A fast and accurate method for discriminating five choices with EOG

Toral Zaveri, Jason Winters, Mamta Wankhede, Il Park

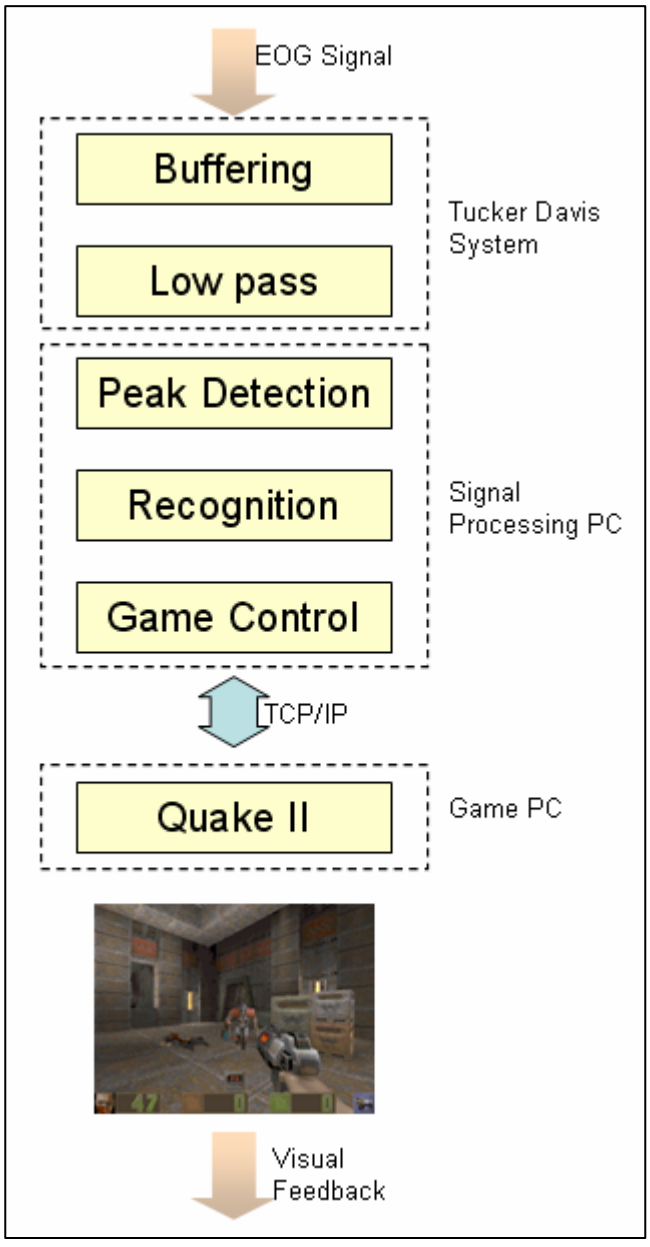
Introduction

Patients suffering from paraplegia have limited or no functionality below their neck. They are locked in their own bodies with almost no communication with the outside world. In order to impart functionality and/or means of communication to such patients, attempts are being made to optimally utilize all the available resources available with the patient such as brain activity, or even eye movement.

In this short experiment that we have performed, we have designed a model which utilizes the signals generated from the eye movement of a person for game control.

Here, we obtained the EOG (Electr-Oculography) signals from a subject for five different eye movements (up, down, left, right, blink) and then designed a model to identify these distinct movements. After this, these signals were used for control of a computer game (Quake II). So that as a result we could achieve that the subject was controlling the computer game simply by moving his eyes.

The algorithm works in the time domain. First, peak detection, and then recognition of each peak pattern into actual eye movements, and finally it is translated into game controlling commands. Users can control the game in real-time with visual feedback from the game PC. We also had a simple procedure to calibrate the parameters after electrode positioning.



Electro-Oculography

Human ocular movement has been widely studied in neurophysiology and psychology. These studies indicate that there are four types of eye movements, called *vestibular, optokinetic, saccadic,* and *pursuit.* The first two have to do with the largely involuntary head motion. The saccadic movement is used to "jump" from one object of interest to another. This is the fastest type of eye movement. The pursuit movement is used to maintain fixation on a moving object.

Deliberate eye movement can convey useful information in basically two ways: (1) through the six extra-ocular muscles (by absolute eye position, speed, and direction of eye movement), (2) through the eyelid and other periorbital muscles (by unilateral or bilateral blinking or blink duration).

The eye maintains a voltage of +0.4 ~ 1.0 mV with respect to retina (due to the higher metabolic rate at the retina compared to the cornea). This corneoretinal potential is roughly aligned with the optic axis. Hence it rotates with the direction of gaze. It can be measured by surface electrodes placed on the skin around the eyes. These recorded signals are smaller (15 to 200 μV), so they are amplified before processing.

