Correlating Head Gestures of Speakers in political debates with Audience Engagement

Divyansh Agarwal (da2629)

January 5, 2015

1 Introduction

1.1 Problem Statement

The work is based on the hypothesis that certain speaker gestures can convey significant information that are correlated to audience engagement, and that people's engagement can be effectively reported by EEG data. The domain is focused on political debates, particularly the 2012 U.S. Presidential debates. The method works under the hypothesis that neural activity would be most similar across subjects at moments when something piques their interest. Therefore, the method identifies these moments by computing the points of maximum correlation of EEG measurements across subjects and comparing it against a significance derived from a permutation test. In this project we have tried to extend the work done in Correlating Speaker Gestures in Political Debates with Audience Engagement Measured via EEG(John R. Zhang, Jason Sherwin, et al) [1] by testing a new hypothesis, i.e. 'Correlating Head Gestures in Political Debates with Audience Engagement Measured via EEG'.
The data used in the work was collected by John R. Zhang et al. To collect audience engagement data, 6 videos totaling 61 minutes were shown to each of 20 human subjects in an electrostatically shielded room. 28 clips are selected out of the 291 clips in the recorded video of the first (original air date October 3, 2012) and third (original air date October 22, 2012) debates between Obama and Romney: 8 from the first debate and 6 from the third debate featuring Obama, and 8 from the first debate and 6 from the third debate featuring Romney. The total duration of the clips is approximately 30.5 minutes, so each clip is 65 seconds on average. The 28 clips are randomly shuffled and 28 silent versions were made to make the subjects more concentrated on the image features rather than the words and sound. The EEG data were smoothed over a 5-second window and the frequency was finally 1HZ.

1.2 Possibilities of other features influencing audience engagement

To identify the relevant features piquing audience attention, an effort was made to see the clips multiple times and empirically identify the moments that captured the attention of the viewer. However, with so many different and concurring movements, it was difficult to acertain whether these features triggered an interest collectively or in a stand alone manner. Also, some of the movements were too recurring and might not always trigger interest. Following are some interesting observations about the dataset and the various facial features:

1. Eyebrows: Romney had a prominent eyebrow movement whenever he was emphasizing a point. A rising up of eyebrows was observed.
2. Head movement: changes in direction of head, head shaking and nodding. In first debate obama's static head position lead to boredom. Once again the clips belonging to Romney showed a much greater movement in head movement.
3. Fluency and pace of speaking could be tested as one of the cue capturing attention

1.3 Head gestures role in day to day conversation

- dialogue structuring, nodding and shaking to show agreement and disagreement
- changes in head position right before a current listener takes turn at speaking
- gaze and head direction can be used for referencing to objects or points
- as a reinforcement agent
- quick head movement as a sign of alarm
- mutual gaze - a sign of awareness

2 Exploration of affective attributes influencing audience engagement

During the initial phases of the project, we explored the possibility of correlating speaker's emotion with audience engagement. The heuristic was derived from the simple understanding that in normal human interaction often the emotions conveyed by the speaker do influence our perception and interest, for instance a smiling and confident person can charm the audience easily as opposed to a angry or a disinterested person. The dive into literature led to a number of interesting approaches and challenges.

2.1 Description of Affect

Affect has been described by psychologists in terms of six discrete categories. These basic emotion categories include happiness, sadness, fear, anger, disgust and surprise. The advantage of such a classification is that people use this categorical scheme to describe observed emotional displays in daily life. However, this discrete lists of emotions fails to describe the range of emotions in natural communication settings which involve a lot of subtle and interchanging signals. An alternative to the categorical description is the dimensional description where an affective state is characterized in terms of small number of latent

dimensions rather than in terms of a small number of discrete emotion categories. These dimensions include Activation, Expectation, Power and Valence. Activation (Arousal) is the individual's global feeling of dynamism or lethargy. Power (Dominance) dimension subsumes two related concepts, power and control. Valence is whether the person feels positive or negative. Expectation is whether the person is expecting something or is caught unaware (surprised). In contrast to categorical representation, dimensional representation enables to capture a range of emotions as a combiantion of the four dimensions. However, this representation is non-intuitive.[2]
There have been work addressing both categories of emotion description. The efforts on learning classifiers for different discrete emotion categories have been persistent for a while. The Extended Cohn-Kanade Dataset (CK+)[3] provides a complete dataset for action unit and emotion-specified expression. The target expression for each sequence is fully FACS coded and emotion labels have been revised and validated. Baseline result with Active Appearance Model and Support Vector Machine has been provided. The motivation behind the dataset is to remove ambiguity regarding the different methods reporting accuracy and results on different datasets. Though the results are promising for emotions detection but the dataset is meant for exaggerated expressions.
Recently AVEC (Audio/Visual Emotion Challenge)-2011 and 2012[4][5] have made a conscious effort towards advancing emotion recognition systems to be able to deal with naturalistic behavior in large volumes of un-segmented, non-prototypical and non-preselected data as this is exactly the type of data that both multimedia retrieval and human-machine/human-robot communication interfaces have to face in the real world. The Audio/Visual Emotion Challenge 2011/2012 (AVEC2012) data contains 32 continuous, frontal face videos of an interview subject being emotionally engaged. It is more challenging than previous data sets because the data is not acted or sponsored by the interviewer. The subject is speaking with an interviewer. As a result, emotions are subtle and more difficult to detect than in other data sets. AVEC-2011[4] put forth the problem of learning binary classifier for both valence and arousal domain. AVEC - 2012[5] put fourth the regression problem as opposed to the classification problem in AVEC - 2011. But, the results are only able to achieve an accuracy of around 50 percent, thus the idea of correlating speaker's emotions with audience engagement, though interesting, looks a bit premature.

