laptop with a webcam running Windows 7. Code was written in C++ with the Microsoft Visual Studios 2010 Professional IDE, and data was handled with Microsoft Excel 2010. Computer vision was done using the OpenCV open source computer vision library.

The first step of the project was to find a library for computer vision. After some research, OpenCV was deemed as the best candidate, as it is free and has a great deal of functionality. C++ was the language of choice because OpenCV offers a great deal of support for it. OpenCV for Windows offers installation with pre built libraries if using Microsoft Visual Studios, so that was selected as the IDE.

After the installation of all of the necessary software, the next step was to research motion detection algorithms. Frame subtraction, background subtraction, and the modified frame subtraction method were chosen because of their simplicity and functionality. The first algorithm written was the simple frame subtraction.

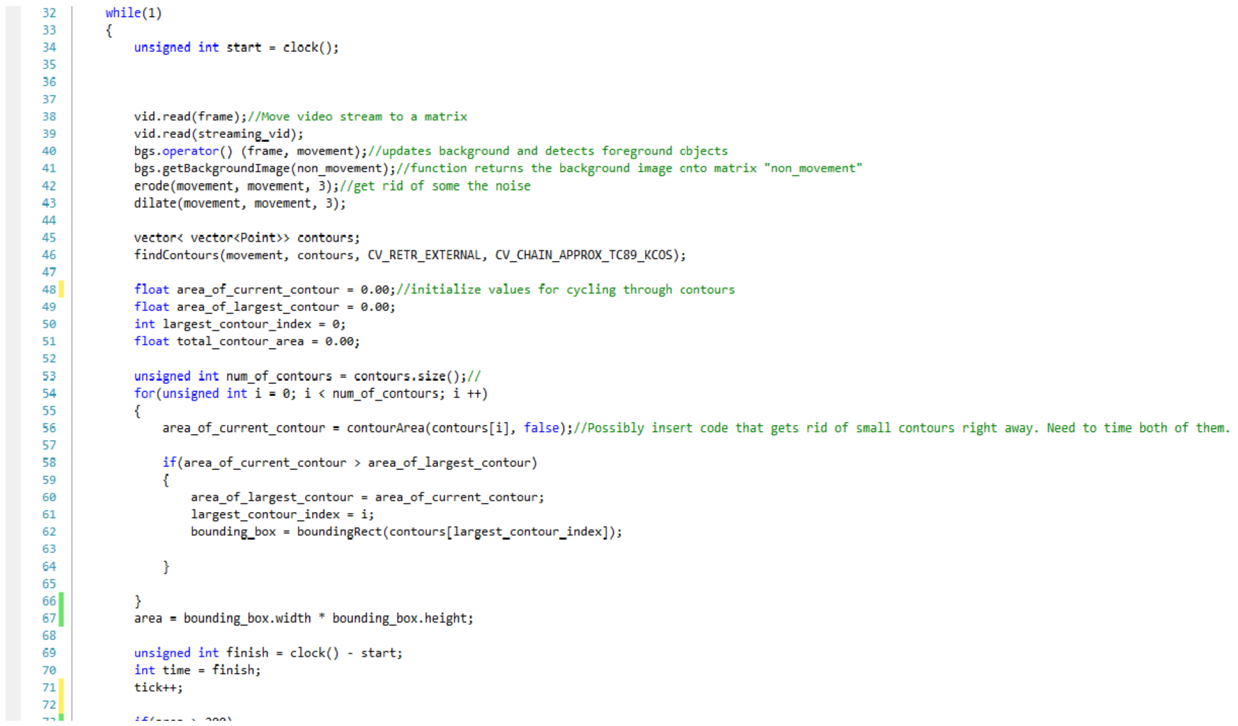
Frame subtraction is the simplest method. In summary, it stores a frame as a matrix with a value for each pixel, stores a second frame as a matrix, and subtracts the two matrices. Differences in pixel values indicate movement.



```
    while(1)
    {
        unsigned int start = clock();

        vid.read(frame_1);//camera feed to frame
        cvtColor(frame_1, frame_1, CV_BGR2GRAY);//convert to gray scale

        vid.read(frame_2);//camera to frame
        cvtColor(frame_2, frame_2, CV_BGR2GRAY);//convert to gray scale so frames can be subtracted

        absdiff(frame_1, frame_2, movement);//subtract the two frames, output to matrix movement

        threshold(movement, movement, 10, 255, THRESH_BINARY);//apply a threshold. Any pixel above absolute value 10 is changed to white

        vector< vector<Point>> contours;//create vector to store contours
        findContours(movement, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_TC89_KCOS);

        float area_of_current_contour = 0.00;
        float area_of_largest_contour = 0.00;
        int largest_contour_index = 0;
        float total_contour_area = 0.00;

        unsigned int num_of_contours = contours.size();
        for(unsigned int i = 0; i < num_of_contours; i ++)//Cycle through all contours to find the largest one
        {
            area_of_current_contour = contourArea(contours[i], false);

            if(area_of_current_contour > area_of_largest_contour)
            {
                area_of_largest_contour = area_of_current_contour;
                largest_contour_index = i;
                bounding_box = boundingRect(contours[largest_contour_index]);//Set bounding box equal to largest contour

            }

        }

        area  = bounding_box.width * bounding_box.height;

        unsigned int finish = clock() - start;
```

