Low-cost Solutions for Making Hands-free Video Games

GRADUATE PROJECT REPORT

Submitted to the Faculty of
the Department of Computing and Mathematical Sciences
Texas A&M University – Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Long K. Huynh
Fall 2010

Committee Members

Dr. Scott A. King
Committee Chairperson

Dr. Ahmed M. Mahdy
Committee Member

Dr. Michael Scherger
Committee Member
ABSTRACT

The game industry is huge with enormous profits. Currently, about sixty five percent of US households play video games. Gaming keeps becoming more and more advanced with new technologies. However, there are very few games provide support to gamers with disabilities. This lack of support prevents many disabled gamers from being able to play games. With the goals of hands-free, no expensive equipment, and relatively low implementation cost, this project studies the available technologies that can be applied to games to support people who cannot use their hands. The performance in speed and accuracy of the technologies when applied to games are evaluated in this project. Based on the experimented results, this project points out the advantages, disadvantages, problems encountered with the technologies, and proposes solutions to solve those problems. The results of evaluation show that speech recognition and face detection are very promising technologies that have high potential for increasing game accessibility.
# TABLE OF CONTENTS

Abstract .............................................................................................................................. ii

Table of Contents ............................................................................................................ iii

List of figures ..................................................................................................................... vii

1. Introduction and Background ....................................................................................... 1
   1.1 Introduction ........................................................................................................... 1
   1.2 Related Works ....................................................................................................... 2
       1.2.1 Special software or equipment ................................................................. 2
       1.2.2 Specially Designed Games ........................................................................... 5

2. Target Audience ........................................................................................................... 6
   2.1 Disabled gamers – potential and avid consumers ................................................... 6
   2.2 What disabled gamers need and what can be done ................................................. 8
   2.3 The goals of this project ....................................................................................... 9

3. Potential Technologies .................................................................................................. 11
   3.1 Sound based technologies .................................................................................... 11
       3.1.1 Speech recognition ...................................................................................... 11
       3.1.2 Pitch detection ............................................................................................ 12
   3.2 Visual tracking ...................................................................................................... 12
       3.2.1 Face Detection ............................................................................................. 13
       3.2.2 SURF Detection ......................................................................................... 13
       3.2.3 Motion Detection ......................................................................................... 13
3.1.3 Brain-wave technology ............................................................... 13
3.1.4 Using nanos inserted into the body ........................................... 14
4. Speech recognition ........................................................................ 15
   4.1 Introduce Speech Recognition Technology and SAPI .................. 15
      4.1.1 Concept ........................................................................ 15
      4.1.2 Microsoft SAPI ............................................................... 16
   4.2 Evaluation ................................................................................. 18
      4.1.3 Advantages .................................................................... 18
      4.1.3 Disadvantages and problems ........................................... 18
5. Visual Tracking ............................................................................. 21
   5.1 Introduction .............................................................................. 21
   5.2 Introduce OpenCV and Emgu CV ............................................ 22
      5.2.1 OpenCV ........................................................................ 22
      5.2.2 Emgu CV ....................................................................... 22
      5.2.3 Notes ............................................................................. 22
   5.3 Face detection ........................................................................... 23
      5.3.1 Concept ......................................................................... 23
      5.3.2 Implementation Guide ..................................................... 25
      5.3.3 Performance ................................................................... 27
      5.3.4 Advantages ..................................................................... 28
      5.3.5 Disadvantages/problems and proposed solutions ............... 28
   5.4 Speeded Up Robust Features (SURF) detection ......................... 30
5.4.1 Concept .................................................................30
5.4.2 Implementation Guide............................................30
5.4.3 Performance ..........................................................34
5.4.4 Advantages ............................................................35
5.4.5 Disadvantages/problems and proposed solutions..........35
5.5 Motion detection..........................................................36
5.5.1 Concept .................................................................36
5.5.2 Implementation Guide............................................36
5.5.3 Performance ..........................................................38
5.5.4 Advantages ............................................................39
5.5.5 Disadvantages/problems and proposed solutions..........39
5.6 Multithreading.............................................................40
5.6.1 Concept .................................................................40
5.6.2 Implementation guide..............................................41
5.6.3 Performance ..........................................................42
6. OCZ’s Neural Impulse Actuator (NIA)...............................45
6.1 Introduction...............................................................45
6.2 Advantages .............................................................45
6.2.1 Price .................................................................45
6.2.2 Customization and the range of usage........................46
6.2.3 Good support from the vendor and community..............46
6.3 Disadvantages ...........................................................47
LIST OF FIGURES

Figure 1.1. Robert Florio playing Psychonauts Using a Quad Controller ............2
Figure 1.2. Sip & Puff Mouth Joystick Game Controller...........................3
Figure 1.3. Playing Game With Head Tracker........................................4
Figure 2.1. Numbers and percaents on Amputations.................................7
Figure 4.1. Reference of the Speech Library..........................................16
Figure 5.1. Face Detection Speed..........................................................27
Figure 5.2. SURF Detection Speed.........................................................34
Figure 5.3. Motion Detection Speed.........................................................39
Figure 5.4. Performance on the Laptop without Multithreading.................43
Figure 5.5. Performance on the Desktop without Multithreading..............43
Figure 5.6. Performance on the Laptop with Multithreading....................44
Figure 5.7. Performance on the Desktop with Multithreading....................44
1. INTRODUCTION AND BACKGROUND

1.1 Introduction

As the game industry grows, electronic games are becoming more popular and addictive. Many universities and academies have game-oriented studies to teach and research electronic games. Thanks to that, new advanced technologies are rapidly introduced to accelerate the games’ growth even more.

However, while games are growing in depth and complexity, there are few that have enough support for gamers with disabilities. Unlike normal gamers, many people cannot play games properly due to their disabilities such as: blindness, deafness, or mobility limitations. For example, there is no way a person who lost one of his arms can play an action game with a complex control system that requires both hands to control the mouse and the keyboard at the same time. Compared to the mainstream electronic games, those that support accessibility for disabled gamers are very few. Thus, a large number of people are being excluded from gaming due to this lack of support. To make it worse, the difference between regular games and accessible games keeps increasing. For that reason, it is important to study and improve game accessibility.

Currently, there are several technologies that can be applied to games to provide accessibility. This project studied those technologies, analyzed them and proposed solutions that can be applied to games to help gamers who cannot use their hands.
1.2 Related Works

In the past few years, there are several projects developed to provide game accessibilities to physically disabled gamers. They can be categorized into two groups: special software or equipment that can assist with mainstream electronic games and games specially designed for accessibility.

1.2.1 Special Software or Equipment

These are equipment/software that provide support for mainstream games that have no built-in accessibility options. These included mouth-controller, head/eye-tracker, and voice control interactive software.

**Mouth-controller**

The mouth-controller is special equipment that allows the player to use their mouth to play (see Figure 1.1).

![Figure 1.1: Robert Florio playing Psychonauts Using a Quad Controller](image-url)
The equipment shown in Figure 1.2 is called “Sip & Puff Mouth Joystick” and can support PlayStation 2, PlayStation 3, PC USB Computer, Xbox, and GameCube. The controller has one mouth piece (operates as a joystick), 3 holes for "sip & puff" (6 switches) and 4 lip buttons. So, there are a total of 12 buttons (the same amount of buttons on a Playstation® 2 controller).

