２０１８年度　修士論文

Running Support System in Realistic 3D Map Using Body Movements

指導教員　田中二郎　教授

早稲田大学大学院情報生産システム研究科

情報生産システム工学専攻

インタラクティブプログラミング　研究

44161017-1　　ADHI YUDANA SVARAJATI

Abstract

We developed a running support system using body movements as input in a realistic 3D Map. In our system, the user uses their body movements to control movements in the 3D map. We used depth-camera sensor to track the user's body joints movement as the user moves. The location changes are tracked real-time and the system calculates the speed. The speed derived from the system mechanism will later be used as speed control in our system. This means when the user run in the real life, they will also feel like running in the system naturally. The user will also feel the realistic aspect of our system because the speed can keep up with user's speed in the real life.

Realistic 3D map from Zenrin is used in our system. The map consists of several detailed featured such as traffic information, train station, and another natural feature such as weather, lightning, and shadow. Therefore, we hope that by using this virtual environment in our research, we can help the user to feel more realistic and fun experiences. Besides, the map also holds various possibilities for rich and interactive system in the future.

In our system, we use a head-mounted display to assist the user's experience while using the system. The user will wear the head-mounted display while running and can see the realistic 3D map. We aim to provide immersive experience so the user can feel like going to the real place and keep motivates the user to give better running performances.

Integrating realistic 3D map as part of the system will enrich human experience, keep user motivated, and provide natural environment that is similar with the real world. We hope our system can assist the users as one of possible alternatives running in a virtual environment.

Keyword: Realistic 3D map, Running Support System, Body Movement

TABLE OF CONTENTS

Table of Contents 3

List of Figures 5

Chapter 1 Introduction 7

1.1 Research Background 7

1.2 Motivation 8

Chapter 2 Target & Approach 9

2.1 Goal Description 9

2.2 Target System 9

2.3 Approach 10

Chapter 3 System Design 12

3.1 System Overview 12

3.2 Realistic 3D Map 13

3.3 Body Movement as Input 14

3.3.1 Moving Forward Mechanism 15

3.3.2 Speed Control 16

3.3.3 Horizontal Rotation 18

3.3.4 Posture Control 19

3.4 Collision Control 20

3.5 Head Mounted Display Settings 22

3.6 Additional Interfaces 23

Chapter 4 System Implementation 25

4.1 Realistic 3D Map 25
4.2 System Settings 31
4.3 Body Movement as Input 35
4.3.1 Moving Forward Mechanism 38
4.3.2 Speed Control 39
4.3.3 Horizontal Rotation 42
4.3.4 Posture Control 43
4.4 Collision Control 44
4.5 Additional Interfaces 45
Chapter 5 Evaluation and Discussion 46
5.1 ISIPS 2017 46
Chapter 6 Related Works 51
6.1 Using Body Movements in VR System 51
6.2 Realistic 3D Map in VR System 52
Chapter 7 Conclusion 53
7.1 Summary 53
7.2 Future Works 53
Acknowledgement
References

LIST OF FIGURES

Figure 1. Target System Illustration 10
Figure 2. System Illustration 12
Figure 3. Akihabara 3D Map by Zenrin 13
Figure 4. Akihabara 3D Map by Zenrin 13
Figure 5. Body Skeletal Data Captured by Kinect 14
Figure 6. Body Movement Integration 15
Figure 7. Running and Walking Posture 15
Figure 8. Changing Point of View 16
Figure 9. User's Orientation Toward Depth-Camera 18
Figure 10. Good Posture and Bad Posture in Running 19
Figure 11. Reminding User to Correct Their posture 20
Figure 12. Moving Area in The Room 20
Figure 13. User Getting Too Near an Object in The Room 21
Figure 14. The System reminds the User to Move Backward 21 21
Figure 15 Samsung Gear VR Head Mounted Dsiplay. 22
Figure 16. Server Streaming Kinect Data 22
Figure 17. Sever-Client Component in Unity 22
Figure 18. Running Performance Display 23
Figure 19. Zenrin 3D Map Menu 25
Figure 20. Different Weather Configuration 26
Figure 21. Turning on Shadow Control (a) and Off (b) 26
Figure 22. Rain Interface in The City Environment 27
Figure 23. Several Point of View Selections 27
Figure 24. Fly Through Mode (a) and Driving Mode (b) 28
Figure 25. Moving Avatar Code 28
Figure 26. Car "NavMesh" Code 29
Figure 27. FPS Code 29
Figure 28. Post-Edit 3D Map Asset 30
Figure 29. Imported Avatar in the 3D Map 31

Figure 30. System Setup 32
Figure 31. Devices Relationship Illustration 32
Figure 32. Kinect SDK Browser 33
Figure 33. Kinect studio v2.0 34
Figure 34. Kinect v2 Unity Plugin as Unity Package 34
Figure 35. Capturing Data Using Kinect in Unity 35
Figure 36. Avatar Connected with User's Body Joints 36
Figure 37. Avatar Joints Structure 37
Figure 38. Avatar Component 37
Figure 39. Attaching Avatar's Feet with A Game Object 38
Figure 40. First Person Controller Code 39
Figure 41. Position Tracking Code 40
Figure 42. Avatar Joint Position Controller 40
Figure 43. Tracked Joint and Speed Calculation 41
Figure 44. Unity Object Movement Speed Control Window 41
Figure 45. Quaternion Calculation Code 43
Figure 46. Calculating Distance Between User and Object 44
Figure 47. "Time" Feature Code 45
Figure 48. ISIPS Symposium 2017 46

Chapter 1 Introduction

1.1 Research Background

Realistic 3D map and Head Mounted Display developments open new possibilities for creating interactive and immersive interface that enables user to explore and enjoy new experiences in the virtual world. On the other hand, depth-camera is a well-known device that can locate body joints and recognize body movements based on body joints' location changes. There are several depth-cameras in the market, such as Leap Motion which specialized in detecting hands and finger movements We are interested in full body movements tracking using Kinect depth-camera in a realistic 3D map environment.

Realistic 3D map recently becoming more popular due to the increasing need of virtual environment that is similar with the real world. Zenrin, a map company from Japan, created realistic 3d map of cities in Japan. The Zenrin 3D map includes detailed information such as traffic and road sign, public spaces, and natural landscapes [1]. Several cities already developed and many more will come, giving opportunities for virtual reality (VR) system development.

Virtual reality provides many possibilities for developing interactive and immersive applications. Although the use of VR holds risk, such as VR sickness [2], it certainly provides opportunities for new applications and provides a unique experience to the user.

Exercise is one of the most important aspect of human life. We need to exercise in daily routine to keep our body healthy and fit. There are many varieties of exercise, but among them running is popular because people consider it fun and easy to do. Despite the popularity of running, in this modern time, many people do not exercise enough for maintaining their health. There are lots of reasons for keeping people from doing routine exercise, therefore we aim to create a system that people can use for exercising, especially running, anytime and anywhere they want.

In this paper, we propose a support system for running in a realistic 3D map. The user's body movements will be used as input by mapping it using Kinect depth camera [3]. The 3D map used in this system is a replication of the real environment of a city in Japan. The use of realistic 3D map will keep the user motivated and provide familiarity with the real environment.

1.2 Motivation

We are aiming to create a running support system in which a user uses his body movement to control movements on the 3D map. When the user runs in the real world, the user will also feel like running in the map naturally. Integrating realistic 3D map will enrich the human experience, keep user motivated, and provide a natural environment, which is similar to the real world. We will use a realistic 3D Map of a city in Japan by Zenrin (Map Company) to provide a similar environment with the real world.

A system using a realistic 3D map as its environment is still limited, due to the lack of suitable environment. The development of 3D map of cities in Japan by Zenrin provides and opens new ways for presenting immersive experience.

Body movement is used as input to the system by mapping the user's body joints into a character in the system [3]. By using body movement as input, the user will feel more connected and realistic in the way they move and control the system. The body joints movement data will be used to calculate velocity and recognize action [4]. The action recognized will determine the movement in the system.

Exploring a large virtual environment usually required lots of space in the real world. Consequently, many systems related to virtual environment exploration could only be realized in small scale or using a controller to move in the virtual world. Using controller to control movement in the virtual world will provide one solution, but consequently the system will not provide natural feeling for the user. Another possible solution is to use walking in place principle to explore a large virtual reality environment, so we do not need to provide lots of space in the real world [5].

Chapter 2 Target & Approach

2.1 Goal Description

The synergy between virtual reality and human motion can be applicable for improving human health, such as creating running support application. Currently available sport systems provide various features and experiences to the user and various devices also available. However, the currently available systems in the market still holds problems that we need to solve, such as:

1. Lack of interactive ways to explore a virtual world using realistic body movement
2. Realistic 3D map which is like the real world has not been used for enhancing human experiences

Solving these problems becomes our research's goal. We are aiming to solve the lack of ways for virtual exploration and use the currently available realistic 3D map. We summarize our goal in the following:

1. Provide ways for interacting and exploring virtual world through creating a running support system using body movement as input
2. Create the running support system using realistic 3D map by Zenrin so the user can get rich experiences and feels like going to the real place in the real world

2.2 Target System

The target system of our research is a running support system that uses body movement as input while exploring the realistic 3D map in immersive way. We will use several devices such as Kinect Depth-camera and Samsung Gear VR Head mounted display in our research. Figure 1 illustrates our target system.

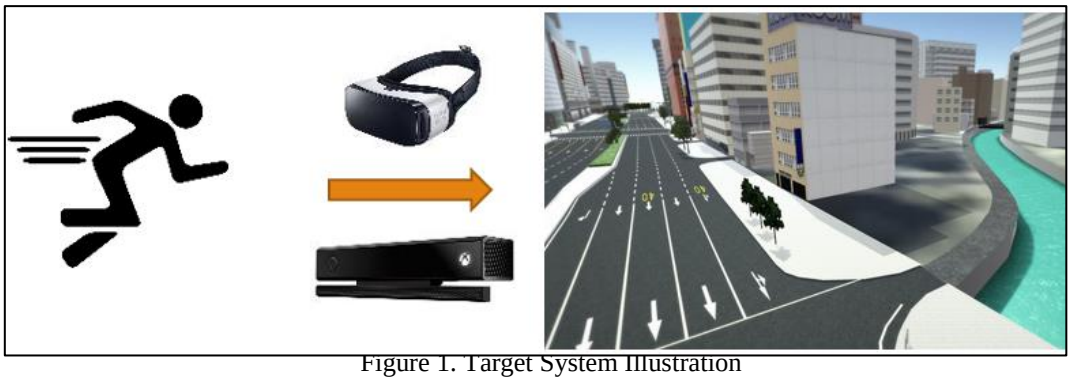

Figure 1. Target System Illustration

As illustrated in figure 1, the user's body movement is tracked by Kinect depth-camera real-time. The body movement acts as input to the system, so when the user moves their body according to certain manner, the system will give some feedbacks. The user body is integrated with the system and they will be able to look around the system using head-mounted display. In the end, we integrate our system with a realistic 3D map as virtual world so the user can do exercise while exploring the environment.

