Head Orientation APP on iPhone

Weiming Lu
U5287127@anu.edu.au

Supervisors:
Prof. Tom Gedeon
Dr. Dingyun Zhu

Master of Computing
Research School of Computer Science
Australian National University
Acknowledgments

First and foremost, I would like to thank my supervisors of this project, Prof. Tom Gedeon and Dr. Dingyun Zhu. They were abundantly helpful and offered invaluable assistance, support and guidance for my work. My special thanks to Prof. Tom Gedeon for his patient proofreading of this report. I also would like to express my gratitude towards my project-meeting partner Sida Li for his suggestions and kind encouragement. In addition, I would like to thank Dr. Weifa Liang for his weekly guidelines about report writing and the presentation.
Abstract

The purpose of this project was to extend an existing IOS iPhone face-tracking application with a new face detection library and to upgrade control and drive methods of the face tracking control mode to achieve better performance. In addition, I implemented a new human computer interaction method called head orientation. To evaluate the performance of the new application, I conducted a quantitative analysis based on interaction experiments and a qualitative analysis based on post-test questionnaires. To summarize, the new application achieved a significantly better performance than the old version. However, the face tracking control mode still performed worse than the accelerometer control mode. Lastly, I proposed a solution for the face tracking control mode for further improvement work.
1. Introduction

1.1. Motivation
When it comes to mobile devices, we are very familiar with two kinds of interaction methods: Interaction by using a keypad or interaction by using a stylus (Hansen, Eriksson & Lykke-Olesen 2005). Either of these methods has its limitation since user’s hands are forced to participate on them. Imagine a situation that a user is cooking in the kitchen; both of his hands are stained with oil. At this time, scrolling the cookbook displayed on his tablet becomes impossible. To address this issue, varieties of new interaction methods have come out to free user’s hands. Among these methods, we believe that the face tracking control achieved by using face detection technology is a good choice. More specifically, in face tracking control, a mobile device is able to detect the motion of the user’s head captured by its front camera as user’s input and respond accordingly.

1.2. Background
To realize the face tracking idea, an existing program called FaceTrack runs on the iPhone platform was designed and implemented by a Computer Science student (Tian, 2012). In general, the program integrated two basic human-device interaction modes to make comparisons, which were face tracking control mode and accelerometer control mode, to rotate and zoom a 3-dimensional cube displayed on the iPhone screen. For face tracking control mode, the program invoked the iPhone’s front camera to detect the position of the user’s face. For accelerometer control mode, the program fetched motion data from the iPhone’s accelerometer and gyroscope to detect the user’s device-holding status. Both modes enabled the user to control movement and scale of the 3-dimensional cube. In order to test the effectiveness of two interaction modes, the program included an experiment module for four crossover trials, which were ARAZ (Accelerometer Rotate Accelerometer Zoom), HRHZ (Head Rotate Head Zoom), ARHZ (Accelerometer Rotate Head Zoom) and HRAZ (Head Rotate Accelerometer Zoom), respectively. Based on a previous study (Maciel, 2013), the experiment results and user feedback showed that the face tracking control mode had relatively poor effectiveness and was difficult to use compared to the accelerometer control mode.

1.3. Contribution
The aim of my project was to improve the existing face tracking application to achieve a better performance. For this specific purpose, I did some research on the public announced face detection APIs and replaced the face detection library used in the existing application with a better one. Moreover, I upgraded the drive, control method of the face tracking control mode of the existing application to meet shortfall, and hence achieved a new interaction method called head orientation.
2. Deficiencies of Existing Program

By trialing the application and reverse analysis of the code, two main deficiencies that may have caused the problem were identified. The first deficiency manifested in the poor face recognition supported by the face detection library used in the application. The second deficiency may be due to poor design of the control and drive method. This section will discuss these two deficiencies in detail to undertake the further improvement work.

2.1. Face Detection Library

Face tracking control quality is highly dependent on the face detection library used. The existing application uses Apple’s build-in image processing and analysis framework (library in Objective-C environment) named ‘Core Image’ to handle real-time video captured by the front camera. Face detection capability is achieved by the invoking of Core Image’s internal class CIDetector and CIFaceFeature. In general, a CIDetector object detects features (faces) in a picture; a CIFaceFeature object finds the positions of eyes and mouth in features (faces) that are detected with CIDetector (IOS Developer Library 2013). By testing on a separate face detection program which also uses the Core Image framework, three main problems are exposed. Firstly, this library has a high failure rate of face detection in dark environments. The detector appears to lose faces frequently in an inadequate lighting condition. Secondly, this library has weak noise tolerance. Ideally, a good detector should be able to filter out all other objects instead of faces. However, CIDetector does not respond well when the camera view’s background is complex. For instance, the condition that multiple light sources radiating in the background of the camera view may interfere with face recognition accuracy badly. Lastly, CIDetector and CIFaceFeature are designed for front face detection only. However, a good detector should be tolerant to faces inclined or rotated to some extent. In other words, a good detector can detect a face even if the face is slightly to the side and so on. This is a very important technical indicator because we cannot guarantee that the user is always perfectly front face to the camera. Moreover, in order to achieve the head orientation interaction method mentioned previously, side face tolerance is an essential capability.