![Figure 1.2: Sip & Puff Mouth Joystick Game Controller](http://www.broadenedhorizons.com/mouthjoystick.htm)

The Sip & Puff Mouth Joystick Game controller is available on Gimpgear (http://www.broadenedhorizons.com/ mouthjoystick.htm) for $289.
Head/eye-tracker

Head/eye tracker can be either special equipment or software (or a combination of both) to track the movements of the head or the eye of the player and use them to control the game. Figure 1.3 shows an example of gaming with head tracker.

Figure 1.3: Playing Game With Head Tracker
(Source: mms://www.accessibility.nl/games/video/exampleheadtracker.wmv)

Voice control interactive software

These software can map the player voice to certain commands in the games. An example is Say2Play (http://www.gameaccessibility.com/index.php?pagefile=motoric) which can support a wide range of electronic games.

Gamers can download the trial version of Say2Play or buy it for $29.95 on say2play official website (http://www.say2play.com/). Some demos of say2play can be found on YouTube (http://www.youtube.com/results?search_query=say2play). Note that this application is mainly for supporting normal gamers with voice control to ease the
need of using hotkeys, it does not have enough functions to provide a hands-free environment for disabled gamers.

1.2.2 Specially Designed Games

The external special equipment listed above can provide support for mainstream games which originally do not have extended accessibility. Using them, physically disabled gamers can, to some extent, play mainstream games like normal gamers. The downside of those gadgets is that the players have to pay an additional price to have them. Unfortunately, their prices are usually not cheap and not everyone can afford to pay for these gadgets. There is another option that is specially designed games with built-in accessibilities.

Instead of having to buy special equipment/software for accessibility, disabled players can play games with built-in accessibility. These games can be categorized into two main groups: games with limited controls (fewer buttons that can be rearranged to be more accessible. This is for gamers who can still use one of their arms.), and games that can be controlled with other methods instead of using the hands (Examples: built-in voice control or head/eye-tracking). The latter group is the main focus of this project.
2. TARGET AUDIENCES

Disability, according to Americans with Disabilities Act of 1990 (ADA), “is a physical or mental impairment that substantially limits one or more of the major life activities of such individual”. Disabilities are varied, including: visual disability, auditory disability, physical disability, and learning disability. Each type of disability has different characteristics, so it is very hard to have a “one-size-fits-all” solution for all of them. Thus, this project will only focus on people who have a physical disability (mainly the people who lost the control of their hands). From here by, the term “disabled gamers” will especially be directed to people who cannot use their hands.

This section is about whom will be the main users that should be focused on, their potential as gamers/consumers, what their needs are, and what can be done to satisfy those needs. This will base on the author’s own experience as a programmer and a gamer, as well as information provided by other researches.

2.1 Disabled Gamers – Potential and Avid Consumers

To see why disabled people are potential consumers for video games producers, let us take a look at some statistics. According to the Amputee Coalition of America, in the United States alone, there were approximately 1.7 million people with limb loss (excluding fingers and toes) in 2007 [ACA 2010]. Also, according to Kulley, the annual increase of the number of people having upper limb loss is quite high (see Figure 2.1 for more details) [Kulley 2003]. With these numbers are just in USA alone, the number of
people who lost their hands/arms (the target audience) in the world is assumedly very large.

- 50,000 new amputations every year in USA based on information from National Center for Health Statistics
- ratio of upper limb to lower limb amputation is 1:4 (4)
- most common is partial hand amputation with loss of 1 or more fingers, 61,000
- next common is loss of one arm, 25,000
- existence of 350,000 persons with amputations in USA, 30% have upper limb loss
- of this, wrist and hand amputations are estimated to make up 10% of upper limb population
- transradial amputations make up 60% of total wrist and hand amputations
- which means 70% of all persons with upper limb amputations have amputations distal to the elbow(3)
- In US 41,500 persons are registered who had an amputation of hand or complete arm (5)
- 60% of arm amputations are between ages 21 and 64 years and 10% are under 21 years of age (4)

**Figure 2.1: Numbers and Percentages for Amputations**

Moreover, disabled gamers seem to be more devoted to games than many non-disabled gamers. According to the survey conducted by Information Solutions Group on behalf of PopCap, compared to the normal casual gamers, “those with disabilities play more frequently, for more hours per week, and for longer periods of time per gaming session” [PopCap 2008]. Not only spending more time on games than normal casual gamers, disabled gamers also state that they feel significant benefits from playing and view playing games as an important factor in their life [PopCap 2008]. Based on her study about disabled gamers, Kalning stated in her article posted on MSNBC, “For most of us, playing ‘Rock Band,’ or any other video game, is pure fun — a leisure activity. For disabled gamers, playing games can be more than just play. It’s a community. It’s a connection. It’s a life line” [Kalning 2003]. From all of those information, we can see that
the passion for game of the disabled gamers can be compared to that of hard-core gamers (far surpass the passion of non-disabled casual gamers). Currently, there is no accurate statistic about how many of the disabled people actually play game. However, with such large number of disabled people and their positive view on video games (as Kalning said, “For disabled, video games can be a lifesaver” [Kalning 2003]), the number of disabled gamers and the number of disabled people who want to play games are certainly not a small number.

In summary, with their number and the passion they have for games, disabled people are undeniable potential and avid consumers. Game developers who hesitate to invest more in game accessibility because they afraid the market is not big enough should take this into their consideration. The potential of disabled people as consumers is too big to be overlooked.

2.2 What Disabled Gamers Need and What Can Be Done

Because the target audience here is the disabled gamers who lost their hands, they certainly need accessibility support to allow them to control the game without using hands.

There are some disabled gamers who can play games with their feet instead of hands. In the case of those gamers, methods such as making simpler controller (less buttons, better arrangement of buttons) and remapable control (allow user to redefine the control key/button) can help to a certain degree. However, such gamers are very few because it requires a huge amount of practice.
Fortunately, there are current technologies such as speech recognition, head/eye-tracking, etc. that can be applied to replace the mouse, keyboard or controller. With them, it is possible for games to be played without the need of using hands; in other words, “hands-free games”. Because these methods also do not require intense training from the gamers, they can be viable solutions to satisfy the need of disabled gamers.

2.3 The Goals of This Project

Because many of the current specially designed equipment for accessibility are hand-made and most of them are still expensive, not many people can afford them. Thus, this project will focus more on “software-based” solutions and avoid using expensive equipment. Nowadays, a decent webcam with built-in microphone can be bought with just $60 or even less. Also, on the current market, most laptops come with built-in webcam and microphone. This project will mainly focus on finding solutions for hands-free games without the need of any extra devices except those inexpensive equipment. Still, other inexpensive equipment will not entirely be ruled out of this project. Equipment that do not cost more than a hundred of dollars and can provide acceptable performance will also be introduced in this project as alternative options for the disabled gamers to choose from.

We also have to keep in mind that game developers make games mainly for profits; they are not doing charity. A solution that requires a high implementation cost would be simply not viable from the game developer’s point of view. That is why avoiding too much additional work for the game developers is also very important. This
project will only study and introduce technologies that have the potential of being implemented with relatively low cost.

In summary, finding solutions to make hands-free games that do not require expensive equipment with relatively low cost is the main goal of this project. Potential technologies will be evaluated with 3 key factors: hands-free, inexpensive, and low implementation cost.
3. POTENTIAL TECHNOLOGIES

There are several current technologies that can be applied to make hands-free games. This section will briefly discuss the available technologies to find those most suitable. Technologies that require expensive extra devices or have high implementation costs will be deemed unsuitable.

3.1 Sound Based Technologies

This type of technology allows the user to use specific voice commands to control the game. There are two different approaches for this technology. The first approach is speech recognition which uses a list of different phases or words with each mapped to a command in the game. The second approach is using sounds with different magnitudes or frequencies.