2.3 Approach

1. We use 3D map from Zenrin company as our virtual world environment. The 3D map from Zenrin provides detailed and accurate information so the user will feel like going to the real place in the real world. The user will also be easier to remember the location and maintain their running performance. User will feel like running in real place, therefore the performance is expected to be similar with the real running.

2. In this system, we use body movement as input. The User's body movement is tracked and processed into another output in the system. We use body movement to provide natural feeling toward the user, so the user will move at ease and feel like connected into the system through their body. The body movement is represented by body joint's movement in real time.

3. Among several depth cameras, we choose to use Kinect depth-camera. We choose Kinect as our tool because of its capability to capture whole body data. Kinect can capture the user's body movement through tracking the body joints. Kinect also provides tracking of more than 36 body joints and another feature such as gesture. We would like to use the whole-body movement to produce movement in the system, speed control, posture control, and another feature that require full body movement

4. User will use head-mounted display to look inside the 3D map. We decided to use HMD to provide the user with immersive feeling and in more natural way. The user can run in longer period and get the feeling of the certain area the user explore in the map

Chapter 3 System Design

3.1 System Overview

We developed a running support system in a realistic 3D map using body movements as input. The user body movement's such as knee and hip will be tracked and processed by the system. When the user moves such as running or walking in the real world, the user can explore the realistic 3D map. Figure 2 illustrates our system.

Figure 2. System Illustration

The user can run, walk, and rotate inside the realistic 3D map. Besides those features, our system will also have speed control for controlling movement speed in the 3D map, horizontal rotation for moving the point of view of user horizontally, posture control for maintaining user's good posture while running, collision control for preventing injury, and several additional interfaces.

We are using depth camera, pc, head mounted display in our system. Those devices are installed, integrated, and programmed for creating our system. In the following sections, we will explain features included in our system and Chapter 4 will presents the detailed information about our features such as installation, scripting, and many more.

3.2 Realistic 3D Map

We implemented the system for running support system in a realistic 3D map. We use a 3D map of city in Japan provided by Zenrin. In our system, we use the Otaku city 3D map (Akihabara) because of its popularity worldwide. The 3D map provides a detailed environment and smooth images of the real Akihabara City as shown in figure 3. We downloaded the 3D map file from Unity Asset Store.

Figure 3. Akihabara 3D Map by Zenrin

Information in the 3D map consists of several detailed infrastructure, public spaces, and natural landscapes. We can observe traffic and road signs, train station, and rivers just like in the real world as shown in figure 4. By using this detailed map, we hope can keep the user's motivation high and make the user feel like exploring the location in the real world.

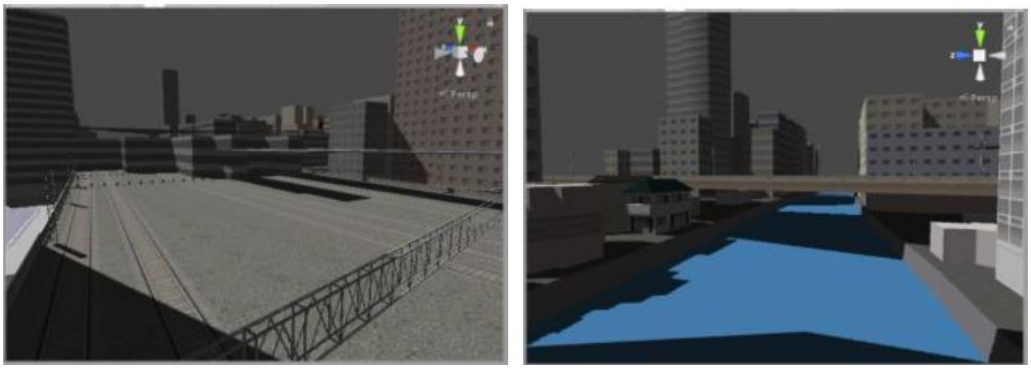

Figure 4. Akihabara 3D Map by Zenrin

The raw file downloaded from Unity asset store still contains several features provided by the company. To develop system of our own, we need to delete the current features and install our own. Section 4 will present the detailed explanations.

3.3 Body Movement as Input

We are tracking body movements of the user and use it as input to the system. For realizing the system, we need to track the user's body joints and observe its movements. In this case, we use Kinect depth-camera to capture the user's body joints as shown in figure 5.

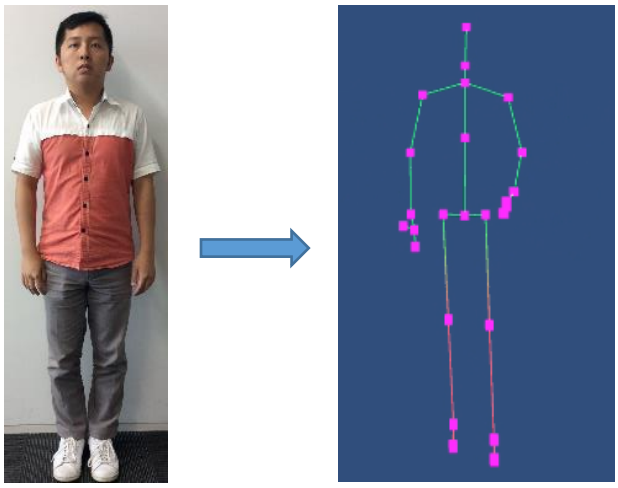

Figure 5. Body Skeletal Data Captured by Kinect

By using Kinect depth-camera, we can capture the whole-body structure of the user in the matter of capturing body joints. Kinect Microsoft SDK provides data for capturing 25 body joints, and by tracking those joints we can use its movements as input for the system. After tracking certain joints movement, we will implement several algorithms to calculate their velocity and use it as variable in the movement mechanism, speed control, etc.

We connect each joint to a character model in the 3D map using unity and "Kinect v2 Examples with MS-SDK" asset. The user's movement is integrated with the character, as shown in figure 6. By integrating the user's movement to the character, we want to make the user feel connected and get "feedback" feeling.

Figure 6. Body Movement Integration

The body joint movements will be used for creating several features in the system such as running/walking movements, rotation, running/walking and rotation speed integration, and posture control. In our system, the movements from several body joints is observed to make a more realistic running support system.

3.3.1 Moving Forward Mechanism

The first feature of the system is the moving forward. The purpose is how to make the user runs as if running in the real world by moving their leg, arms, and create certain posture that are similar. If the user makes running or walking like movements as shown in figure 7, the point of view in the virtual world will change as if the users move in the real world, as represented in figure 8.

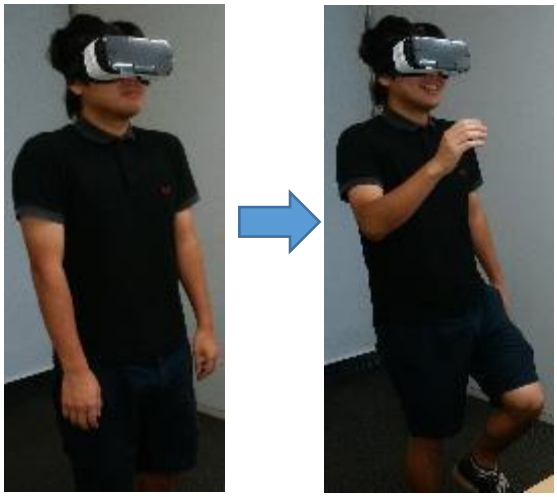

Figure 7. Running and Walking Posture

Figure 8. Changing Point of View

A Basic moving forward mechanism will be used in the unity studio. We will use the key control in unity for moving forward by inputting button “w” for moving forward. When the system detects the button input, the avatar will move also in forward movement. Later, we will link the moving mechanism with speed control feature to reflect more realistic movement by the user.

Both running and walking movement used the same starting position as illustrated in figure 7, but the difference is with the speed of the joints movement. If the system tracked the joints movement as fast, it means the user is considered as running, vice versa.

3.3.2 Speed Control

The system will also have speed control for the running movement. Speed in this context means the avatar speed of moving through the map. We can integrate the velocity obtained from equation (1), (2), and (3) with the avatar speed by using f variable in the unity. F variable is a float variable, and the value can be set as much and as big as we want.

$$\text{avg(l\_knee)} = \sum_{i=1}^{n-1} f_i^{l\_knee} - f_{i-1}^{l\_knee} \quad (1)$$

$$\text{avg(l\_knee)} = \sum_{i=1}^{n-1} f_i^{l\_knee} - f_{i-1}^{l\_knee} \quad (2)$$

$$\max(\text{avg(l\_knee)}, \text{avg (r\_knee)}) \quad (3)$$

First, we use depth-camera to track knee joints. Then, we calculate the velocity of knee movements (x, y, z axis). We calculate the velocity by comparing the position changes of those joints for each frame as illustrated in equation (1) and (2) [4]. After acquiring velocity, we will take the maximum value from r_knee and l_knee. In equation (1) and (2), f is frame, n is the number of buffer, and time is the time taken by Kinect between frame i and frame i-1 to obtain skeleton data. The velocity obtained will be used as speed control in the system, therefore the avatar's movement speed will be same with the velocity. We will not use the threshold value presented in equation (3) because we only developed the speed calculation and not the recognition between walking or running.

We will use knee movement as our input value because of human running posture. In our system, the users do not run like in the real world, instead they will do running in place movement. We observe that the knee is the part that will move the most when people running in the real world. Consequently, knee joints are tracked by our system and its changing during the time will be used as speed control input.

In Unity 3D studio, we can set the speed of a moving object by adding f as the speed. If we set the object speed as 2f, the object will move 2 times the f variable. So, we need to divide the velocity with the f value and set it into the moving object. For example, we set the f with 3, then we obtain the velocity is 15, so we will get 15/3 = 5 from dividing velocity with f. Then we can use 5f to assign the speed into the moving object. After acquiring the movement speed calculated by equation (1), we linked the value with the moving forward mechanism. When the speed calculator produces certain value of speed, it will be converted into another value and times it with the f in the unity. So, the speed is constantly changing throughout the use and will follow user's hip movement.

Speed controller is added into the system to realize a realistic running support system. Human run in certain speed in certain period and the speed is subject to change due to various reasons. Integrating the speed into the system will provide more realistic feeling to the user, so we can expect linear performances of duration using our system with the benefit of running.

3.3.3 Horizontal Rotation

Another feature in the system is the horizontal rotation. This feature enables the user to change his point of view in horizontal direction. Since we limit the space of user in our system, we need another way to realize the horizontal movement. In this case, the Horizontal rotation movement is produced by the user's orientation toward the depth-camera as shown in figure 9 (a).