2.2. Drive and Control

The choice of face detection library is not the only factor which can affect the face tracking quality. Fundamentally, the movement of the 3-dimensional cube is driven by the face motion data. The face motion data is calculated form the face tracing data collected with the face detector. A good design of drive method and control method can result in good tracking. The drive method may consist of calculations
such as face displacement calculation, rotation speed conversion and rotation angle conversion. The control method may include action trigger design and threshold settings. From the operation experiment, we find two main problems of drive and control methods used that may have led to the bad tracking result. The most obvious problem is the laggy feeling of the cube rotation. During the operation, the movement of the cube is not smooth. This seemed to be due to the frames lost at the first time steps. But eventually, we found that it is actually caused by the inappropriate drive method. The second problem is that it is difficult for the user to stop the cube from rotating due to a defective of control method. This problem can result in bad user results in the HCI experiment. Detailed causes and improvements will be mentioned in sections 4.2 and 4.3.
3. Technology Comparison

As mentioned in the previous section, the face tracking quality of the proposed application is highly dependent on the choice of face detection library. A good library not only has a high recognition success rate, but also has good noise handling capability and can be adapted to a variety of complex environments. There are a number of face detection APIs that can be used in this field and they have different properties. The existing application does face detection using the Core Image framework, which has been available for Mac OS for years and has been integrated into iOS environment since iOS version 5. Another popular API is openCV, which is free to use under an open source BSD license (OpenCV 2013). The library is cross-platform and has been published as a framework for development under iOS environment, which could be a very good option for our application (OpenCV 2013). In addition, a new face analysis API called ‘CI2CV’ designed by CSIRO ICT Centre Computer Lab is available, which has some interesting features and could be another alternative. In this section, I will explore the characteristics of these three libraries and make comparisons by using demo face detection applications to find the most suitable library for our proposed application.

3.1. Core Image

A new feature of iOS version 5 is that it provides a face detection API, which is the Core Image framework. Using this API we can quickly detect the size of the face along with the locations of the mouth and nose in a single Image (video frame) (Dowa 2013). To achieve face detection in real-time video on the iPhone, another framework called AVFoundation should be used. It basically gets video streams from iPhone’s front camera and processes each frame real-timely. To detect faces in each video frame, a class called CIDetector belonging to Core Image is used on the frame to look for features. After that, the class CIFaceFeature should be used to find eye and mouth positions in faces that are detected with CIDetector. By using these position data, we can draw a box on the frame to box up faces that have been detected. In this way, the face detection process can be visualized. Because the Core Image is iOS’s built-in API, it works seamlessly for iOS applications. Some argue that the speed of using Core Image is faster than openCV but the accuracy is lower. In the following section we will do some technical comparisons between these two APIs.

3.2. OpenCv

OpenCv is a library mainly used for real-time image processing. The library is cross-platform. It supports iOS perfectly and has released a framework for iOS environment development. With the framework, I can directly drag-and-drop it into my XCode project. Some pre-configurations such as compiler settings, file suffix
changing are needed because the library is written in C++ rather than Objective-C. Overall, this is a library that very easy to use. In order to compare the performance of OpenCV and Core Image, I find one face tracking iOS app using Core Image framework and implement another face tracking app using OpenCV (Tom 2013). To control variables, the camera captures rates of both two applications have been set to 30FPS. To examine performance in dealing with the three main problems I have found in the existing application on face detection, the comparative trials of face detection library will run in four cases: detection in normal background, detection in dark background, detection in complex background and detection for side face. The screenshot of each application on each case is shown as follows:

**Comparative trial 1:**

<table>
<thead>
<tr>
<th>Core Image Framework</th>
<th>OpenCV Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Core Image" /></td>
<td><img src="image2.png" alt="OpenCV" /></td>
</tr>
</tbody>
</table>

**Figure 3.2.1. Face detection in normal background**

Figure 3.2.1 shows the face detection trial in normal background. From this comparison set, we can see that under normal conditions, both programs produce good results. My face is boxed up properly on both applications.
Comparative trial 2:

<table>
<thead>
<tr>
<th>Core Image Framework</th>
<th>OpenCV Framework</th>
</tr>
</thead>
</table>

Figure 3.2.2. Face detection in dark background

Figure 3.2.2 shows the face detection testing in a dark background. From this comparison set, we can see that in the dark condition, OpenCV performs better than Core Image. Although there is only a vague outline of the face in the view, the powerful OpenCV library can still detect the position of the human face.
### Comparative trial 3:

<table>
<thead>
<tr>
<th>Core Image Framework</th>
<th>OpenCV Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Core Image" /></td>
<td><img src="image2.png" alt="OpenCV" /></td>
</tr>
</tbody>
</table>

Then, we test face detection in complex background (with light source in the background). From this comparison set, we can see that in this complex condition, OpenCV performs better than Core Image. The strong light source in the background interferes with face detection on Core Image, so that my face has not been boxed.