Electro-oculography has both important advantages and disadvantages over other eye tracking methods. On the positive side, the equipment is cheap, readily available, and can be used with glasses or contact lenses, unlike some reflection methods. The necessary fixtures do not obstruct the visual field, and are completely insensitive to head movement, although significant deviation form the last calibrated position would require the user to repeat a calibration sequence for accurate tracking. On the other hand, the measured signals are subject to drift from several sources: changing skin resistance, electrode slippage or polarization, even a variable corneoretinal potential due to light accommodation and level awareness. Noise pickup from other electrical devices can be minimized by careful shielding, but action potentials of the other facial muscles can mask the desired signal. The most obvious shortcoming, uncorrectable by its very design, is the need for attachments directly on the user's face - six electrodes under the proposed scheme. Set-up is cumbersome, and although actual discomfort is low, mental and physical awareness can be very high, creating a large long-term "annoyance factor"; this method may be unacceptable to some subjects.

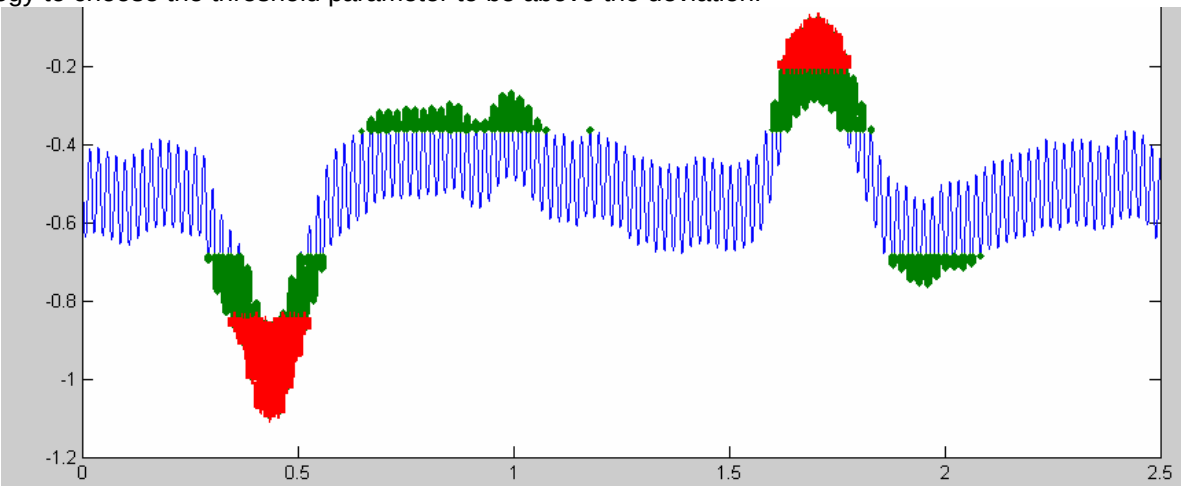
Straightforward signal processing steps can be devised to condition the data so it can be reliably interpreted. Some of the noise patterns such as the 60 Hz line frequency can be easily removed, using a notch filter. Other noise artifacts are mostly transients caused, for example, by the turning of an electrical switch on/off in the vicinity of the electrodes, contraction of the facial or neck muscles, slippage of the electrode due to sweat and eye blinking. However, the signals produced by eye blinks are, in fact, quite regular. This makes it easy to recognize and eliminate them. On the other hand, because this type of signal is quite distinct from the usual data from pursuit or saccadic movements, they can be recognized and categorized as such. In other words, the EOG technique can potentially recognize eye "gestures" such as winking, blinking or a combination thereof.

It is possible to obtain independent measurements from the two eyes. However, the two eyes move in conjunction in the vertical direction. Hence it is sufficient to measure the vertical motion of only one eye together with the horizontal motion of both eyes.

Data Analysis

Noise Reduction

The eye movement signals are band limited due to the fact that there is a speed limit on eye movements. Thus, a low pass filter with 20Hz cutoff could remove most of the high frequency noises. The largest noise we observed was the 60Hz noise from the power line. We compared the standard deviation of the signal in order to discriminate meaningful signals from noise (see figure below). It turned out that the whole signal is within 2 times of the deviation, and the base level noise was mostly within the deviation. This led to our calibration strategy to choose the threshold parameter to be above the deviation.

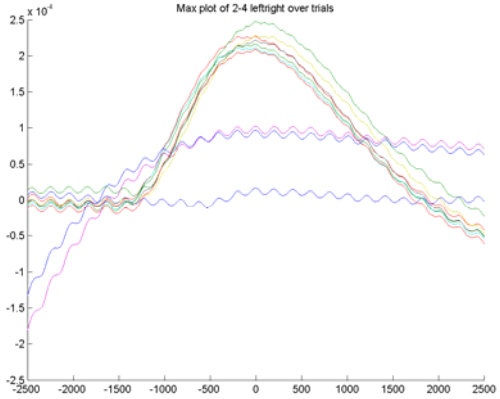


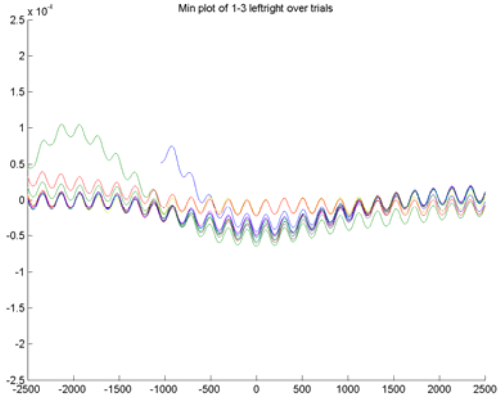
Characteristics of Eye Movements

In this experiment, we tried to characterize each type of eye movements by visually inspecting the spikes (see figures on page 4 and 5). The result is summarized in the following decision table. '+'/'-' indicates positive/negative peak, '0' means below certain level, and N/A means does not matter. Blink is characterized as either a consecutive '+' and '-', or '+', '0', '-'.

