Eye Connect

The Portfolio
Initial Observations

Upper-body paralysis is a severe issue in our society. According to cerebralpalsy.org.au, 1 in 500 babies are diagnosed with cerebral palsy every year without any known cure. Research conducted by the Christopher & Dana Reeve Foundation, showed that 1 in 50 people are living with paralysis. This information suggests that there are over 140 million people in the world showing signs of paralysis, with over half from low socio-economic backgrounds. Furthermore, according to the Amputee Coalition of America, 4.9 out of 1000 people suffer from upper limb loss.

People suffering from limited muscular control or limb loss can often lose the ability to interact with their environment or communicate. This is why there are currently various forms of assistive technology, designed to facilitate their needs.

Research into Current Models

Current assistive technology targeting upper body paralysis is extremely expensive. Eye-based control cannot be reliably done without expensive technology. Professional peripheral eye-trackers, like the Tobii PCEye Go cost $1,995 USD, without achieving exceptional results. Enthusiasts have mounted webcams on glasses to control a computer, but these solutions are inaccurate and impossible to mass-produce.

Personal brain-computer interfaces such as EEG headsets cost over $500 and are in their juvenile stage. Current technology requires immense training and calibration, often with unsatisfying results. Switch technology can support certain forms of paralysis but still costs over $100. Assistive technology is too expensive for the majority of the world to use. There is a pervasive need for low-cost, convenient and accurate assistive technology.

My Plan

For most people suffering limb loss or upper body paralysis, the extra-ocular muscles are not affected. Even in the most severe instances of ALS, eye movement remains in control of the patient. Therefore it makes sense to use the eye as a means of assistive technology.

Almost every new laptop will contain a built-in webcam. USB web cameras can be bought for a few dollars online. Assistive technology has always been expensive and hardware focused. I want to make it free and software focused. Assistive technology should be available to everyone irrespective of financial situations. That is why I intend on making Eye Connect.
My plan is to create Eye Connect, the first cross-platform suite of assistive software that runs on a built-in (or external) web cam. I plan to do this by familiarizing and contributing to the science of computer vision.

My first aim is to create my own blink detection algorithm using techniques of computer vision. After the blink detection algorithm is finished, I intend on adding blink-based mouse control. If I have sufficient time, I will try to create a motion-based solution as well, so that my project caters to people suffering from varying scales of paralysis.

13/6/15

**Progress on Face Detection**

**Creating Moustachiface**

In my first ventures into computer vision, I quickly realized that the face must be detected first. Eyes must lie within the boundaries of a face. Isolating the face will result in a higher level of accuracy for eye detection.

I researched and read many papers on face detection techniques, investigating and comparing algorithms like the Viola-Jones detection framework and Eigenface (PCA) method. After researching I chose to implement Viola-Jones detection, which I programmed in ActionScript 3.

Moustachiface is a cross-platform face detection system, written in ActionScript 3. By interfacing between the device camera, the Viola-Jones detection algorithm, and a Haar Classifier class, I was able to make a real-time face detection system. It uses artificial intelligence to decipher faces from a webcam, while positioning and resizing a moustache at the face’s detected region.

**View Moustachiface Source Code:**
https://goo.gl/6Vo7b0

**Papers and Articles Consulted:**

Progress on Blink Detection

Creating Eye Recognise
After successfully creating real-time face detection, I created my first blink detection algorithm, “Eye Recognise” in ActionScript 3. Detecting open eyes and closed eyes using the process of computer vision is a challenging and daunting task. I broke the problem down into steps to make the process easier.

1) The program loads in a sample image
2) The face is detected using Viola-Jones (Based on Moustachiface)
3) Pixel clustering:

   In computing each pixel has an associated Red, Green, and Blue value from 0-255 that determines its color. Using my sample data, I formed a conclusion that the eye pixels have red values smaller than 60, and green and blue values smaller than 50.

