COMP4801 Final Year Project
2018-19
Department of Computer Science,
The University of Hong Kong

Final Report

Project Title: A Motion Tracking Game Based On VR

Supervisor:
Dr. T.W. Chim

Student:
Ng Cheuk Hung (3035281093)

Date of Submission: 14/04/2019
Abstract

Virtual reality with motion tracking technology has been utilized in different industries around the world recently, from engineering simulation to gaming industry. Retrieved motion can be reconstructed inside the virtual world built by the software. It can then be used for simulating the interaction between users and computer-generated virtual world.

However, the rudimentary implementation of this simulated interaction in video games can be improved further to elevate the whole gaming experience. In this paper, a new model named motion-driven player-game interaction is introduced as a conceptual solution to improve current virtual reality games with motion tracking devices. A detailed motion pattern matching algorithm is designed and a demonstrative virtual reality game is developed at the end of the project. HTC VIVE is chosen as the core virtual reality and motion tracking device in this project. The game is built in Unity Engine with the HTC VIVE.

The finished motion pattern algorithm can successfully recognize most of the motion and match them with predefined motion patterns. Therefore, the project makes contributions on solving the shortage of input methods of hand controller in virtual reality system, such as HTC VIVE. Users are no longer limited to the control the game behaviours using the buttons on the hand controller since they can now also control the game behaviours by their hand motion. Currently, the motion pattern algorithm can only be used by hand controller, but not full-body motion tracking system. It is hoped that the algorithm can be further improved to allow the implementation of those complicated systems in the foreseeable future.
Acknowledgement

I would like to express my deepest appreciation to those who have offer their help to this project. I would like to specifically thank my supervisor, Dr. T. W. Chim for his advices and support throughout the project.

I would also like to express my gratitude towards my friend, Mr. Dicky Tang, who helped me to test the game and algorithm during the development. I would also like to express my gratitude towards the Department of Computer Science, Faculty of Engineering, The University of Hong Kong for providing the equipments needed for this project.
# Contents

Abstract 2  
Acknowledgement 3  
List of Figures 6  
Abbreviations 7  
1 Introduction 8  
2 Background 10  
  2.1 Motivation 10  
  2.2 Motion-Driven Player-Game Interaction 11  
  2.3 Definition of Motion 12  
3 Objective 15  
4 Scope 17  
5 Deliverables 19  
6 Methodology 20  
  6.1 Development Tools 20  
    6.1.1 Virtual Reality System and Motion Tracking Devices 20  
    6.1.2 Game Engine 20  
    6.1.3 Software Development Kit 21  
    6.1.4 Agile Methodology 21  
  6.2 Algorithm Design and Implementation 22  
    6.2.1 Design on Motion Tracking Algorithm 22  
    6.2.2 Point Motion Tracking Process 23  
    6.2.3 Feature Extraction Process 24  
    6.2.4 Pattern Recognition Process 29  
    6.2.5 Thresholding Process 33  
    6.2.6 Clearing Linked List of Motion Sequence 34  
    6.2.7 Algorithm Analysis 34  
  6.3 Game Design 35  
    6.3.1 Overview 35  
    6.3.2 Scene Design 36  
    6.3.3 Enemy AI Design 37  
    6.3.4 Motion-Driven Events Design 38  

7 Discussion of Results 39
  7.1 Game Implementation 39
    7.1.1 Game Map 39
    7.1.2 Enemy AI Implementation 40
    7.1.3 Motion-Driven Events Implementation 41
  7.2 Algorithm Testing 43
    7.2.1 Reliability Test 43
    7.2.2 Thresholding Test 44
8 Future Plan 46
9 Conclusion 47
References 48
Appendix I - Project Schedule in Project Plan 49
Appendix II - Hyperlink to Game Walkthrough video on Youtube 50
List of Figures

1  Screenshot of player before turning over his hand
2  Screenshot of player after turning over his hand
3  A graph representing the domain of motion
4  A graphical representation of point motion
5  A flow chart of the motion tracking algorithm
6  A sequence of points on a two-dimensional coordinate system
7  A 8*8*8 matrix in 2D to represent the space around the player
8  A graph to show how to calculate $\phi$
9  A linked list of matrix elements representing the motion
10 A graphical illustration of the motion represented in Figure 9
11 Matching between pattern node and tracked node on tolerance level $= 1$
12 Matching between pattern node and tracked node on tolerance level $= 2$
13 A case of mismatching due to isolated node
14 The tutorial scene of Virtual Fighter
15 The main story scene of Virtual Fighter
16 The boss fight scene of Virtual Fighter
17 A image of the script of IState.cs
18 The game map of tutorial scene
19 The game map of boss fight scene
20 The game map of main story scene
21 A graph of enemy chasing the player
22 A graph of enemy attacking the player
23 An image of player perform motion pattern 2 to shoot fireball
24 An image of player perform motion pattern 3 to cast the ultimate attack
25 The box graph of the mismatching score distribution
Abbreviations

VR : Virtual Reality
AI : Artificial Intelligence
2D : Two-dimensional
3D : Three-dimensional
1 Introduction

Virtual reality games have recently become more popular in the gaming industry. Steuer’s article pointed out that virtual reality is a technology to provide user an environment for telepresence experience [1]. Different game development companies and studios have introduced motion tracking technology into their virtual reality games and this has been a hot topic in the gaming industry. The journal article from Rosenberg and Page described motion tracking device as a type of computer input devices for converting human motion into data which can be interpreted by computers [2]. With the use of motion tracking devices, virtual reality game can bring evolution to the video game industry.

Currently, most of the virtual reality games make use of the motion tracking technology to solely capture the human motion and reproduce the body movements on in-game characters. Therefore, this project aims to expand the usage of motion tracking technology on virtual reality games besides the reconstruction of player’s body movements inside the game. It is known that there is a shortage of input methods of many virtual reality systems. For instance, there are few different buttons on the hand controller of HTC VIVE and Oculus Rift, but the amount of buttons is relatively low compared to keyboard and mouse. It is suggested by the project that the motion tracking technology can contribute on solving this problem by adding motion as an input method. There are a lot of information the game can retrieved from the body movements. Body motion can be interpreted as a sequence of human body postures to complete a set of particular tasks [2]. The motion sequences and patterns of the players can be used as a type of data input to control different elements and accomplish different tasks inside the game. As a result, the interaction between the players and the games is enhanced by the enlarged and improved implementation of motion tracking technology on the virtual reality games. Players can then have a better control on different game objects in the virtual reality world with the enhanced player-game interactions.