---

**Figure 3.2.3.** Face detection in complex background

---
Comparative trial 4:

<table>
<thead>
<tr>
<th>Core Image Framework</th>
<th>OpenCV Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 3.2.4. Side face detection

Lastly, we test side face detection on both libraries. From this comparison set, we can see that OpenCV can detect my face even if my face is slightly rotated to the side. By contrast, Core Image loses my face in the same condition.

To sum up, comparing to Core Image, OpenCV is significantly better under various conditions. This may because that OpenCV’s face detection classifier has a better classification algorithm. Another possible reason could be the trained data I used for OpenCV is stronger than Core Image’s default one. (For the OpenCV application I implemented, I use haarcascade_frontalface_alt2.xml file as the training set) This is another advantage of the OpenCV library, the developer can choose the training set in use or even create a new training set using their own data.

3.3. CI2CV

CI2CV is a face analysis SDK contains useful components to extract and utilize the features of the face found in video (CI2CV computer vision lab 2013). The Non-rigid Face Registration API of CI2CV can be used in third party C++ applications to process video frames and return 66 landmarks for faces detected in those frames (CI2CV computer vision lab 2013). The API contains three basic methods. The method ‘NewFrame’ performs tracking on a grayscale frame using the tracking parameters ‘params’ and return an integer value between 0 and 10 indicating quality of a frame
The method ‘getShape’ processes on the frame to detect faces and returns 2D facial landmarks to a vector (CI2CV computer vision lab 2013). The method ‘Reset’ resets the tracker when tracking quality is poor (CI2CV computer vision lab 2013). A demonstration program can be found in the SDK’s official web site:

![Figure 3.3.1. CI2CV Face Tracking Demonstration](image)

The SDK is developed based on OpenCV, hence the performance of its face detection should be same as OpenCV. The only difference is that it can detect a face in more detail.
4. Design Choice and Technical Improvements

Since the existing application has its limitations, some work was done to improve it. In this section, I will discuss the design choices made to improve the application, to have a better face tracking performance and introduce some technical improvements that have been done to achieve a better interaction.

4.1. Face Detection Technology

According to the results of comparative trials on three different face tracking libraries, we can conclude that Core Image is not a good option due to its low performance in all experimental conditions. Although it appears the CI2CV API has a lot of interesting features, but those features are not what we actually need. The aim of this project is firstly to improve the face tracking quality of an existing application, and secondly to achieve a new head orientation interaction method of the face-tracking control mode. Since CI2CV is basically an extension of OpenCV, The actual tracking ability makes no differences compared to OpenCV. Moreover, according to the side face detection trial, OpenCV has a very good face inclining tolerance, which is good enough to achieve head orientation. Therefore, I decided to use the OpenCV library to perform real-time video processing and face tracking.

4.2. Drive Method

By using the face detection library, a user’s face in the real-time camera video can be detected and tracked. The application uses the displacement of user’s face from the centre of the view to drive the 3-Dimensional cube displayed on the screen. The face tracking library replacement work has been done based on the knowledge of drive method and drive interface of the cube’s movement in the existing application. To improve the effectiveness, I have also done some improvement work on the drive method. In general, the new application’s driven process can be divided into three steps:

**Step 1: Calculate the relative displacement of user’s face.**
As shown in Figure 4.2.1, we assume that when user faces this face to the camera, the center of detected face coincides with the center of camera view. When user moves his face, a relative displacement between detected face center and default view center occurs. The basic idea of the application is to capture this relative displacement as user’s motion data to drive the 3-Dimensional cube. This calculation is presented in Objective-C code as follows:

```objective-c
self.diff = CGPointMake(self.facePosition.x - self.centerPoint2.x, self.facePosition.y - self.centerPoint2.y);
```
Where `diff` is a `CGPoint` structure type of instance variable indicates the relative displacement; `facePosition` is a `CGPoint` structure type of instance variable indicates the detected face position in the camera view; `centerPoint2` is a `CGPoint` structure type of instance variable indicates the default view center.