The code above is the algorithm for frame subtraction. First, frames are stored to a matrix. They are then converted to greyscale. This process is repeated for two frames. They are then subtracted. A threshold is applied to change the values to either black or white, which makes movement stand out. Movement is seen as contours, so a vector is created to store contours. Contours are then found, and their areas are stored to variables. Contours are then compared in order to find the largest, which is then marked with a bounding box.

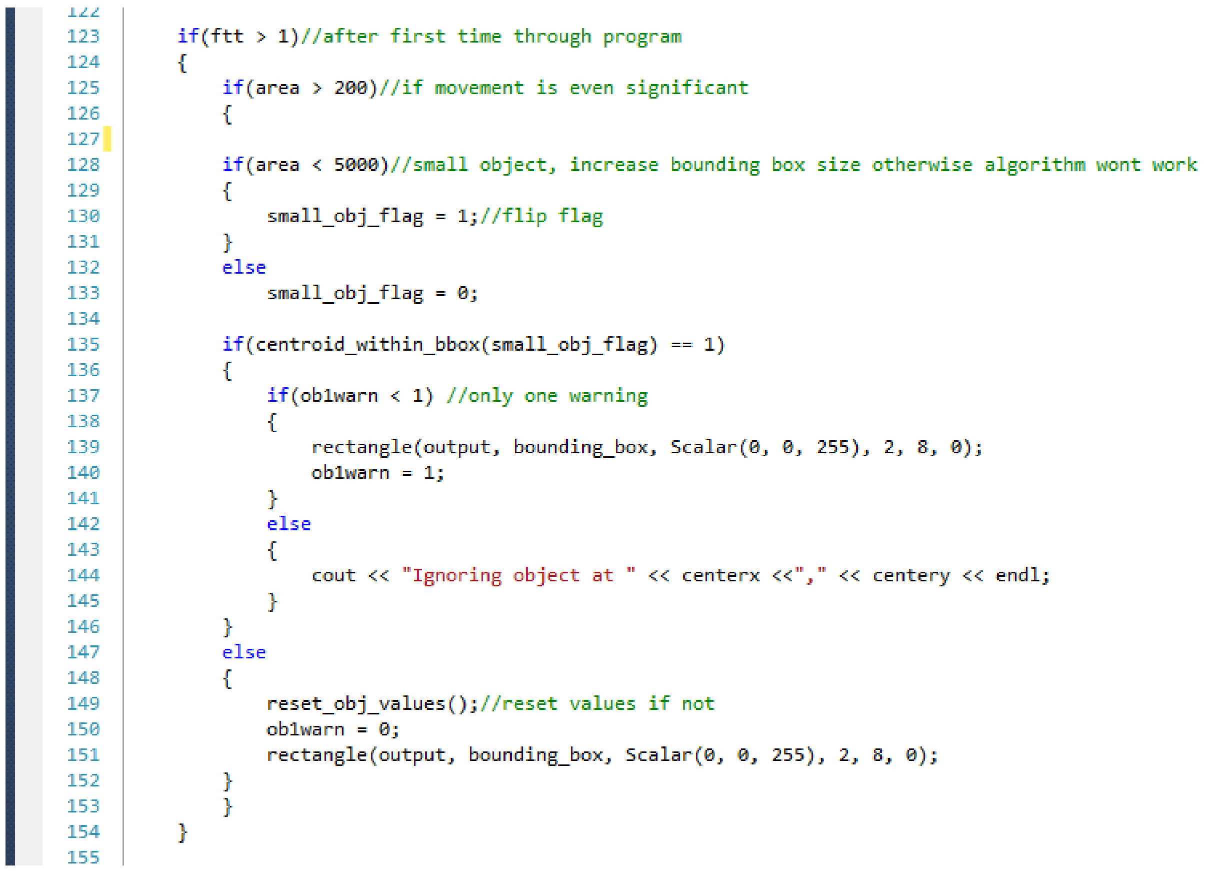
The second method constructed was background subtraction. Background subtraction is a much more complex method, as it performs mathematical operations for each pixel.



```
while(1)
{
    unsigned int start = clock();



    vid.read(frame);//Move video stream to a matrix
    vid.read(streaming_vid);
    bgs.operator() (frame, movement);//updates background and detects foreground objects
    bgs.getBackgroundImage(non_movement);//function returns the background image onto matrix "non_movement"
    erode(movement, movement, 3);//get rid of some the noise
    dilate(movement, movement, 3);

    vector< vector<Point>> contours;
    findContours(movement, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_TC89_KCOS);

    float area_of_current_contour = 0.00;//initialize values for cycling through contours
    float area_of_largest_contour = 0.00;
    int largest_contour_index = 0;
    float total_contour_area = 0.00;

    unsigned int num_of_contours = contours.size();//
    for(unsigned int i = 0; i < num_of_contours; i ++)
    {
        area_of_current_contour = contourArea(contours[i], false);//Possibly insert code that gets rid of small contours right away. Need to time both of them.

        if(area_of_current_contour > area_of_largest_contour)
        {
            area_of_largest_contour = area_of_current_contour;
            largest_contour_index = i;
            bounding_box = boundingRect(contours[largest_contour_index]);

        }

    }
    area = bounding_box.width * bounding_box.height;

    unsigned int finish = clock() - start;
    int time = finish;
    tick++;

    if(area > 200)
```