A motion tracking game based on virtual reality is delivered in the final stage of this project in order to demonstrate how to utilize motion tracking devices in virtual
reality game by interpreting the motion sequences and patterns to greatly improve the player-game interactions. This concept of player-game interaction based on motion tracking will be named as motion-driven player-game interaction in this project. The game is called Virtual Fighter. It is fighting game which users need to use different hand motion to trigger different abilities to defeat their enemies. An efficient algorithm for transforming body motion into in-game commands is introduced as well in this project. The motion pattern recognition algorithm is designed for distinguishing different body movements given a set of predefined motion patterns in the database.

The remainder of this paper proceeds as follow. In chapter 2, the background of the motion tracking algorithm and the motivation of this project will be introduced. In chapter 3, the objectives of the project will be described clearly. The project scope will be included in chapter 4. It will give some information on the project focus and define the works to be done. Chapter 5 of this paper will describe the methodologies of the project, from the game engine to the motion tracking devices. After that, chapter 6 will give detailed explanation on current findings on the motion tracking algorithm. The limitation of the designed algorithm and the remaining challenges of this project will be included in chapter 7. The paper will end with a short conclusion in chapter 8.
2 Background

In this section, the background of motion tracking game based on virtual reality and the motivation of this project will be explained in detail. A clear introduction to the concept of motion-driven player-game interaction with examples will be included. The definition of motion is also included to clarify the types of motion being handled in this project.

2.1 Motivation

Virtual reality technology in gaming industry brings a new level of experiences to gamers. One of the main goals of video games is to provide new experience to the players. Virtual reality can change users’ perception to their surrounding environment [1]. While the players enjoy their new experiences inside the virtual world simulated by the virtual reality games, there are needs of interaction between the players and the game objects. Motion tracking devices is then introduced as a method for players to interact with the game.

As stated in the article from Smith, a modern virtual reality system contains a game station composed by input and output devices, including motion tracking devices for sending output signals by detecting the movements of the user [3]. Most of these motion tracking devices can precisely capture the motion of the users and send the corresponding information to the computer. The game can then interpret the received signal and simulate the interaction between the player and related game objects. Therefore, players can interact with the game with their body movements now while players in the past can only interact with the game with traditional computer input devices, like keyboard, mouse and console controller.

Motion tracking games based on virtual reality have become the spotlight among the gaming industry in the past few years. All the top-tier virtual reality games in the Best of 2017 Virtual Reality Rewards from Steam make use of the motion tracking controller [4]. Steam is one of the biggest digital distribution platform for video games in the world. Its statistical data and attention on virtual reality games shows that the market of virtual reality games has grown quickly in recent years and game
development professionals start using motion tracking technology in their virtual reality games.

2.2 Motion-Driven Player-Game Interaction

However, most of these virtual reality games solely utilize the motion tracking controller to capture the hand movements and reconstruct the motions inside the game. This project suggests that more advanced utilization of motion tracking devices can be used in the game implementation for better player-game interaction improvement. Player should be able to interact with the user interface or even control different game objects through body motions.

![Figure 1: A screenshot of player before turning over his left hand](image1.png)

![Figure 2: A screenshot of player after turning over his left hand to open a menu](image2.png)

There are some examples to clearly explain the suggested advanced implementation of motion tracking devices and the concept of motion-driven player-game interaction. Leap motion is a company producing hand motion tracking devices in the gaming market. They published some videos showcasing their hand motion tracking devices.
(See Figure 1 and 2). In the selected figures, player can turn over his left hand to open a menu. This is an example of utilizing player motion to interact with the user interface. The player may then turn over his right hand for other game events which depends on the actual game implementation. Player motion becomes the “language” between players and the game. The game can interpret the player motion and understand what game events should be raised correspondingly. This paper describes this type of player-game interaction as motion-driven player-game interaction. Its name implies that this type of human-machine interaction starts from the user motion as the input.

2.3 Definition of Motion

![Figure 3: A graph representing the domain of motion](image)

In order to understand the algorithm design and implementation of the later parts, there is a need to define motion clearly. Motion means the positional and orientational change of an object within an given interval of time. This general definition is not enough to define the representation of motion in computer program. A more clear definition on motion is needed for this project to design the representation of motion in the algorithm and computer program.
There are many types of motion in the reality world. First of all, only human motion is interested in this project. Only the motion of the player is considered in the model of motion-driven player-game interaction. However, there are still several types of human motion (See Figure 3). The most complicated human motion is full-body motion. It represents the movements of every parts of the body. There is also hand motion which only represents the movements of different parts of the hand. It would be great if the project can develop an algorithm and build a game to support the motion pattern detection of all these types of human motion.

However, they are not handled in this project because of the hardware limitation. It is because HTC VIVE is chosen as the virtual reality system in this project and it can only track the movement of the hand controllers. This type of motion is called point motion, which the basis of all other types of motion. The position of an object in any given time is represented as a point in a coordinate system and its movement is illustrated as the point-to-point transformation (See Figure 4). All the motion discussed in this project will only be point motion. In this paper, point also means the position of the HTC VIVE controller in a three-dimensional coordinate system with a given time.

In short, the motivation of this project is to improve the current motion tracking technology implementation on virtual reality games based on the concept of motion-driven player-game interaction. The project aims to design an efficient algorithm for
implementing this motion-driven player-game interaction on virtual reality games. The motion to be handled in this project is point motion. The design and implementation of the motion pattern matching algorithm is performed based on the concept of motion-driven player-game interaction and point motion.
3 Objective

The following section describes the objectives of this project clearly. There are two main objectives in the project, namely the algorithm design and the game demonstration.