   The code loops through all the pixels of the face. It finds the color of a pixel, by interacting with the ActionScript 3 BitmapData class. The pixels that match the color range of eye pixels are then grouped together in clusters with nearby pixels that are also a positive match.

   These clusters are stored as a list of y values within a list of x values.

4) Each cluster must satisfy certain dimensions to be considered a potential eye blink. The width of a cluster must be smaller than half of the face width. The height of a cluster must be smaller than half the height of the face.

   Additionally the area of the cluster must exceed 20 pixels to be checked.

5) Then clusters are tested for truncation. If there are any horizontal gaps within a third of the width from the center, the object is not considered a blink.

6) The average rise of height in the clusters must be sufficiently close to 0

As can be seen, assuming an eye is horizontal, the height difference between
2 adjacent x values is relatively small, when the eye is closed. However when the eye is opened, the distance between points is constantly changing at a rapid rate.

**In other words:** \( |H_1 - H_2| \) is significantly larger than \( |H_3 - H_4| \)

My code loops through x values from left to right and monitors the change of height. Then the average change of height is computed. If the average height difference is greater than 15, the shape is not considered a blink.

7) The last test checks for vertical holes in the cluster. If more then a quarter of the image is comprised of gaps, the cluster can’t be considered an eye blink.

**Initial Results**

Here is a sample of a generated result:

![Original Image](image1) ![Analysed Image](image2)

The black rectangle is the detected face boundaries. Each separate region of blue on my face is one of the “cluster” objects. The yellow text is the outcome of tests. The text “True 😊” appears correctly on the left eye, suggesting that it is closed.

**Difficulties**

Although the algorithm worked well for several images in my sample data, there were an overwhelming amount of false positive closed eye detections. The algorithm is clearly too temperamental to be considered effective.

![Original Image](image3) ![Analysed Image](image4)

After image analysis, the algorithm falsely identified the eyebrows, lips and both open eyes as closed eyes.
The “Eye Recognise” blink detection algorithm was not nearly effective enough or reliable for my purposes. Furthermore each image analysis took several seconds. As I desired a real-time solution for my assistive technology, these were highly disappointing initial results.

Similar Tests

17/6/15
“Eye Recognise Face”

I began to think that an inverse of my original algorithm could be of more use. “Eye Recognise Face” forms clusters through ranges. A new pixel is introduced to a cluster if its colors are within the range of the current clusters maximum and minimum color values. This system effectively detected the background skin. The original plan was to try detecting the open eyes in the gap of the background skin, but this test was aborted because testing one image required over 20 seconds of analysis. It was not feasible to be used in real-time.

19/6/15
“Eye Recognise Dark Shapes”

In my next attempt I tried to group and create the clusters in a different way. Any sufficiently dark pixel would recursively group with surrounding pixels that had close-matching colors. This essentially failed, leaving fragmented eyes and facial features, while proving successful at detecting the main groupings of skin.

View Source Code of “Eye Recognise”, other test algorithms and Sample Data: https://goo.gl/Xe1M5w
New Phase of Development

I came to the realisation that ActionScript 3 was not the most effective programming language for computer vision applications. I needed a more fast and robust language for real-time blink detection. It seemed befitting to continue development on Python as it has access to many powerful modules that could make development easier. I specifically desired easy access to OpenCV, a powerful library for real-time computer vision. I still want the Eye Connect suite to be cross-platform, and will ensure that all dependencies can be used on Mac, Windows, or Linux computer systems.

Initial Tests With OpenCV

My first tests with the Open Computer Vision library were successful. I managed to set up and display camera footage very easily. I then implemented various Haar feature-based cascade classifiers. This allowed me to detect and localize the eyes on a face in real-time. The eye is detected regardless of weather it is closed or open. Although eye detection was relatively easy with OpenCV and Python, eye blink detection is still an extremely difficult and an unsolved ordeal.