The code above is for background subtraction. OpenCV contains a class specifically for background subtraction so much of the code utilizes that. First, frames are read and stored as matrices. The background area is considered the first seen frames, and the mean and confidence interval for each pixel is established from that. The background is updated in each iteration of the code, which allows the algorithm to ignore movement that becomes stationary. Frames are compared to the background, and changes are seen as the foreground. In order to mark the movement, the algorithm looks for contours and marks the largest with a bounding box.

The final algorithm written was the modified frame subtraction method. It is identical to frame subtraction in its method for detecting motion, but it contains an implementation for ignoring repetitive movement. This method is simpler than background subtraction yet slightly more complex than the original frame subtraction.

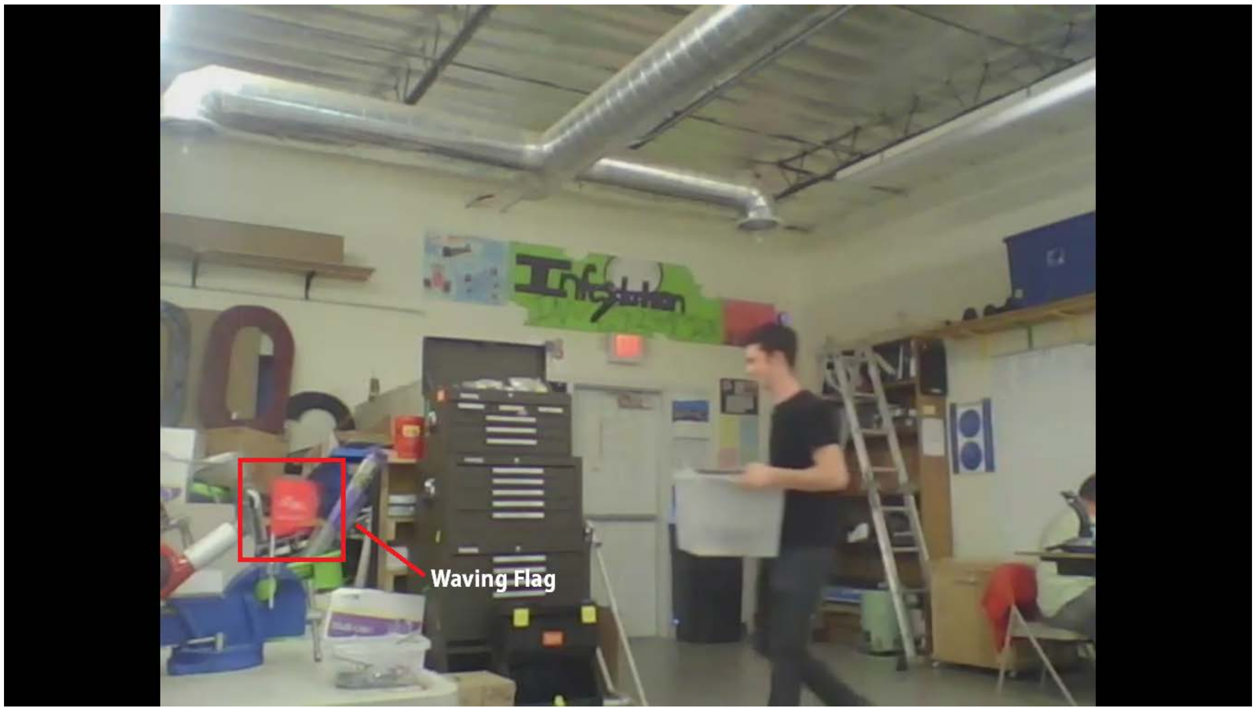


```
    if(ftt > 1)//after first time through program
    {
        if(area > 200)//if movement is even significant
        {

        if(area < 5000)//small object, increase bounding box size otherwise algorithm wont work
        {
            small_obj_flag = 1;//flip flag
        }
        else
            small_obj_flag = 0;

        if(centroid_within_bbox(small_obj_flag) == 1)
        {
            if(ob1warn < 1) //only one warning
            {
                rectangle(output, bounding_box, Scalar(0, 0, 255), 2, 8, 0);
                ob1warn = 1;
            }
            else
            {
                cout << "Ignoring object at " << centerx <<"," << centery << endl;
            }
        }
        else
        {
            reset_obj_values();//reset values if not
            ob1warn = 0;
            rectangle(output, bounding_box, Scalar(0, 0, 255), 2, 8, 0);
        }
        }
    }