As I mentioned in section 3.2, The OpenCv face detection library is tolerant to side face. Based on this feature, we can use the same method to drive the 3-Dimensional cube with head orientation. The Figure 4.2.2 shows relative displacement in a user’s head orientation action. The face box in dotted line indicates the position of user’s head before orientation. From the ‘Current frame’ of the figure, we can see a relative displacement occurs when a user makes a head orientation action. This displacement was used to drive the cube in our application.
**Figure 4.2.2. Relative Displacement in Head Orientation**

**Step 2: Convert the relative displacement to rotation radian.**

Once we get the relative displacement, we want to convert it to a rotation radian. The 3-Dimensional cube will rotate according to this rotation radian. Another important factor should be taken into account is the distance between the user’s face and the camera. For example, if a user holds the device very close to his face, a slightly movement of his head could cause a very large relative displacement. We do not want this to happen. In other words, we do not want the distance to affect the system’s response to a user’s movement action. Hence, a ratio of the relative displacement and the size of detected face is calculated and used to drive the 3-Dimensional cube. This calculation is presented in Objective-C code as follows:

```objective-c
rotX = GLKMathDegreesToRadians(self.diff.y / self.faceSize.height * 20);
rotY = GLKMathDegreesToRadians(self.diff.x / self.faceSize.width * 20);
```

Where `faceSize` is the size of box drawing on the detected face; `GLKMathDegreesToRadians` is a method used to convert an angle measured in degrees to radians.

**Step 3: Drive the cube to rotate.**

In the application, the 3-Dimensional cube was implemented as a floating-point matrix in OpenGL ES 2.0. Mathematical operations are commonly used to manipulate
matrices to represent a graphical transformation (Neider 1997). These transformations can be scaling and rotation. The rotation operation is presented in Objective-C code as follows:

```objective-c
_rotationMatrix = GLKMatrix4Rotate(_rotationMatrix, rotX, xAxis.x, xAxis.y, xAxis.z);
_rotationMatrix = GLKMatrix4Rotate(_rotationMatrix, rotY, yAxis.x, yAxis.y, yAxis.z);
```

Where method ‘GLKMatrix4MakeRotate’ was used to perform a rotation around an arbitrary coordinate vector. It takes the converted rotation radian (rotX, rotY), rotation center (xAxis.x, xAxis.y, xAxis.z, yAxis.x, yAxis.y, yAxis.z), existing matrix(_rotationMatrix) as input, and returns a transformed matrix as output.

By following the aforementioned processes, I replaced the face detection library of existing library with the OpenCV one. From the performance testing, we found that the problem of laggy feeling of the cube rotation still existed. By observing the camera view, we found that this problem was due to lost detection of the user’s face. In other words, the tracking of the user’s face is not continuous due to limitation of the face detection library. The loss of detection causes the loss of relative displacement, thereby causing the 3-Dimensional cube failing to respond in a timely fashion to the user’s motion inputs. To solve this problem, I set the relative displacement diff to a global variable. In this case, when lost tracking happens, the relative displacement of the previous tracking drives the cube to move, creating much smoother and not laggy rotation.

4.3. **Control Method**

Control method is the interaction design of the application which determines in what ways a user could operate the application. A good control method can lead to a good user experience. As mentioned in section 2.2, there were some defects, such as difficulty of stopping the cube from rotating, in the existing application. In this section, I will introduce some improvement work I have performed to improve the application with regard to the user’s control.

4.3.1. **Camera View Panel**

In the existing application, the real-time camera video view is hidden in the background. A smiling face graphic responding to the detected face position is visible for user on the top layer. Because of the increased functionality of the improved application, more details should be drawn into that layer and may affect the user’s ability to see the 3-Dimensional cube on the second layer. Hence, I decided to use a separate camera view panel which is located at the bottom of the view and visible to the user to show the operation status. According to Nielsen’s Usability Heuristics for User Interface Design, visibility of system status is a very important heuristic, which means the system should always keep users informed about what is going on,
through appropriate feedback within reasonable time (Nielsen, 1994). Figure 4.3.1.1 shows the user interface design for face tracking control mode. From the figure, inside the camera view panel, we can see the blue box shows the position of the user’s face; the Red Cross point indicates the center of user’s detected face; the blue circle is the safe-zone (will be explained in section 4.3.2); the blue line is the Scale line (will be explained in section 4.3.4); the blue line connecting from the detected face center point to the initial face center point indicates the relative displacement of the user’s face. These details clearly show the system status and are always visible to user.

Figure 4.3.1.1. User Interface Design for Face Tracking Control Mode

4.3.2. Safe-zone

According to the design, only when the detected face center coincides with the default view center, then the 3-Dimensional cube has a zero speed (See figure 4.2.1). This is difficult for users to apply in practice, thereby resulting in the problem of
difficulty of stopping the cube from rotating. The existing application provided a solution to this problem. The solution is to add a Freeze/Unfreeze switch button that user can press to stop/rerun the cube. In my design, I brought a new concept called ‘Safe-zone’ to solve the problem and to eliminate the need for Freeze/Unfreeze switch button. The idea is to set a threshold value of the relative displacement to one-seventeen of the width of detected face box, the value was chosen based on some initial experiments. The 3-Dimensional cube will have no speed if the relative displacement is lower than the threshold. The code for this is shown as follows:

```python
if (sqrt(pow(self.diff.x,2)+pow(self.diff.y,2)) < self.faceSize.width/17)
{
    rotX=0;
    rotY=0;
}
```

From Figure 4.3.2.1, we can see the Safe-zone threshold is presented as a circle in the Camera View Panel. The screenshot on the left is the state that user’s face center is inside the Safe-zone; hence the 3-Dimensional cube stops. The screenshot on the right is the state that user’s face center is outside the Safe-zone; hence the 3-Dimensional cube starts to rotate.