3.1.1 Speech recognition

*Speech recognition* is a broad term which can describe either a system that can recognize anybody’s speech (what the user speaks) or a system that is trained to recognize a specific person (who is speaking). *Speech recognition* technology which will be discussed in this paper is the technology to implement a system that can recognize what the user speaks.

With Microsoft Speech Application Programming Interface (SAPI) (see section 4.1.2), this technology is simple to implement with relatively low cost. So, *speech recognition* is a promising technology with SAPI arguably the best way to implement it.
3.1.2 Pitch detection

Instead of using a speech, a player can use a specific pitch to issue a command. In this technique, we will analyze the sound made by a player to get its frequency, magnitude and the length. Then, these characteristics are used to compare and find the matched sound command to determine which command the player want to issue. Due to the complexness of handling sound, for now, this technique will not be discussed in this project. However, it will be included in the future work and will be studied by this project in the near future.

3.2 Visual Tracking

This type of technology will track the user movements and use them to control the game. Since the target users are people who cannot use their arms, tracking the hands or arms is out of the question. Another restriction is the technology must not require expensive equipment. This restriction rules out the possibility of eye-tracking. The reason why is the cameras that are good enough to track the eyes are rather expensive. The only options left are either the body or the head/face. We evaluate using the head/face as the main target of the tracking. The three technologies for tracking that will be discussed are face detection, Speeded Up Robust Features (SURF) detection, and motion detection. All those three technologies are supported by OpenCV (see section 5.12), which is a free library, and are viable for this project’s objectives.
3.2.1 Face detection

This technology detects the user’s face using HaarCascade (see section 5.3.1) and uses its position to determine the movements of the user head/face. Those movements will be used to control the game.

3.2.2 SURF detection

This technology will use the special features of an object, which in this case is the user’s head/face, to detect it. While it seems that this technology is another version of face detection, it does have its own advantages such as being able to distinguish the user’s face from other faces (face recognition).

3.2.3 Motion detection

This technology measures the vector of the object in the field of view, which is the user’s head/face/body, to detect its movement. While the accuracy of this technology is questionable, it has a high potential in speed.

3.3 Brain-wave Technology

Brain-wave technology is a very high level technique that allows the communication between the human brain and external devices. As a natural biologic occurrence, a human brain consistently cycles through several states which are called brainwave states. At any given time, a brain does not operate in only one brainwave state but simultaneously pulses in all states, with one of them being dominant [Immrama 2010]. This technology will capture the waves produced by a brain when it pulses in the brainwave states, then analyze and use them as user input. Because of its high level,
implementing this technology is certainly not a simple task. That is why it would be very hard to find a way to implement this technology with low cost. Another downside of this technology is that it requires a considerable training and customization from the user. Therefore, this project deems that finding a software-based solution for this technology is not viable. For hardware options, there is actually a viable candidate which is the OCZ’s Neural Impulse Actuator (see section 6). This device is inexpensive and can be purchased for under $100. Also, its creator, OCZ, claims that it can be played with all PC games and it can even reduce the reaction time by 50 percent. Based on the price and what OCZ claims it can do, the NIA seems to be a very promising solution for this technology. Whether it can actually live up to the expectation will be discussed in Section 7 – OCZ’s Neural Impulse Actuator.

3.4 Using Nanos Inserted Into the Body

Nanotechnology uses very tiny objects implanted into the body for specific purposes such as diagnosis and treatment. Theoretically, with the diagnosis capability of the nano-devices, this technology can be a great way to provide game accessibility. However, this technology is very costly and certainly is not suitable for this project which aims for low cost solutions. Moreover, while inserting nano-devices into the body for medical purpose is understandable, doubtfully any gamer would want to have them inserted into their bodies so they can play games. In summary, currently, nanotechnology is a very promising technology for medical purpose, but it is not a suitable method for gaming purpose.
4. SPEECH RECOGNITION

This section provides detailed information about the speech recognition technology and how to implement it. There will also be an evaluation of the technology, its advantages, disadvantages and the possible solutions for solving the encountered problems.

4.1 Introduce Speech Recognition Technology and SAPI

This section will introduce the concept of speech recognition, Microsoft SAPI and how to implement speech recognition using SAPI.

4.1.1 Concept

Recognizing speech is actually quite difficult. The main reason is that natural language is very complicated. First of all, the syntax of natural language is much more complicated than programming languages. It has too many exceptions and irregularities. For example, a word can have more than one meaning and can be understood in many ways depending upon the context. This may be easily understood by a person, but a computer is an entirely different case. How could it understand what meaning you want to express? To implement a system that can recognize what is being said with high accuracy is certainly not a simple task.

Fortunately, we do not really want to make the system recognize everything the player says. What we really want is to make the system recognize some, possibly very few, voice commands. Assuming that each button on a controller is mapped to a voice
command, the total number of voice commands would not be too big. Making a system to recognize a command from a small dictionary of voice commands is not that hard. Microsoft provides a Speech Application Programming Interface (SAPI) that has very nice support for this technology.

4.1.2 Microsoft SAPI

SAPI was developed by Microsoft for handling Speech Recognition and Speech Synthesis. For the purpose of this project, Speech Recognition will be the main focus.

Using SAPI

First, we need to add a reference to the speech library. This reference can be found in the first tab of references (.NET) (see figure 4.1). Then, we need to add a using directive using System.Speech.Recognition.

![Figure 4.1: Reference of the Speech Library](image)
After that, we need to build a grammar as following.

```csharp
Choices myChoices = new Choices(commandList.ToArray());
GrammarBuilder builder = new GrammarBuilder(myChoices);
Grammar gram = new Grammar(builder);
```

Now, we have to load the grammar and add an event handler to trigger an event whenever a command is recognized. The `speechReco.Enabled` is used to set the value controlling whether a Speech Recognizer is enabled and running.

`speechReco_SpeechRecognized` is the name of a function that will be called when the event is triggered.

```csharp
speechReco.LoadGrammar(gram);
speechReco.Enabled = true;
speechReco.SpeechRecognized += new
    EventHandler<SpeechRecognizedEventArgs>
    (speechReco_SpeechRecognized);
```

The last thing left to do is defining the `speechReco_SpeechRecognized` function.

With `SpeechRecognizedEventArgs`, we can get the recognized speech command.

```csharp
void speechReco_SpeechRecognized(object sender,
    SpeechRecognizedEventArgs e)
{
    string text = e.Result.Text;
    ...
}
```
4.2 Evaluation

This section will discuss the advantages and disadvantages that speech recognition has.

4.2.1 Advantages

Simple to implement

With the support of Microsoft SAPI, game developers only have to write several lines of code to implement this feature (as seen in the implementation guide).

Fast speed

Speech recognition provided by SAPI is fast. Compared to visual tracking, the slow down it causes to the game is hardly noticeable. With visual tracking, trying to find solutions to improve the speed is a big problem due to the high computation cost. Speed is really a big advantage speech recognition has over visual tracking.

4.2.2 Disadvantages and Problems

The speech recognition surely has its own advantages and is a viable solution for hands-free games. However, it has several disadvantages such as very susceptible to noise, voice distinguish, problems with fast-paced games, and the discomfort that gamers may feel when playing for a long period of time.