Creating My Own Descriptors

In my research, I came across the Closed Eyes In The Wild (CEW) dataset. It had thousands of ordered open eye images and closed eye images, which can be used in the creation of a blink descriptor. I originally tried to use Haar-training to create an accurate classifier. I came across a website “coding-robin.de” that explained the process of creating Haar descriptors. Although the guide worked, my computer was simply not powerful enough to effectively perform the training.
I then successfully created a HOG Descriptor using trainHOG, but the resulting descriptor was still an unreliable form of blink detection. It became clear that training descriptors was not a viable means of blink detection.

12/7/15
Blink Detection through Template Matching

I came across a fascinating article on “machinelearningmastery.com” that demonstrated how to solve “Where’s Wally?” using OpenCV. The method used template matching. Template matching is the process of finding potential matches of smaller images inside of larger images. I realized that I could use the OpenCV template function to check for an open eye within the face.

The first step was to take a photo of my open eye to use as a template. I then created a sample program to test the accuracy of template matching.

A Brief Workflow of my sample Python script

1) Template is loaded and converted to gray scale
2) Sets up camera and constantly reads new frames
3) Frames are converted to gray scale and the face is detected
4) The left region of the face is used to search for an eye classifier
5) The matchTemplate function is called with mode TM_CCOEFF_NORMED, and a template of the open eye image. Then the best match is found using the function minMaxLoc.
6) If the best match was greater then the threshold of 0.6 and the location of the best match is sufficiently close to the location of actual eye on the face, the eye is considered open. Otherwise it is considered closed.

Results
The template matching approach has performed substantially in initial tests.

When the eye is analyzed as open, a box surrounds the location of the eye. Initial results
are extremely accurate in real-time eye blink detection.

Further Expansion of the Sample

I added support for both eyes and sides of the face. In this example frame, the eye on the right is correctly identified as open, and the eye on the left is correctly identified as blinking. The large boxes represent the overlapping boundary sides of the face. These boundaries ensure that the matching template detects the right eye.

Source Code to the Template Matching Approach
https://goo.gl/x37GaJ (Files “sampletestold.py” and “samptesttest.py”)

My initial template-matching technique is a viable solution for assistive technology. The only problem is that the template must be specific to each user. In other words, I would need to create a form of calibration that customizes the eye templates for the eyes of each individual user.

21/7/2015

Mouse Motion

Since I am now able to accurately detect eye blinks, it seemed important to start considering a mechanism of full-scale system control. The ideal way to control the system would be through the mouse. Thinking of a means of mouse control that relies only on blinks is not an easy task. After much thought I came up with an easy-to-use method of mouse control.

The Method

Direction Controller
The mouse follows a path to and from all 4 included directions. It goes in the direction “UP” and continues on clockwise all the way to “LEFT” before repeating the pattern.

Motion
When a user blinks right the mouse moves at a constant speed in the current direction. When a right blink occurs again, the direction controller is redisplayed.

Suspension
A left blink stops the mouse from moving. From this point, a right blink will reopen the direction controller, a left blink will execute a left click, and both eyes close will cause a right click.
Implementation of the Method

I wrote a prototype of this mouse motion technique, by using the keyboard’s right and left arrows. The cross-platform modules, PyMouse and Tkinter were used to conduct mouse operations, and detect the keyboard inputs. The initial prototype worked substantially well. It also implements a thread, as in the final program the mouse motion will need to be done at the same time as eye detection.

Source Code to the Mouse Control Demonstration
https://goo.gl/x37GaJ (File called “retina detect 2.py”)

3/8/2015
Blink Detector with a Calibrator and GUI

I managed to make a calibration system that automatically creates the necessary templates for blink detection. The calibration and blink detection system spanned almost 1000 lines of Python code. To increase the simplicity for the user I have added a high quality graphical user interface that outlines the various steps in the calibration process.

Using my prior knowledge of Photoshop I created some helpful pictures that assist in the calibration process. The video is scaled and aligned to the right of the graphics.

This is a screen shot from the calibration application. The small window on the left is one of the images that could become a template.
The workflow of the calibration is fairly simple to use

1) The Program is launched. If the eye template files already exist, the program skips calibration. The camera and face detection is loaded.