3 Head Gesture Detection

3.1 Model for identifying key facial landmarks:

An open-source adaptation of CVPR-2014 paper is used to identify 68 key facial points [One Millisecond Face Alignment with an Ensemble of Regression Trees by Vahid Kazemi and Josephine Sullivan][6]. It uses an ensemble of regression trees to estimate the key facial points directly from a sparse subset of pixel intensities. The landmark detection takes approximately milliseconds per frame and achieves accuracy greater than or comparable to state of the art methods on standard dataset.
The open source implementation (in dlib C++ library) uses a small dataset to train a face land-marking model using the program **trainShapePredic-**

tor.cpp. The learnt model is represented as a .dat file. For the purpose of the project the pre learnt sp.dat file was used. The Library can be installed using the following link: `http://sourceforge.net/projects/dclib/files/latest/download` and is critical for working of the later parts of the project. The Figure 1 shows the location of various landmarks. Individual frames of the video can be processed to simulataneously track different facial landmarks thereby capturing the various movement of head, eye, nose, mouth, etc as shown in Figure 2.

Listing 1: C++ code to get the facial landmark information for each frame

```cpp
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>
#include <dlib/image_processing.h>
#include <dlib/gui_widgets.h>
#include <dlib/image_io.h>
#include <iostream>
#include <cv.h>
#include <highgui.h>
#include <dlib/opencv.h>
#include<fstream>
using namespace dlib;
using namespace std;
#define SKIP_RATE 1


int main(int argc, char** argv)
{
    try
    {
        // The code takes in a shape model file (.dat) and name of the video to
        if (argc == 1)
        {
            cout << "Call this program like this:" << endl;
            cout << "./face_landmark_detection_ex shape_predictor_68_face_landm
            return 0;
        }

        // We need a face detector.
        frontal_face_detector detector = get_frontal_face_detector();
        // And we also need a shape_predictor.
        // loading the model from the shape_predictor_68_face_landmarks.dat
        shape_predictor sp;
        deserialize(argv[1]) >> sp;

        ofstream myfile;
        string filename;
        cout << "Enter file to process: \n";
        cin >> filename;
        string landmarkInfo = filename + ".txt";
        myfile.open(&landmarkInfo[0]);
```

```
        int frameCount = 0;
        string videoName = filename + ".avi";
        cv::VideoCapture cap( videoName); // open the video file for reading
        if ( !cap.isOpened() )
        {
            cout << "Cannot open the video file" << endl;
            return -1;
        }
        double fps = cap.get(CV_CAP_PROP_FPS); //get the frames per seconds of
        double totalFramesInVideo = cap.get(CV_CAP_PROP_FRAME_COUNT);
        double capturedFrames = (int)(totalFramesInVideo/SKIP_RATE);
        cout << "Frame per seconds : " << fps << ",  Total Frames: " << totalFr
        cv::Mat frame;

        myfile << capturedFrames  << endl;
        while(1)
        {
            bool bSuccess = cap.read(frame); // read a new frame from video
            if (!bSuccess) //if not success, break loop
            {
                cout << "Cannot read the frame from video file" << endl;
                break;
            }
            frameCount++;

            myfile << frameCount << endl;
            cout << "Processing frame " << frameCount << endl;
            array2d<rgb_pixel> img;
            assign_image(img, cv_image<bgr_pixel>(frame));

            // Now tell the face detector to give us a list of bounding boxes
            // around all the faces in the image.
            std::vector<rectangle> dets = detector(img);

            // Now we will go ask the shape_predictor to tell us the pose of
            // each face we detected.
            std::vector<full_object_detection> shapes;
            full_object_detection shape;
            // Save the position of each facial landmark for each frame in a fi
            if(dets.size() > 0)
            {
                for (unsigned long j = 0; j < 1; ++j) // Chk for just the first
                {
                    shape = sp(img, dets[j]);
                    for( int parts = 0; parts < shape.num_parts(); parts++)
                        myfile << shape.part(parts) << endl;
                    shapes.push_back(shape);
                }
            }
            else
```