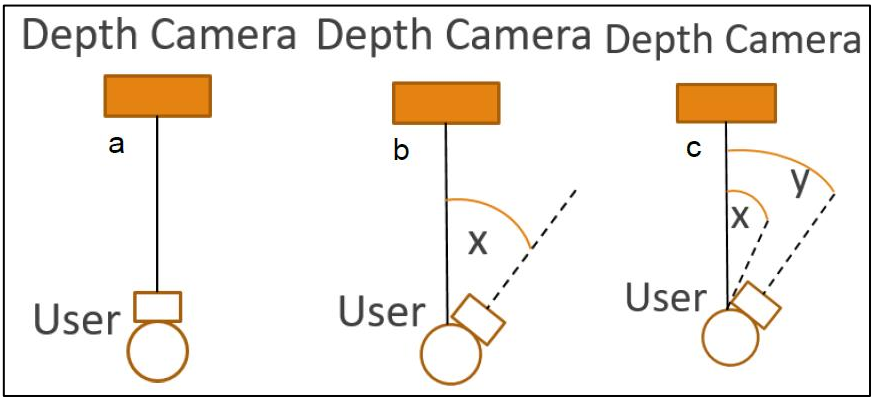

Figure 9. User's Orientation Toward Depth-Camera: (a) User facing Straight (b) User faces x degrees toward the right side (c) User faces larger x and y degrees toward the right side

If the user wants to change the direction in the system, the user needs to change direction toward the depth-camera. For example, as shown in figure 9 (b), if the orientation exceeds x degrees, the point of view in the system will also change toward that direction. This feature is designed to compensate the vast 3D map environment with the limited space in the real world. By using this feature, the user will able to change direction and explore another area without changing permanent direction in real life. In this sense, the user will maintain his original position when starting to run or walk again.

This feature will also have speed control of the movement. The user can choose to rotate faster or slower depending on their orientation toward the depth-camera. For example, on figure 9 (c), the user decides to change his direction toward Kinect for y degrees. Compared to the previous degrees (x), the rotation will become faster.

3.3.4 Posture Control

Posture control is a mechanism for comparing the user's posture while running with a standard. The feature will help the user maintain good posture while running, such as asking the user to keep their legs higher while running so that the user will feel exhausted, similar with running in real world.

The user's body posture is captured by Kinect and we will set certain algorithm to measure the position of body joints real time. For example, if the user's knee and hands were not in the correct position assumed for good running posture, the system will add reminder in the screen so the user can correct their posture all the time. Figure 10 illustrates the good posture and bad posture in running.

Figure 10. Good Posture (Left) and Bad Posture in Running (Right)

The system will remind the user of their posture by comparing with the standard posture for running. In our system when the user makes bad posture as illustrated in figure, the system will recognize the posture as bad and reminds the user. For example, in figure the user's running with arching back and is considered a bad

posture. The system will remind the user by popping a text in the head mounted display as represented in figure 11.

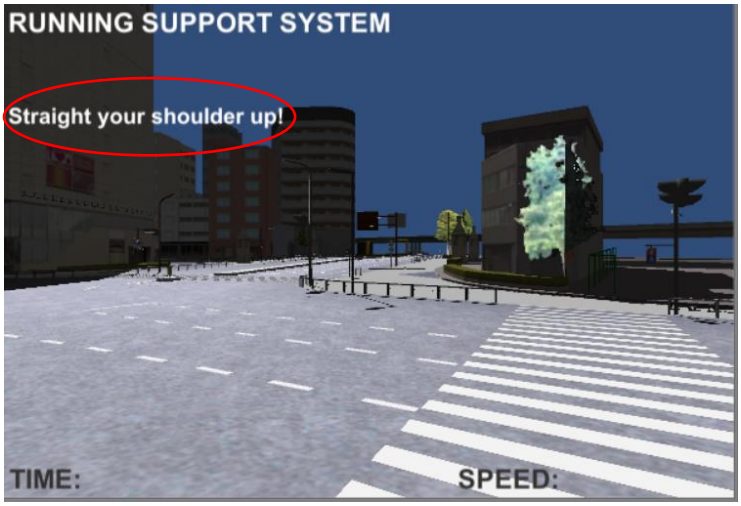

Figure 11. Reminding User to Correct Their posture

3.4 Collision Control

Collision Control is a feature that will prevents the user to have a collision with another object in the room and makes the user stay in the proper distance of Kinect camera. We will use the depth data (z coordinates) of the user's distance from Kinect to their position, and x coordinates data to track the user's movement relative to the sensor in the room.

The system will recognize the distance between the user and another object. In our research, we limited the area of the user movements into a square area of 1.5 meters squares as illustrated in figure 12. The limitation is meant to prevent the user going out from the Kinect's reading area and prevents the user from making collision or injury by hitting with surrounding objects.

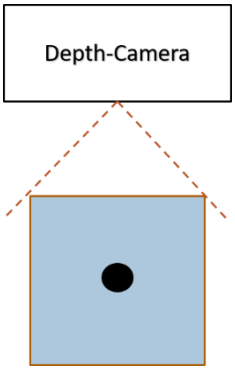

Figure 12. Moving Area in The Room

When the user's position become to near around 0.5 m from the Kinect as presented in figure, the system will warn the user to move slightly backward. At the same time, if the user moves too far, the system will also warn the user to move forward until they reach the original position. The warning will be presented by the system in the head mounted display as illustrated in figure 13.

Figure 13. User Getting Too Near an Object in The Room

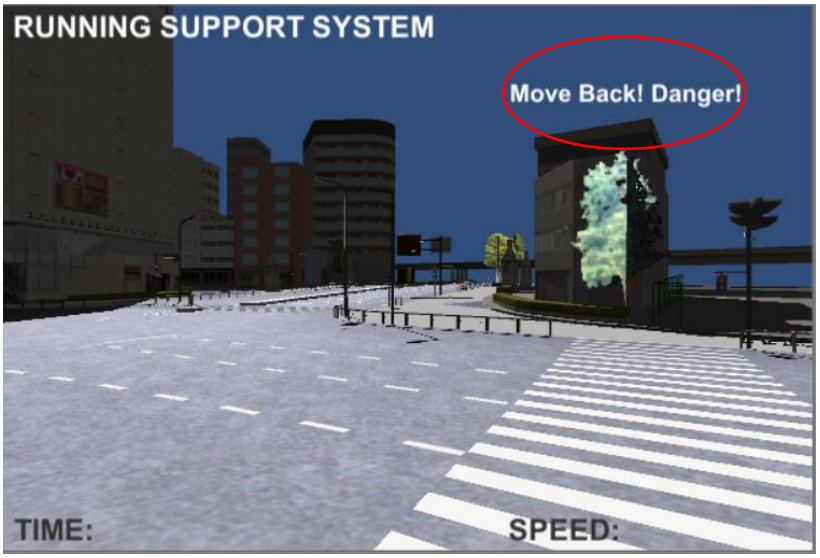

Figure 14. The System reminds the User to Move Backward

3.5 Head Mounted Display Setting

We use Samsung Gear VR for the head mounted display device. The unit consisted of a smartphone and one head mounted display, as represented in figure 15. The system is created in Unity 3D studio and built for the android device. After install the system in the smartphone, we also use server-based interface for streaming the data from Kinect to the android smartphone because between the computer and the smartphone is not connected. The server based interface is presented in figure 16.

Figure 15. Samsung Gear VR Head Mounted Dsiplay

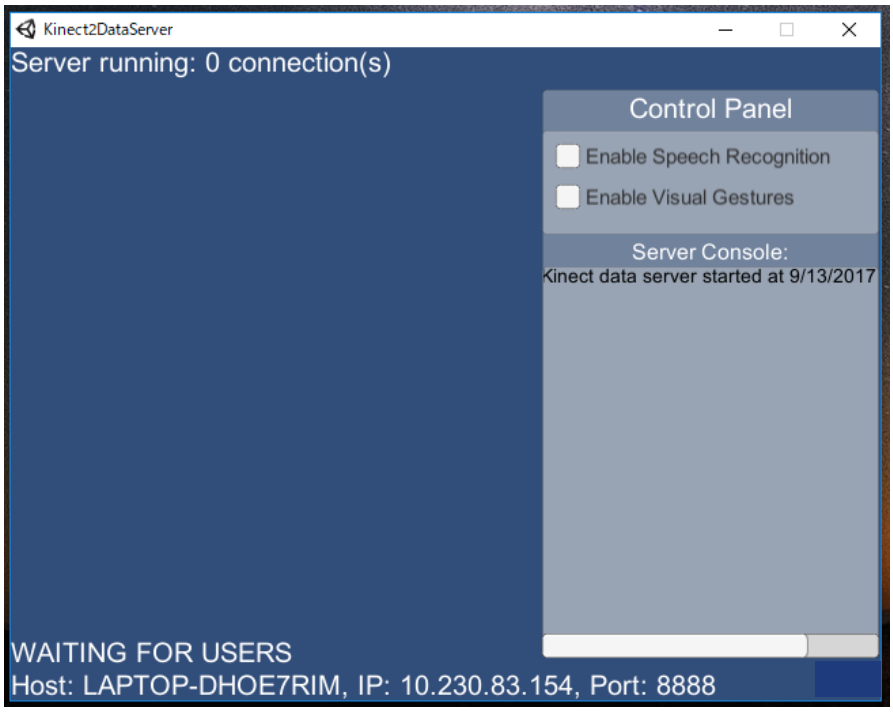

Figure 16. Server Streaming Kinect Data

The system is created with using Unity and all the devices must be connected into the same WIFI Network. When we want to use the system, we must first connect all the devices and run the server in the PC as represented in figure. We can also use the same PC with the running application. The application .exe from figure must be opened and run in the background. To realize this feature, we need to add some feature to the unity so the system can stream the data into a server and linked with the head-mounted display. Figure 17 represents the component in the system we can use to set the server and client relationship in the system.

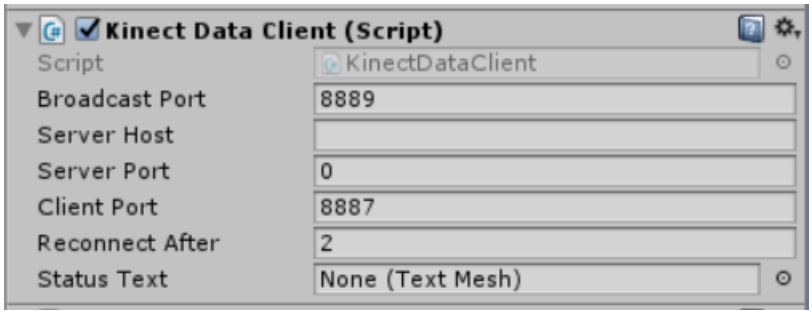

Figure 17. Sever-Client Component in Unity

3.6 Additional Interfaces

We also developed additional interfaces to the system. Additional information such as time, distance, and speed will be presented on the head mounted display when the user is using the system as shown in figure 18.