2) The user is asked to open their eyes. One second later the calibrator begins selecting several images of the user’s open eyes. For the next few seconds, all of those images are matched against web cam. Any images tested with more than 50% false negative are considered unreliable and removed. If there are no images remaining, the calibration process restarts.

3) The user is asked to close their eyes. For every frame from the camera, template matching is performed with each captured image from step 2 and a tally of false positives is created for every image.

4) The image with the lowest amount of false positives for each eye is saved as a template.

5) The template-matching blink detection algorithm is performed and executed using the saved templates.

Calibrator Reflection
The calibrator effectively and intelligently chooses templates. However one flaw within the program is that the user might partially blink during step 2. When this occurs, the templates are unreliable and will not work as expected. The calibrator is heavily dependent on the user and very cumbersome to use. I would prefer to try creating a system that can work without user interaction.

Source Code to the Calibration System
https://goo.gl/x37GaJ (File called “calibratorBACKUPnew.py”)
Calibration and Assistive Control System

I successfully combined the calibration system, the template-based blink detection algorithm, and the mouse control to create my first fully featured assistive technology software. To see the mouse move and click based on my blinks was an exceptionally rewarding milestone.

Source Code to this script
https://goo.gl/x37GaJ (File called “debugversion.py”)

A Final Blink-Detection Algorithm

As I began testing the calibration system, results were not satisfying enough for me. Blinks would often not be registered, and the requirement of calibration resulted in a complicated user experience. Although I had created a usable, cross-platform blink detection system, I had not reached my goal. I wanted to create a solution that was convenient and reliable. I wanted a solution that did not require calibration. By incorporating all the knowledge I had gained through the inventive process, I tried to create a truly accurate blink detection system. After immense amount of work I finally succeeded. The final blink detector is an extremely sophisticated, reliable and innovative method, that is unparalleled by any other current solution.

My Final Blink Detection Algorithm

Main program:
1) The default web cam is initialized. Frames from the video are constantly read in a loop.
2) The frame is converted into gray scale and faces are detected using Haar cascades. The face with the largest area is chosen for further processing.
3) Eyes are detected using Haar Cascades, and the eyes are only considered valid if they are sufficiently parallel and sufficiently dispersed. Then detected eyes are cropped slightly to reduce eyebrow intrusion.
4) The right eye is reflected horizontally so that it can match the left eye structure. The left eye and right eye both check for blinks using my function, findregions.
5) The mouse thread receives the information and moves the mouse accordingly.
Findregions Workflow:

1) Checks for the most prevalent color, which will be the skin surrounding the eye. Pixels colors are accessed using Numpy arrays and have a single color value from 0-255, as the image is in gray scale. For every pixel the related color and other close colors are incremented by 1. The color with the largest count at the end is considered the most prevalent color.

2) The Image is resized to a height of 20 pixels. This makes processing possible at real-time speeds.

3) Sometimes the contrast of an image can be affected by lighting conditions. In my algorithm, I found a proportional relationship between the skin color and the eye color to make the color-matching conditions adapt to different lighting conditions.

4) If a pixel passes the color-matching conditions and is not a prevalent color, it can join other nearby positive matches to become a cluster.

5) Filtration is applied to clusters. Clusters are removed if they are too close to the boundaries of the eye shape and do not cross the center. This gets rid of other nearby detected objects, such as eyebrows. If clusters are too small to be considered they are also removed.

6) If the amount of clusters is not one, then the eye is automatically considered open.

7) If the overall eye shape has more then 10 vertical holes or the number of holes is greater then a 16th of its area, the eye is considered open.

8) If there is more then one horizontal gap, the eye is considered open

9) The next step is to find the peak height and compare it to the width.
The eye can be at varying levels of rotation. This means that the peak height is not simply (maximum y – minimum y) at the center x as it would be for an eye symmetrical to the x-axis.