<table>
<thead>
<tr>
<th>Face center is inside the Safe-zone</th>
<th>Face center is outside the Safe-zone</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Safe-zone Demonstration" /></td>
<td><img src="image" alt="Safe-zone Demonstration" /></td>
</tr>
</tbody>
</table>

Figure 4.3.2.1. Safe-zone Demonstration
4.3.3. Reset Button of View Center

According to the design of the existing application, the default view center was set to the initial center of a user’s face. But in practice, the user’s initial face center is variable and highly based on the user’s device holding posture. It is necessary to add a mechanism to allow the user to reset his preferred initial face center. To achieve this, I added a ‘Reset’ button into the application. The idea is that when a user presses the ‘Reset’ button, the initial face center is set to the current position of the user’s face center. Figure 4.3.2.1 shows the demonstration of this functionality.

<table>
<thead>
<tr>
<th>Initial face center by default</th>
<th>User’s preferred Initial face center</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Initial face center by default" /></td>
<td><img src="image2.jpg" alt="User’s preferred Initial face center" /></td>
</tr>
</tbody>
</table>

**Figure 4.3.3.1. Initial Face Center Resetting Demonstration**

4.3.4. New Scale Method

In the existing application, when a user wants to zoom in the cube, he has to hold the ‘Scale’ button and moves his face toward to the camera. Zooming out is achieved by a reversed operation. This interaction method is achieved by calculating the area difference of detected face box. If the area difference value is positive, the cube scales to a lager shape; if the area difference value is negative, the cube scales to a smaller shape. The method causes a problem in the new application: after the zooming operation, the user’s face center will be no longer being in the Safe-zone. This causes a scaling operation happens along with an unexpected rotation. To solve this problem, a new scale method was designed. In the new scale method, a user’s pitch-up action of his face leads to a zoom in; a user’s pitch-down action of his face.
leads to a zoom out. This new method will not cause any unexpected rotation after a zooming operation is done, because that the human’s head always rotates around human’s neck (as a rotation axis). Hence, user’s face center will always be back to the Safe-zone after a head pitch. This new method is achieved by calculating the height difference between detected face center and the initial face center, where a positive difference value leads to a zoom in and a negative difference value leads to a zoom out. In order to make the control visible to user, a ‘Scale line’ is displayed on the camera view panel when user holds the ‘Scale’ button. The demonstration of this new scale method is shown in Figure 4.3.4.1.

<table>
<thead>
<tr>
<th>Zoom in</th>
<th>Zoom out</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Zoom in" /></td>
<td><img src="image2.png" alt="Zoom out" /></td>
</tr>
</tbody>
</table>

**Figure 4.3.4.1. Cube Scaling Demonstration**

### 4.4. Noise Handling

During the application performance testing, we encountered some unexpected movements of the cube. We found that those unexpected movements are due to noise in face detection. Sometimes, the face detector takes a non-face object as a face, and this affects the user control. Figure 4.4.1 presents two samples of face detection noise.
From the two samples, we can see that the error-detections (noise) are often relatively smaller than normal detections. By using this feature, I can filter out all detected faces which are smaller than a specific size. This specific is determined by the detected face size of a maximum possible distance of a user’s device, by holding the device at a plausible maximum arm length (Shown in Figure 4.4.2).
The filtering process in the code is shown as follows:

```cpp
faceCascade.detectMultiScale(grayscaleFrame, faces, 1.1, 2, HaarOptions, cv::Size(140, 140));
```

Where the `cv::Size(140, 140)` is the specific size mentioned above.
5. Evaluation

Evaluation in the HCI field typically involves asking participants to use the interactive system and gathering feedback from them so that the evaluators can observe the interaction process and measure the outcomes of the interaction (Stevenson 2013). To evaluate the new application I implemented, two types of evaluation have been carried out. Firstly, we ran an interaction experiment with participants and did a quantitative analysis on the experimental data. Secondly, we asked those participants to fill in post-test questionnaires and did a qualitative analysis on their subjective thoughts. This section introduces those two evaluations and their analytical results.

5.1. Quantitative Analysis

The quantitative analysis was carried out using the same mechanism as the previous study which used the existing application. In the previous experimental design, 12 participants were asked to participate in an interaction experiment by using the head tracking application. The designer implemented an independent module for evaluation. In that module the texture of the 3-Dimensional cube was replaced by a new one with small alphabetic letters on it. Participants were asked to follow instructions displayed on the screen and to rotate the cube to locate a specific letter on the cube (Shown in Figure 5.1.1).