Figure 18. Running Performance Display

“Time” will indicate the duration of user using the system, “Distance” will record the user’s running distance in the 3D map. “Speed” will be used for presenting user’s running speed. Presenting several information about user’s running performance will help to motivate the user and the user will also know their performance. The user can decide when to correct their posture, increase their speed, and set their own running target such as distance.

Chapter 4 System Implementation

In this chapter, we explain several important modules from our system. Modules that needed detailed information will be presented including our way of implementation.

4.1 Realistic 3D Map

Zenrin provides free 3D Map file downloadable through Unity Asset Store. The original file contains several features such as presented below and presented in one menu, as illustrated in figure 19.

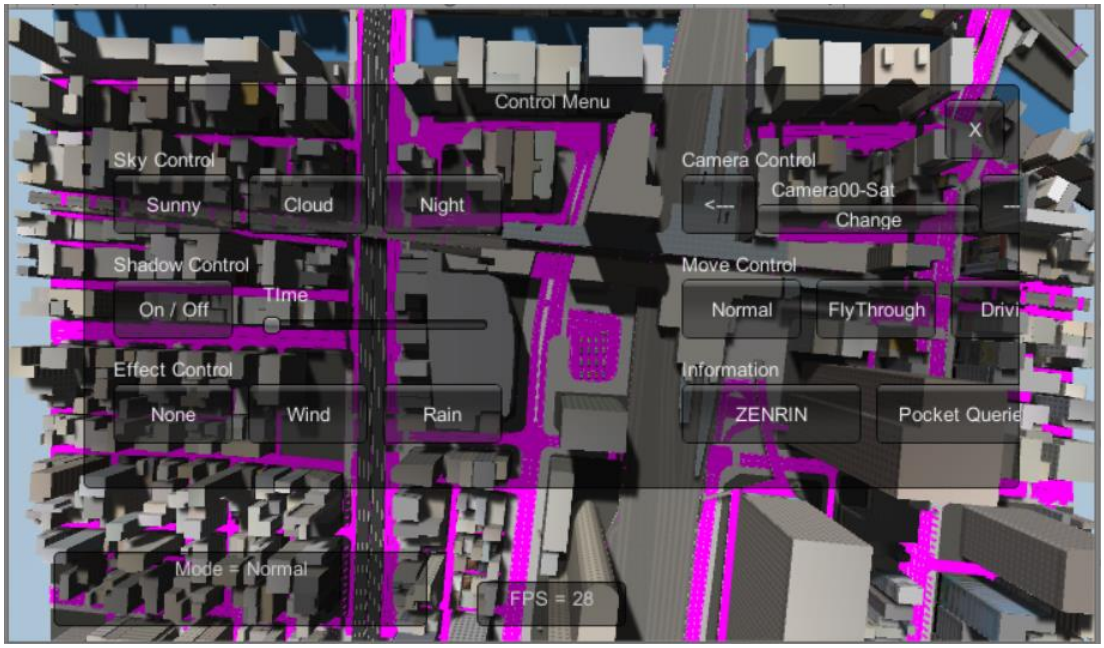

Figure 19. Zenrin 3D Map Menu

1. Sky control : Control the weather in the system. We can change to sunny, cloud, and night by changing the lightning in the system. Figure illustrates the differences, left side is the morning view and the right figure 20 is the night view.

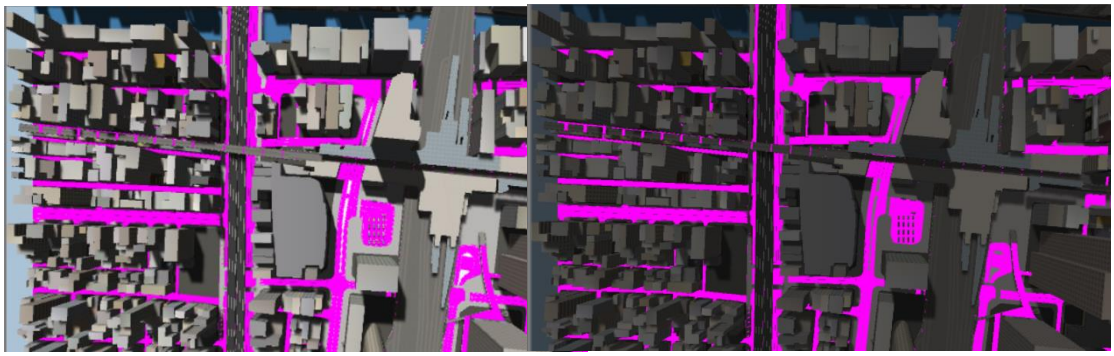

Figure 20. Different Weather Configuration

2. Shadow control : Turn on or off the shadow effect in the system. We can also adjust the time in the system by changing the shadow and lightning compared to periods of time in the day. Figure 21 illustrates the difference between using shadow or without shadow. As we can see, by using shadow in figure (a) provides

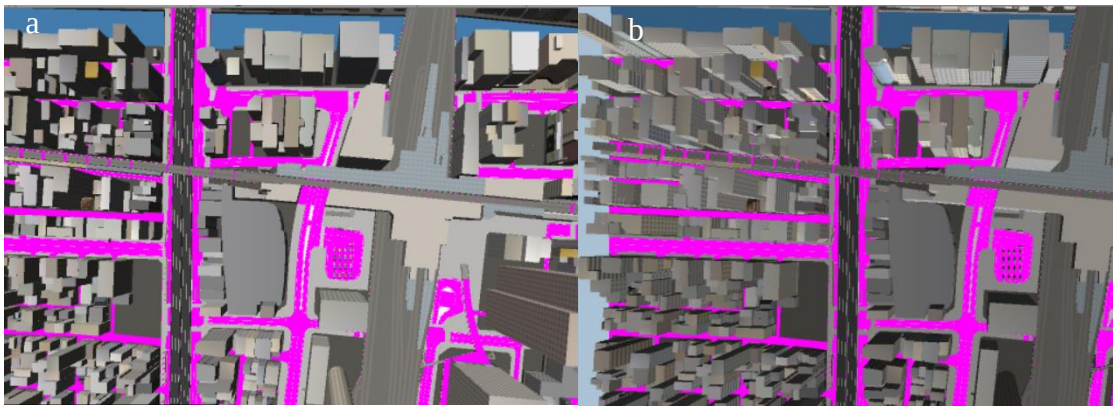

Figure 21. Turning on Shadow Control (a) and Off (b)

3. Effect control : Add wind or rain feature in the system to provide more realistic

environment in everyday life. As we can see in figure 22, zenrin

provides realistic environment by also adding rain feature so the user can enjoy the realistic environment similar with the real world.

Figure 22. Rain Interface in The City Environment

4. Camera control : Change the starting point of view in the system. Zenrin provides 10 starting spots. As illustrated in figure 23, these are examples of the point of view we can choose to see and observe things in the map. Several camera points of view have automatic move mechanism so the user can sit back and enjoy automatically.

Figure 23. Several Point of View Selections

5. Move control : Switch between normal, Fly Through, and Driving mode. These modes linked with certain character or vehicle provided. As illustrated in figure 24, zenrin provides several modes so the user can explore the world.

Figure 24. Fly Through Mode (a) and Driving Mode (b)

In figure 24 (a), the avatar is moveable throughout the environment. We can choose to move forward or stop, and we can also change the speed. The control is provided, by using “z” button for decelerate, “x” button for accelerate, arrow key up down left right to change direction. The scripting process for this feature is quite complex and figure 25 represents a part of the script used for realizing this feature, especially the moving mechanism.

```
// Move Forward
Vector3 forwardSpeed = transform.TransformDirection(Vector3.forward * Time.deltaTime * speed);
controller.Move (forwardSpeed);

// Speed Control
if (Input.GetKey("x"))
{
    speed += ACCELERATE * Time.deltaTime;
    if (speed >  MAX_SPEED)
    {
        speed = MAX_SPEED;
    }
}
else if (Input.GetKey("z"))
{
    speed -= DECELERATE * Time.deltaTime;
    if (speed < 0.0f)
    {
        speed = 0.0f;
    }
}

if (speed == 0.0f)
{
```

Figure 25. Moving Avatar Code

In Figure 24 (b), user can observe the movement of the car automatically and get a tour around the environment. Zenrin uses “navMesh” feature to set track for the car. Figure 26 represents a part of the script used in this feature.

```
    // Set destination
    targetNavMeshObjectNow = 1;
    navMeshAgentCompornent.SetDestination(targetNavMeshObjects[targetNavMeshObjectNow].transform.localPosition);
    this.transform.localPosition = startPos;
    this.transform.localEulerAngles = startRot;

    yield return new WaitForSeconds(0.5f);
    navMeshAgentCompornent.speed = CAR_SPEED_MAX;
    targetAICar.GetComponent<Animation>().Play("01_Run");   }

// Update is called once per frame
void Update () {
    if (navMeshAgentCompornent.remainingDistance < 0.1f)
    {
        targetNavMeshObjectNow ++;
        if (targetNavMeshObjectNow <= targetNavMeshObjectCounts)
        {
            navMeshAgentCompornent.SetDestination(targetNavMeshObjects[targetNavMeshObjectNow].transform.localPosition);
        }
        else if (targetNavMeshObjectNow >  targetNavMeshObjectCounts)
        {
            targetNavMeshObjectNow = 1;
            navMeshAgentCompornent.SetDestination(targetNavMeshObjects[targetNavMeshObjectNow].transform.localPosition); }}}}
```

Figure 26. Car “NavMesh” Code

6. FPS : Observe the frame per second rate of the system. FPS feature can be used by the user to observe the frame rate of the system. The user can decide to use this information for their advantages. Figure 27 presents the code used for presenting the FPS in the zenrin map.

```
using UnityEngine;
using System.Collections;

public class ViewFPS : MonoBehaviour {
    float timeA;
    int fps;
    int lastFPS;

    // Use this for initialization
    void Start () {
        timeA = Time.timeSinceLevelLoad;
    }
    // Update is called once per frame
    void Update () {
        if(Time.timeSinceLevelLoad  - timeA <= 1)
        {fps++;}
        else
        {
            lastFPS = fps + 1;
            timeA = Time.timeSinceLevelLoad;
            fps = 0;
        }}
    void OnGUI () {

    GUI.Box( new Rect(Screen.width / 2 - 50 , Screen.height - 40, 100, 30), "FPS = "+lastFPS); }}
```

Figure. 27. FPS Code

Illustrated in figure 19, the original file produced several bugs and incompatibilities with our environment such as missing asset link. We can see in the map that the map is missing color for game object streets and came out as pink color in the figure. To realize our own system, we need to solve the errors and revise the original file.

First, we solve the error by adding several codes to the current file to fix the shading problem. Next, we delete features that are not relevant to our system such as sky control, shadow control, effect control, camera control, and move control. We will use the FPS feature for calculation later. Several features in the original file relate to each other, therefore we need to trace the link and edit it one at the time to prevent bugs and errors.