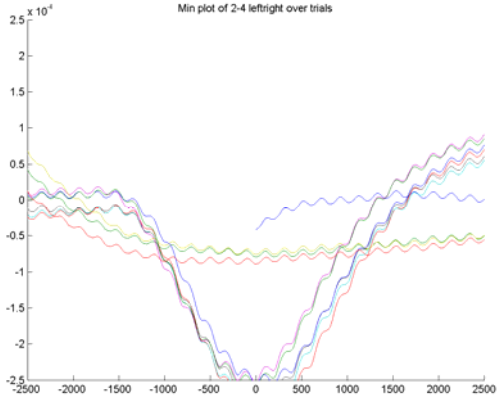
	Horizontal Bipolar	Vertical Bipolar
Left	+	0
Right	-	0
Up	N/A	+
Down	N/A	-
Blink	+(0)?-	N/A
Nothing	0	0
Artifact	+/-	+/-

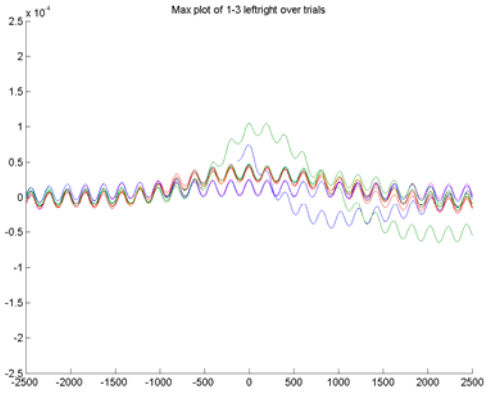
Figure
Left Movement





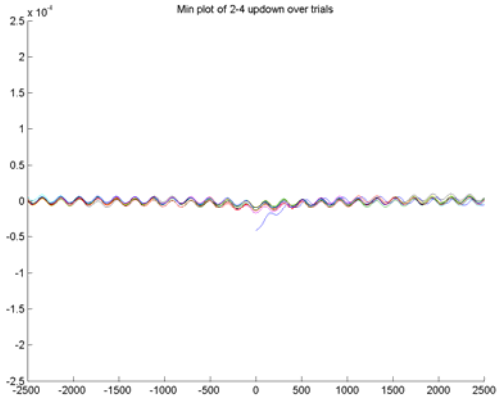
Right Movement

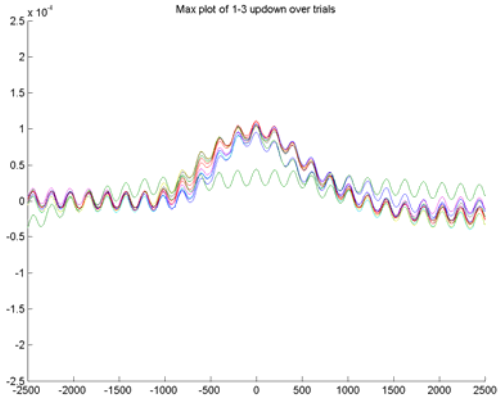




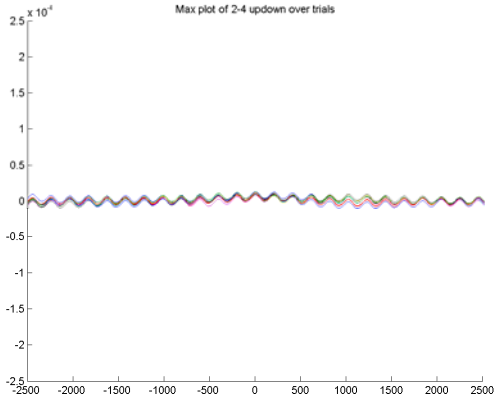
Each pair of graphs is a result of aligning and overlaying 10 trials for each movement. The left and right graph corresponds to horizontal and vertical bipolar measurement, respectively. You can see the trials are overlapping strongly, which means that the shape and strength of signals are stationary.
When the eye is moving fast towards left or right, we get a strong peak in the horizontal bipolar measurement, and in the case of up or down movement, the peak was strong in the vertical bipolar measurement, as expected by physiological insights. Note that for the blink, a positive peak followed by a slight negative peak on the vertical bipolar measurement is observed.