![Figure 5.1.1. Evaluation Section](image)
When a participant finds a letter by following the instructions, he has to input the letter into the application and prepare for the next round. The time cost of each round and the correctness of participants’ inputs were recorded. When a trial finishes, a text file recording the result is generated and exported to the computer for analysis. Four crossover trials were carried out with the 12 participants, each trial containing 60 sub-trials. The trials are ARAZ (Accelerometer Rotate Accelerometer Zoom), HRHZ (Head Rotate Head Zoom), ARHZ (Accelerometer Rotate Head Zoom) and HRAZ (Head Rotate Accelerometer Zoom), respectively. Once I got approval from the Human Research Ethics Committee, I ran those 4 trials on a new group of participants following the same procedure. Again, 12 experimental results from the new group were obtained and exported to my computer. In the previous study, experimental results were arranged and analyzed manually. For my study, I wrote a python program to automate this task. The python program is able to read all experimental results exported from the application and generate a data summary. Figure 5.1.2 shows the data summary on the previous experiment’s result.

![Figure 5.1.2. Data Summary of Previous Experiment Results](image)

The first table shows the mean value and the standard deviation of participants’ time.
cost when performing a task on each crossover trial. The second table shows the mean value and the standard deviation of participants’ time cost when performing a task on each crossover trial where errors are not included. The third table shows total errors and error rate on each trial. To demonstrate the new technology for face tracking control is better than the previous one, the python program was run on the new experimental results as well. Figure 5.1.3 shows the data summary on the new experiment results.

Since face tracking control is the part we upgraded, we can compare the result of HRHZ (Head Rotate Head Zoom) between the 2 applications to determine whether the improvements have been successful. By looking at the Error Rate, we can conclude that the new application is better than the old one (0.15 < 0.23). By looking at the mean value of time cost to perform the HRHZ task using the two applications, we can see that the new face tracking control mode costs less time than the old one on average (31.89 < 63.97). In the next step we conducted a hypothesis testing following the formal scientific method to compare the performance of the two face tracking control modes by measuring the time cost to complete a task. We have 2 versions of face tracking control modes, the old version and the new version; we
want to find out if the new version is better than the other. The hypothesis t-testing processes is as follows.

**Step 1:**
Null Hypothesis: There is no significant difference in performance between the old version and the new version.

**Step 2:**
Alternative Hypothesis: There is a significant difference in performance between the old version and the new version.

**Step 3:**
In this step, we used a two-tailed t-test to compare the results. Since python requires an additional package for hypothesis testing and this is difficult to set up, I decided to use R to conduct the testing. Firstly, I exported the data to a csv file containing two columns, being the pairs of time costs for the two control modes for all 60 trials. The file follows the following format:

<table>
<thead>
<tr>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>115.2584</td>
<td>14.11941</td>
</tr>
<tr>
<td>113.0922</td>
<td>7.99026</td>
</tr>
<tr>
<td>34.34363</td>
<td>17.2305</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>24.82874</td>
<td>12.43268</td>
</tr>
<tr>
<td>37.32453</td>
<td>12.66337</td>
</tr>
<tr>
<td>34.39805</td>
<td>18.17058</td>
</tr>
</tbody>
</table>

Then, I used R to perform the hypothesis testing. Since the experiments were applied on two different groups of participations, I used the Independent Samples T-Testing. Figure 5.1.3 shows the result of the testing.
Step 4:
From the result, Probability value (p-value = 4.733e-05) is less than our probability limit (p<0.05), so we reject the Null Hypothesis and accept the alternative hypothesis. This says that, at a confidence level of 95% the true difference in mean time costs (for old version of face tracking control model and the new version) will be in the range 17.23511 and 46.92670. Hence, the new version is scientifically better the old version.

From Figure 5.1.2, we can see there was a different of performance between 4 crossover trials. Compared with the results of the old application (See Figure 5.1.1), the difference seemed not significant. It appears that the performance difference between the accelerometer control mode and the face tracking control mode has been reduced due to the improvements in the new application. To figure out whether one or more crossover trials means differ significantly from the others, we conducted an ANOVA (analysis of variance) by computing an F-statistic. The hypothesis f-testing processes is as follows.