After cleaning the file from the previous features, we can start to import another asset such as the “Kinect v2 Examples with MS SDK” and install the avatar in our system. We then change the “Main Camera” to the avatar point of view and the current system is ready to use and we can add our own code, game object, and many more. The main point of this step is how to delete the previous features asset without damaging the core part of the system. Figure 28 illustrates the ready 3D map after completing the editing part and figure 29 illustrates the combined avatar into the 3D map.

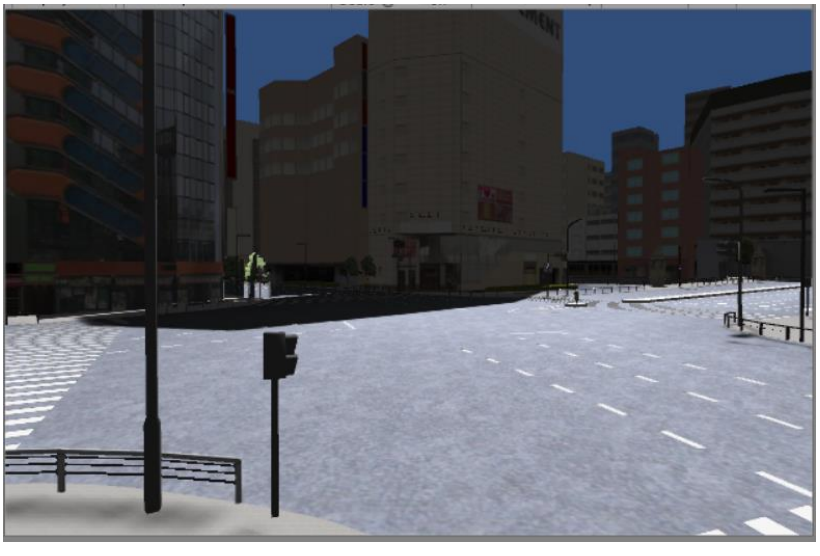

Figure 28. Post-Edit 3D Map Asset

In figure 28, we can see that the issue had been fixed by adding additional "UNITY_INITIALIZE_OUTPUT(Input,o); " line into the asset script. The line is functioned to link the environment city input each other.

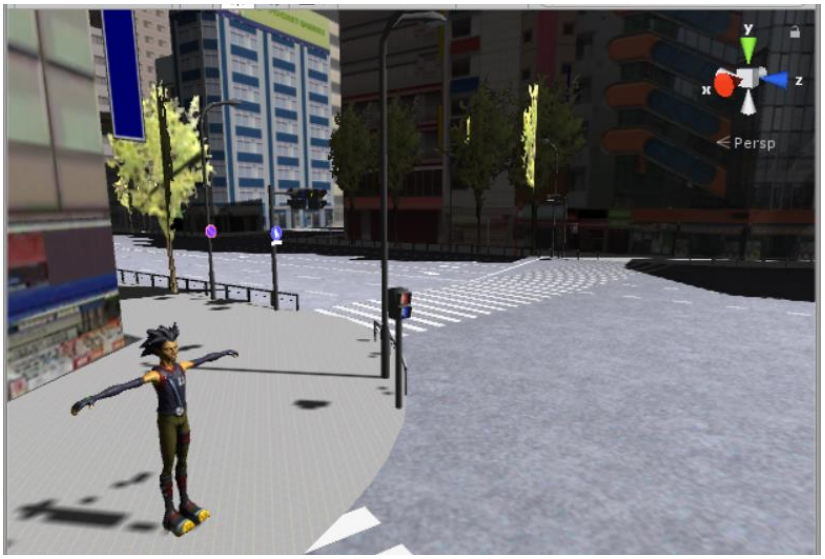

Figure 29. Imported Avatar in the 3D Map

4.2 System Settings

We use several devices for implementing our system. We choose to use Kinect depth-camera for capturing full-body skeletal data. We will use Samsung Gear VR as head-mounted display for providing an immersive experience while using our system. We will also use one unit computer with an Intel i7 processor, 8GB RAM, CPU speed 2.5 GHz. The system will be integrated with Unity and Microsoft Visual Studio for scripting.

Several SDK will also be used for integrating all the devices. We will use Microsoft Kinect for Windows SDK, Kinect Unity SDK and "Kinect v2 Examples with MS-SDK". After creating the system, we installed it to the HMD by using

server-based system that streams the Kinect data to the HMD in real time. Figure 30 shows the setup of the system.

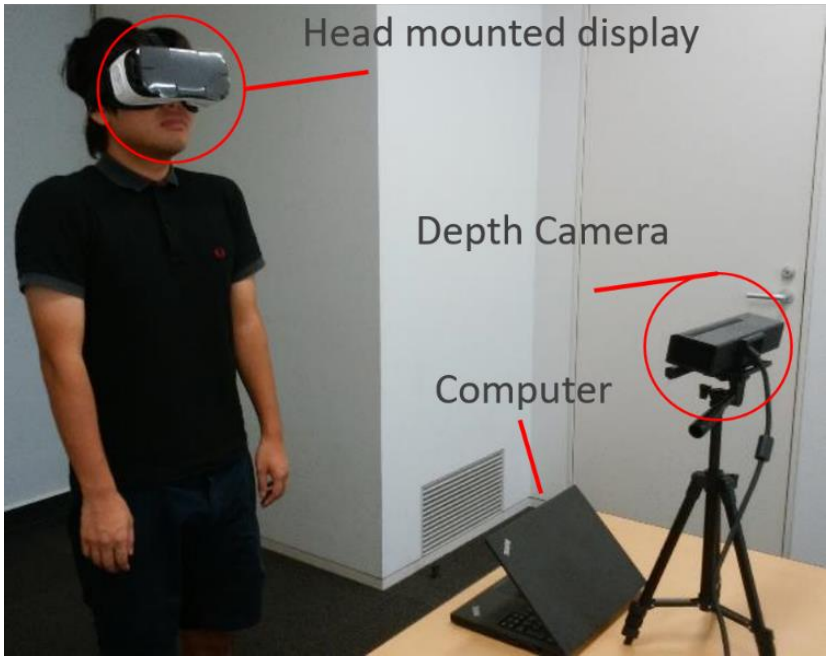

Figure 30. System Setup

We use Unity 3D Studio for creating our system, including connecting all the devices including Kinect Depth camera and Samsung Gear VR Head Mounted Display. Therefore, the first step we need to do is connecting all the devices in the Unity 3D Studio environment using several SDK. Figure 31 provides the illustration of devices in our system.

Figure 31. Devices Relationship Illustration

Between Kinect Depth-Camera and PC we use several SDK, “Kinect for Windows SDK 2.0” from the official Microsoft developer website, Kinect Unity plugin and “Kinect v2 Examples with MS-SDK”. We downloaded those SDK and plugins from the official website and then installed it.

“Kinect for Windows SDK 2.0” can be downloaded from https://go.microsoft.com/fwlink/p/?LinkId=403899 for free. After downloaded, we must install it in the computer and explore it features. There are various features in this SDK, but the focus is we are using the “Coordinate Mapper” and “Depth” data input to extract body joints and depth data for later use. After installing the SDK, we will get access to “Kinect studio v2.0” that has the basic feature to see depth, infrared, color, and 3D data. We can use this to do preliminary check of Kinect and record data of Kinect reading. Figure 32 illustrates the SDK browser that we can use to choose several examples of code and features from Kinect’s capability and figure 33 represents the “Kinect studio v2.0”.

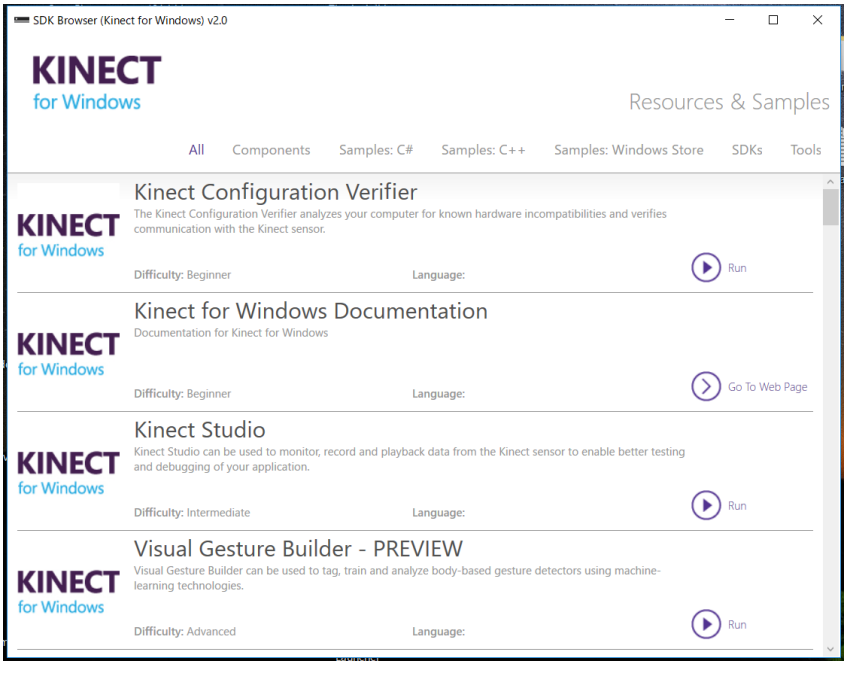

Figure 32. Kinect SDK Browser

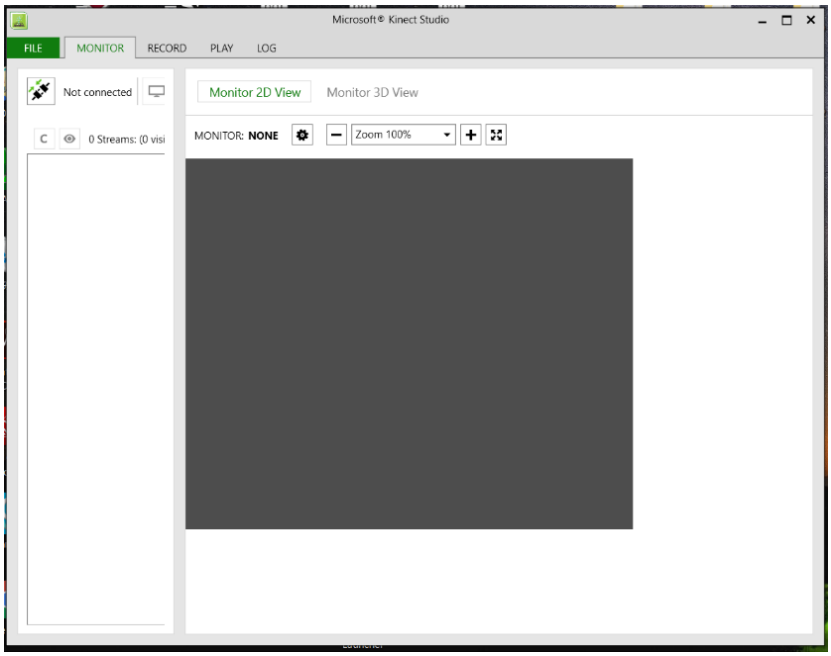

Figure 33. Kinect studio v2.0

The next step is how to connect the Kinect SDK with Unity 3D studio so we can use Kinect in the Unity environment. We would like to download "Kinect v2 Unity Plugin" from this link https://go.microsoft.com/fwlink/p/?LinkId=513177. After download the plugin, we integrate the plugin by importing it as "Custom Package" in the Unity environment. Figure 34 illustrates the asset being imported.