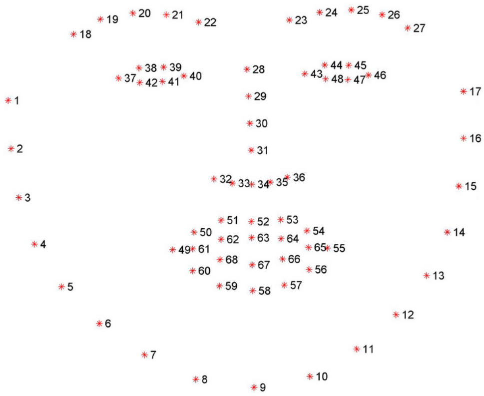
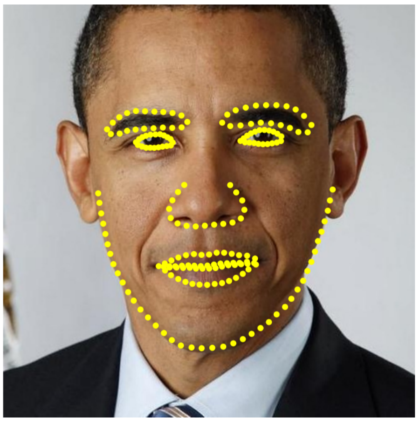


Figure 1: Anatomy of Key facial landmarks (68)



(a) Subfigure

(b) Subfigure

Figure 2: 68-Key facial landmarks annotated

```
                shapes.push_back(shape);

        }
        myfile.close();

    }
    catch (exception& e)
    {
        cout << "\nexception thrown!" << endl;
        cout << e.what() << endl;
    }
}
```

3.2 Head pose/head normal estimation using facial landmarks

3.2.1 Background for human head pose estimation

Human head pose estimation is a hard problem because it suffers from classical problems in computer vision such as viewpoint variation, illumination variation and occlusion. It has been a topic of active research and number of approaches has been tried. For the purpose of the project, we felt that the geometric approaches are best suited. Nonetheless, a succinct analysis for different head pose estimation methods is provided in the following paragraphs. Refer to [7] for detailed discussion.

Appearance based modeling methods generally consider the head pose as a collection of face images from multiple viewpoints. In a training phase, a model is presented with a set of face images of known pose. In an online phase, a query face image is compared to a leaned model to find the most faithful match. However without interpolation, these appearance based approaches are only capable of estimating discrete pose locations. Moreover, they require large amounts of training data, and are highly sensitive to small changes in face position and image scaling.

Flexible models such as active appearance models (AAM) show good invariance to head localization error since they adapt to the image and find the location of the facial features. Although precise, using 3D head models and image registration techniques are computationally intensive and require a high image resolution that could slow down realtime performance.

Geometric based approaches can potentially achieve accurate pose measurements by tracking the location of only a few facial features such as the eyes, nose, and mouth. The computational efficiency of tracking a few features and determining the head orientation from the their configuration allows real-time performance and is well suited for our application. The downside is that feature detection and tracking are critical to the accuracy of the pose estimation results, and can lead to failure when features are lost. However, the implementation for landmark detection, as discussed in the previous section, is fairly robust and provide a near real time estimates of facial landmarks.

3.2.2 Geometric methods for Head pose estimation

There are various ways in which the facial configuration can be exploited for head pose estimation. For the purpose of this project, we adopted the approach of [8].

The facial model is based on the rations of four world lengths L_f, L_n, L_e and L_m as shown in figure 4. The far corners of the eyes and mouth demarcate the facial plane. L_f and L_m are measured on the symmetry axis of the facial plane while L_e is measured perpendicular to the symmetry axis of the facial plane. L_n is the distance between the nose tip and the nose base. It is measured along the normal to the facial plane, the facial normal. The corresponding distances in the image are denoted by small case letters. The method also required two model rations: $R_m = L_m/L_f$ and $R_n = L_n/L_f$

The symmetry axis is found by joining the midpoint between the eyes to the mouth, and the nose base is located using the model ratio $R_m = L_m/L_f$. The angle τ is the angle between the projection of face normal in the image and

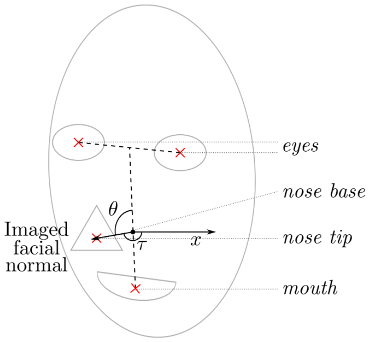


Figure 3: Facial Model [8]