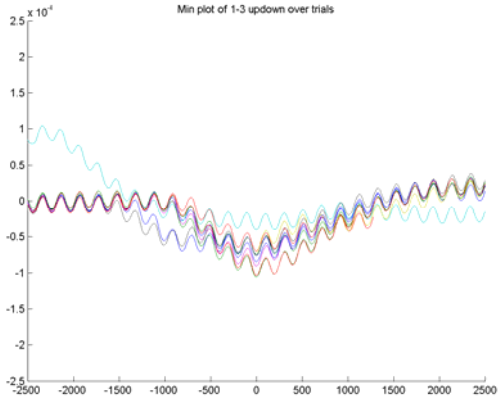
Up Movement



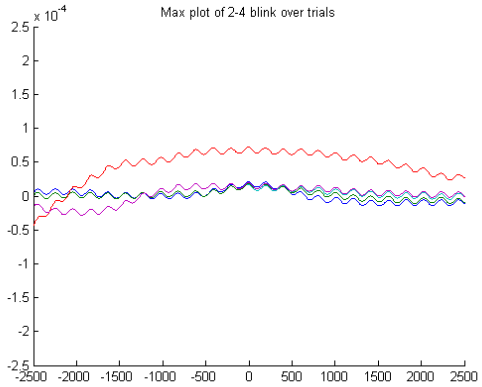


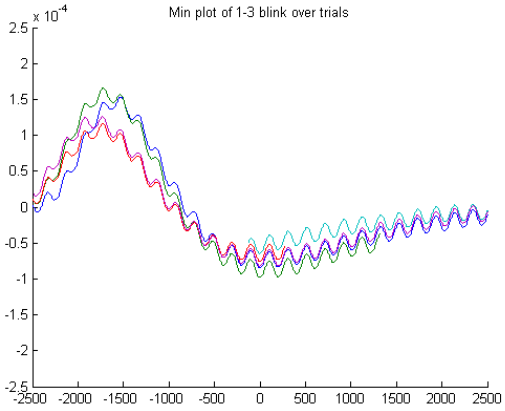
Down Movement





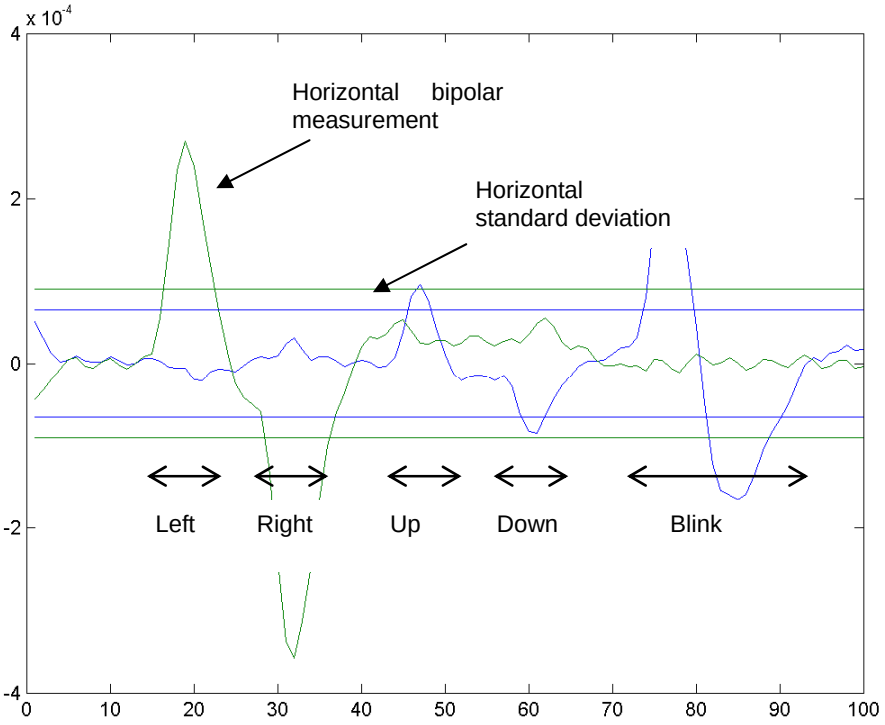
Blink





Calibration

Since the EOG signal varies depending on several uncontrollable factors, such as placement and conductance of the electrodes, and also the amplitude pattern which differs across subjects, it is essential to have a process to calibrate the parameters of the detection and recognition system.
We had a semi-automatic procedure for calibration.



This is a typical calibration graph we generated. There are all five movements in the time frame. The subject was asked to move his or her eyes to the left, center, up, center, and then blink in one second after the cue was given. Both vertical and horizontal signals were plotted in one figure along with the calculated standard deviation. The deviations are used as a guideline for choosing threshold; it should be at least larger than the deviation. After a couple of stable calibration graph is obtained, we decided the parameters for correct and robust discrimination of each eye movement.