This project ultimately aims to enhance the implementation of motion tracking devices on virtual reality games to provide a better player-game interaction. This is because most of the current implementation of motion tracking technology on games based on virtual reality is too simple and straightforward. For instance, hand movements captured by the hand motion tracking devices are usually merely used to reproduce the tracked motions on the hands of an in-game character. However, more advanced utilization of the body motion data can be implemented to the game, such as triggering different in-game commands and effects through different body motion sequences and patterns. This can also help to tackle the shortage of input methods of virtual reality system. The amount of input methods is relatively low compared to that of traditional computer systems, such as mouse and keyboard. After implementing the model of motion-driven player-game interaction, motion of the player can be used as an new input method to control the game behaviours.

Therefore, there are several sub-tasks to be completed in this project. Firstly, an efficient motion pattern matching algorithm will be introduced for matching tracked motion with predefined motion patterns. Then, a motion tracking game based on virtual reality will be made to demonstrate how virtual reality games can make use of those useful data information generated by motion tracking devices on different game elements to provide a better player-game interaction. For example, drawing a circle with the motion tracking device can open a setting menu while most of the current virtual reality games require players to stare at the setting menu icon for a few seconds to open the menu.

In order to demonstrate how the motion tracking technology can be used on video games, the game product will be a virtual-reality fighting game. The motion of the player will be used on weapon control. Different specific motion sequences will be
used to trigger different abilities to fight against the enemies. For instance, when a player pretends to throw something using hand, the game can detect the motion and correspondingly shooting a fireball according to the direction which the player is facing.
4 Scope

In this section, the scope of this project will be described. This part of the paper will define clearly the focus of this project. Some out-of-scope tasks are listed as examples to discuss the project scope.

The aim of this project is to study how to use motion pattern matching to facilitate advanced implementation of motion tracking devices in a virtual reality game for improving the current player-game interaction. Therefore, the algorithm study is focused on how to transform the fast-changing positional data to simple abstract movement sequences. For instance, there are hundreds of evenly distributed points (as defined above, point means the position of controller in a three-dimensional system in this paper) starting from \((0, 0, 0)\) to endpoint \((0, 0, 1)\), i.e. \((0, 0, 0) \rightarrow (0, 0, 0.01) \rightarrow \ldots \rightarrow (0, 0, 0.99) \rightarrow (0, 0, 1)\). This point-to-point transformation can be simplified by grouping them as \((0, 0, 0) \rightarrow (0, 0, 1)\). Both of them represent the same motion pattern, but the latter representation has much smaller size. Then these motion sequences are used to match with predefined motion patterns and trigger corresponding predefined in-game commands. The next focus is hence how to correctly compare the simplified motion sequences with the predefined motion patterns. These are the two main focus of the algorithm design of this project.

Apart from the algorithm study, a game called Virtual Fighter will be made as a practical application to illustrate the whole process of motion-driven player-game interaction. The generalized process is proposed to start from motion capturing with motion tracking devices to interpretation and simplification of the input motion data, then to pattern matching with predefined motion patterns in the database, and finally to trigger customized in-game events. Therefore, the focus of this game product is to implement this model of motion-driven player-game interaction with the designed motion pattern matching algorithm.

The algorithm study does not aim to improve motion tracking devices but to propose a model to utilize the captured data efficiently on a virtual reality game in a larger extent. The project will not solve the problem of precision and accuracy of the motion.
tracking devices and presume that the motion tracking device is capable of tracking
the movement of hand controller precisely and accurately. The details of motion
capturing is also not interested in this project since it is the basic implementation of
motion tracking devices. The project is about the advanced implementation of motion
tracking technology on how to make use of the captured motion for motion pattern
matching.

Although Virtual Fighter is delivered in the final stage of the project to demonstrate
the proposed motion-driven player-game interaction implementation, it will not be
comprehensive enough to be comparable with the commercial games. The delivered
game is supposed to showcase the advanced implementation of motion tracking
technology on virtual reality games and its effects on raising the game enjoyability,
therefore the other elements of the delivered game, such as storyline and aesthetics,
are not weighed as much as the aspects of technology and mechanism.
5 Deliverables

In this section of the paper, all the finished deliverables of the project are listed.

- Model of motion-driven player-game interaction
- Design of a motion tracking algorithm for point motion, which can be further divided into a motion feature extraction algorithm and a motion pattern matching algorithm
- Development of a virtual-reality motion tracking fighting game, which the core mechanism is based on the model of motion-driven player-game interaction
- Recording of a video on playing and testing the game (See Appendix II)
- Writing this paper to explain all details of the project

There will be one more deliverable by the end of the project, which is listed as below.

- Poster design for the poster exhibition
6 Methodology

In this section, the methodology in this project is discussed. Different tools used to implement the project, including the virtual reality system and game engine for implementing the virtual reality game, will be introduced in the following part. The motion tracking algorithm and its implementation will be explained in details in the next part. Moreover, the overview of the game is included. The design of the artificial intelligence (AI) in the game will also be discussed.

6.1 Development Tools

In this subsection, the tools to develop the game is introduced. Reasons of choosing them over their counterparts are given for references.

6.1.1 Virtual Reality System and Motion Tracking Devices

The virtual reality system chosen for this project is HTC VIVE. HTC VIVE is a virtual reality system developed by HTC and Valve Corporation. It consists of a headset, two hand controllers and lighthouses as the sensors. The controller can be served as a simple point motion tracking device. It can detect the position of the player’s hand indirectly when the player is holding it. The reason why it is chosen for this project is that HTC VIVE has a high accuracy on position and orientation calculation and its end-to-end latency is only 22ms [5]. It is also well supported by the game engine and software development kit to be discussed below. Therefore, the development of the delivered game with HTC VIVE is more convenient and has less development risks.