Noise

This technology is based on voice which is very susceptible to noise. Noise caused by the environment or a bad microphone will drastically reduce the accuracy of the recognition.
Theoretically, we can solve this problem by using a threshold to filter out sound that has a magnitude lower than the threshold we use. However, it is not easy to do this and finding a way to implement it with low cost is very hard. Another solution, using more than one microphone, is also not viable. First, forcing gamers to buy additional microphones is against the goal of inexpensive that this project aims for. Second, synchronizing the microphones will make the implementation harder. Increasing the implementation cost is against the goal of low implementation cost that this project aims for. In the end, there is currently no good solution for this problem. For now, the simplest and maybe the most viable solution is that the players can try to move the microphone closer to their mouth to reduce noise.

**Speaker recognition**

Similar to noise, when a gamer is playing, if someone nearby says something that includes one of the voice commands, the system will recognize it as a command. This is because the system cannot distinguish the voices, so it cannot know who is talking. Due to the complexity of distinguishing voices, there is currently no good solution to solve this problem.

**Problems with action games**

Action games, which require fast-paced control, will be a hard problem to solve with this technology alone. Using only voice, it is too difficult for the player to issue multiple commands quickly enough due to the nature of talking.
Discomfort

To have to keep talking when playing a game for a long period of time is tiring. The longer the period of time the gamer plays the more discomfort the gamer will feel. Because this is a nature of using voice to play games, there is no solution for this. As the users get used to the feature, they will slightly have less problem with this.
5. VISUAL TRACKING

This section will provide more detailed introduction on visual tracking technologies, what OpenCV and EmguCV are, and how to use them. Three technologies that will be introduced are face detection, SURF detection, and motion detection. There will be information about the concept, implementation, and an evaluation for each technology. The evaluation will use the game Platformer (a starterkit for XNA 3.1). The computers used for testing are a mid-low end Dell laptop (Intel Core 2 Duo 1.40 GHz, 2 GB DDR2, 256 MB GeForce 8400M GS, Windows XP Professional SP3) and a much more powerful desktop (Intel Xeon X5482 x2, 16 GB DDR3, NVIDIA Quadro FX5600 x2, Windows 7 Enterprise). The evaluation will be based on how fast the tracking is (or how much the computation cost for the tracking will slow down the game) and its accuracy. The camera used for testing is a Logitech Webcam 600 (2.0 Megapixel, 720p, up to 30 frames per second). The project will also propose solutions for problems with each technique.

5.1 Introduction

Head/eye-tracking technology is one of the current trends of research for accessibility. This technology uses camera(s) to capture pictures and then analyzes them to track the movements of the head or the eye gaze. Unlike speech-recognition, this technology has fewer problems with the constant movement control. Thus, this technology is more suitable for controlling the constant movements of the characters in a
game than the speech-recognition technology. To implement the tracking technology, this project will use OpenCV and its .NET wrapper Emgu CV.

5.2 Introduce OpenCV and Emgu CV

5.2.1 OpenCV

OpenCV (Open Source Computer Vision), originally developed by Intel and now supported by WillowGarage, is a cross-platform library of programming functions for real time computer vision. OpenCV is released under a BSD license and is free for both academic and commercial use. The current version of OpenCV is 2.1 and it can be downloaded from http://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.1/. An installation guide and more information can be found at http://opencv.willowgarage.com.

5.2.2 Emgu CV

EmguCV is a .NET wrapper that allows OpenCV to be used with .NET compatible languages. The current version of Emgu CV is 2.1 which is now able to support 64-bit windows. An installation guide, examples, and more information can be found at http://www.emgu.com/wiki/index.php/Main_Page. Emgu CV can be downloaded from http://sourceforge.net/projects/emgucv/files/.

5.2.3 Notes

A developer does not need to install OpenCV and EmguCV. To use them, they can just simply copy the needed library files to the execution directory. Those library files are:
For Emgu CV version 2.1 and older: *cvXXX.dll, cvauxXXX.dll, cxcoreXXX.dll, highguiXXX.dll, opencv_ffmpegXXX.dll, mlXXX.dll* and *cvextern.dll* (XXX is the OpenCV version number).

For Emgu CV version 2.2 and newer: *opencv_calib3dXXX.dll, opencv_contribXXX.dll, opencv_coreXXX.dll, opencv_features2dXXX.dll, opencv_highguiXXX.dll, opencv_imgprocXXX.dll, opencv_legacyXXX.dll, opencv_mlXXX.dll, opencv_objectdetectXXX.dll, and opencv_videoXXX.dll* (XXX is the OpenCV version number).

Another note, if the developer is using a 64-bit OS, they have to do either:

- Download the package for 64-bit instead.
- Recompile OpenCV from source for 64-bit platform.
- Force the .NET application to run in 32-bit mode.

### 5.3 Face Detection

#### 5.3.1 Concept

The system will constantly detect the user’s face and tracks the movement direction of the face to determine the movement of the character. In order to detect the user’s face, Haar Cascade Classifiers and Canny Pruning will be used in this project.
**Haar Cascade Classifiers**

The Haar-like features are digital image features used in object detection. They are computed similar to the coefficient in a Haar wavelet transform which was proposed in 1909 by Alfred Haar. The Haar Cascade classifier uses the Haar-like features as the core in object detection. OpenCV provides several classifiers as XML files. Those cascades are not only for face but also for eyes, nose, mouth, body, etc. However, since this project only uses an inexpensive camera for image capture, small parts such as the eyes, nose, and mouth cannot be tracked with the desired accuracy. Based on the result of several experiments, we chose the face as the most suitable object for detection.

**Canny pruning**

This is an edge detection technique developed by John F. Canny in 1986. Canny pruning can significantly reduce the amount of data in an image without losing its structural properties. With many edges around the eyes, nose, mouth, etc. in a picture of human face, Canny pruning can be used to boost the performance of the detection (by eliminating the candidates that do not have enough edges). Canny edge detection is quite old but currently it is the only method that is supported by the enumeration `HAAR_DETECTION_TYPE` (provided by EmguCV). Due to the support that EmguCV provides, developers can implement face detection with Canny pruning with little problems. This is the main reason the project chose Canny as the main edge detection technique. In the future, other edge detection techniques (such as Kalman) will be evaluated.
5.3.2 Implementation Guide

To add the face detection feature into an XNA game, a developer can follow several simple steps.

Add Emgu.CV, Emgu.CV.UI, and Emgu.Util to the references. Also add using directive for them. We also need to add Emgu.CV.Structure (for namespaces like Bgr, Gray, etc.) and Emgu.CV.CvEnum (for the enumeration HAAR_DETECTION_TYPE)

```csharp
using Emgu.CV;
using Emgu.Util;
using Emgu.CV.UI;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;
```

Make sure the Haar cascade file is copied to the execution directory. This project uses the file haarcascade_frontalface_alt2.xml for face detection.