```

Above is the portion of the modified frame subtraction that works to ignore repeated movement. The actual detection code is not included because it is identical to the frame subtraction method. In order to ignore repetitive movement, this algorithm uses values from the bounding boxes. First, it checks the area of proposed boxes to deem whether or not movement is large enough to care about. The algorithm saves information about each places bounding box. Specifically, it stores the x and y values of the top left and bottom right corners. It then checks to see if the center of future bounding boxes sits within those x and y values. If so, the movement is ignored because it can be assumed to be repetitive.

Each algorithm was put through two tests. The tests were completed using a video containing a waving flag and a moving human. The first test was to take an average of the time in milliseconds to run through an iteration of the part of the code that completes the detection. The second test was to find the percentage of accurate motion detections.

A video was made that acted as the standard test for each algorithm. The video includes simple repetitive movement, specifically a flag waving, and also includes movement from a person. The person enters the shot, picks up a box, leaves, returns, sets the box down, moves again, stops moving entirely, and then leaves the shot. This gives a number of different movements for testing. Ideally, an algorithm would ignore the flag waving, detect the person at every step taken, detect nothing when the person is absent or standing still, and have no false detections.



Above is a screenshot taken from the video, with the flag marked. This is not an example of an algorithm in action.

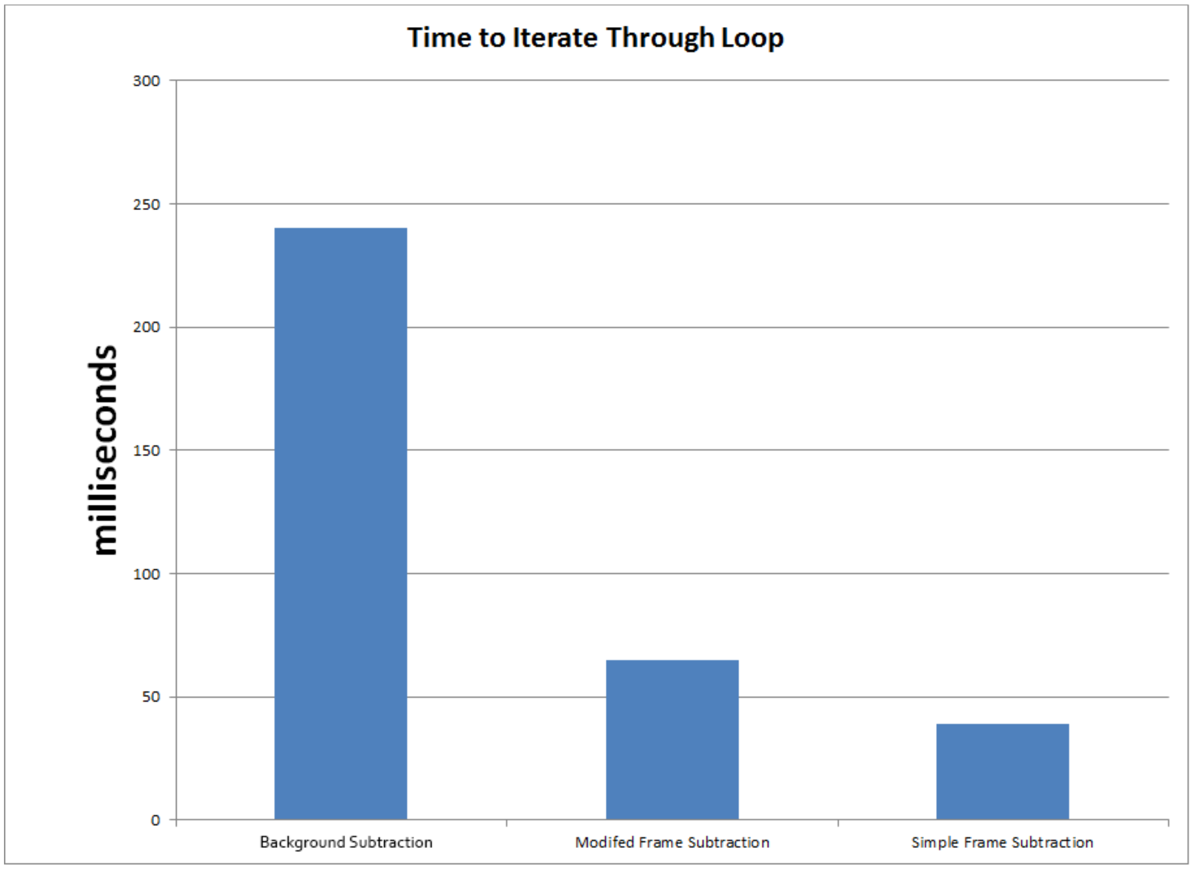
In order to test the iteration time, code was implemented that would return the time in milliseconds it took for the program to step through the loop while detecting motion in the video. The times were then exported to an Excel spreadsheet and averaged. This gave an idea of how the algorithms perform against each other currently and how they would perform against each other on a microcontroller. It also showed a good example of how the update times change between the three different algorithms.