6.1.2 Game Engine

The game engine used to implement the delivered game is Unity Engine. Unity Engine is a one of the most popular game engine for creating three-dimensional games. It provides a set of scripting application programming interfaces in C# for developers to implement custom game logic. Unity Engine has a great support on virtual reality game development and development with HTC VIVE. It provides an input system for mapping the input from motion tracking devices. Moreover, C# is an object-oriented programming language, so it is easier to implement game logic since it treats the elements inside the game as an object. The interaction between the objects
can be modified to illustrate the motion-driven player-game interaction. Therefore, Unity Engine is chosen for this project.

6.1.3 Software Development Kit
SteamVR is chosen as the software development kit and the run-time environment of the game when implementing the delivered game. SteamVR consists of OpenVR, a set of application programming interfaces for the virtual reality game development with the feature of independent update for supporting hardware updates according to its documentation [6], and a run-time environment for virtual reality games on PC. SteamVR is required as a way for mapping device input to game-usable data in Unity Engine when the game is developed with HTC VIVE. Therefore, this software development kit is included in the development tools in this project.

6.1.4 Agile Methodology
This project follows the agile methodology. In order to perform the iterative development, preliminary games were development after the algorithm design to test the reliability of the motion tracking algorithm and the feasibility of motion-driven player-game interaction. After the successful build of preliminary games, the official game were developed on top of the preliminary games by adding more game elements and more advanced game logic implementation. For each game builds above, the preliminary game were tested by some voluntary gamers for the feedback on the enjoyability of motion-driven player-game interaction and the stability of the game and the motion tracking algorithm implementation. The game was then further improved and enhanced according to the feedback. Therefore, the project quality is ensured by following the agile methodology.
6.2 Algorithm Design and Implementation

In this subsection, the algorithm design and its detailed implementation in this project are explained in detail.

6.2.1 Design on Motion Tracking Algorithm

![Flow chart of the motion tracking algorithm](image)

**Figure 5:** A flow chart of the motion tracking algorithm

In order to bring the motion-driven player-game interaction to the virtual reality game, a motion tracking and pattern matching pipeline is designed (See Figure 5). The pipeline consists of four main parts, namely the point motion tracking process, feature extraction process, pattern recognition process and the thresholding process.

Implementing the motion tracking mechanism using this pipeline can allow the game to distinguish different motion sequences and map them to predefined motion patterns if there is enough similarity between the motion sequence and the predefined motion pattern. A corresponding game event will be raised after each successful matching. The flow of pipeline will be completed within a frame in Unity Engine, which is about 0.03 second, and then repeat again. Therefore, an efficient and fast algorithm is very important for this pipeline.

In short, the program measures the motion of the controllers and updates their position in the game. After that, the feature of the positional data is extracted and added to a
list of positional data representing the point motion in the previous time intervals. Then, the motion sequences are compared with all the predefined motion patterns in the database. A mismatching score is given for each comparison. The name gives out its meaning that the score is indicating how similar are the motion sequence and the given motion pattern. If the mismatching score is zero, then the motion sequence is said to be a perfect match with the given motion pattern. For the most perfectly matched motion pattern among the database, if the mismatching score is lower than the threshold value, then the motion sequence is successfully matched with that pattern and therefore the corresponding game events will be raised. This is the basic flow of logic for the motion tracking and pattern matching pipeline. There are much more to explain on each process in order to understand the actual implementation. Each process of the algorithm will be explained in detail in the following part.

6.2.2 Point Motion Tracking Process
The starting point of the algorithm is to measure the positional change of the controller, which also represents the positional change of the player’s hand. Motion can be described as a sequence of changing position. However, a sequence of points in a three-dimensional coordinate system is a more suitable way to represent motion in this case according to the definition of point motion discussed before. In the Unity Engine, position of the controller is stored as a transform object which contains the three-dimensional coordinates. This process can be done by the API provided by SteamVR. The API will convert the data from HTC VIVE controller to three-dimensional coordinates.

However, this is not the end of the process after getting the new position of the controller. For each frame, the position of the controller is updated and this forms a point-to-point sequence to represents the motion. The coordinate values of these points are world coordinates in the Unity Engine. Consider a motion sequence which the player lift his hand up while standing on the origin of the world coordinate system, \((1, 0, 0) \rightarrow (1, 1, 0)\) is a fairly good point motion sequence to represent this motion (Y-axis is the upward direction in Unity Engine). The player now rotates ninety degree clockwise and lift up his hand again. The motion sequence becomes \((0, 0, 1) \rightarrow (0, 1, 1)\). The player move forward by 2 units along the Z-axis in world coordinate
system and lift his hand up again. The motion sequence is tracked as $(0, 0, 3) \rightarrow (0, 1, 3)$. It is shown that the tracked motion sequence will vary a lot if the coordinate values are taken based on the world coordinate system. The motion sequences should be the same no matter which direction the player is facing or which location the player is standing. The same motion of the controller (player’s hand) should not result in two different motion sequence.

The solution is not complicated. Instead of taking the coordinate values based on the world coordinate, the coordinate values should be taken based on the position and rotation of the player. This means the points becomes the local coordinates in a local coordinate system which the player is always located at the origin $(0, 0, 0)$. In this case, the motion of lifting hand is always from $(1, 0, 0)$ to $(1, 1, 0)$ no matter how the player moves around the playground. In order to implement this function, the position of the headset is used as the position of the player. This means the position of the headset is always $(0, 0, 0)$ on the local coordinate system in the program.

After getting the local coordinate of the controller with respect to the player (headset), the positional data is ready for the next process in the motion tracking algorithm.

### 6.2.3 Feature Extraction Process

![Figure 6: A sequence of points on a two-dimensional coordinate system](image)

The next part, which is also one of the most important part of the algorithm, is how to extract the feature of the motion and stored them in a form that can be used for pattern
matching in the later stage. The naive solution is to directly use the local coordinates obtained from the point motion tracking process. The program can store a sequence of points in the local coordinate system in the past few seconds (See Figure 6). However, this method has several problems.