The next step is to declare the variables

```csharp
private Capture cap;  //for capturing image from camera
private HaarCascade haar;  //for object detection
```

Next, we have to initialize the variables. The code can be put in a constructor function (or any function that will be called first).

```csharp
cap = new Capture(0);
haar = new HaarCascade("haarcascade_frontalface_alt2.xml");
```

Add the following code to detect the face. The method cap.QueryFrame() is used to capture the image from the camera or video file as a Bgr image frame. We then store the image in the variable nextFrame as a 8-bit unsigned Bgr color image. To reduce the
computation cost and improve the performance, we convert the image to grayscale and store it in variable `grayframe`. The method `DetectHaarCascade` is used for object detection (the face in this case). More information about this method can be found at http://www.emgu.com/wiki/files/2.0.1.0/html/11c784fc-7d30-a921-07ec-7d217bbe.htm. The next part is to find the biggest face. The reason of doing this is because the `DetectHaarCascade` method will detect all faces appear in the captured image. Since the player’s face is the only one we need to track, we would want to follow the biggest one (usually, it would most likely be the player’s face). After getting the face, we can use it position for the tracking. Depending on the game’s characteristic, there are several ways to handle the position for tracking.

```csharp
using (Image<Bgr, byte> nextFrame = cap.QueryFrame())
{
    if (nextFrame != null)
    {
        Image<Gray, byte> grayframe =
            nextFrame.Convert<Gray, byte>();
        var faces = grayframe.DetectHaarCascade
            (haar, 1.4, 2,
             HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
             new Size(nextFrame.Width / 8,
                      nextFrame.Height / 8))
                        [0];
        if (faces.Length > 0)
        {
            int biggestFace = 0;
            for (int i = 0; i < faces.Length; i++)
            {
                if (faces[i].rect.Height >
                    faces[bigestFaceNum].rect.Height &&
                    faces[i].rect.Width >
                    faces[bigestFaceNum].rect.Width)
                {
                    bigestFaceNum = i;
                }
            }
            ...}
```
5.3.3 Performance

Speed

As seen in Figure 5.1, on an average computer, face detection will slow down games significantly. Powerful computers, however, do not seem to be slowed down by the computation cost of face detection.

![Figure 5.1: Face Detection Speed](image)

Accuracy

The accuracy of the face detection technique depends heavily on how good the chosen Haar Cascade Classifier is. From the experiments, with the classifiers provided by OpenCV, this project has concluded that face detection yield pretty good accuracy (probably the best out of the three discussed techniques). However, the tracking still cannot be as precise as a mouse or a keyboard control (all three techniques suffer this problem). Thus, games that require pinpoint accuracy are not supported well by this technique.
5.3.4 Advantages

Simple to implement

With the support of OpenCV and EmguCV, developers only have to write several lines of code. Note that more work is needed to be done on translating the captured movement to game input in order to have better accuracy.

Has good accuracy

Overall, the face detection has pretty good accuracy (probably the best out of three techniques).

5.3.5 Disadvantages / Problems and Proposed Solutions to Solve Them

Kind of slow

The speed is pretty slow due to high computation require to detect the face. This will be clearly noticeable if the part for tracking is put in a function that is looped constantly like Update.

Proposed solution(s): avoid putting the tracking part in the Update function and use multi-threading (details will be explained in section 5.6)

Cannot distinguish faces

The DetectHaarCascade method detects all faces that appear in the captured image regardless of whose they are. If more than one face is in view of the camera, the system will not be able to tell which one is the player’s face.

Proposed solution(s): only track the biggest face (it will most likely be the player’s face) or combine with SURF detection to recognize special features on the player’s face (this will cause very high computation cost). Theoretically, there is another
solution for this problem that is using special Haar Cascade classifiers (made especially for a player’s face). With these special classifiers, the program will only detect the player’s face, not a face of anyone else. However, this is not viable in reality. Such classifiers must be done by the player because there is no way developers can design them for each player. Including an option for players to create their own classifier is not a good solution. First, the implementation cost for that will be high. Second, making a Haar Cascade Classifier is not a simple task; it requires training with a large number of negative and positive samples to make a good classifier. There are few players willing to spend so much time and effort to make a classifier.

*Cannot be as precisely as mouse or keyboard control*

   Even though the accuracy provides by face detection is good, it is still inferior to that of a mouse or a keyboard. Therefore, this technique would be unsuitable for games that require pinpoint accuracy.

   Proposed solution(s): currently, making the tracking as precise as mouse or keyboard control is not possible (especially when one of the main goals of this project is to use inexpensive equipment and low implementation methods). The proposed solutions here are only for improving the accuracy as much as possible. First, we can try different methods of handling the position of the face to track the movement. Depending on the game, each method has its own advantages/disadvantages. We have to find the best suitable method for the game we want to modify. Another method is to use a threshold value to increase/decrease the sensitive of the tracking. We can also make a tradeoff between the accuracy and the computation cost. Different options can be provided to the
player to let them choose the most suitable one. For example, if the player has a powerful
computer, they can choose an option that has better accuracy but requires higher
computation cost (more hardware demanding).

5.4 Speeded Up Robust Features (SURF) Detection

5.4.1 Concept

The concept of this technique is to detect and extract some special features of an
image (which would be the player’s face), and then use them to compare with another
image to detect if the first image is part of the second image (to get the position of the
player’s face in the captured image). Currently, SURF is an excellent method for the
interest point detection and description task; it outperforms the older schemes Scale-
Invariant Feature Transform (SIFT). Thus, SURF would be the best method for detecting
and extracting features from an image.

Basically, this is another approach for face detection. Instead of using
HaarCascade, we will use SURF. The advantages of using SURF over HaarCascade are
the ability to handle transformation/rotation better and the potential to do face recognition
(will be explained in section 5.4.4).

5.4.2 Implementation Guide

First, we need to add Emgu.CV, Emgu.CV.UI, and Emgu.Util to the references.
Also add using directive for them. We also need to add Emgu.CV.Structure (for
namespaces like Bgr, Gray, etc.) and Emgu.CV.CvEnum (for the enumeration
CAP_PROP which is an enumeration for capture property identifier)
Make sure the model image file is copied to the execution directory (this is the image contain the features that the program will extract from). Because we use the player’s face as the object for detection, the model image should be the picture of the player’s face. This image should have at least background as possible (ideally, this image should only contain the player’s face) to prevent the program from extracting the features of the background (we do not need them for our detection; they are only noise).

Declare the variables. The variable cap is for capturing image from either camera or video file. MCvSURFParams method takes two parameters (double hessianThresh, bool extendedFlag). The number hessianThresh is the threshold number; only features with keypoint.hessian larger than that are extracted (good default value is about 300-500). The extendedFlag is for indicating whether we will use basic descriptor or extended descriptor (we only use the basic one so it will be set as false). The variable modelFeatures will be used to store the key features that will be extracted from the model image. The variable modelImage is for the image that we copied to the execution directory.
Initialize the variables, extract the key features from the model image and store it in the variable `modelFeatures`. Similar to the face detection, we will convert the original image to grayscale to reduce the computation cost. We also use the method `_SmoothGaussian` to “blur” the image and remove the noise (optional). Then, we need to extract and store the key features from the model image.

```csharp
cap = new Capture(0);
cap.SetCaptureProperty(CAP_PROP.CV_CAP_PROP_FRAME_WIDTH, 320);
cap.SetCaptureProperty(CAP_PROP.CV_CAP_PROP_FRAME_HEIGHT, 240);
Bitmap markerBit = new Bitmap("marker.png");
modelImage = new Image<Gray, byte>(markerBit);
modelImage._SmoothGaussian(3, 3, 1.5, 1.5);
modelFeatures = modelImage.ExtractSURF(ref surfParam);
```

Add the following code to detect the face. First, we will use the captured image from the camera and process it (convert to grayscale, remove noise). Then, we also extract the key features from the observed image. After that, we will create a SURF tracker from the key features extract from the model image. The method `MatchFeature` is used for matching the SURF features from the observed image to the features from the model image. This method takes three parameters (`SURFFeature[] observedFeatures, int k, int emax`). The number `k` indicates the number of neighbors to find. The number `emax` is used for indicating the number of leaves to visit (for k-dimentional tree). Now, we will filter the matched features. First, we will eliminate any match that is not unique. This is done by using the method `VoteForUniqueness`. The second parameter of this method is a threshold number which indicate the distance different ratio which a match is considered unique (a good number is 0.8). We also need to eliminate the matched features whose scale and rotation do not agree with the majority's scale and rotation by using the method
VoteForSizeAndOrientation. This method takes three parameters