In order to find the percentage of accurate detections, code was implemented that took user input after every complete iteration of the detection loop. If the displayed behavior was expected and desirable, the user input a one. If the displayed behavior was unexpected or undesirable, a zero was passed. Ones were considered a good detection, zeros bad. This gave a value for each iteration, and those values were put into the form of a percentage.

Results and Discussion

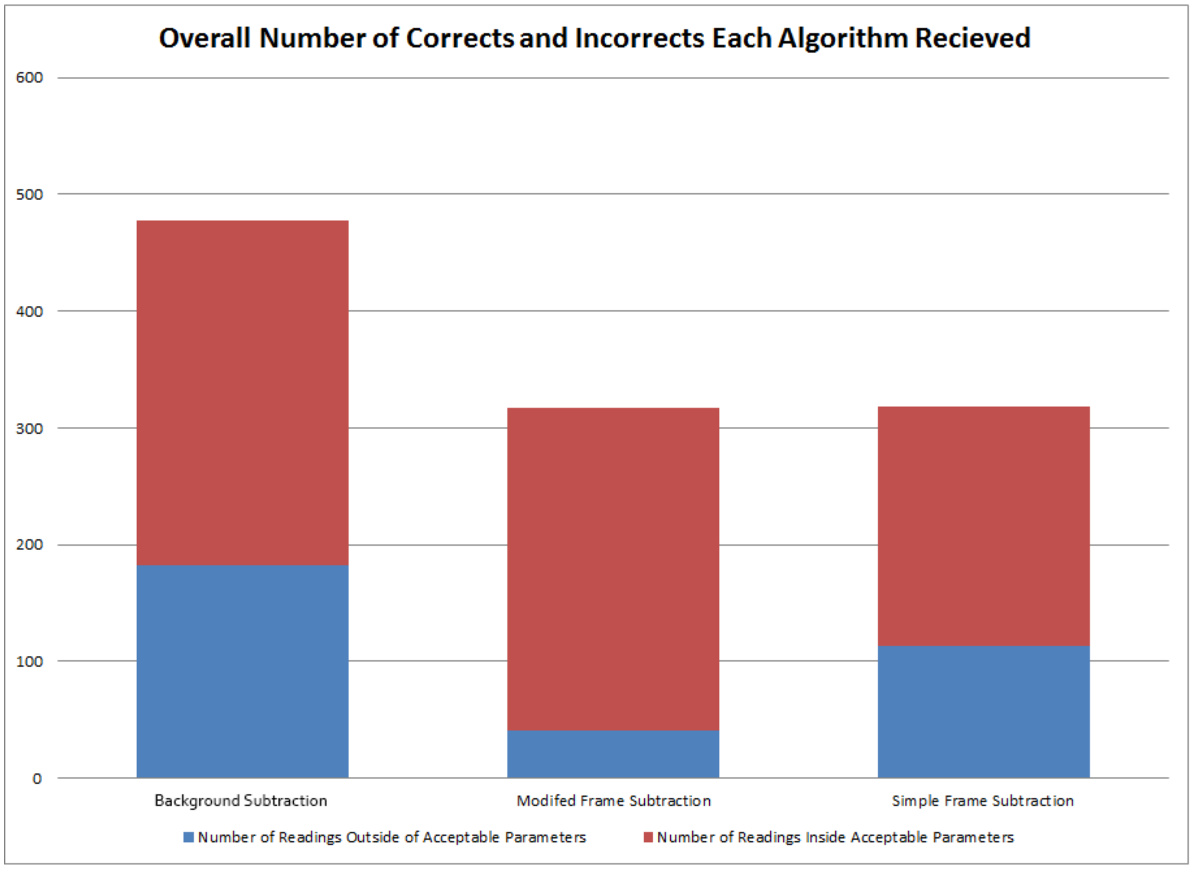
After testing, the modified frame subtraction method was deemed to be the most effective considering the standards. While the standard frame subtraction was the fastest, it contained too many undesirable detections. Background subtraction was the most adept at ignoring repetitive movement, but its traits were overall undesirable. The modified frame subtraction offered an acceptable medium.