What I noticed is that the line between the endpoints of the eye always tends to act as a line of symmetry. Both sides of the line tend to open at relative levels.

If the eye width is treated as a line of symmetry, it follows that both endpoints of the peak height must be an equal distance from the left endpoint of the eye width.

By averaging all y values at the leftmost x value of the cluster, the left endpoint of the eye width can be determined.

Using the distance formula, I created a mapped list of all minimum y values and a list of maximum y values each containing their distance from the left endpoint and corresponding x values (the green lines). The maximum y values are paired with the minimum y value that is the same difference from the left endpoint (such as x1,y1 and x2,y2). The distance between both points is then calculated. The largest of these distances is determined as the peak height.

If the ratio between the peak height and width is greater then a fifth, the eye is considered open.

The midpoint of the peak height is then calculated, and then the linear equation of the eye width is formed. If there are any holes above the eye’s “line of symmetry”, the eye will be considered open.

If the eye is still not considered open, it qualifies as a blink.
Special cases:
Sometimes an eye will connect to its eyebrow, forming a dual eyebrow-eye cluster.

Generally when a dual grouping is created, there will be a substantial gap through the shape until the very right edge, where the eyebrow and eye join. To filter out the eyebrow from calculations, there is a maximum vertical gap. Once this gap is no longer reached the program does not perform any calculations. Then when there is no longer a maximum vertical gap, the program will stop performing tests to try avoiding inconsistencies. This allows accurate eye blink detection even in a joint eyebrow.

Results:
The final blink detection algorithm is more effective then I had originally thought. I have screen-recorded many people using the Blink Detection Algorithm and analyzed the data in videos to determine some results. The output was compared side by side to the original eye to identify correct and incorrect blink detections.

Blink Detection Statistics
Each person was tested for approximately 40 seconds. Positives are results that identify a blinking eye. Negative results identify an open eye.

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Number of Frames</th>
<th>True Positive</th>
<th>False Positives</th>
<th>True Negatives</th>
<th>False Negatives</th>
<th>Total False Detections</th>
<th>Total True Detections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>601</td>
<td>31</td>
<td>4</td>
<td>564</td>
<td>2</td>
<td>6</td>
<td>595</td>
</tr>
<tr>
<td>2</td>
<td>548</td>
<td>49</td>
<td>5</td>
<td>494</td>
<td>0</td>
<td>5</td>
<td>543</td>
</tr>
<tr>
<td>3</td>
<td>614</td>
<td>15</td>
<td>0</td>
<td>597</td>
<td>2</td>
<td>2</td>
<td>612</td>
</tr>
<tr>
<td>4</td>
<td>557</td>
<td>41</td>
<td>2</td>
<td>513</td>
<td>1</td>
<td>3</td>
<td>554</td>
</tr>
<tr>
<td>5</td>
<td>553</td>
<td>25</td>
<td>3</td>
<td>524</td>
<td>1</td>
<td>4</td>
<td>549</td>
</tr>
<tr>
<td>6</td>
<td>603</td>
<td>1</td>
<td>10</td>
<td>590</td>
<td>2</td>
<td>12</td>
<td>591</td>
</tr>
</tbody>
</table>
The accuracy can be calculated as total true detections/number of frames – Initial tests suggest my algorithm has a 99% blink detection accuracy. Although the amount of test data is relatively small, these results are substantial. The final blink detection algorithm performs at a pleasing level.

**Source Code to the Final Blink Detection Algorithm**

https://goo.gl/x37GaJ (File called “Final Blink Detection Algorithm.py”)

20/8/15

**Developing a Motion-Based Control System**

My original plan for the Eye Connect suite was to include solutions for various different forms of paralysis. Some disabled people may have upper limb paralysis, but still have control over certain areas on their head. It may sometimes be easier and more convenient for the user to control their computer through motion. I have successfully created a cross-platform system that can control the mouse using head-based motion.

**The Workflow of My Motion-Based Control System**

1) The eyes and face are detected using the same methods as previously outlined.