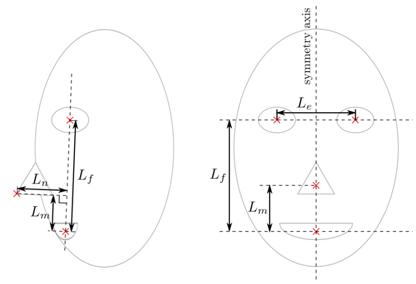


Figure 4: Profile (left) and frontal (right) views of a face. The face model is composed of distance ratios between the eye, nose and mouth positions of a typical face. The facial plane is defined by the locations of the eyes and the mouth. The pitch and yaw angles are estimated by assuming that the face is planar, and the nose is a vector normal to this plane.[8]

the x-axis as can be seen 3. The angle σ which the facial normal makes with the vector d normal to the image plane is called the slant, as shown in figure 5. This angle may be found by using the image measurements θ, l_n, l_m and the model ratio $R_n = L_n/L_f$. Please refer to [8] for supporting equations. It can be shown that in camera centered coordinates, the facial normal $\hat{n}$ is given by:

$$\hat{n} = [\sin\sigma\cos\tau, \sin\sigma\sin\tau, -\cos\sigma]$$

Listing 2: C++ code to get the normal estimates (normalEstimation.cpp)

```

#include<iostream>
#include<cv.h>
#include<highgui.h>
#include<fstream>
#include<vector>
#include<utility>
#include<string>
```

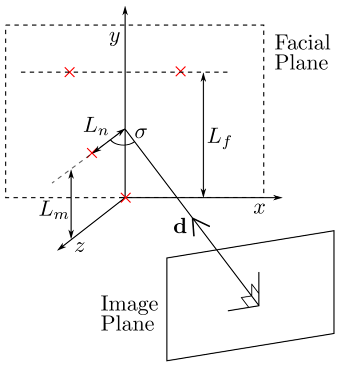


Figure 5: The face centred coordinate system used to calculate the slant angle σ. The facial slant σ is the angle between the vector d normal to the image plane, and the facial normal located along the z axis of the facial plane. [8]

```
#include<math.h>
using namespace std;
using namespace cv;

void facialLandmarks(string landmarkInfo, vector< vector< pair<int,int> > >& la
    numFramesCaptured)
{
    ifstream infile(&landmarkInfo[0]);
    //vector< vector< pair<int,int> > > landmarks; // max no of frames 100 and
    vector<int> frameIndex;
    //int numFramesCaptured = 0;
    infile >> numFramesCaptured;
    cout << numFramesCaptured << endl;
    for( int i=0; i<numFramesCaptured; i++)
    {
        int f;
        infile >> f;
        frameIndex.push_back(f);
        vector< pair<int,int> > myvec;
        for( int parts=0; parts < NUMPARTS; parts++)
        {
            int r,c;
            string s2,s3;
            char c1;
            infile >> c1 >> r >> s2 >> c >> s3;
            myvec.push_back(make_pair(r,c));
        }
        landmarks.push_back(myvec);
    }
}
void artifacts(Point eyeMid, Point mouthMid, Point noseTip, Point noseBase,
                Mat& frame)
```

```
{
    circle(frame, eyeMid, 1 , CV_RGB(255,255,0), 1);
    circle(frame, mouthMid, 1 , CV_RGB(255,255,0), 1);
    circle(frame, noseBase, 1 , CV_RGB(255,255,0), 1);
    circle(frame, noseTip, 1 , CV_RGB(255,255,0), 1);
    line(frame, noseBase, noseBase + Point(10,0), CV_RGB(0,0,255),1);
    line(frame, noseBase, noseBase - Point(0,-10), CV_RGB(0,0,255),1);

}
void normalEstimates(double nVec[], Point eyeMid, Point mouthMid,
                    Point noseTip, Point noseBase )
{
    double tilt = acos( (noseTip - noseBase).dot(Point(1,0))
                    /norm(noseTip - noseBase) );
    double len_nose = norm(noseTip - noseBase);
    double len_face = norm(eyeMid - mouthMid);
    double m1 = pow(len_nose/len_face,2);
    double m2 = pow( (noseTip - noseBase).dot(Point(0,-1))/
                    norm(noseTip - noseBase), 2);
    double Rn = RATIO_NOSE;
    double dz = ( Rn*Rn - m1 - 2*m2*Rn*Rn + sqrt( pow( m1 - Rn*Rn, 2)+4*m1*m2*R
                    / ( 2*Rn*Rn*(1-m2) );
    dz = sqrt(dz);
    double sigma = acos( dz );
    nVec[0] = sin(sigma)*cos(tilt);
    nVec[1] = sin(sigma)*sin(tilt);
    nVec[2] = -cos(sigma);
}
int main()
{
    string filename, landmarkFilename, normalFilename;
    int numFramesCaptured = 0;
    vector< vector< pair<int,int> > > landmarks; // max no of frames 100 and o
    cout << "Video to be processed: \n";
    cin >> filename;
    landmarkFilename = filename + ".txt";

    //Get Facial Landmark Information for all the frames
    facialLandmarks(landmarkFilename, landmarks, numFramesCaptured);

    ofstream normals;
    normalFilename = filename + "_normal.txt";
    normals.open(&normalFilename[0]);
    normals << numFramesCaptured << endl;
    int frameCount = 0;
    while( frameCount < numFramesCaptured - 1 )
    {
        frameCount++;
        Point eyeMid( (landmarks.at(frameCount).at(37).first +
                        landmarks.at(frameCount).at(46).first)/2,
```

```
            (landmarks.at(frameCount).at(37).second +
                landmarks.at(frameCount).at(46).second)/2 );

            Point mouthMid((landmarks.at(frameCount).at(49).first +
                            landmarks.at(frameCount).at(55).first)/2,
                    (landmarks.at(frameCount).at(49).second +
                    landmarks.at(frameCount).at(55).second)/2 );
            Point noseTip(landmarks.at(frameCount).at(30).first,
                        landmarks.at(frameCount).at(30).second );
            Point noseBase( 0.6*mouthMid.x + 0.4*eyeMid.x ,
                            0.6*mouthMid.y+ 0.4*eyeMid.y);

            //To plot various facial artifacts
            artifacts( eyeMid, mouthMid, noseTip, noseBase, frame);

            // Normal Vector Estimation
            double nVec[3];
            normalEstimates( &nVec[0], eyeMid, mouthMid, noseTip, noseBase );

            // IF nVec[0] = NaN then assign a default 0 value
            if( nVec[0] != nVec[0])
                normals << 0 << " " <<  0 << " " << 0 << endl;
            else
                normals << nVec[0] << " " <<  nVec[1] << " " << nVec[2] << endl;
        }
        normals.close();
}
```