The above process will be repeated for every single frame. In Unity Engine, a frame lasts for about 0.03 seconds. Each frame generated a point from the point motion tracking process. Therefore, the distance between points and points is very close. There are other processes in the algorithm which are time consuming, such as the pattern recognition process. The ideal solution is that the feature extraction process can filter some points out by “grouping” some close-distance points into a single point. The pattern recognition process is then performed for each group of point rather than being performed for every single point generated. This is because if the motion sequence failed to match with any predefined motion pattern, adding a point very close to its endpoint to the motion sequence is very likely to fail the pattern matching again. This can greatly decrease the number of pattern recognitions per second while keeping the feature of the motion sequence without losing too much information. The size of the motion sequence is also greatly decreased by grouping the points. Therefore, one of the goals of feature extraction is to improve the time and space efficiency of the algorithm by filtering out the unnecessary points from the motion sequence.

Another problem is that human motion is imperfect, therefore using exact position of the controller will make it difficult to match with predefined motion patterns because it is guarantee there will be slight positional differences between the tracked points from controller and the point in the predefined motion patterns. The speed of the controller may also result in large positional differences for some points in the motion sequences. For instance, player A and B want to make the lift their hand as much as possible. The ideal motion sequence should be (1, 0, 0) → (1, 2.5, 0) → (1, 5, 0). Player A moves the controller faster and therefore get the motion sequence (1, 0, 0) → (1, 3, 0) → (1, 5, 0) while player B moves slower and perform a motion (1, 0, 0) → (1, 1, 0) → (1, 5, 0). Both of them are moving the same direction but the speed of their movement results in the difference of the second points. It is very hard to define
a set of criteria to compare the similarity between motion sequence and the predefined motion pattern or other motion sequences using the exact position of points.

From the previous method of taking the exact position of points for comparison, there are two properties can be utilized to design a better method to extract the motion features. The first one is imperfect human motion. The motion does not need to represent the human motion precisely since the algorithm do not need precise motion comparison. This allow us to “group” the points together to a abstract “point” to represent the same point motion using a smaller motion sequence. The second one is the different speed of the human movements. In order to solve the problem, instead of storing the exact intermediate point of the controller in the motion sequences, storing which area the controller has visited should be a better approach. Therefore, a three-dimensional matrix is used to represent different space around the player. The points located inside the same element in the matrix is grouped as one point. The motion sequence is then become the sequence of these matrix elements.

**Figure 7:** A 8*8*8 matrix in 2D to represent the space around the player
The matrix is used to represent the space around the player (See Figure 7). In the figure, it is a three-dimensional matrix, but in a two-dimensional graphical illustration in this paper.

Therefore, all the matrix in this paper about the algorithm is a three-dimensional matrix. If the two-dimensional graphical illustration is used, the third value of the element is always 4 in this paper. The matrix size is up to developer. The larger the matrix size is, the longer the mapped motion sequences are for the same motion.

Therefore, the complexity of the predefined motion pattern should be considered to decide the matrix size before the implementation. In this project, the matrix is 8.

In the figure, the headset is located in the center of four elements, (3, 3, 4), (4, 3, 4), (3, 4, 4) and (4, 4, 4). The controllers are located in some elements in the matrix. Each controller will be handled separately and therefore there are two point motion representing the movement of the controllers in the game, but now just consider one of them for the algorithm explanation.

\[ \phi = \text{arm span} \times 1.5 \]

**Figure 8:** A graph to show how to calculate $\phi$

In each frame of Unity, the local coordinate of a controller is measured from the point motion tracking process. In this process, this local coordinate is mapped to one of the
elements in the matrix. Therefore, the actual size of the three-coordinate matrix in the local coordinate system is needed in order to do the mapping. \( \phi \) is then calculated to find the actual length of the three-coordinate matrix. \( \phi \) should be proportional to the arm span of the player (See Figure 8). It is because the human size should not affect the mapping of the local coordinate to the elements in matrix. In this project, \( \phi \) is 1.5 times larger than the arm span and almost all the possible position of the controller with respect to the headset can then be mapped to the matrix. After having the \( \phi \), the mapping from local coordinate to matrix element is easy which can be done in the formula of

\[
\text{element}(x, y, z) = (\text{local coordinate}(x, y, z) + \frac{\phi}{2}) \times \frac{\text{matrix size}}{\phi}
\]

One of the goals of feature extraction is to filter out the unnecessary points out to minimize the number of pattern recognition process performed for better time efficiency. After mapping to the elements in matrix, element-to-element distance is larger than the point-to-point distance. The slight positional change away from the previous position is very likely to be the same element in matrix. Therefore, the best way is to proceed to the next process only if the controller is moving from one element to other elements. The program holds the element in the previous frame and compare it with the matrix element calculated in this frame. The process will only proceed to the pattern recognition if these two elements are not equal, which means the controller has some relatively large positional change.
The motion sequence is now kept in the form as linked list of matrix element in the program (See Figure 9). The feature extraction process only proceeds to this stage of the algorithm with the change of matrix element found, which means a new element is found different from the previous one. Therefore, the program holds a linked list of such matrix elements representing the motion sequence and adds the new element to the beginning of the linked list when a new element is found. Each node is different.
from their next node or previous node since the element added to the list must be found different from their previous node. The matrix element retrieved from the previous process represents the position of the controller inside the matrix. Therefore, the first node represents the current location of the controller inside the matrix and the last node represents the oldest location of the controller within the time interval the program is measuring.

In Figure 10, it is a graphical illustration of the motion represented in Figure 9. The current element is (4, 3, 4) while the oldest element is (2, 5, 4). Therefore, the motion sequence is (2, 5, 4) → (2, 4, 4) → (2, 3, 4) → (3, 3, 4) → (4, 3, 4). After that, given a motion pattern (2, 5, 4) → (2, 4, 4) → (2, 3, 4) → (3, 3, 4) → (4, 3, 4), it is obvious that the motion sequence is perfectly matched with the motion pattern here because every node is the same. The next question is how to match the motion sequence with the pattern when there are some difference in the node. As discussed above, the human motion is imperfect. Asking a player to draw a circle in air for ten times will properly result in more than one different motion sequences. Therefore, a tolerance level is needed for the comparison.