2) On the first detection, an anchor point is automatically created in between both eyes. This is represented on the viewer as a dot. If a person can only control certain parts of their head, the anchor point can be moved by clicking, to focus on an area that the user can move more easily on.

3) The distance between the two points is calculated using the distance formula. If the distance is sufficiently close to the anchor point, the mouse will be in a suspended until a larger distance occurs. The required proximity is relative to the eye width.
4) The y difference is calculated as eye’s y – the anchor point’s y. The x difference is calculated as eye’s x – the anchor point’s x. The absolute values of the y and x differences are compared against each other and the largest distance becomes the direction. This restricts the direction of the mouse to up, right, down and left, which makes the program more easy to use. In the above example the current eye has a larger y difference than x difference. Therefore it will move the mouse in an upward direction.

5) When no motion is detected (in other words, the distance from the anchor point is sufficiently close to 0), mouse movement is suspended. At this point a timer occurs. After approximately 1 second, if the user is still in suspended state a left click is performed. Another second later, a double click is conducted. In the final second, a right click occurs. This flow allows the user to easily access and control their computer, purely out of head movement.

Results

My head-based motion technique is extremely easy for a user to use, and it is more efficient to use than a blink detector. For less severe levels of paralysis, this method allows a greater control speed and ease of use.

Source Code to the Final Motion-Based Control System
https://goo.gl/x37GaJ (File called “Motion-Tracking Assistive Technology.py”)
21/8/15

Creating an Interactive Demonstration

To help users familiarize themselves with my technology, I have created a comprehensive demonstration. The demonstration shows a flying balloon and the goal of the user is to try navigating to the balloon and popping it.

This results in a fun and interactive demonstration that allows the user to learn how to use my assistive technology for mouse control. The program is written in ActionScript 3.

The Demonstration System
A Mac-compiled version of the demonstration is available online https://goo.gl/x37GaJ (File called “Demo Features.app”)

23/8/15

Additional New User Features, Options and Tools

Now that I had created two powerful forms of assistive technology I wanted to increase the user experience. Therefore it seemed necessary to add some options and tools.

I created a sample TkInter window, containing various buttons that I would later implement in the final eye connect suite.
The buttons created allow several new features.

1) The user can now toggle between modes of assistive technology
2) The software can be set to run on login
3) A quit button allows the user to easily quit my program
4) A final button allows access to the interactive demonstration

The current mode of assistive technology is saved and remembered for the next execution,

Source Code
https://goo.gl/x37GaJ (File called “TkInter Button Sample.py”)

24/8/15

Final Combined Project

After months of hard work, I have finally finished the first version of the Eye Connect. I combined the motion-based detection system with the blink detection system, and added several buttons to create a fully featured suite of assistive technology.

The final script can be launched and used like a general application

Evaluation
The final Eye Connect suite is a new frontier in assistive technology. An accurate blink-detection algorithm has never been created using a mere web-cam as hardware. Eye Connect is the first form of assistive control that can run purely on blinks. Eye Connect is also the first cross-platform form of assistive technology capable of using any standard webcam. Overall I have achieved my aim. Now anyone, anywhere in the world, can use Eye Connect for free, powerful and convenient assistive control. There is finally a cost-effective and viable method of assistive control for the disabled.

Source Code
https://goo.gl/x37GaJ (File called “EyeConnect Final.py”)

Future Plans

I hope to soon create a website and release Eye Connect online as free, open source and cross-platform software. I am already in the process of communicating with hospitals and other facilities to try help share my assistive technology.

In the future I plan on adding support for low-cost environmental control, so that the user can control TV’s and other infrared appliances. This would be created using an Arduino, serial communication, IR transmitters and receivers.

I also intend on conducting more trials and research to even further improve the blink detection algorithm.

Conclusion

Creating Eye Connect has been an exceptionally rewarding experience. I passionately feel that it can be used to positively assist disabled people throughout the world, so that they can communicate and use technology in a reliable, cost-effective way.