When testing for time, the results were as follows:



The background subtraction had the longest average time taken to iterate through the loop. The modified frame subtraction method took more time than the simple frame subtraction, although the difference was small.

The test for accurate detection showed that the modified frame subtraction was overall more accurate than the other two algorithms. The total correct detections look as so:



Overall, modified frame subtraction had the lowest number of undesirable detections. Background subtraction had the greatest, and the simple frame subtraction method had the median score.

Conclusion

The results of these tests give many interesting points of data. In the begging, it seemed as if the most suitable algorithm was the background subtraction, as it can ignore repetitive movement and focus on one object. Upon testing and applying standards, it was shown that the modified frame subtraction was the most suitable. This was shown especially through the results from the accuracy test.

Although the background subtraction is adept at detecting motion and following one detected target that moves over the background, this is undesirable for the future application as the intention of the Police A.L.E.R.T device is to point one instance of movement, not to track the movement of one object. The modified frame subtraction method meets that need, and can ignore repetitive movement almost as well. It works better overall because it is faster and it will only mark one instance of movement, as opposed to marking and tracking the moving object.

The simple frame subtraction method, although the quickest, was all too inaccurate. The nature of the algorithm as it is written has no implementation for ignoring repetitive movement, which makes it undesirable from the start. This is the reason why the modified frame subtraction method was adapted from the original. It takes the simplicity of frame subtraction and adds a simple method of ignoring repetitive movement.

Personal Statement

The main accomplishment of this project was successfully developing and testing three different algorithms for our needs. This will enable us to track motion using the Police ALERT effectively.

Acknowledgements

We would like to thank our teacher and mentor Creighton Edington. He has helped us the development of our project thus far. In addition, thank you to Talysa Ogas who helped us prepare our presentation slides and reviewed our writings.

Works Cited

"How to Use Background Subtraction Methods¶." *How to Use Background Subtraction Methods — OpenCV 3.0.0-dev Documentation*. Open CV Dev Team, n.d. Web. 01 Apr. 2014.

"Installation in Windows¶." *Installation in Windows — OpenCV 2.4.8.0 Documentation*. Open CV Dev Team, n.d. Web. 01 Apr. 2014.

"Introduction to Motion Detection." - *RidgeRun Developer Connection*. N.p., n.d. Web. 01 Apr. 2014.

Appendices

Installing Open CV on Windows 7:

1. Launch a web browser of choice and go to our page on Sourceforge.
2. Choose a build you want to use and download it.
3. Make sure you have admin rights. Unpack the self-extracting archive.
4. You can check the installation at the chosen path.
5. To finalize the installation go to the Set the OpenCV enviroment variable and add it to the systems path section.