3.2.3 Head movement estimation: Speed and Acceleration

The velocity and acceleration estimation is done in y any x direction, i.e., along the face symmetry axis and the axis perpendicular to it on the facial plane by utilizing the y and x component of the estimated normal vector. The normal estimation is done at 30 fps. However, we need the sample the data to 1 Hz since the EEG data is sampled at 1 Hz. Also, the difference between two frames is insignificant ($\frac{1}{30}$ th of a second) to register any significant movement between two consecutive frames. Thus, the speed is sampled at 1 Hz and is given by:

$$speed = abs(\text{avg. disp. in first 15 frames} - \text{avg. disp in last 15 frames})$$

By extension, the acceleration in both directions are sampled at 1 Hz by utilizing the sampled speed. A couple of different experiments/approaches were tried for sampling the data. However, as part of the project we had difficulty in ascertaing the usefulness of one sampling method over other due to lack of quantitative measures. Finally, the above method was adopted for sampling.

Listing 3: C++ code to get the head motion estimates and output video (motionDetection.cpp)

```

#include<iostream>
```

```
#include<cv.h>
#include<highgui.h>
#include<fstream>
#include<vector>
#include<utility>
#include<string>
#include<math.h>
using namespace std;
using namespace cv;
#define XVELOCITY 0.08
#define WINDOW 15

void displayOnFrame( double xVelocity, double yVelocity, double xAcc, double yA
{
    char xvelStr[100], xaccStr[100];
    char yvelStr[100], yaccStr[100];
    sprintf(xvelStr,"X Velocity: %lf ", xVelocity );
    sprintf(xaccStr,"X Acceleration: %lf ", xAcc );
    sprintf(yvelStr,"Y Velocity: %lf ", yVelocity );
    sprintf(yaccStr,"Y Acceleration: %lf ", yAcc );
    if( xVelocity > 0.05)
        putText(frame, xvelStr, Point(20,20), CV_FONT_NORMAL, 0.5,
        Scalar(255,255,0),1,1);
    else
        putText(frame, xvelStr, Point(20,20), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,255),1,1);
    if( yVelocity > 0.05)
        putText(frame, yvelStr, Point(300,20), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,0),1,1);
    else
        putText(frame, yvelStr, Point(300,20), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,255),1,1);
    if( xAcc > 0.05)
        putText(frame, xaccStr, Point(20,40), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,0),1,1);
    else
        putText(frame, xaccStr, Point(20,40), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,255),1,1);
    if( yAcc > 0.05)
        putText(frame, yaccStr, Point(300,40), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,0),1,1);
    else
        putText(frame, yaccStr, Point(300,40), CV_FONT_NORMAL, 0.5,
            Scalar(255,255,255),1,1);
}
void velocity1(string velocity1Filename, int &numFramesProcessed, vector< vecto
{
    int numSeconds = numFramesProcessed / 30;
    ofstream velocityInfo(&velocity1Filename[0]);
    velocityInfo << numSeconds << endl;
```

```
    int frameCount = 0;
    cout << numFramesProcessed << endl;
    double prevXvelocity = 0;
    double prevYvelocity = 0;
    while(frameCount < numFramesProcessed)
    {
        double Xdisp1 = 0, Xdisp2 = 0;
        double Ydisp1 = 0, Ydisp2 = 0;
        for( int i = 0; i < WINDOW; i++)
        {
            Xdisp1 = Xdisp1 + nVector.at(frameCount).at(0);
            Ydisp1 = Ydisp1 + nVector.at(frameCount).at(1);
            frameCount++;
        }
        for( int i = 0; i < WINDOW; i++)
        {
            Xdisp2 = Xdisp2 + nVector.at(frameCount).at(0);
            Ydisp2 = Ydisp2 + nVector.at(frameCount).at(1);
            frameCount++;
        }
        double Xvelocity = abs(Xdisp1-Xdisp2)/WINDOW;
        double Yvelocity = abs(Ydisp1-Ydisp2)/WINDOW;
        double Xacc = Xvelocity - prevXvelocity;
        double Yacc = Yvelocity - prevYvelocity;
        velocityInfo << frameCount/30  << " " << Xvelocity << " " << Yvelocity
        prevXvelocity = Xvelocity;
        prevYvelocity = Yvelocity;
    }
    velocityInfo.close();
}
void outputVideo(string inputVideoname, string outputVideoname, string velocityI
{

    // open the video files for reading and writing
    cv::VideoCapture cap(inputVideoname);
    cv::VideoWriter output(outputVideoname, cap.get(CV_CAP_PROP_FOURCC),cap.get
            cv::Size(cap.get(CV_CAP_PROP_FRAME_WIDTH),cap.get(CV_CAP_PROP_FRAM
    if (!output.isOpened())
    {
        std::cout << "!!! Output video could not be opened" << std::endl;
        return;
    }

    if ( !cap.isOpened() )
    {
        cout << "Cannot open the video file" << endl;
        return;
    }