(SURFTracker.MatchedSURFFeature[] matchedFeatures, double scaleIncrement, int rotationBins). The number rotationBins is the numbers of bins for rotation. The number scaleIncrement is used to determine the different in scale for neighborhood bins (scaleIncrement = x means matched features in bin i+1 is scaled x times larger than matched features in bin i). The method GetHomographyMatrixFromMatchedFeatures is used to recover the homography matrix using RANDSAC (RANdom SAmple Consensus). The method will return null if the matrix cannot be recovered. Then, we will get the region of interest from the model image by getting the four corner points (bottom left, bottom right, top right, and top left). Now, we will get the homography projection of the points. With this homography projection, we can calculate the wanted position of the player’s face.

```csharp
using (Image<Bgr, byte> nextFrame = cap.QueryFrame())
{
    if (nextFrame != null)
    {
        Bitmap camBit = new Bitmap(nextFrame.ToBitmap());
        Image<Gray, Byte> observedImage =
            new Image<Gray, byte>(camBit);
        observedImage._SmoothGaussian(3, 3, 1.5, 1.5);
        SURFFeature[] imageFeatures =
            observedImage.ExtractSURF(ref surfParam);
        SURFTracker tracker = new SURFTracker(modelFeatures);
        SURFTracker.MatchedSURFFeature[] matchedFeatures =
            tracker.MatchFeature(imageFeatures, 2, 20);
        matchedFeatures =
            SURFTracker.VoteForUniqueness(matchedFeatures, 0.8);
        matchedFeatures =
            SURFTracker.VoteForSizeAndOrientation
                (matchedFeatures, 5, 20);
        HomographyMatrix homography =
            SURFTracker.GetHomographyMatrixFromMatchedFeatures
                (matchedFeatures);
        Image<Gray, Byte> res = observedImage;
        if (homography != null)
```
5.4.3 Performance

Speed

As seen in Figure 5.2, this technique has a huge issue with speed due to high computation cost. Even with a powerful computer, games will be slowed down significantly.

![Figure 5.2 SURF Detection Speed](image)

**Figure 5.2 SURF Detection Speed**

Accuracy

SURF detection has low accuracy and cannot keep up with the real time tracking. With low cost camera, the quality of the captured image will not be good enough for the technique to actually recognize the special features we need to track.
5.4.4 Advantages

Better transformation and rotation handling

SURF can handle transformation and rotation better.

Less susceptible to noise

Because SURF uses unique patterns for matching and recognition, it will be less affected by noise compared to other visual tracking techniques. We can further reduce the effect of noise by using markers to make the object (player’s face in this case) more unique.

Potential for face recognition

With HaarCascade, to recognize a specific face, a classifier especially made for that face is required. This is not a simple task (see section 5.3.5). Because SURF will detect and use special features of a face, we only have to allow the player to register his face as the tracked object (which we have to do anyway). Compared with making a special Haar classifier, this is much easier to do.

5.4.5 Disadvantages/Problems and Proposed Solution

Very slow due to high computation cost

As seen in Figure 5.2, speed is a huge problem of SURF. Even with powerful computer, games will be slowed down significantly due to high computation cost SURF requires.

Proposed solution(s): use multi-threading (details will be explained in section 5.6)
Low accuracy in real time

Cannot keep track with the fast movement. This is the biggest problem of SURF detection (even harder to solve than the problem with speed). The only way to improve this is using more expensive camera with faster speed.

Low accuracy due to the quality of captured images

With inexpensive camera, the quality of the captured image will be low. This will greatly affect accuracy (harder for the program to recognize and match the special features).

Proposed solution(s): instead of using more expensive camera, we can use some markers to create more special/unique features.

5.5 Motion Detection

5.5.1 Concept

The concept of this technique is to detect the motion by measuring the vector of the object in the field of view. Instead of having to detect the position of the face and use it to determine the face’s movement, we can directly detect the face movement with the motion detection technique provided by OpenCV.

5.5.2 Implementation Guide

Add Emgu.CV, Emgu.CV.UI, and Emgu.Util to the references. Also add using directive for them. We also need to add Emgu.CV.Structure (for namespaces like Bgr, Gray, etc.) and Emgu.CV.CvEnum (for the enumeration CAP_PROP which is an enumeration for capture property identifier)
using Emgu.CV;
using Emgu.Util;
using Emgu.CV.UI;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;

Declare the variables. The variable *Capture cap* is for capturing image from either camera or video file. *IBGFGDetector* is the interface for background / foreground detector.

```csharp
private Capture capture;
private MotionHistory motionHistory;
private IBGFGDetector<Bgr> forgroundDetector;
```

The next step is to initialize the variables.

```csharp
capture.SetCaptureProperty(CAP_PROP.CV_CAP_PROP_FPS, 30);
capture.SetCaptureProperty(CAP_PROP.CV_CAP_PROP_FRAME_WIDTH, 320);
capture.SetCaptureProperty(CAP_PROP.CV_CAP_PROP_FRAME_HEIGHT, 240);
motionHistory = new MotionHistory(
    1.0, // the duration of motion history you wants to keep
    0.5, // parameter for cvCalcMotionGradient
    0.05); // parameter for cvCalcMotionGradient
```

Add the following code to detect the motion. For the *BG_STAT_TYPE*, we can either use *FGD_STAT_MODEL* or *GAUSSIAN_BG_MODEL*. The reason for finding the maxComp (the biggest component) is because it will more likely be the player than other smaller components. This is for the purpose of reducing noise. With all the information we can get from *MotionInfo*, it would be simple to calculate the movement. The code

```csharp
xDirection = (int)(Math.Cos(angle * (Math.PI / 180.0)) * circleRadius)
```

is for calculating
the horizontal movement. To calculate the vertical movement, we can use \( \sin \) instead of \( \cos \).

```csharp
using (Image<Bgr, Byte> image = capture.QueryFrame())
    //create storage for motion components
using (MemStorage storage = new MemStorage())
{
    if (foregroundDetector == null)
    {
        foregroundDetector = new BGStatModel<Bgr>(image,
            Emgu.CV.CvEnum.BG_STAT_TYPE.FGD_STAT_MODEL);
    }
    foregroundDetector.Update(image);
    //update the motion history
    motionHistory.Update(foregroundDetector.ForgroundMask);
    storage.Clear(); //clear the storage
    Seq<MCvConnectedComp> motionComponents =
        motionHistory.GetMotionComponents(storage);
    double maxArea = 0;
    MCvConnectedComp maxComp = new MCvConnectedComp();
    foreach (MCvConnectedComp comp in motionComponents)
    {
        if (comp.area > maxArea)
        {
            maxArea = comp.area;
            maxComp = comp;
        }
    }
    //Threshold to define a motion area
    double minArea = 500;
    if (maxArea > minArea)
    {
        double angle, motionPixelCount;
        motionHistory.MotionInfo(maxComp.rect, out angle,
            out motionPixelCount);
        float circleRadius = (maxComp.rect.Width +
            maxComp.rect.Height) >> 2;
        int xDirection = (int)(Math.Cos(angle * (Math.PI /
            180.0)) * circleRadius);
        ...
    }
}
```

### 5.5.3 Performance

**Speed**

Figure 5.3 shows the speed achieved with motion detection technique.
Motion detection is very susceptible to noise. This significantly reduces the accuracy. A good way to filter the noise is required to improve the accuracy.

5.5.4 Advantages

Good speed

On average, Motion detection has arguably the fastest speed out of the three techniques.