![Figure 11: Matching between pattern node and tracked node on tolerance level = 1](image)

**Figure 11**: Matching between pattern node and tracked node on tolerance level = 1
Figure 12: Matching between pattern node and tracked node on tolerance level = 2

The tolerance level defines how far the pattern node allows the matched element is away from it. For a tolerance level 1, the distance between the pattern node and the tracked node along any axis must not larger than 1 (See Figure 11). Similarly, for a tolerance level 2, the distance between the pattern node and the tracked node along any axis must not larger than 2 (See Figure 12). In this project, since the matrix size is just 8, the tolerance level is decided to be 1, which means the pattern node only matches with its 26-neighbors in the three-dimensional matrix.

The predefined motion patterns in the database are also in the form of a linked list of matrix elements. The only difference from the tracked motion sequence in the implementation is that the first node of motion pattern is the oldest position while the first node of the motion sequence is the most updated position in the matrix. Therefore, the linked list of the motion pattern needs to start from the last node and going backwards during the comparison. It is called back tracing.
Although there is a tolerance level for comparison to tackle the imperfect human motion, it is shown that there are still cases where the motion sequence is mismatched with the desired motion pattern (See Figure 13). It is because there are some isolated elements even though most of the nodes are matched. The algorithm is not encouraged to ignore this kind of case. Therefore, there is one more variable to be considered in the algorithm, which is the mismatching score. For each isolated node, the mismatching score will be increased by the distance between the element in pattern node and the element in the isolated node. The higher the mismatching score is, the more dissimilar are the motion pattern and the motion sequence. However, the motion sequence still needs to match with all the pattern nodes during the linked list comparison. If there is any pattern node is not matched, then the sequence will still be mismatched with the motion pattern.

Therefore, the whole picture of the process is explained as follow. The tracked linked list would go forwards from the first node. The pattern linked list would go backwards from the last node. If they are matched, then both of them move to next node. If they are not matched, the pattern node would not go to the next node while the tracked node would keep on going. The mismatching score would be increased in this case. It would stops when one of the linked list goes to the end of the list. If the pattern linked list does not go to the end of the list here, it means there is a pattern node is not matched.
matched by the nodes in tracked linked list. Then, the motion sequence is considered mismatched with the motion pattern. In another case, the motion sequence is considered matched with the motion pattern.

The motion sequence is compared with all the motion patterns in the database. The program will select the mostly matched patterns according to the mismatching score and proceed to the thresholding process.

6.2.5 Thresholding Process

In the previous process, the mismatching score is increase for each isolated node. Therefore, the mismatching score is zero for a perfect matching. A high mismatching score means the high amount of difference between the predefined motion pattern and the tracked motion sequence. A highly dissimilar motion sequence and motion pattern should not be matched even if it passed the pattern recognition process, therefore the thresholding process is necessary. Therefore, if the mismatching score is higher than the threshold value, then the motion sequence should be considered as mismatched with the pattern. The actual value of the mismatching score should be proportional to the size of linked list of the matched motion pattern. The proper choice of thresholding value was experimented and discussed in the part of project result discussion.

If the motion sequence passed the thresholding test, then the corresponding game events is triggered. The linked list of the tracked motion is then cleared back to empty linked list. This is necessary to avoid the consecutive matching of the motion sequence with the same pattern. Consider the motion sequence \((0, 0, 1) \rightarrow (0, 1, 1) \rightarrow (0, 2, 1)\) and the motion pattern \((0, 0, 1) \rightarrow (0, 1, 1) \rightarrow (0, 2, 1)\). The matching result is obviously correct in this case. However, when the player moves his hand to other element in the matrix after the successful matching, the motion sequence becomes something like \((0, 0, 1) \rightarrow (0, 1, 1) \rightarrow (0, 2, 1) \rightarrow (1, 2, 1)\) and this will properly matched with pattern again. The game events will then be called again. Therefore, the linked list of the motion sequence is cleared for every successful matching,
6.2.6 Clearing Linked List of Motion Sequence

Apart from the situation above, the linked list of motion sequence should also be cleared every fixed amount of time. For instance, the program clears the linked list for every three seconds in this project. This is because the linked list will grow with time if it is not cleared. The linked list will become much more time consuming with a large linked list. Therefore, it is necessary to clear the linked list of the motion sequence in order to keep the efficiency of the algorithm.

6.2.7 Algorithm Analysis

The time complexity of the algorithm is $O(nm)$, which $n$ is the size of the linked list of the motion sequence and $m$ is the number of the predefined motion patterns. Normally speaking, the number of the predefined motion patterns is small. The size of linked list is limited by the constant linked list clearing of the algorithm. Therefore, the efficiency of the motion tracking algorithm is better than most of the other motion tracking algorithms in $O(nm)$, such as the point-to-point comparison algorithm introduced in the beginning of this subsection.
6.3 Game Design
This subsection includes all the design of the game, such as the theme and game events to be triggered by the motion tracking algorithm.

6.3.1 Overview
The name of game is Virtual Fighter. It is a mixture of fighting game and adventure game. The player needs to move around the town and find the treasure chest. There are a lot of monsters in the town. Player can use their controller as a weapon to hurt the monster. Moreover, there are some specific motion-driven events implemented with the motion tracking algorithm above to demonstrate the motion-driven player-game interaction. The details of those event will be discussed in that subsection.

6.3.2 Scene Design
There are three scenes in Virtual Fighter, namely the tutorial scene, the main story scene and the boss fight scene.

Figure 14: The tutorial scene of Virtual Fighter

The tutorial scene includes dialogues to guide the player how to play the game (See Figure 14). The guidelines include a instruction to teach players to set their arm span,
which is used to calculate the actual size of the matrix in local coordinate system according to the motion tracking algorithm. The tutorial scene is short and lasts for about three minutes to inform every necessary information of the game to the players, including how to use the abilities and how to defeat the monster.