```

```
// Reading from velocityInfo file
    ifstream velocityInfo(&velocityFilename[0]);
    int numSeconds;
    vector< vector<double> > motionVector;
    if(!velocityInfo.is_open() )
    {
        cout << "ERROR: Cant open normal information file \n";
        exit(1);
    }
    velocityInfo >> numSeconds;
    char c1;
    for( int i = 0; i < numSeconds; i++)
    {
        vector<double> myvec;
        double val;
        for( int j = 0; j < 5; j++ )
        {
            velocityInfo >> val;
            myvec.push_back(val);
        }
        motionVector.push_back(myvec);
    }
    velocityInfo.close();
// Writing Video
    double fps = cap.get(CV_CAP_PROP_FPS); //get the frames per seconds of the
    double width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
    double height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
    cout << "Height: " << height << " Width: " << width;
    cout << " Frame per seconds : " << fps << endl;
    cv::namedWindow("MyVideo"); //create a window called "MyVideo"
    cv::Mat frame;
    int frameCount = 0;
    while(1)
    {

        int second = frameCount / 30;
        bool bSuccess = cap.read(frame); // read a new frame from video
        if (!bSuccess) //if not success, break loop
        {
            cout << "Cannot read the frame from video file" << endl;
            break;
        }
        if( frameCount >= numSeconds*30)
            break;
        //if( frameCount % 5 != 0 )
        //  continue;
        displayOnFrame( motionVector.at(second).at(1), motionVector.at(second).
                motionVector.at(second).at(3), motionVector.at(second).at(4),
frame);
        output.write(frame);
```

```
        frameCount++;
    }
    output.release();
    cap.release();

}
int main()
{
    string videoname;
    cout << "Video name: \n";
    cin >> videoname;
    string inputVideoname = videoname + ".avi";
    string outputVideoname = videoname + "_out.avi";
    string normalFilename = videoname + "_normal.txt";
    string velocity1Filename = videoname + "_velocity1.txt";
    cout << "Files: " << inputVideoname << "  " << outputVideoname << "  " << normalFilename << "  " << velocity1Filename<<  endl;

    //Getting the nVec's
    int numFramesCaptured;
    vector< vector<double> > nVector;
    ifstream normalInfo(&normalFilename[0]);
    if(!normalInfo.is_open() )
    {
        cout << "ERROR: Cant open normal information file \n";
        exit(1);
    }
    normalInfo >> numFramesCaptured;
    for( int i = 0; i < numFramesCaptured-1; i++)
    {
        vector<double> myvec;
        char c1;
        double val;
        for( int j = 0; j < 3; j++ )
        {
            normalInfo >> val;
            myvec.push_back(val);
        }
        normalInfo >> c1;
        nVector.push_back(myvec);
    }
    normalInfo.close();
    cout << numFramesCaptured << endl;

    int numFramesProcessed = numFramesCaptured - 1 - (numFramesCaptured-1)%30;
    velocity1(velocity1Filename, numFramesProcessed, nVector);
    outputVideo(inputVideoname, outputVideoname, velocity1Filename);

}
```

Listing 4: Matlab code for correlation between the head motion estimates and output video (headEEGcorrelation.m)

```
% remove the first line from velocity_1.txt file
file = input('Filename: ', 's');

EEG = csvread(strcat(file,'_G.txt'));
% fp = fopen( strcat( file , '_velocity1.txt'));
% velocityInfo = textscan(fp,'%d %f %f %f %f');

f1 = fopen(strcat(file,'_normal.txt'));
positionInfo = textscan(f1,'%f %f %f');
meanX = mean(positionInfo{1})
stdX = std(positionInfo{1})

figure, plot(smooth(positionInfo{1},10));
title(' X position over all frames (Moving Avg)');
% disp(['In X dimension, Mean: ', num2str(meanX), ' and standard Deviation: ',


% figure, plot(smooth(positionInfo{2}));
% title('Y position over all frames (Moving Avg)');
% disp(['In Y dimension, Mean: ', num2str(mean(positionInfo{2})), ' and standar


PC1 = EEG(1,:);
% velocityX = velocityMode(positionInfo,10,10,30);
velocityX = velocityMean(positionInfo,10,10,30);

figure;
subplot(2,1,1);
plot(PC1(1:size(velocityX,2)));
title('PC1');

subplot(2,1,2);
plot(velocityX);
title('X velocity ');
line([0,size(velocityX,2)],[mean(velocityX),mean(velocityX)],'Color',[1,0,0]);
saveas(gcf,strcat(file,'_velocityX_EEG.fig'));
meanVelocityX = mean(velocityX)
stdVelocityX = std(velocityX)
CorrelationVelX = corr2(PC1(1:size(velocityX,2)),velocityX)
```