Method

Experimental Setup

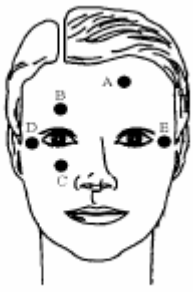
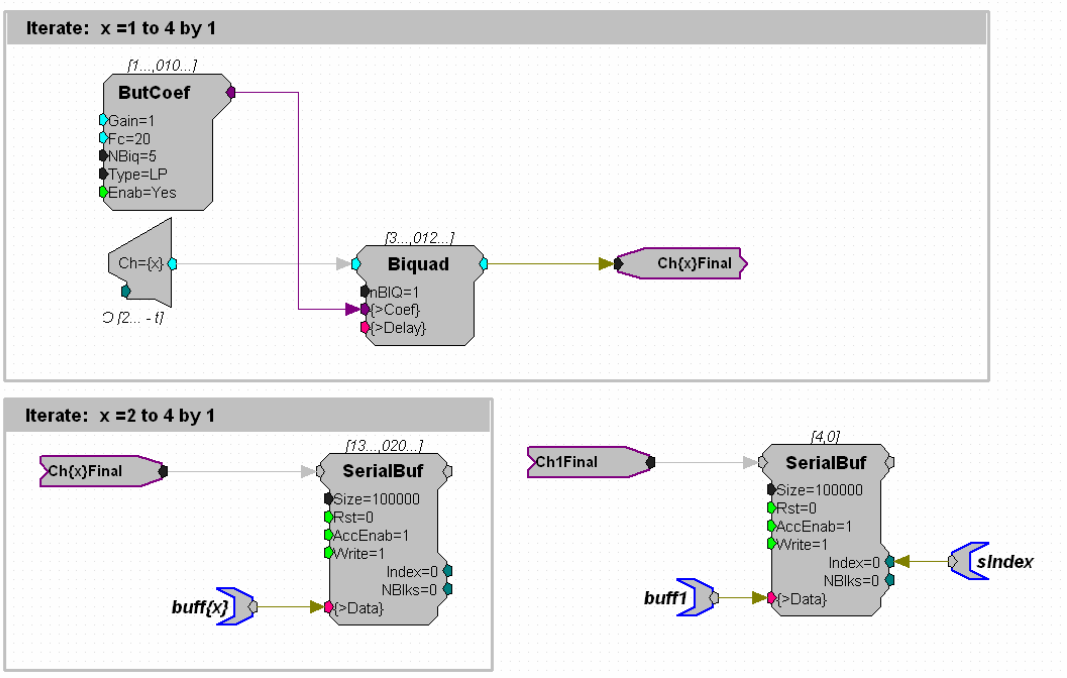


Fig 1.- Electrodes placement.

Electrode Placement

The electrodes used for the system and their connection were as shown below
Channel 1: Electrode B (HEOR)
Channel 2: Electrode E (HEOL)
Channel 3: Electrode C (VEOU)
Channel 4: Electrode D (VEOL)
Reference: Electrode A (M1/M2)
Note: We have shifted the position of reference from forehead to the auricle on the side of Electrode E and also incorporated another reference electrode on the second ear in order to stabilize the ground.

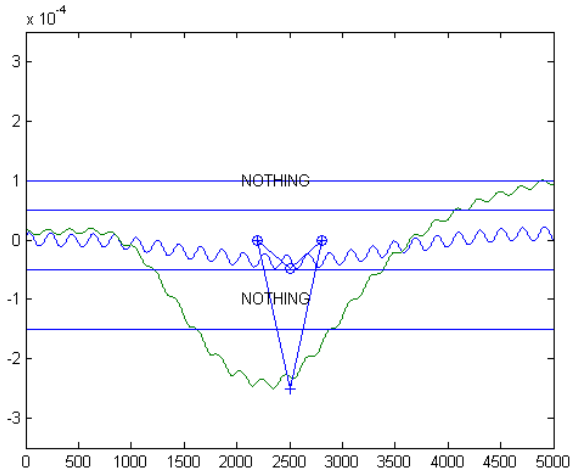
Tucker Davis System



We used RA16 Medusa system (Tucker Davis) to record the EOG. The sampling rate was 25 kHz. The EOG data from the four channels is first given to a low pass filter. The filter is a Butterworth filter of order 5 and cutoff frequency of 20Hz. We selected the cutoff frequency as 20 so as to eliminate the 60 Hz line frequency. In EOG signals the useful frequency range lies within 5 Hz. The filtered data is then given to a file buffer of which will be accessed from the implemented MATLAB code on demand. The ‘sIndex’ represents the number of samples stored in the buffer.

Spike Detection

We observed that the spike generated from saccadic eye movement has wave length less than 5000 samples, and thus we used a window size of 5000 samples to detect these spikes. When the peak of the signal is found within the center region of 1000 samples, it is considered as a spike, and than passed to the pattern recognition algorithm. Each update retrieved 1000 samples from the Tucker Davis System. For the data to be in center, three frames of update is needed, thus the spike detection algorithm introduces 3/25 second of delay.



Pattern Recognition

According to the detection table described in the data analysis section, the pattern recognition algorithm categorizes the spike stream into 7 types. In order to detect blink, three consecutive frames of spike detection data is needed. Hence, the algorithm is introducing additional 2/25 second of delay, which leads to 1/5 second of total delay.