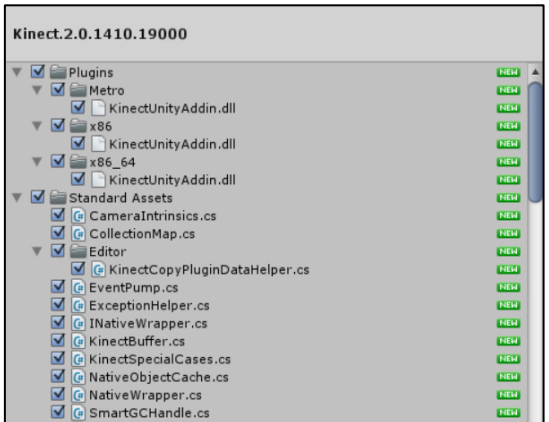

Figure 34. Kinect v2 Unity Plugin as Unity Package

The plugin provides several features such as the basic depth and body joints recognition, face recognition, and speech. We will use the first feature and would like to test it. We created unity sample application to check whether we can use the Kinect and its data in unity environment or not. Figure represents the trial unity application for checking the connectivity between Unity and Kinect camera. In figure 35 we can see that we successfully use Kinect to track body data and presents it in the unity environment.

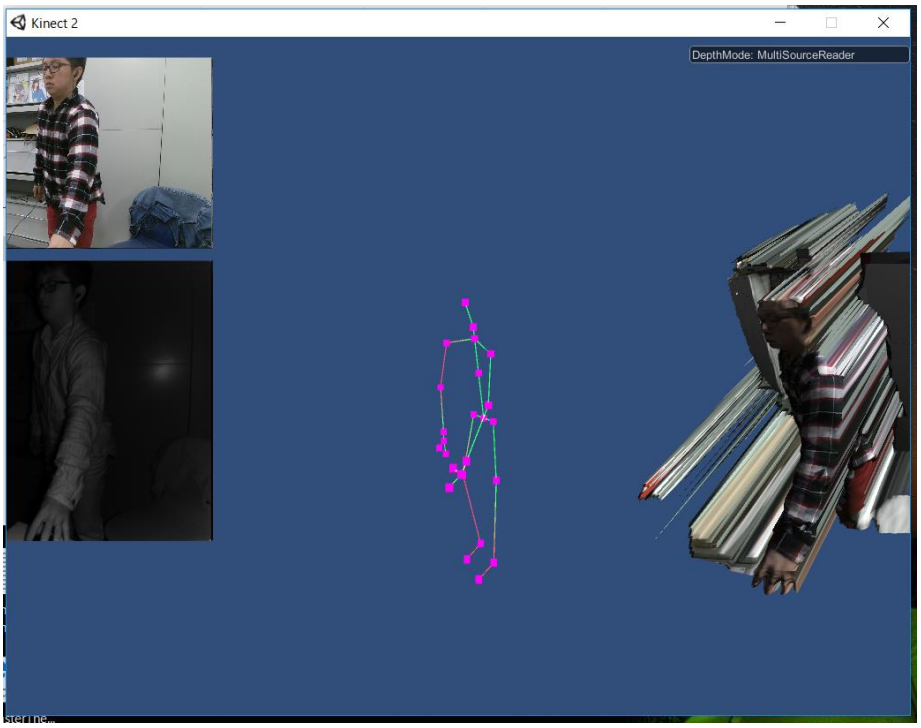

Figure 35. Capturing Data Using Kinect in Unity

4.3 Body Movement as Input

The next part is how we are going to exploit the Kinect feature of capturing body data into our system, in realistic 3D map. In our research, we decided to connect Kinect with an avatar created in Unity environment. The avatar must possess body joints that are similar with body joints tracked by using Kinect. After the model is created, we then add some script using C# programming language within the Unity to each of the avatar's joints game object and connect it to the user's body joints captured by Kinect real-time.

In our research, we use "Kinect v2 Examples with MS-SDK" asset to ease the connection of user's body joints with the avatar. This asset provides us with the avatar designed and integrated with each of body joints recognized by Kinect Microsoft SDK. We import the asset into Unity and set the Kinect in position so the system can recognize the user's body movement. Although the asset provides good integration of body joints and avatar, the avatar only able to move in very limited space with size same in the real life. Figure 36 illustrates the avatar in the asset

Figure 36. Avatar Connected with User's Body Joints

The white line block in the ground each represents 1 meter in real world, in other word the movement space of this avatar is very limited. In our system, we

would like to explore wide range of environment, so we need to add several adjustments to the avatar regarding with the movement area.

First, avatar integration with the user body should be done to move the avatar by using user's body movement. To realize this feature, the avatar created joint by joint as illustrated in figure 37. Afterward, some script to control the avatar need to be added in the unity. Figure 38 represents the component output in the avatar menu presented in Unity.

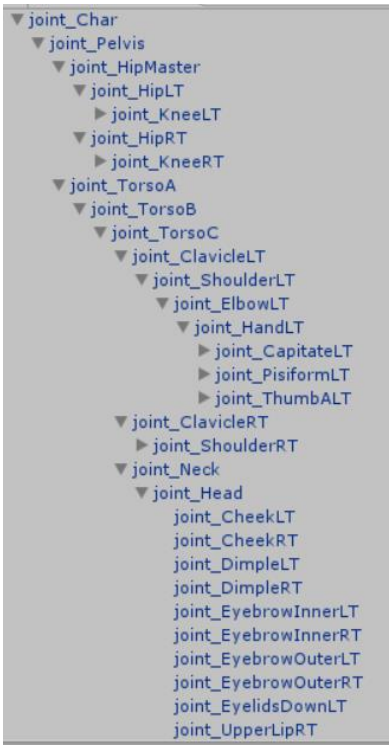

Figure 37. Avatar Joints Structure

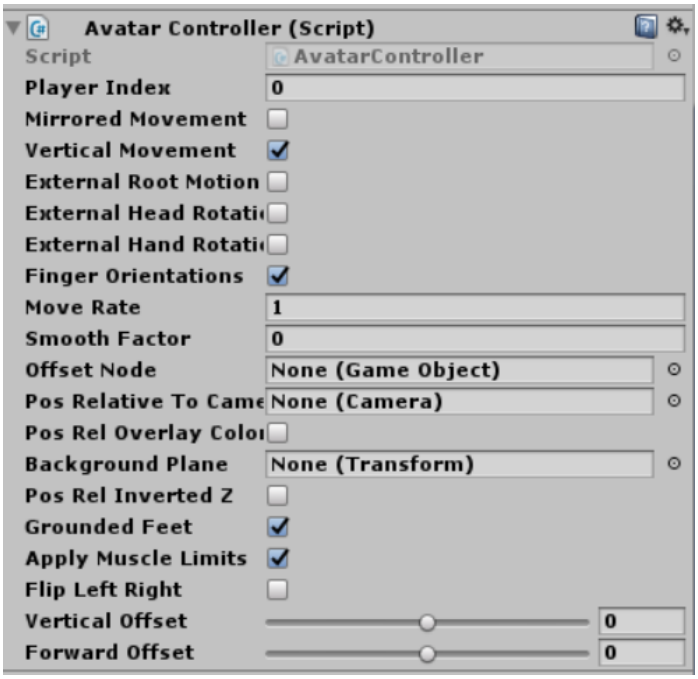

Figure 38. Avatar Component

By using this asset, we can achieve ease of integration between user body joints and the avatar. The asset also requires the user to calibrate their pose by posing like illustrated in figure 36. After the calibration, the system will match all the joints with the user's using data from Microsoft Kinect for Windows SDK.

4.3.1 Moving forward Mechanism

The avatar in the system acts as a game object in the unity and connected with the user body. "Kinect v2 Examples with MS-SDK" provides the connectivity between an avatar and user, but restricted in movement. The avatar corresponds with limited moving area as represented in figure 36 with blocks on the floor. One block represents 1 meter space in the real world, therefore if we want to move the avatar in a big virtual world, we also need big spaces in the real world [5].

To avoid using too many spaces, we use a "virtual traveling techniques" to cover and travel in such a large distance in the virtual environment [6]. This method allows the user to exploit only small place for their movement. In our research, we implement this method by making the user walking or running in place and the system will recognize it as movement in the virtual environment.

One remaining problem is how to make the avatar moves all around the virtual environment. In our research, we will use the basic First-Person Controller movement in the Unity studio to create the basic moving forward mechanism. We will use an GUI object from the Unity that can be moved by using keyboard key such as "w". When there is input of that button, the object will move in certain direction, as in our research moving forward. We attached the avatar into a cube (GUI Object) at its feet as illustrated in figure 39. We make the cube size so small barely seen, so we make as like the avatar's feet is touching the ground.