Figure 15: The main story scene of Virtual Fighter

The main story scene is where the player started their adventures in the town (See Figure 15). There will be thirty houses in the town for player to discover. The treasure chest is hided in one of the houses in the town. The main mission of the player is to find the treasure chest and proceed to the boss fight scene. However, there are a lot of strong monsters around the town who can chase the players and attack them to death. Therefore, the player needs to carefully make use of the controllers as a weapon and a ability casting machine to defeat the incoming enemies and find the treasure chest without dying.
The boss scene is a small scene where the player needs to defeat the boss to win the game (See Figure 16). The boss is exceptionally strong and has the ability to cast long-range magic attack. The players may need to keep a distance between them and the boss and win the fight by casting long-range magic attack back.

### 6.3.3 Enemy AI Design

```csharp
public interface IState
{
    // This method will be called when the object is entering this state
    void OnStateEnter();

    // This method will be called in each frame when the object is inside this state
    void OnStateExecution();

    // This method will be called when the object is exiting this state
    void OnStateExit();

    // This is a method for entering other state
    void Transit(IState state);
}
```

Figure 16: The boss fight scene of Virtual Fighter

Figure 17: A image of the script of IState.cs
The AI of the enemy is designed based on the finite-state machine model. Each enemy contains a number of states, which implements the IState interface (See Figure 17). Each state defines the behaviours of the enemy object when it is inside the given state. The AI of the enemy can change from state to state, therefore the enemy can perform different behaviours in different situations.

The IState interface defines the state class must implement OnStateEnter(), OnStateExecute(), OnStateExit() and Transit(). Most of the regular behaviours of the enemy inside the class will be implemented on OnStateExecute(). The OnStateEnter() and OnStateExit() will only be called when the enemy AI is entering or leaving this state. Finally, the state class must give the fundamental implementation on Transit() to define how to transit from this state to other states in the finite-state machine.

6.3.4 Motion-Driven Events Design

There are three motion patterns to be recognised in Virtual Fighter. The motion sequence is list below in the form introduced above.

- Motion pattern 1 (for left hand controller): (1, 3, 3) → (1, 4, 4) → (2, 5, 4) → (3, 5, 5) → (4, 5, 5) → (4, 5, 4) → (5, 4, 4) → (5, 3, 3). Simply speaking, this is a motion pattern of drawing a half circle from left side to right side of the body.

- Motion pattern 2 (for right hand controller): (5, 5, 3) → (5, 4, 4) → (4, 4, 5) → (3, 4, 5) → (3, 3, 5) → (2, 3, 5) → (2, 2, 4) → (2, 1, 4). Simply speaking, this is a motion of right hand from top right to bottom left.

- Motion pattern 3 (for right hand controller): (5, 1, 5) → (5, 2, 5) → (5, 3, 5) → (5, 4, 5) → (5, 5, 5) → (5, 4, 5) → (5, 3, 5) → (5, 2, 5) → (5, 1, 5). Simply speaking, this is a overlapping motion by lifting right hand and put it down.

There are corresponding game events to be triggered when the player performs a hand motion which is matched with one of the motion patterns above. For instance, the player may shoot a fireball when doing a hand motion according to motion pattern 1.

7 Discussion of Results
In the following section, the current result of the project are discussed. Most of the current results in the project is about the algorithm testing and game implementation.

7.1 Game Implementation
In this subsection, the results of the game implementation are discussed. The implementation of different aspects of the game will be written in details.

7.1.1 Game Map

Figure 18: The game map of tutorial scene

Figure 19: The game map of boss fight scene

Figure 20: The game map of main story scene
The maps are built on top of the free 3D asset found online. The game map of the tutorial scene is just a small area surrounded by fences and houses (See Figure 18). The game map of the boss fight scene is larger and built on top a beautiful castle (See Figure 19). The game map of the main story scene is very large with about thirty houses and many open area (See Figure 20). Player can spend many time discovering the town inside the main story scene.

### 7.1.2 Enemy AI Implementation

**Figure 21:** A graph of enemy chasing the player

**Figure 22:** A graph of enemy attacking the player
The implementation of the AI of the enemy is based on the finite-state machine design. A normal enemy has four states, including the idle state, chasing state, returning state and fighting state. The enemy in idle state stands still unless the player is closing by. It changes to the chasing state when the player is inside the detection range and starts chasing the player (See Figure 21). If the player runs away and exit the detection range, then the enemy transits to returning state which it will runs back to its spawn location. If the player is too close, the enemy will transit to fighting state and start attacking the player (See Figure 22).

The implementation of AI of the boss is more advanced. It has one more state to cast the magic attack. The state is called the casting state. If the player is inside the detection range but not close enough for melee attack, the boss will transits to casting state and shooting fireballs to kill the player.

7.1.3 Motion-Driven Events Implementation

![Figure 23: An image of player perform motion pattern 2 to shoot fireball](image)
There are two magic attacks the player can use, namely the fireball attack and the ultimate attack. Both the motion pattern 1 and motion pattern 2 are mapped to the fireball attack. Therefore, there are two ways, one for right hand and one for left hand, to shoot the fireball attack in Virtual Fighter (See Figure 23). The motion pattern 3 is mapped with the ultimate attack. When the player lift up and put down his right hand, the player object will cast the ultimate attack and deal massive damage to the enemy in front (See Figure 24).
7.2 Algorithm Testing
In this subsection, the result of the algorithm testing will be discussed. The correctness and reliability of the algorithm was tested. The proper choice of thresholding value will also be introduced with the experimental result.

7.2.1 Reliability Test
In order to test the reliability of the algorithm, the quality assurance engineer in the project performed a reliability test on the algorithm implementation. There are three predefined motion patterns in Virtual Fighter. The quality assurance engineer performed the three motion sequences a hundred time to test whether the algorithm can match the motion sequence with the corrected pattern in most cases. The result is list below in a table.

<table>
<thead>
<tr>
<th>Motion Pattern</th>
<th>Success</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Pattern 1</td>
<td>91 (91%)</td>
<td>9 (9%)</td>
</tr>
<tr>
<td>Motion Pattern 2</td>
<td>93 (93%)</td>
<td>7 (7%)</td>
</tr>
<tr>
<td>Motion Pattern 3</td>
<td>97 (97%)</td>
<td>3 (3%)</td>
</tr>
</tbody>
</table>

It is shown that all tests on the three motion patterns has been successfully matched in over ninety percent of the time. This is a very satisfactory result. However, this may be biased since the developer is already very familiar with the motion being tested.