5.5.5 Disadvantages

Very susceptible to noise

The system will capture any movement of any object in the field of view. This lead to the confusion about which is the movement should be use as the user input. One suggestion to solve this problem is to only track the motion of the biggest object. Hopefully, that biggest object will be the user’s face. However, this will lead to another problem that is when the user stands still and something else in the background moves. In this case, the movement of that object will be track instead.
Proposed solution(s): track only the biggest object and also use a threshold to ignore the small objects.

*Over sensitive*

The tracking is pretty sensitive and it is hard for the user to stand still to stop any movement (even an eye-blink will be captured). A simple solution is to have some thresholds to eliminate those unwanted movement tracking.

### 5.6 Multithreading

Because of the heavy computation cost, all three visual tracking techniques suffer from performance degradation. This project proposes using multi-threading to improve the performance. This section will introduce the concept of the idea using multithreading, implementation guide, the performance, and because this project is focusing on XNA games (especially Windows games), the concept of multithreading will base on the architecture of PCs (with Windows OS).

#### 5.6.1 Concept

Due to the need of constant tracking, the part of code for visual tracking must be put in a section that will be looped constantly. However, placing the tracking part into section such as the *update* method will drastically reduce the game’s speed because of the heavy computation cost. This project proposes the use of a dedicated thread to handle the tracking task. Game developers have to keep in mind that doing multithreading is not a simple task especially with games. Instead of enhancing the game’s performance, doing multithreading incorrectly may slow the game down or even lead to serious problems.
such as deadlocks. It is also much harder to debug the game with multithreading. To keep it simple (as low cost implementation is a goal of this project), we will just make a dedicated thread for tracking and ignore the synchronization (there will also be no blocking or whatsoever). The only one rule that we need to follow is: All variables needed for tracking will be modified by this thread exclusively; all other threads can only read those variables without modifying them. By doing this, the communication between the thread for tracking and other threads is one-way. This will help avoid the conflict between threads when they try to modify the same data at the same time. There is one problem with this proposed implementation. Because we ignore the synchronization, other threads may use out of date output of the tracking thread. This will result in a reduction of accuracy. Fortunately, based on the experiments, this change in accuracy is hardly noticeable while the gain in speed is significant (especially with slow computer). Basically, this is a tradeoff in which we sacrifice some accuracy to gain a big boost in performance.

5.6.2 Implementation Guide

The implementation of the proposed multithreading can be done by doing the following steps

*Add using directive for threading*

```csharp
using System.Threading;
```

*The next step is to declare new variables*

```csharp
Thread camThread; //for a new thread
```
bool endThread = false; // remember to change this variable to // true to stop the thread when the game // ends or when we do not need the // tracking anymore

**Initialization**

trackingThread = new Thread(new ThreadStart(TrackingThreadFunc));
trackingThread.Start();

**Define the function for tracking**

```csharp
void CamThreadFunc()
{
    while (endThread == false)
    {
        ... // code for tracking...
    }
}
```

### 5.6.3 Performance

Figure 5.4 and Figure 5.6 show the speed achieved when apply the visual tracking techniques without using multithreading (on the laptop and the desktop respectively). As shown in Figure 5.4, with average computers, games’ speed will be reduced significantly by the tracking. In Figure 5.5, even though powerful computers seem to not be affected much by face detection and motion detection, they still are slowed down by SURF detection.
With multithreading, the speed is greatly improved (as shown in Figure 5.6 and Figure 5.7). Results from experiments with various games also prove that using multithreading will significantly reduce the slow down caused by the visual tracking.
Figure 5.6: Performance on the Laptop with Multithreading

Figure 5.7: Performance on the Desktop with Multithreading
6. OCZ’S NEURAL IMPULSE ACTUATOR (NIA)

This section will introduce the OCZ’s NIA and discuss about its advantages and disadvantages.

6.1 Introduction

NIA was created by OCZ as a device to be used in conjunction with the mouse to reduce the reaction time in gaming. NIA uses a headband with 3 diamond shaped sensors to track eyes movement, muscle movement and even brain-wave and translate those signals to game input. How the captured signals will be translated to game input will be specified by the “profiles”. NIA allow gamers to customize their own profiles or use the predefined profiles that come with the device. OCZ claims that NIA can be used with any PC game and can cut the reaction time by 50 percent.

6.1 Advantages

6.2.1 Price

Currently NIA is the least expensive out of all brainwave devices that can be used with games. Following is prices of the NIA’s commercial-competitors.

Mindset

This device is made by NeuroSky and costs around $199 (http://store.neurosky.com/products/mindset).
Brainfingers

Brainfingers is a production of Brain Actuated Technologies, Inc. Brainfingers is very expensive with the price of $2,100 (http://www.brainfingers.com/price.htm).

Emotiv EPOC

The price of Emotiv EPOC is $299 (http://www.emotiv.com/store/). Emotiv also has an SDK that allow users to create their own application to use with Emotiv EPOC (the SDK can be bought with around $500).

Others

There are a few other devices that use brainwave technologies such as Star Wars The Force Trainer (made by Uncle Milton) and Mindball EEG. However, they are actually games by themselves. They are not devices that can be applied to other games to support accessibility.


The price of Mindball EEG is $23.900 (http://bio-medical.com/products/mindball.html)

6.2.2 Customization and the Range of Usage

NIA allow users to create their own custom profiles that can be used with any PC game.

6.2.3 Good Support From the Vendor and Community

OCZ has a nice support forum for discussing and helping with the problems consumers have with the NIA (http://www.ocztechnologyforum.com/forum/

6.3 Disadvantages

6.3.1 Not as Easy to Use as Advertised

The NIA actually requires a lot of training from the users to use it. It can take hours just to have a grasp of how to use the device. To master the basic functions of NIA, it may take weeks of training and maybe even months to effectively use it. As Kowalisky said, “The firm later claims players must adjust because the NIA provides shorter reaction times and higher immersion, but my experience showed quite the opposite - even after more than a week of practice, I had to coax the NIA to behave as I intended” [Kowalisky 2008]. For users who want to get into the games immediately without spending time on instruction and pre-training, NIA is certainly not a wise choice.

6.3.2 Comfortability

Since the users are required to have an intense focus to use the NIA accurately (especially users who are new to the device), it would be very tiring to use NIA. In a research about NIA, Lin said, “We also felt that the NIA was quite tiring to use, especially at the beginning. When just starting, we had to make large exaggerated movements to trigger the NIA’s sensors. This quickly tired us out and we felt fatigued after only 20 minutes of gameplay” [Lin 2008]. Another research also addressed this
problem, its author stated, “After 20 minutes of fragging, we needed to take a mental break; while the headset was comfortable to wear, the focus needed to move accurately about in the game was draining” [Wilson 2009]. The intense focus needed to use the NIA will be reduced as the users getting used to the device. Then again, much practicing and training is needed.
7. FUTURE WORKS AND CONCLUSIONS

7.1 Future Works

7.1.1 Sound Based Technologies

Finding a good way to reducing the noise is needed to increase the accuracy of the sound based technologies.

There is also a need to figure out a solution for the problem that sound based technologies have with fast pace games.

As said in section 3.1.1, there is another approach for sound based technology which is using pitch. Instead of using SPAI to recognize the speeches, we can use different pitches as input for game control. In the future, this project will study this approach to see its advantages/disadvantages, and viable methods to apply this technique to games.

7.1.2 Visual Tracking Technologies

The performance in both speed and accuracy will be further improved to provide disabled gamer better gaming experience.