3.3 Results and Observations

The results have been pretty disappointing as we fail to observe any consistent pattern. The best case correlation with EEG data was 0.32 for clip TPO1.avi (The clips have been renamed but an index is provided mapping each file to its orignal name) and for some clips (obama2, obama3) even a negative correlation is observed suggesting that the rise in one signal coincides with fall in other

Filename	X-Mean Pos.	X-Std. Dev. Pos	X-Mean Speed	X S.D. Speed	Correlation
TPO1	- 0.086	0.13	0.15	0.13	0.32
TPO2	0.013	0.13	0.13	0.09	0.18
TPO4	0.060	0.11	0.12	0.06	0.20
obama1	-0.030	0.23	0.13	0.09	0.12
obama2	0.200	0.15	0.17	0.11	-0.16
obama3	0.060	0.15	0.18	0.12	-0.08
obama4	0.12	0.14	0.16	0.11	0.15
romney5	-0.300	0.21	0.18	0.19	0.02
romney7	0.34	0.20	0.14	0.16	0.21
romney8	-0.320	0.16	0.15	0.15	0.17

Table 1: Results

signal. The results for the video clips are in Table:1. In Fig: 6 - 12, the red line corresponds to mean speed in x-direction.

We cannot confidently comment that the head gestures does not inluence audience engagement because:

- Small set of people, i.e only twenty people, might be too small a sample set
- Small size of clips - 65 seconds is to small an interval to make an inference
- A different sampling algorithm can be applied to capture the head movement from position estimates of face landmarks.
- We are not sure how good the speed (velocity) and acceleration features are in capturing the higher level features of head movement, i.e nodding and shaking. It would be interesting to perform some experiments to preprocess the raw head movement signal.
- We understand in the Zang et al paper, a different correlation algorithm was applied, therefore it might be a good idea to experiment with different correlation algorithms, particularly something that is concerned only with the growth and decay of signals at corresponding points rather than their magnitudes.

4 References

1. John R. Zhang, Jason Sherwin et al. Correlating Speaker Gestures in Political Debates with Audience Engagement Measured via EEG. Proceedings of the ACM International Conference on Multimedia, 387-396.

2. Zhihong Zeng, Maja Pantic et al. A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions. IEEE Transactions on Pattern Analysis And Machine Intelligence