Figure 39. Attaching Avatar's Feet with A Game Object

The next step is doing some scripting for making the cube attached avatar's feed moving. The script is basically making the object move by certain input. For example, when the user hit "w" button in the keyboard, the cube will move forward. The code is represented in figure 40.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FPSController : MonoBehaviour {
    public float speed = 2f;
    CharacterController player;
    public GameObject eyes;
    float moveFB; float moveLR; float rotX; float rotY;

    void Start () {
        player = GetComponent<CharacterController> (); }

    void Update () {
        moveFB = Input.GetAxis ("Vertical") * speed;
        moveLR = Input.GetAxis ("Horizontal") * speed;
        Vector3 movement = new Vector3 (moveLR, 0, moveFB);
        movement = transform.rotation * movement;
        player.Move (movement * Time.deltaTime); }}
```

Figure 40. First Person Controller Code

After finished adding the simple moving mechanism, we would like to connect the cube with the speed control mechanism. Running speed acquired will make the cube move in certain speed by assigning speed value in the code and times it with the f variable in the unity. By using this method, we can make the avatar move in the realistic map. We chose to use this approach because of the avatar cannot move directly in a large area such as our realistic map.

4.3.2 Speed Control

Speed control obtained by using the speed data acquired from the equation (1) as the movement speed in the virtual world. The system first need to calculate hip and knee joint movement each frame and calculate its speed. The speed will be translated into input movement for First-person controller attached to the avatar.

First, we need to get the position of the joints and tracked it in real-time situation. We will track “central hip”, “left knee”, “right knee” to satisfy equation (1), (2), (3), (4) and get the speed of the user’s. All the joints are available in Microsoft’s official Kinect SDK and integrated with another SDK in our system. To get the

```
// get the joint position
KinectManager manager = KinectManager.Instance;
if(manager && manager.IsInitialized())
{if(manager.IsUserDetected(playerIndex))
{long userId = manager.GetUserIdByIndex(playerIndex);
if(manager.IsJointTracked(userId, (int)joint))
{
// output the joint position for easy tracking
Vector3 jointPos = manager.GetJointPosition(userId, (int)joint);
jointPosition = jointPos; if(isSaving)
{if((secondsToSave == 0f) ||
((Time.time - saveStartTime) <= secondsToSave))
{using(StreamWriter writer = File.AppendText(saveFilePath))
{string sLine = string.Format("{0:F3},{1},{2:F3},{3:F3},{4:F3}",
Time.time,
((KinectInterop.JointType)joint).ToString(),
jointPos.x, jointPos.y, jointPos.z);
writer.WriteLine(sLine); }
string sLine = string.Format("{0:F3},{1},{2:F3},{3:F3},{4:F3}",
Time.time,
((KinectInterop.JointType)joint).ToString(),
jointPos.x, jointPos.y, jointPos.z); Debug.Log(sLine);
```

Figure 41. Position Tracking Code

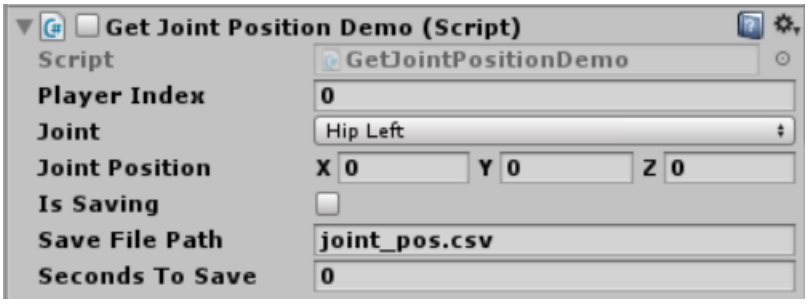

Figure 42. Avatar Joint Position Controller

Next, we will use the tracked joint and added it into equation (2) and (3) respectively for each knee. After acquiring two values of speed for each knee, we use equation (4) to get the maximum number from both of speeds. The largest number will be used as our running speed. Figure 43 illustrates the tracked position: pos_x, pos_y, and pos_z; and its speed: speed x, speed y, and speed z.

time	joint	pos_x	pos_y	poz_z	speed x	speed y	speed z
2.191	KneeRight	0.142	0.599	0.904	0	0	0
2.208	KneeRight	0.391	0.228	1.154	14.6471	21.8235	14.7059

Figure 43. Tracked Joint and Speed Calculation

For example, we obtained the speed y for KneeRight on time 2.208 is 21.8235 and 20.0003 for KneeLeft on the same time. After applying equation (3), we then conclude that the value 21.8235 is the speed of the user at that time. This data then will be used to control the speed of avatar in our system.

After acquiring the speed, we can use the speed as input to the avatar moving mechanism. The speed needed to be translated into unity configuration to get balance speed within the system (pixel) and real speed (m/s). The value presented in figure 43 is calculated in cm/s and we want to convert it into m/s unit for convenient. The method is to divide the value by 100 in the system and reduce the decimal number to make it simpler.

The next step is we link the speed obtained to the GUI cube object moving speed. This is done by adding the value of speed by our “running speed” variable value. The information is presented in the unity window real-time as represented in figure 44. The speed column will change according to the speed calculated by our system.

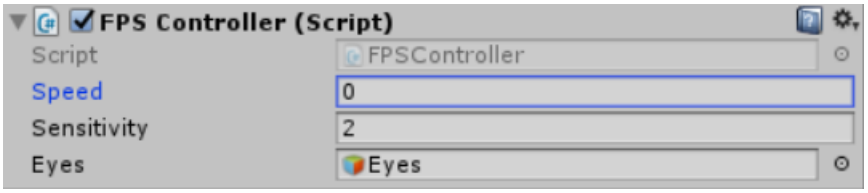

Figure 44. Unity Object Movement Speed Control Window

The running speed obtained in our system represents not the whole accurate speed of the user's running activity. Restrictions and assumptions, such as detection rate, joint tracking limitation, and devices specifications also play some roles in our system development.

4.3.3 Horizontal Rotation

Horizontal Rotation feature provides the system with capability to turn right or left and change direction of the user. There are lot of alternatives how to realize the horizontal movement, but we chose to let the user facing the depth-camera and keep their torso posture readable by the sensor. The sensor will then track the user's torso direction toward the sensor. If the user's facing certain direction, the point of view will also change to certain direction.

The features realized by making the user facing straight toward the depth-camera as illustrated in figure. This method is chosen because the users tend to make more accurate movement in exploring virtual world by translating the direction in which the users are looking [7]. In our system, we assume and set certain limitation due to the complexity and another prioritized feature. The users can only change their direction after they stopped the running activity. We

To realize this feature, we must measure the quaternion and is represented as Vector4 in the C# language. Quaternion is measured by calculating the rotation of the joint around the x, y, and z axis. Initially, we acquire the spine positions (x, y, and z). Next, we use the spine position to calculate the angle of movement such as illustrated by figure 9 in subchapter 3.3.3.

Next, we would like to set certain threshold for control. If the user orientation change to the right or left side more than the threshold value, we will change the point of view of the system according to user's direction. We set the threshold 45 degrees so the system will not easily detect the user's movement as command for changing direction. When the angle value corresponds with the threshold, the system will trigger the command to the GUI object attached to the avatar. We imbued the GUI

object with another command for mouse movement input and limit the moving axis just for x axis. Another axis already covered by the head-mounted display setting for looking around the environment. Figure 45 represents the core code for calculating the quaternion angle.

```
using System;
using Microsoft.Kinect;
namespace LightBuzz.Vitruvius
{   /// Provides extension methods for transforming quaternions to rotations.
    public static class OrientationExtensions
    {   /// Rotates the specified quaternion around the X axis.
        /// <param name="quaternion">The orientation quaternion.</param>
        /// <returns>The rotation in degrees.</returns>
public static double Pitch(this Vector4 quaternion)
{   double value1 = 2.0 * (quaternion.W * quaternion.X + quaternion.Y * quaternion.Z);
    double value2 = 1.0 - 2.0 * (quaternion.X * quaternion.X + quaternion.Y * quaternion.Y);
    double roll = Math.Atan2(value1, value2);
    return roll * (180.0 / Math.PI); }
/// Rotates the specified quaternion around the Y axis.
/// <param name="quaternion">The orientation quaternion.</param>
/// <returns>The rotation in degrees.</returns>
public static double Yaw(this Vector4 quaternion)
{   double value = +2.0 * (quaternion.W * quaternion.Y - quaternion.Z * quaternion.X);
    value = value > 1.0 ? 1.0 : value;
    value = value < -1.0 ? -1.0 : value;
    double pitch = Math.Asin(value);
    return pitch * (180.0 / Math.PI);}
/// Rotates the specified quaternion around the Z axis.
/// <param name="quaternion">The orientation quaternion.</param>
/// <returns>The rotation in degrees.</returns>
public static double Roll(this Vector4 quaternion)
{   double value1 = 2.0 * (quaternion.W * quaternion.Z + quaternion.X * quaternion.Y);
    double value2 = 1.0 - 2.0 * (quaternion.Y * quaternion.Y + quaternion.Z * quaternion.Z);
    double yaw = Math.Atan2(value1, value2);
    return yaw * (180.0 / Math.PI); } } }
```

Figure 45. Quaternion Calculation Code

4.3.4 Posture Control

We developed posture control feature to maintain the user's posture while running. User exercising alone may not be so clear for them how to maintain good posture [9]. The aim of our research is to provide realistic running support system for the user. We would like to exploit as many body movement as possible to make a more realistic system.

The key point of good running posture is the tilt angle of the upper body [14]. If the user's upper body posture is not properly done, the force produced when running will not be effective and lower the running performance. To prevent the loss

of force and energy while running, we developed a posture control feature to minimize this effect.

In our system the avatar and the users body joints are connected. We can obtain the position of each body joint (x, y, and z) by using the SDK. After obtaining the position, we will calculate this following angle:

1. Neck to head angle
2. Torso to neck angle
3. Angle between shoulders

We set thresholds within each calculation to keep the angle in certain position so the users will run in good posture. The angle calculation is similar with calculating the quaternion angle, we only need to adjust the equation variable.

4.4 Collision Control

As explained in chapter 3, collision control is designed to prevent the user hitting another object. We use the depth camera to extract the distance between user and object (in this case is the table we use for placing the depth-camera). We use the length of a vector in a 3D case by using equation (4) as presented below.

$$\sqrt{X^2 + Y^2 + Z^2}$$ Equation (4)

The next step is we need to convert the formula into C# language. The value derived will be then compared with the threshold in our research (0.5) for meters. Therefore, if the value is detected less than the threshold we will initiate the system to give warning to the user to step back. Figure 46 illustrates the used code.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class distance_from_object : MonoBehaviour {

public double Length(CameraSpacePoint point)
    {
        return Math.Sqrt(
            point.X * point.X +
            point.Y * point.Y +
            point.Z * point.Z
        );
    }

    // Use this for initialization
    void Start () {
        var point = body.Joints[JointType.SpineBase].Position;
        var distance = Length(point);
    } }
```

Figure 46. Calculating Distance Between User and Object

4.5 Additional Interfaces

We added several interfaces in our system such as “Time” and “speed”. All that information is presented to inform the user about their performances when using our system. In this section, we are going to explain several details of each section.