7.1.3 Combining the Technologies

Each technology discussed in this project has its own advantages and disadvantage. With a good combining method, the technologies can complement each other so that they can help solving each other’s disadvantages. This project will further research to figure out viable methods to combine the technologies.
7.1.4 Other Technologies

There may remain other technologies that have the potential for game accessibility that this project has not discovered. Further research is needed to be done for this matter.

7.1.5 Drivers

Currently, in this project, the propose solutions for hands-free game is built-in accessibility. Developers have to include the technologies into their game to support disabled gamers. Another solution is to write drivers that can translate the captured voice/head movement to game input. Instead of having to customize each game to add the accessibility, the drivers can be used for any PC game without the need of modifying the game itself. This project will study this solution to see the advantages/disadvantages it has and make a comparison between driver and built-in accessibility.

7.2 Conclusions

This project studied and introduced the available technologies that can be applied into video games to provide accessibility for people who cannot use their hands. This project evaluated the technologies’ performance base on speed and accuracy. From the evaluation, this project pointed out the advantages, disadvantages, problems that can be encountered when using the technologies and proposed solutions that can be used to solve those problems. From the experiment results, even though the speech recognition and visual tracking technologies (with the face detection stands at the top currently) still have issues with the accuracy and can be a little uncomfortable to use at first, they are still very promising and have great potential for game accessibility. With further
improvement to increase their performance, they can be good solutions for making hands-free games that do not require any expensive equipment with relatively low implementation cost.
BIBLIOGRAPHY AND REFERENCES


This section will provide examples and source codes of the modified games.

1. A Simple Example with Microsoft Speech API and XNA Game Studio 3.1

To demonstrate the application of this technology to game accessibility, the starter kit (a sample game which is provided in XNA by default) Platformer will be used.

In this game, there are only 3 commands: move left, move right, and jump. The file Player.cs will be modified to add voice commands into this game. First, a dictionary needs to be built. In the function LoadContent, add the following code

```csharp
commandList.Add("left");
commandList.Add("right");
commandList.Add("jump");
Choices myChoices = new Choices(commandList.ToArray());
GrammarBuilder builder = new GrammarBuilder(myChoices);
Grammar gram = new Grammar(builder);
speechReco.LoadGrammar(gram);
speechReco.Enabled = true;
speechReco.SpeechRecognized += new EventHandler
    <SpeechRecognizedEventArgs>(speechReco_SpeechRecognized);
```

Now, there are 3 voice commands in the dictionary: “left”, “right”, and “jump”. Every time the player says one of those commands, a Speech Recognition Event will be triggered. A new function `speechReco_SpeechRecognized()` will be added to handle the Speech Recognition Event.
void speechReco_SpeechRecognized(object sender,
    SpeechRecognizedEventArgs e)
{
    string text = e.Result.Text;
    if (e.Result.Text == "jump")
    {
        jumpCommand = "jump";
    }
    else
    {
        moveCommand = e.Result.Text;
    }
}

Now, the only thing left is to add some codes to the function GetInput() to actually handle those voice commands

if (gamePadState.IsButtonDown(Buttons.DPadLeft) ||
    keyboardState.IsKeyDown(Keys.Left) ||
    keyboardState.IsKeyDown(Keys.A) ||
    moveCommand == "left")
{
    movement = -1.0f;
}
else if (gamePadState.IsButtonDown(Buttons.DPadRight) ||
    keyboardState.IsKeyDown(Keys.Right) ||
    keyboardState.IsKeyDown(Keys.D) ||
    moveCommand == "right")
{
    movement = 1.0f;
}
// Check if the player wants to jump.
isJumping =
gamePadState.IsButtonDown(JumpButton) ||
keyboardState.IsKeyDown(Keys.Space) ||
keyboardState.IsKeyDown(Keys.Up) ||
keyboardState.IsKeyDown(Keys.W) ||
jumpCommand == "jump";

That is it, now a player can say “left”, “right”, or “jump” to make the character moving left, right, or jump. The game is now become a hands-freed game that can be controlled by voice without using the hands.
2. Codes for Modifying the Game Platformer to Add Face Detection

We will modify the file player.cs to add face detection to the game. First, we need to add the following variables

```csharp
private Capture cap;
private HaarCascade haar;
private int moveThreshold = 20;
private int jumpThreshold = 10;
private Boolean stopThread = false;
private String moveCommand = "nothing";
private String jumpCommand = "nothing";
private Thread camThread;
```

Then, we have to initialize those variables. The following codes should be put in the Player function

```csharp
if (cap == null)
{
    cap = new Capture();
    haar = new HaarCascade("haarcascade_frontalface_alt2.xml");
}
if (camThread == null)
{
    camThread = new Thread(new ThreadStart(CamThreadFunc));
    camThread.Start();
    stopThread = false;
}
```

The next step is to define the CamThreadFunc function

```csharp
void CamThreadFunc()
{
    while (stopThread == false)
    {
        if (cap != null)
        {
            using (Image<Bgr, byte> nextFrame =
                cap.QueryFrame())
            {
                if (nextFrame != null)
                {
```
Image<Gray, byte> grayframe = nextFrame.Convert<Gray, byte>();
var faces = grayframe.DetectHaarCascade(
    haar, 1.4, 2, HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
    new Size(nextFrame.Width / 8, nextFrame.Height / 8))[0];
int biggestFaceNum = 0;
if (faces.Length > 0)
{
    for (int i = 0; i < faces.Length; i++)
    {
        if (faces[i].rect.Height > faces[biggestFaceNum].rect.Height
        {
            biggestFaceNum = i;
        }
    }
    int capCenterX = cap.Width / 2;
    int capCenterY = cap.Height / 2;
    int faceCenterX = faces[biggestFaceNum].rect.X + faces[biggestFaceNum].rect.Size.Width / 2;
    int faceCenterY = faces[biggestFaceNum].rect.Y + faces[biggestFaceNum].rect.Size.Height / 2;
    if (faceCenterX - capCenterX > moveThreshold)
    {
        moveCommand = "right";
    }
    else if (capCenterX - faceCenterX > moveThreshold)
    {
        moveCommand = "left";
    }
    else
    {
        moveCommand = "nothing";
    }
    if (capCenterY - faceCenterY > jumpThreshold)
    {
        jumpCommand = "jump";
    }
    else
    {
        jumpCommand = "nothing";
    }
}
}
Now, the only thing left is to add some codes to the function `GetInput()` to actually handle the tracking:

```csharp
if (gamePadState.IsButtonDown(Buttons.DPadLeft) ||
    keyboardState.IsKeyDown(Keys.Left) ||
    keyboardState.IsKeyDown(Keys.A)    ||
    moveCommand == "left")
{
    movement = -1.0f;
}
else if (gamePadState.IsButtonDown(Buttons.DPadRight) ||
    keyboardState.IsKeyDown(Keys.Right) ||
    keyboardState.IsKeyDown(Keys.D)     ||
    moveCommand == "right")
{
    movement = 1.0f;
}
// Check if the player wants to jump.
isJumping =
    gamePadState.IsButtonDown(JumpButton) ||
    keyboardState.IsKeyDown(Keys.Space)   ||
    keyboardState.IsKeyDown(Keys.Up)      ||
    keyboardState.IsKeyDown(Keys.W)       ||
    jumpCommand == "jump";
```
APPENDIX B – LINKS

1. **Links for the Information about the Technologies Used in This Project**

   - OpenCV: http://opencv.willowgarage.com/wiki/
   - SURF: http://www.vision.ee.ethz.ch/~surf/

2. **Website of this project**

   Updates of the project include source codes, needed libraries and other information can be found at http://sci.tamucc.edu/~lhuyhn/index.html.