Background Subtraction Code:

```
#include <opencv2\opencv.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>
#include<vector>
#include <ctime>
#include <fstream>
#include <stdio.h>
using namespace cv;
using namespace std;


int area;
int main()
{
	Mat frame;
	Mat non_movement;
	Mat movement;
	Mat streaming_vid;

	Rect bounding_box;

	VideoCapture vid("C:/Users/Albert/Desktop/Zackmoving.avi");

	BackgroundSubtractorMOG2 bgs;//create class member bgs

	namedWindow("Movement", CV_WINDOW_AUTOSIZE);

	namedWindow("actual_vid", CV_WINDOW_AUTOSIZE);


	if(!vid.isOpened())

	{

	  cout << "unable to open webcam";

	}

	int tick = 0;

	int repeat = 0;

	ofstream file;

	file.open("C:\\testcomplicated.csv");

	while(1)

	{

		unsigned int start = clock();




		vid.read(frame);//Move video stream to a matrix

		vid.read(streaming_vid);

		bgs.operator() (frame, movement);//updates background and detects foreground objects

		bgs.getBackgroundImage(non_movement);//function returns the background image onto
matrix "non_movement"

		erode(movement, movement, 3);//get rid of some the noise

		dilate(movement, movement, 3);
```

```
                    vector< vector<Point>> contours;

                    findContours(movement, contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_TC89_KCOS);


                    float area_of_current_contour = 0.00;//initialize values for cycling through contours

                    float area_of_largest_contour = 0.00;

                    int largest_contour_index = 0;

                    float total_contour_area = 0.00;


                    unsigned int num_of_contours = contours.size();//

                    for(unsigned int i = 0; i < num_of_contours; i ++)

                    {

                              area_of_current_contour = contourArea(contours[i], false);//Possibly insert code
that gets rid of small contours right away. Need to time both of them.


                              if(area_of_current_contour > area_of_largest_contour)

                              {

                                        area_of_largest_contour = area_of_current_contour;

                                        largest_contour_index = i;

                                        bounding_box = boundingRect(contours[largest_contour_index]);


                              }


                    }

                    area = bounding_box.width * bounding_box.height;


                    unsigned int finish = clock() - start;

                    int time = finish;

                    tick++;
```

```
            if(area > 200)
            {
            rectangle(streaming_vid, bounding_box, Scalar(27, 255, 255), 2, 8, 0);
            }
            imshow("Movement", streaming_vid);
            imshow("actual_vid", movement);
            waitKey(33);

        cout << "yes or no?" << endl;
            int correct;
        cin >> correct;



            if(correct == 1 || correct == 0 || correct == 2)
            {

            file << finish << "," << correct << endl;
            }


    }
    file.close();
}
```

Frame Subtraction code:

```
#include <opencv2\opencv.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <iostream>
#include<vector>
```

```
#include <ctime>

#include <Windows.h>

#include <fstream>

#include <stdio.h>


using namespace cv;

using namespace std;


int area;




int main()

{

	Mat frame_1;

	Mat frame_2;

	Mat movement;

	Mat streaming_vid;

	namedWindow("output", CV_WINDOW_AUTOSIZE);

	namedWindow("stream", CV_WINDOW_AUTOSIZE);

	Rect bounding_box;

	char input = 0;


	VideoCapture vid("C:/Users/Albert/Desktop/Zackmoving.avi");


	if(!vid.isOpened())

	{

		printf("Unable to open camera");

	}
```

```
    ofstream file;
        file.open("C:\\test.csv");


        while(1)
        {
                unsigned int start = clock();


                vid.read(frame_1);//camera feed to frame
                cvtColor(frame_1, frame_1, CV_BGR2GRAY);//convert to gray scale


                vid.read(frame_2);//camera to frame
                cvtColor(frame_2, frame_2, CV_BGR2GRAY);//convert to gray scale so frames can be
subtracted


                absdiff(frame_1, frame_2, movement);//subtract the two frames, output to matrix
movement


                threshold(movement, movement, 10, 255, THRESH_BINARY);//apply a threshold. Any
pixel above absolute value 10 is changed to white


                vector< vector<Point>> contours;//create vector to store contours
                findContours(movement, contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_TC89_KCOS);


                float area_of_current_contour = 0.00;
                float area_of_largest_contour = 0.00;
                int largest_contour_index = 0;
                float total_contour_area = 0.00;


                unsigned int num_of_contours = contours.size();
                for(unsigned int i = 0; i < num_of_contours; i ++)//Cycle through all contours to find the
largest one

                {

                    area_of_current_contour = contourArea(contours[i], false);


                    if(area_of_current_contour > area_of_largest_contour)

                    {

                        area_of_largest_contour = area_of_current_contour;

                        largest_contour_index = i;

                        bounding_box = boundingRect(contours[largest_contour_index]);//Set
bounding box equal to largest contour


                    }


                }


                area  = bounding_box.width * bounding_box.height;


                unsigned int finish = clock() - start;


                cout << "elapsed time is " << finish << "ms" << endl;

                int time = finish;


                vid.read(streaming_vid);

             if(area > 200)

                {

                rectangle(streaming_vid, bounding_box, Scalar(0, 0, 255), 2, 8, 0);

                }

                imshow("stream", streaming_vid);
```

```
                        imshow("output", movement);

                        waitKey(33);

                        cout << "yes or no?" << endl;

                        int correct;

                cin >> correct;



                        if(correct == 1 || correct == 0 || correct == 2)

                        {


                        file << finish << "," << correct << endl;

                        }




            }

            file.close();


}
```