**Step 1:**
Null Hypothesis: There is no difference in performance between four crossover trials.

**Step 2:**
Alternative Hypothesis: One or more crossover trials significantly different from the others.

**Step 3:**

<table>
<thead>
<tr>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>15.09</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>29.28</td>
</tr>
<tr>
<td>Median</td>
<td>44.11</td>
</tr>
<tr>
<td>Mean</td>
<td>63.97</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>97.60</td>
</tr>
<tr>
<td>Max.</td>
<td>312.85</td>
</tr>
</tbody>
</table>

```r
> ExperimentData = read.csv(ExperimentFile)
> summary(ExperimentData)

<table>
<thead>
<tr>
<th></th>
<th>OLD</th>
<th>NEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>15.09</td>
<td>7.99</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>29.28</td>
<td>15.52</td>
</tr>
<tr>
<td>Median</td>
<td>44.11</td>
<td>26.49</td>
</tr>
<tr>
<td>Mean</td>
<td>63.97</td>
<td>31.89</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>97.60</td>
<td>39.83</td>
</tr>
<tr>
<td>Max.</td>
<td>312.85</td>
<td>132.34</td>
</tr>
</tbody>
</table>
```

> t.test(ExperimentData$OLD, ExperimentData$NEW, paired=FALSE)

Welch Two Sample t-test

```
data:  ExperimentData$OLD and ExperimentData$NEW
t = 4.2995, df = 81.097, p-value = 4.733e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:   
17.23511 46.92670
sample estimates:   
mean of x mean of y
 63.97168 31.88977
```

**Figure 5.1.3. Independent Samples T-Testing on HRHZ**
In this step, we used a one-way repeated f-test to compare the results. I found a python script which can compute the f score, so I merged the code into my data analysis program. Figure 5.1.4 shows the result of the testing.

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ___________________________________________ RESTART =___________
>>> ___________________________________________
>>> dic={'9': [], 'F':[], 't':[], 'l':[]}
>>> readAll(dic)
>>> printANOVA()
F_stat 4.3222319266

Figure 5.1.4. One-way Repeated F-test on 4 Trails
```

To get the right F-distribution, we need to know that the DOF (degrees of freedom) are:

\[ k-1 = 3 \quad \# \text{k = number of trails} \]
\[ N-k = 236 \quad \# \text{N = total observations} \]

To get the probability value of an F-test by given the DOF and the F-score, I found a p-Value Calculator online (Soper 2013). Figure 5.1.5 shows the result of probability value.

```

Figure 5.1.5. P-value calculation for F-test on 4 Trails
```

**Step 4:**

From the result, Probability value \( (p-value = 0.0055)\) is in between our probability limit \( (0.05 > p > 0.001)\), then we reject the Null Hypothesis. This says that there were
one or more crossover trials significantly different but not highly significantly different from the others.

By looking at the data summary on the new application (Figure 5.1.2), it appears ARAZ is highly significantly different from the other crossover trials. To test this hypothesis, an f-testing on the other three crossover trials was conducted.

**Step 1:**
Null Hypothesis: There is no difference in performance between HRHZ, ARHZ and HRAZ.

**Step 2:**
Alternative Hypothesis: One or more crossover trials significantly different from the others.

**Step 3:**
Figure 5.16 shows the one-way repeated f-test result on HRHZ, ARHZ and HRAZ.

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> =============== RESTART ===============
>>> dic={'S': [],'F':[],'t':[],'l':[]}
>>> readAll(dic)
>>> printANOVA()
F_stat 2.57274317507
```

**Figure 5.1.6.** One-way Repeated F-test on 3 Trails

To get the right F-distribution, we need to know that the DOF (degrees of freedom) are:

k-1 = 2 # k = number of trails
N-k = 177 # N = total observations

Figure 5.1.7 shows the P-value calculation for F-test on 3 Trails.
Step 4:
From the result, Probability value (p-value = 0.079) is greater than our probability limit (0.05>p), then we accept the Null Hypothesis. This says that there were no significant difference on performance between HRHZ, ARHZ and HRAZ (HRHZ=ARHZ=HRAZ).

To repeat the testing process, we found that the probability value of (ARAZ=ARHZ=HRAZ) is 0.0012 < 0.05; the probability value of (ARAZ=HRHZ=HRAZ) is 0.0045 < 0.05; the probability value of (ARAZ=HRHZ=ARAZ) is 0.15 > 0.05. Hence, we can conclude that ARAZ and HRAZ are significant different from the others where ARAZ performed significantly better and HRZA performed significantly worse. This confirmed the differences of the mean values shown in figure 5.1.2. Hence, we can conclude that any interaction combination which involves face tracing control mode in it reduced the efficiency. In other words, the face tracing control mode I implemented was not good as the accelerometer control mode and needs further enhancements. Nevertheless there was clear improvement. Using the old application the worst performance was from HRHZ whereas with the new application the worst performance was with HRAZ. This suggests that the Head Rotate needs more work than Head Zoom.

5.2. Qualitative Analysis

Qualitative analysis was done by asking the 12 participants to do a post-test questionnaire. The questionnaire consists of three sections. The first section gathers a participant’s personal details including name, age, gender and experience with similar applications etc. The second section gathers the participant’s subjective feelings about the application. In this section, preference questions with numerical Likert scales were used. Each question contains a set of choices for the participant to
The choices are from 1 to 7 indicating the agreed level (1 means strongly disagree and 7 means strongly agree). The third section contains some open-ended questions to gather participants’ comments and suggestions. In this section, I will summarize the survey results and make comparisons between the result on the new application and the result on the old one.