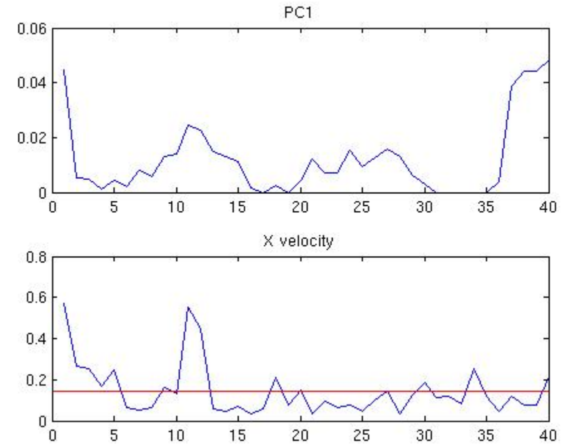


Figure 6: TPO1, Correlation: 0.32

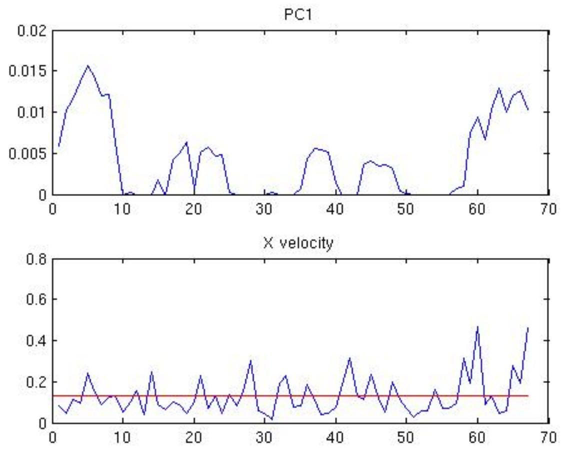


Figure 7: TPO2, Correlation: 0.18

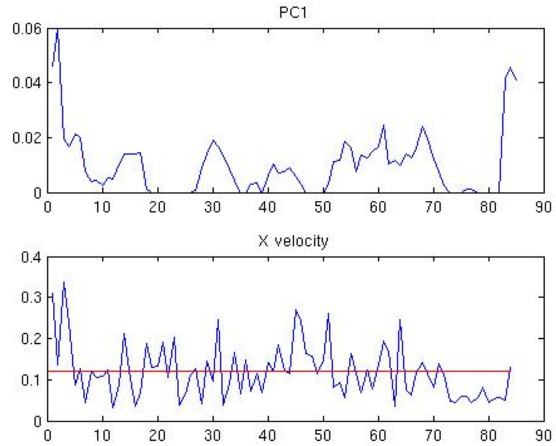


Figure 8: TPO4, Correlation: 0.20

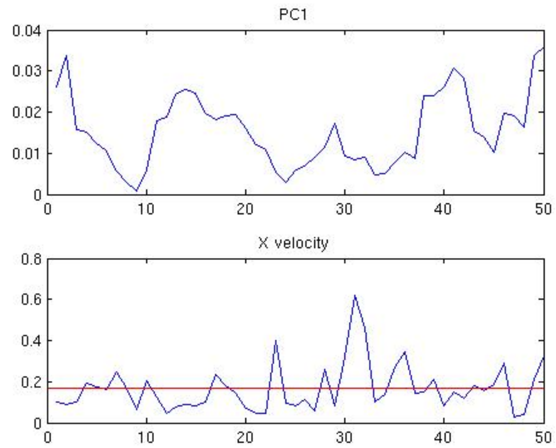


Figure 9: obama2, Correlation: -0.16

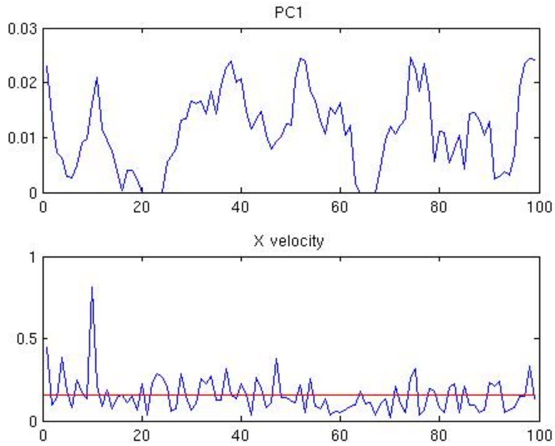


Figure 10: obama4, Correlation: 0.15

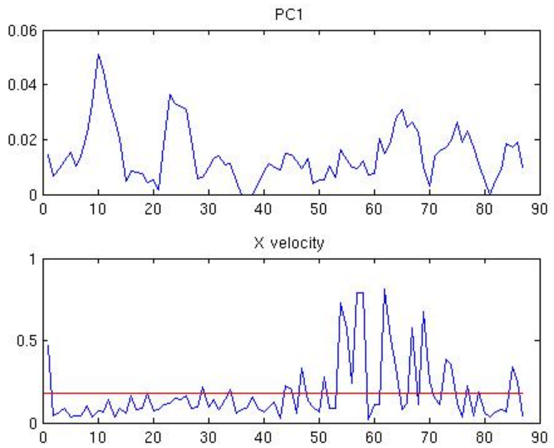


Figure 11: romney5, Correlation: 0.02

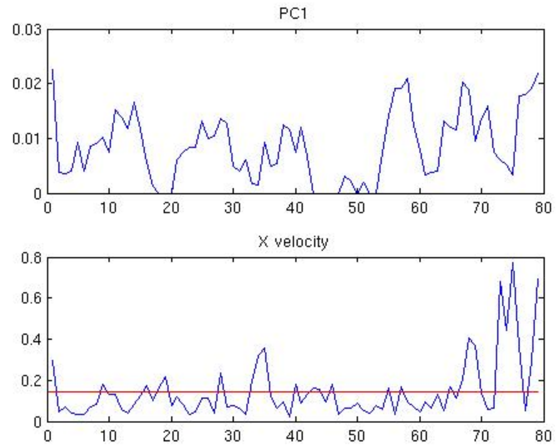


Figure 12: romney8, Correlation: 0.17

3. Patrick Lucey, Jeffrey F. Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. (CVPR - 2010).

4. Bjorn Schuller, Michel Valstar, Maja Pantic et al. AVEC 2011 – The First International Audio/Visual Emotion Challenge.

5. Bjorn Schuller, Michel Valstar, Maja Pantic et al. AVEC 2012 – The Continuous Audio/Visual Emotion Challenge.

6. Kazemi, Vahid, and Sullivan Josephine. "One Millisecond Face Alignment with an Ensemble of Regression Trees." (CVPR - 2014).

7. Murphy-Chutorian, E and Trivedi, M.M. . Head Pose Estimation in Computer Vision: A Survey. Pattern Analysis and Machine Intelligence, IEEE Transactions, 607-626.

8. A. H. Gee and R. Cipolla, Determining The Gaze Of Faces In Images, Image and Vision Computing - 1994, vol 12, 639 - 647.