Game Control

After the signal was decoded and the appropriate action determined, the program used a TCP/IP connection to send the command to the computer running Quake 2. Prior to running the test, the control keys for the game were reprogrammed to “w”, ”z”, ”a”, ”s” , and space which corresponded to the actions of forward, backward, left, right and fire, respectively. Each eye movement was decoded for the following game actions: an up eye movement moved the game forward, a down eye movement moved the game backward, a left eye movement moved the game left, a right eye movement moved the game right and an eye blink fired the gun. For each decoded eye movement, five characters were sent to produce larger game movement. Since the user needed keep the computer screen within his/her visual field while playing the game, controlling the game motion with eye movements posing an interesting problem. More specifically, if a left eye movement is used to move the game left, the user’s eyes would no longer be looking at the screen. Additionally when the user produced an eye movement back to the screen, a right eye movement was detected. To overcome this problem, all decoded EOG signals were ignored for one second after an initial command was decoded and sent to the game. This allows the user, wanting to move left, to move his/her eyes left then back to the screen and it only be interpreted as one forward command. One second after a command is sent, the program is ready to send another command. This scheme worked very well, producing reliable control of left, right, forward, backward game movements as well as a fire command.

Result

	Forward	Backward	Right	Left	Shoot	Total
Subject 1	10/10	9/10	10/10	10/10	10/10	**98%**
Subject 2	10/10	10/10	10/10	10/10	10/10	**100%**
Subject 3	10/10	8/10	10/10	10/10	6/10	**88%**
Total	100%	90%	100%	100%	86%	**95%**

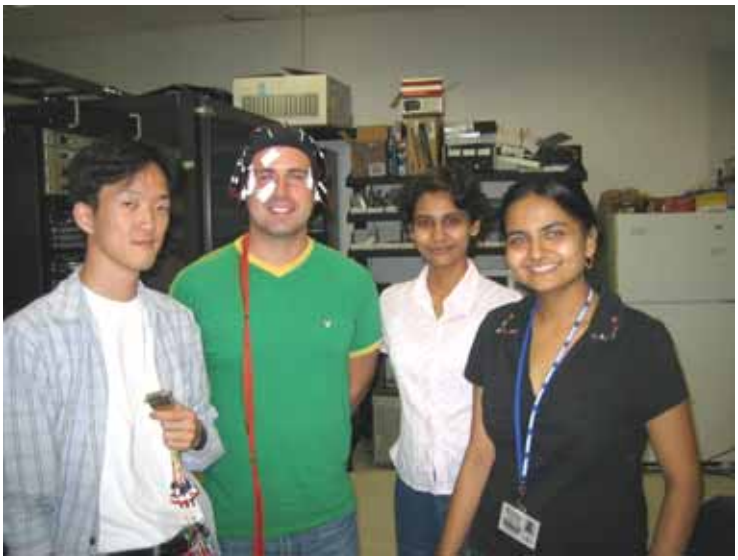
Subjects who volunteered for the experiment were 2 males and 1 female in the age group of 22-27 years. The three subjects belonged to different ethnic groups. Subject 1 underwent maximum training on the system and hence has demonstrated consistently high performance. Subject 2 and 3 were trained for just a few minutes and subjected to the task. However, these two subjects differed especially in performing shooting. This is partially due to the fact that the shooting action depends on the frequency of blinking that is time interval between the positive and negative spike produced from blinking, and only if this interval lies within a specific range, it is recognized as a blink, if not, as a command for up or down action.

The calibration process proved to be very useful. Without the calibration, the variation among subjects was not tolerable (data not shown). Subjects reported that playing the game with EOG interface is exciting and they could not detect any sign of delay, though one subject reported dizziness and pain in the eye probably due to contact lens. The CPU utilization of the processing computer was less than 5% even with real-time plotting via MATLAB. This proves the efficiency of the algorithm.

Discussion

In this report, we described a fast and accurate algorithm for an EOG based control system. However, there are several limitations. First of all, the input rate of the system was limited to one input per second due to the controlling algorithm, it could be improved by devising better method to ignore or utilize the returning to center movement of the eye. But there is not much room for improvement for the nature of eye movement. Second, the calibration process involves experts. Even though it automatically gives clues of threshold, the final decision should be made by human. It is possible to further improve the calibration algorithm to fully do the job automatically. Third, even though artifact detection worked fine, that is the subject could talk or move his or her head around without making any false recognition, however, it is not possible to control the game while talking. Finally, the algorithm assumes stationarity of the signal. Since the electrode position or environment could vary over time, it is not only necessary to calibrate after each positioning of electrodes, but also important to change the parameters over time. This could be achieved by using adaptive signal processing technologies.

Introduction to our team members



Il Park, Jason Winters, Mamta Wankhede, Toral Zaveri (from left to right)

Acknowledgement

We would like to thank Dr. DeMarse for providing such a wonderful opportunity and equipment, the members of the lab for helpful comments and endurance of the noise we generated. (Some noise was indeed our best friend in some sense.) And thank YOU very much for reading!