The project then invite a voluntary gamer to perform the same test to measure the reliability of the algorithm. The number of tests for each motion patterns is 50 in this test. The result is listed in the table below.
<table>
<thead>
<tr>
<th></th>
<th>Success</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Pattern 1</td>
<td>39 (78%)</td>
<td>11 (22%)</td>
</tr>
<tr>
<td>Motion Pattern 2</td>
<td>41 (82%)</td>
<td>9 (18%)</td>
</tr>
<tr>
<td>Motion Pattern 3</td>
<td>45 (90%)</td>
<td>5 (10%)</td>
</tr>
</tbody>
</table>

The result is less satisfactory than the previous one. It may be due to the small sample size. Nonetheless, the average percentage of successful matching for the voluntary gamer is about 83.33%. It is still a fairly good successful rate. Therefore, the reliability test of the algorithm and its implementation is considered pass.

7.2.2 Thresholding Test

![Box Graph of the mismatching value distribution](image)

**Figure 25:** The box graph of the mismatching score distribution

In order to determine the proper choice of the threshold value in the thresholding process in the motion tracking algorithm, a test on the mismatching score distribution was performed. As shown in Figure 24, the box graph shows the basic distribution of 50 non-zero mismatching scores when performing the motion pattern 1. As discussed before, the threshold value should be proportional to the size of the pattern linked list. For motion pattern 1, the size of pattern linked list is 8. The third quartile of the data
set is 12, which means 75% of the mismatching score are equal to or below 12. The factor $\gamma$ after dividing the third quartile of the mismatching scores with the size of the pattern linked list is $12 / 8 = 1.5$. Therefore, according to the experimental result, the proper choice of threshold value is $\gamma \times$ size of pattern linked list.
8 Future Plan

In this section, the project schedule will be examined and the future plan will be also discussed.

The current progress of the project is almost finished (See Appendix I). The only task left is the poster exhibition, which will be finished by the end of April 2019.

The designed motion tracking algorithm does not consider orientational data as part of the motion. It is because it will be challenging to design an algorithm to recognise both the positional and orientational pattern. It will still require a lot of computational power even if an algorithm is designed for the pattern matching on orientational change by motion. The performance of the game will be affected adversely. Therefore, the next plan in the future might be designing an algorithm that can handle the orientational pattern well with good time complexity and add it to Virtual Fighter. Moreover, there are only three predefined motion patterns in the Virtual Fighter. Adding more motion patterns and corresponding game events to improve the gameplay is also an important task.

The next step after that is to start considering other types of motion defined in the background section, such as the hand motion and the full body motion. It would be challenging to design an algorithm to distinguish the full body motion of the players. It is hoped that this can be successfully designed in the foreseeable future.
9 Conclusion

This paper introduces a new concept named motion-driven player-game interaction. Motion-driven player-game interaction is a type of player-game interaction which is initiated by the motion of the players. In order to achieve this kind of interaction, a motion tracking and pattern matching algorithm was proposed for effective implementation. A demonstrative game, called Virtual Fighter, was implemented to show the effectiveness on how motion-driven player-game interaction can improve the gaming experience. Virtual Fighter is a mixture of adventure and fighting game which the player can explore around the map to defeat different monsters and find the hidden treasure chest. The motion-driven player-game interaction might improve the future implementation of virtual reality games. Players can have more controls on the game and more interaction with the games and other players, therefore the proposed type of interaction can be a solution to the problem of rudimentary gameplay design of the existing virtual reality games. Moreover, the motion-driven player-game interaction can also solve the shortage of input methods of the virtual reality system. Using motion as an input method other than the traditional input method, like mouse and keyboard, can bring more improvements to the development of virtual reality game. Currently, the project only handles point motion. Other types of the motion, like the hand motion or the orientational motion, is not handled in the project. Therefore, it is hoped that the next step of the project is to design a fast and efficient motion tracking algorithm for those types of motion in the future.
References


## Appendix I - Project schedule in Project Plan (Page 18)

<table>
<thead>
<tr>
<th>Date</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 September 2018</td>
<td>Deliverables of Phase 1 (Inception)</td>
</tr>
<tr>
<td></td>
<td>• Detailed Project Plan</td>
</tr>
<tr>
<td></td>
<td>• Project Web Page</td>
</tr>
<tr>
<td>October 2018</td>
<td>Research on Virtual Reality and Motion Tracking Devices</td>
</tr>
<tr>
<td>November 2018</td>
<td>Algorithm Study</td>
</tr>
<tr>
<td>December 2018 - January 2019</td>
<td>Development of Demo Game</td>
</tr>
<tr>
<td>7 - 11 January 2019</td>
<td>First Presentation</td>
</tr>
<tr>
<td>20 January 2019</td>
<td>Deliverables of Phase 2 (Elaboration)</td>
</tr>
<tr>
<td></td>
<td>• Preliminary implementation</td>
</tr>
<tr>
<td></td>
<td>• Detailed interim report</td>
</tr>
<tr>
<td>February 2019 - April 2019</td>
<td>Development on Deliverable Game</td>
</tr>
<tr>
<td>14 April 2019</td>
<td>Deliverables of Phase 3 (Construction)</td>
</tr>
<tr>
<td></td>
<td>• Finalized tested implementation</td>
</tr>
<tr>
<td></td>
<td>• Final report</td>
</tr>
<tr>
<td>15 - 19 April 2019</td>
<td>Final Presentation</td>
</tr>
<tr>
<td>29 April 2019</td>
<td>Project Exhibition</td>
</tr>
</tbody>
</table>
Appendix II - Hyperlink to Game Walkthrough video on Youtube

Hyperlink: https://www.youtube.com/watch?v=Bt9tIDMfftK&t=10s
Video Title: HKU CS FYP18021 Game Walkthrough
Video length: 04:20