Frame Subtraction Modified Code:

```
#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>

#include <Windows.h>

#include <time.h>

#include <vector>

#include <fstream>


using namespace cv;

using namespace std;
```

```
Rect bounding_box;
Mat init_frame;
Mat nxt_frame;
Mat diff_frame;
Mat output;

int ftt;//first time through

int *ob1tlx;
int *ob1tly;
int *ob1brx;
int *ob1bry;
int centerx;
int centery;
int ob1warn = 0;
int area = 0;
int small_obj_flag = 0;

void create_windows(){
   namedWindow("output", CV_WINDOW_AUTOSIZE);
}
void display_windows(){
          imshow("output", output);
          waitKey(33);
}

void find_largest_contours(Mat& diff_frame)
{
          vector< vector<Point>> contours;

   findContours(diff_frame, contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);//Find all contours


          int area_of_current_contour = 0;

          int largest_contour_index = 0;

          int area_of_largest_contour = 0;


          for(unsigned int i = 0; i < contours.size(); i ++)

          {

                  area_of_current_contour = contourArea(contours[i], false);//Possibly insert code that gets
rid of small contours right away. Need to time both of them.

                  if(area_of_current_contour > area_of_largest_contour)

                  {

                          area_of_largest_contour = area_of_current_contour;

                          largest_contour_index = i;

                          bounding_box = boundingRect(contours[largest_contour_index]);


                  }


          }

          centerx = (bounding_box.x + (bounding_box.width/2));

          centery = (bounding_box.y + (bounding_box.height/2));

          area =  (bounding_box.width) * (bounding_box.height);

          cout << area << endl;

}


void reset_obj_values(){

          *ob1tlx = bounding_box.x;
    *ob1tly = bounding_box.y;

            *ob1brx = bounding_box.x + bounding_box.width;

            *ob1bry = bounding_box.y + bounding_box.height;

}

int centroid_within_bbox(int flag)

{

            if(flag == 1)//If area is small amplify bounding box before seeing if new bounding box fits inside
of it

            {

            if(((centerx < (*ob1brx + (.9 * *ob1brx))) && (centerx > (*ob1tlx -  (.9 * *ob1tlx)))) &&
((centery < (*ob1bry + (.9 * *ob1bry))) && (centery > (*ob1tly - (.9 * *ob1tly)))))

            {

                        return 1;

            }


            else

            {

                        return 0;

            }

            }

            if(flag == 0)

            {

                        if(((centerx < (*ob1brx)) && (centerx > (*ob1tlx))) && ((centery < (*ob1bry)) &&
(centery > (*ob1tly))))//Check to see if new bounding box fits within previous parameters

                        {

                                    return 1;

                        }

                        else

                                    return 0;

            }
```

```
}

int main(){

	VideoCapture camera("C:/Users/Albert/Desktop/Zackmoving.avi");//Initialize capture
	//VideoCapture camera(0);
	if(!camera.isOpened())//Make sure that capture works
	{
		printf("Failed to open camera");
	}
	ofstream file;
	file.open("C:\\ignore_frame_sub_test.csv");

	create_windows();

	while(1){
	unsigned int start = clock();
	camera.read(init_frame);//take two consecutive frames
	cvtColor(init_frame, init_frame, CV_BGR2GRAY);
	camera.read(nxt_frame);
	cvtColor(nxt_frame, nxt_frame, CV_BGR2GRAY);

	absdiff(nxt_frame, init_frame, diff_frame);//subtract the two frames
	threshold(diff_frame, diff_frame, 10, 255, THRESH_BINARY);//apply a threshold

	find_largest_contours(diff_frame);//find the largest contour, or the most significant movement
	camera.read(output);
```

```
if(ftt > 1)//after first time through program
{
        if(area > 200)//if movement is even significant
        {

        if(area < 5000)//small object, increase bounding box size otherwise algorithm wont work
        {
                small_obj_flag = 1;//flip flag
        }
        else
                small_obj_flag = 0;

        if(centroid_within_bbox(small_obj_flag) == 1)
        {
                if(ob1warn < 1) //only one warning
                {
                        rectangle(output, bounding_box, Scalar(0, 0, 255), 2, 8, 0);
                        ob1warn = 1;
                }
                else
                {
                        cout << "Ignoring object at " << centerx <<"," << centery << endl;
                }
        }
        else
        {
                reset_obj_values();//reset values if not
                ob1warn = 0;
```

```
                    rectangle(output, bounding_box, Scalar(0, 0, 255), 2, 8, 0);
            }
            }
}

else{//only done once at the beginning of the program
            ob1tlx = new int;
            ob1tly = new int;
            ob1brx = new int;
            ob1bry = new int;
            cout << "in else" << endl;
            reset_obj_values();



}
ftt++;
unsigned int finish = clock() - start;

display_windows();
cout << "yes or no?" << endl;
            int correct;
    cin >> correct;



            if(correct == 1 || correct == 0 || correct == 2)
            {

            file << finish << "," << correct << endl;
            }
            }
    }
```