Appendix – MATLAB code

```
function fastNaccurate(lambdaUpDownMax, lambdaUpDownMin, lambdaLeftRightMax, lambdaLeftRightMin)
% EOG control system main routine.
% Copyright 2005 all rights reserved,
% Il Park, Toral Zaveri, Jason Winters, Mamta Wankhede

% Error control block
try
   RP = initializeActiveX('record-eog.rco');
   quake_server_handle = initializeQuakeConnection;
   main(lambdaUpDownMax, lambdaUpDownMin, lambdaLeftRightMax, lambdaLeftRightMin);
catch % if any error, stop circuit and close TCP/IP connection
   finalizeActiveX(RP);
   finalizeQuakeConnection(quake_server_handle);
end


function main(lambdaUpDownMax, lambdaUpDownMin, lambdaLeftRightMax, lambdaLeftRightMin)
% The main routine of the program

tic; % starts the tic-toc mechanism
samplefreq = 25000; % 25 kHz
windowsize = 5000;
% global constants
global NOTHING UP DOWN LEFT RIGHT BLINK ARTIFACT
NOTHING = 0; UP = 1; DOWN = 2; LEFT = 3; RIGHT = 4; BLINK = 5; ARTIFACT = 6;

global windowbuffer
windowbuffer = zeros(2,windowsize);
lrudbnBuffer = zeros(1,3); % fill with NOTHINGS

sum13 = 0; % sum of all data for averaging
sum24 = 0;
totalAccumulatedSamples = 0;
chunksize = 1000;

while true
   dataA = retrieveData(RP, 100000, chunksize, samplefreq);
   totalAccumulatedSamples = totalAccumulatedSamples + chunksize;
   x1 = dataA(1,:)-dataA(3,:);
   sum13 = sum13 + sum(x1);
   avg13 = sum13 / totalAccumulatedSamples;

   x2 = dataA(2,:)-dataA(4,:);
   sum24 = sum24 + sum(x2);
   avg24 = sum24 / totalAccumulatedSamples;

   x1 = x1 - avg13; % remove DC component from the input signal
   x2 = x2 - avg24;

   [r13, r24] = detectSpike(x1, x2);

   % Blink recognition
   lrudbnBuffer(1:2) = lrudbnBuffer(2:3); % shift the buffer
```

```
    lrudbnBuffer(3) = recognize(r13, r24, ...
            lambdaUpDownMax, lambdaUpDownMin, ...
            lambdaLeftRightMax, lambdaLeftRightMin);
    if lrudbnBuffer(1) == UP
        if lrudbnBuffer(2) == DOWN
            lrudbn = BLINK;
            lrudbnBuffer(1) = NOTHING;
            lrudbnBuffer(2) = NOTHING;
        elseif lrudbnBuffer(2) == NOTHING && lrudbnBuffer(3) == DOWN
            lrudbn = BLINK;
            lrudbnBuffer = zeros(1,3);
        else
            lrudbn = lrudbnBuffer(1);
        end
    else
        lrudbn = lrudbnBuffer(1);
    end

    disp(lrudbnToStr(lrudbn));

    % convert the recognized symbols to quake control
    quakeController(quake_server_handle, lrudbn);

    % Plot the recognized results real-time
    plot(windowbuffer');
    hold on
    plot([2200,2500,2800], [0,r13,0], '-o');
    plot([2200,2500,2800], [0,r24,0], '-+');
    plot(lambdaUpDownMax .* ones(1,windowsize));
    plot(lambdaUpDownMin .* ones(1,windowsize));
    plot(lambdaLeftRightMax .* ones(1,windowsize));
    plot(lambdaLeftRightMin .* ones(1,windowsize));
    lrudbnStr = lrudbnToStr(lrudbn);
    text(2000,1e-4,lrudbnStr);
    text(2000,-1e-4,lrudbnStr);
    axis([0,windowsize,-3.5e-4,3.5e-4]);
    drawnow;
    hold off
end

function quakeController(quake_server_handle, lrudbn)
% Converts the recognized signal into quake understandable characters
% and send them to quake via TCP/IP
global NOTHING UP DOWN LEFT RIGHT BLINK ARTIFACT
charArray = [];
switch lrudbn
    case NOTHING
    case LEFT
        charArray = ['a'];
    case RIGHT
        charArray = ['s'];
    case BLINK
        charArray = [' '];
    case UP
        charArray = ['w'];
    case DOWN
```

```
        charArray = ['z'];
    case ARTIFACT
end

if length(charArray) ~= 0
    t = toc;
    if t < 1  % Ignore any input within 1 sec of last action
        return;
    end

    for k = 1:5  % send 5 chars per action to have a greater effect
        tcpip_write(quake_server_handle, charArray);
    end
    tic;
end


function lrudbn = recognize(r13, r24, ...
                            lambdaUpDownMax, lambdaUpDownMin, ...
                            lambdaLeftRightMax, lambdaLeftRightMin)
% From the horizontal/vertical bipolar peaks, recognize left/right/up/down

global NOTHING UP DOWN LEFT RIGHT BLINK ARTIFACT
lrudbn = NOTHING; % basically, it's nothing

if abs(r13) < max(abs([lambdaUpDownMin,lambdaUpDownMax]))
    if r24 < lambdaLeftRightMin