5.2.1. Preference Questions Analysis

In this section of the questionnaire, 11 viewpoints clarifying interactive experiences on the application were stated to each participant to gather the participant’s agreed level on them. The viewpoints are listed as follows.

1. It was natural to use the head-tracking version to rotate the 3D object.
2. It was natural to use the sensor version to view rotate the 3D object.
3. It was easy to learn to use the head-tracking version to view rotate the 3D object.
4. It was easy to learn to use the sensor version to view rotate the 3D object.
5. It was natural to use the head-tracking version to zoom the 3D object.
6. It was natural to use the sensor version to zoom the 3D object.
7. It was easy to learn to use the head-tracking version to zoom the 3D object.
8. It was easy to learn to use the sensor version to zoom the 3D object.
9. It didn’t take long to get used to the using head-tracking version.
10. It didn’t take long to get used to the using the sensor mode version.
11. Overall, I preferred head tracking to sensors.

To summarize the participants’ choices, bar charts were used where the abscissa indicates all questions and the ordinate indicates participants’ agreed level on average. Figure 5.2.1.1 shows the preference summary on the old application. And Figure 5.2.1.2 shows the preference summary on the improved/new application.
Comparing these two preference summaries, we can find that users’ subjective feelings on accelerometer control mode remained basically unchanged. On the other
hand, users’ sense of identity with the new face-tracking control mode significantly improved. From the result, we can say that the effectiveness of the new version of face-tracking control mode was relatively good.

5.2.2. Open-ended Questions Analysis

In the open-ended questions section of the questionnaire, three questions were asked of each participant to gather the participant’s general thoughts and suggestions on the new application. The questions list is as follows.

1. In what way do you feel these interaction methods (head tracking and sensors) either enhanced, or detracted from the performance of the tasks?
2. Do you think these interaction methods could be improved, if so, how?
3. Do you have any other comments about the interaction methods or anything related to them?

To summarize the participants’ answers, users think the accelerometer control mode is easier to learn and use than the face-tracking control mode. But they still found the face-tracking one very interesting. Some improvements / additional functionalities were proposed by users as follows.

1. The idea of face-zone control was good but a freeze button is still needed.
2. It would be better to make the camera view panel smaller.
3. Face-tracking control mode could be further improved.
4. Zooming functionality need to be improved.
5. Further Work

According to the HCI evaluation results, the new application with face-tracking control mode upgraded achieved a lower learning cost and a more precise operation. However, the new head orientation interaction method of the face-tracking control mode still had not reached the same performance as the accelerometer control mode. Our future work will focus on upgrading the head orientation interaction method to enhance the performance. According to users’ feedbacks, the main problem of existing head orientation interaction methods is that the user’s face is forced to stay in the camera's shot and cannot move freely for the entire operation. Any accidental motion will trigger the cube to move. This is because the movement of the cube is driven by a continuous variable which is the relative displacement of the detected face center and view center. To solve this, the drive method should be upgraded to a new version where the displacement detection is replaced with an orientation detection of a user’s head. I propose a solution which could be conducted in further work. Since the orientation detection of user’s head requires more facial details, the CI2CV face detection library mentioned in section 3.3 could be used. Figure 5.1 shows a change of facial landmarks caused by an orientation action.

<table>
<thead>
<tr>
<th>Before Head Orientation</th>
<th>After Head Orientation</th>
</tr>
</thead>
</table>

![Figure 5.1. Change of Facial Landmarks Caused by an Orientation Action](image)

From the figure, we can observe that when an orientation action occurs, the relative distance between the detected landmark in green and the detected landmark in yellow changes. In other words, such distance change indicates an orientation action. Hence, we may use the relative relationship to drive the cube in the proposed application. More specifically, we can set up a threshold value. When the relative distance between the two landmarks mentioned above is larger than the threshold value, a head orientation action is considered to occur. In this case, the orientation motion of user’s head could be detected; moreover, the displacement of the user’s
head will not trigger the cube to move. This is a preliminary idea; the specific driving methods and implementations should be conducted in further work.
6. Conclusion

To sum up, the effectiveness of the face tracking control mode was affected by two main factors: the face detection library in use and the appropriate drive and control method. According to the evaluation, the upgrading work of the existing application in these two aspects succeeded in improving the user experience and effectiveness. However, the face tracking control mode still performed worse than the accelerometer control mode. Using a face detection library, which can detect more facial details, could make a further improvement on the head orientation interaction method of the face-tracking control mode.
References


Neider, Jackie, Tom, D & Mason, W 1997, OpenGL Programming guide, Addison-Wesley