        lrudbn = RIGHT;
        return
    elseif r24 > lambdaLeftRightMax
        lrudbn = LEFT;
        return
    end
end

if r13 < lambdaUpDownMin
    lrudbn = DOWN;
    return
elseif r13 > lambdaUpDownMax
    lrudbn = UP;
    return
end

% If the signal is not recognized and it is still huge, then we consider it to be an artifact.
if max(abs([r13,r24])) > min(abs([lambdaUpDownMin, lambdaUpDownMax, lambdaLeftRightMax,
lambdaLeftRightMin]))
    lrudbn = ARTIFACT;
end


function [r13, r24] = detectSpike(x1, x2)
% Detects a spkie within the window and returns the amplitude of the spikes
% First appends the new data (1000 samples) to the window buffer. And
% within the 5000 window buffer, if the peak (max or min) is in the middle
% range which is from 2000 to 3000 in index, then returns a peak amplitude
% of the spike.
```

```
global windowbuffer

windowbuffer(:,1:4000) = windowbuffer(:,1001:5000);
windowbuffer(:,4001:5000) = [x1;x2];
[max13,maxIndex13] = max(windowbuffer(1,:));
[max24,maxIndex24] = max(windowbuffer(2,:));
[min13,minIndex13] = min(windowbuffer(1,:));
[min24,minIndex24] = min(windowbuffer(2,:));

r13 = 0; % Peak amplitude for channel 1 - 3
r24 = 0; % Peak amplitude for channel 2 - 4

if 2000 <= maxIndex13 && maxIndex13 < 3000
    r13 = max13;
elseif 2000 <= minIndex13 && minIndex13 < 3000
    r13 = min13;
end

if 2000 <= maxIndex24 && maxIndex24 < 3000
    r24 = max24;
elseif 2000 <= minIndex24 && minIndex24 < 3000
    r24 = min24;
end

%%%%%%%%%%%%%%%%%%%%  Helper Functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%

function RP = initializeActiveX(rocfilepath)
% Create a Active X control by loading RPco.x
RP = actxcontrol('RPco.x');
invoke(RP, 'ConnectRA16', 'GB', 1);
invoke(RP, 'ClearCOF');

invoke(RP, 'LoadCOF', rocfilepath);
invoke(RP, 'Run');

status = invoke(RP, 'GetStatus'); % gets the status
if bitget(status, 1) == 0;  % checks for errors in the starting circuit
    er = 'Error connnecting to RP';
elseif bitget(status, 2) == 0;  % checks for connection
    er = 'Error loading circuit';
elseif bitget(status, 3) == 0;
    er = 'Error running circuit';
else
    global bufptr;
    bufptr = 0;
    disp('Circuit loaded and running');
    return;
end
finalizeActiveX(RP);
disp(sprintf('Error! Status is [%s] ', dec2bin(status)));
error(er);


function quake_server_handle = initializeQuakeConnection()
% Open TCP/IP connection with BMI hacked quake server
quake_server_ip = '10.244.10.94'; % nslookup quake.bme.ufl.edu
```

```
quake_server_port = 55501;
quake_server_handle = tcpip_open(quake_server_ip, quake_server_port);
tcpip_write(quake_server_handle, ['w']);


function finalizeActiveX(RP)
% Turn the circuit off
disp('Finializing circuit....');
invoke(RP, 'Halt'); % stop the TDT
invoke(RP, 'clearCOF');  % clear out your program


function finalizeQuakeConnection(quake_server_handle)
% Close TCP/IP connection with BMI hacked quake server
tcpip_close(quake_server_handle);


function s = lrudbnToStr(b)
% converts the global numerical constants to human readable strings
lrudbnStrArr = [' NOTHING'; '      UP'; '    DOWN';...
                '    LEFT'; '   RIGHT'; '   BLINK'; 'ARTIFACT'];
s = deblank(lrudbnStrArr(b+1,:));


function dataA = retrieveData(RP, buffsize, chunksize, samplefreq)
% retrieve data from the TDT system

global bufptr;
readvar = ['buff1'; 'buff2'; 'buff3'; 'buff4'];
dataA = zeros(size(readvar,1), chunksize);
for tries = 1:20 % limited retries
   index = invoke(RP, 'GetTagVal', 'sIndex');
   lack = abs(index - bufptr) - chunksize;
   if lack >= 0 % If we have enough data
      for k = 1:size(readvar,1)
         in = invoke(RP, 'ReadTagVEX', readvar(k, :), ...
                              bufptr, chunksize, 'F32', 'F64', 1);
         if length(in) == 0
            error('No data in channel [%s]', readvar(k,:));
         end
         dataA(k,:) = in;
      end
      bufptr = bufptr + chunksize;
      if (bufptr >= buffsize)
         bufptr = 0;
      end
      return;
   end
   pause(-lack/samplefreq); % this is not busy waiting!
end
error('Data acquisition failed');

% Hey are you still reading? Congratulations, his is the end of the code!
```