The time feature is done by using the stopwatch feature in the unity studio. We create a UI text object, place in into the screen and add a script into the text. When the user starts the system, the time will start calculating and stop when the user decided to pause. We added C# script and attach it to UI text, as presented in figure 47.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6 public class time : MonoBehaviour {
7     public Text timerText;
8     private float startTime;
9     // initialization
10    void Start ()
11    { startTime = Time.time; }
12    // Update is called once per frame
13    void Update () {
14        float t = Time.time - startTime;
15        string minutes = ((int) t / 60).ToString ();
16        string seconds = (t % 60).ToString("f2");
17        timerText.text = minutes + ":" + seconds; } }
```

Figure 47. “Time” Feature Code

The “speed” mode is acquired by presenting the speed acquired from the speed control mechanism. This data is basically connected directly to the speed data derived from equation (3). The scripting process is similar with time and distance interface, we only need to change the data source with the “running speed” variable.

Chapter 5 Evaluation and Discussion

5.1 ISIPS Symposium 2017

In this research, we proposed to evaluate our system by qualitative evaluation. Qualitative evaluation includes ISIPS Symposium 2017 by doing poster presentation and getting feedbacks regarding our system. Responding the visitor's questions and feedbacks, we also had discussions and will be included in the following section.

We attended the ISIPS Symposium 2017 held by Waseda University IPS in Kitakyushu Japan. In this event we presented a poster presentation and asks for visitor's opinion, feedbacks, and suggestions for our research. Figure 48 shows the ISIPS Symposium event atmosphere.

Figure 48. ISIPS Symposium 2017

In this event, we set a poster in a panel board and several devices included one unit laptop, Kinect depth-camera, and Samsung Gear VR Smartphone and Head Mounted Display as represented in figure. Visitors then can try the demo version of our research at the time. Interactive questions and answer session produces several feedbacks that are very useful for our research.

There are lot of questions and suggestions obtained from the visitors, therefore we need to summarize and evaluate each of the comments presented below.

a. “Maybe it’s better to add running belt below the user so will be more natural?”

The visitor mentioned about installing running belt below the user’s running area. The image is like a treadmill on which the user will run above a rubber belt. Through this comment, the user is emphasizing about the feedback from our system. The visitor argues that by giving running belt, user will feel more natural due to feedbacks from the running belt. Thus, the user running movement will be more similar with running in real world.

Giving running belt to provide feedback to the user is one possible alternative that can be considered in our research. Although adding a running belt will not cost too much and relatively easy to be done, we chose not to add a running belt due to several considerations. The first consideration is that our system focus is mainly in the use of human body movement as input. We are interested in using the depth-camera to capture as much body movement as possible to provide a realistic system.

The second consideration is if we add running belt into our system, we will reduce the mobility aspect of our system. It means that we would likely be restricted in terms of our movement and control of the system. If the user wants to move the system installment into another rooms or adapt the system into another virtual world or platform, more adjustments must be made regarding the running belt.

Despite several advantages provided by adding running belt into our system, we decided not to install the running belt because of its risk to reduce our system adaptability and mobility.

b. “How to control the user from hitting table or another thing in the room?”

The visitor places her concern at the safety of the user. She argues that without proper handling, user would likely to hit table or another object in the room. The

visitor also suggested to add features to control this behavior and minimize the risk of user hitting objects and getting injuries.

We found this argument very interesting and helpful. Safety control is also considered important in our system and realizing it also will be a challenge. After some discussions and arguments with the visitor, we summarized the idea by adding some limited area in the room and make the system tracks user position in that area.

As presented in chapter 3 and 4, “Collision Control” is designed to prevent the user for getting injuries and hitting another object in the room. We decided to add this feature because of two reasons, the user’s safety and integration with depth-camera. We wanted to use the depth-camera in our system to accommodate as many features as possible. The depth data acquired from the camera makes it easier to calculate distances between objects. Therefore, we can realize this feature by using our devices.

c. “How to make sure the user runs in good manner is one problem, please consider.”

The visitor argues that we need some control mechanism to make sure the user’s running in good manner. If user’s not running in good posture, the system will not be considered realistic because lack of aspect to influence the user to run like in real life. We agreed with the visitor’s argument.

We decided to develop posture control which presented in chapter 3 and 4. Posture control will keep the user running in good posture and at the same time reminds the user to change their posture. The user will able to get the notification from their head mounted display while using our system.

Posture control is considered important in our system and acting as bridge with running in real world. Main concern regarding this research is the similarity with real world running. The users are expecting by using this system, they will be able to experience and get the same atmosphere. If the system cannot make the user excited or exhausted, the reliability of the system is questionable. Thus, one

way to make sure that the user feels the same way is by checking their posture from time to time.

d. “Why don’t you consider adding another feature such as heart rate so the user performances can be improved?”

The visitor argues that it is better to add features such as heart rate recognizer in the system. Heart rate recognition technology already realized and the consumer product such as smart watches have already been sold in the market. The availability of similar feature in the market makes this feature less important in our research.

Our research mainly focuses on the computer science aspect rather than the mechanical aspect. The research aim is to develop a running support system using body movements. Body movements is used to track the user, connecting the user with an avatar, and move the avatar in the 3D map. Heart rate on the hand, have the possibility to make things more complicated and not related with our research.

Body movement tracking is realized by using Kinect depth-camera in our system. Kinect tracks the body movements by locating body joints and mapped in real-time. The connection between body joints movement and heart rate is not direct and we cannot directly extract heart rate data just from the joints location. Even if we develop certain algorithm to accommodate this feature, the result will not be so satisfying and it would seem better to use the available devices in the market.

e. “Another thing I think you should consider is how to evaluate the system and analyze whether the system is helpful.”

The main question from a visitor is how to evaluate the system so we can know that our system has benefits and give some influence on the user. The visitor stated that our system was still lacked evaluation plan. In this matter, we spent more time and discussing about our research’s evaluation plan.

There are two major evaluations sectors, qualitative evaluation and quantitative evaluation. Qualitative evaluation will be done by asking questionnaire and little interview with several participants about our system. We will ask several potential users to try our system and then give some feedbacks. This evaluation method will produce subjective feedbacks and related to the design and interface of the system itself. Regarding the objective evaluation, will be discussed with the quantitative evaluation.

Quantitative evaluation will be done by comparing the running speed of the user while using our system with the real running. First, we will ask the user to use our system and run for a designated period. Then we track their running speed for those time. The next part, we asked the user to run in the similar environment and measure their running speed. By comparing running speed, we can check the similarity of our system with running in real life and whether our system is supporting the user running activity or not.

Chapter 6 Related Works

6.1 Using Body Movements in VR System

The use of body movements has been used as method for controlling the VR system in the past. Body movements mentioned in this manner is using body joints or data based on body joints such as joints location changes [3]. The location changes are tracked by certain devices and sensors, some divided per framework and some done in real-time. This method is considered popular and used in various research due to its simplicity and feasible to realize [6].

In the past, realizing this feature in real-time was difficult due of device limitations [3]. However, after the development of Kinect, the users can run the system with proper accuracy, robustness, and affordable. Kinect tracks several body joints and translate it as depth-data, so the users can get the information its positions in the real world. Shotton et. al. proposed the principles of Kinect Skeletal Tracker by capturing single input depth image and infer a per-pixel body part distribution [11].

One example of system developed using Kinect is done by Tanaka and Hirakawa [9]. The authors developed a training system using Kinect and head-mounted display as illustrated in figure 50. The users can use the system for strength training such as squat, push up, etc. When the users do certain posture, the system will evaluate their postures and give additional feedbacks such as user performance and posture condition as illustrated in figure 51.

Another main concern is how to implement a system based on a depth-camera in a virtual environment. A VR environment sometime consists of vast and large size of space, such as virtual world, virtual map, etc. Exploring this area requires similar movement spaces in the real world, which will lead to using up too much spaces. To solve this problem, [6] proposes using “virtual walking” method for minimizing the required space. Wilson et. al. use “walking in place” method for realizing VR system exploration [5].

In our research, we would like to develop a system that can recognize the running motion from the users. Initially, we must define the parameters for good running to improve the usability of our system. There are several criterions for determining a good running posture, but the main point is the body angle [16]. The body angle is a key point of force distribution. If the user's body angle while running defects from the proper running posture, then the users would likely to get low performances. For minimizing this effect, several body angle controls are needed. Taylor et. al. uses head, neck, shoulders, and hands angle to maintain the users running posture [17].

6.2 Realistic 3D Map in VR System

The use of realistic 3D map in a VR system is implemented for fire evacuation training system [4]. Sookhanaphibarn and Paliyawan use a university based virtual environment in their system, so the users can walk around the area and train themselves regarding evacuation process when a fire emerges.

In the game industry, there are various type of game that uses realistic 3D environment [18]. Most of the environments are artificial environment for fictional purpose. The availability of realistic 3D environment that is like the real-world environment still quite low. On the other hand, Zenrin with its 3D Map asset in Unity 3D Studio provides vast and realistic environment of Japan's cities. The availability of realistic environment opens the possibilities for research to be done in various area.

Chapter 7 Conclusion

7.1 Summary

We designed a running support system in a realistic 3D Map using body movement as input. The user can explore the vast and detailed 3D map while running. The next step is we will add more movements features in the system to realize more realistic body movement tracking. We will compare the user's speed while running using our system running with running in real life. By comparing speed, we can prove that our system can be used as alternative for running in real life.

7.2 Future Works

In our research, we explored the use of depth-camera to capture realistic body movement and implemented it to create a running support system in realistic 3D map. During our work, we found several limitations and potential improvement points, such as:

1. In the future, we can explore more body movements to be included in the feature to provide more realistic experiences for the user
2. We will do thorough user testing to measure the system objectively
3. We can try to implement our system in different map platform such as google street view and explore it benefits for the user

Acknowledgement

Special Thanks goes to Professor Tanaka for guiding me through my research from the beginning until the end. His judgement, advices, and insights proven invaluable to my research and my life as a Graduate School student in Waseda University IPS, Japan.

Thank You also for my parents, several fund sources that had been kindly supported me financially throughout the years. Without your help and supports, I would not be able to graduate and focus my mind toward the goals.

To all my friends, especially IPLAB members who have been with me from the beginning, IPLAB graduated members, and currently active IPLAB member, Thank You for your support. Good luck for your future and let use keep our contact.

Lastly, I would like to Thank you all, people I have met up until today. You have made my life full of precious stories, memories, and life-time experiences. I will not forget all those lessons life has taught me and wish me luck for my following years.

REFERENCE

[1] Zenrin 3D Asset. http://www.zenrin.co.jp/product/service/3d/asset/

[2] A. Karina, T. Wada, and K. Tsukamoto, "Study on VR Sickness by Virtual Reality Snowboard," Transactions of the Virtual Reality Society of Japan, vol.11, no.2, pp.331-338, 2006.

[3] C.Subodha, "Real-Time Human Movement Mapping to a Virtual Environment," Region 10 Symposium (TENSYMP), IEEE, pp. 150-154, 2016.

[4] K. Sookhanaphibarn and P. Paliyawan, "Virtual Reality System for Fire Evacuation Training in a 3D Virtual World," IEEE 5thGlobal Conference on Consumer Elevtronics, pp. 1-2, 2016.

[5] P. T. Wilson, K. Nguyen, A. Harris, B. Williams, "Walking in place using the Microsoft Kinect to explore a large VE," SAP'13 Proceedings of the ACM Symposium on Applied Perception, pp. 27-33, 2014.

[6] G. Bruder, F. Steninicke, "Implementing Walking in Virtual Environments," Human Walking in Virtual Environments, Springer Science+Business Media New York, Chapter 10 pp. 227, 2013.

[7] B. Williams, M. McCaleb, et. al, "Torso versus Gaze Direction to Mavigate a VE by Walking in Place," pp. 67-70, SAP 2013, ACM, 2013.

[8] K. Tollmar, D. Demirdijan, and T. Darrell, "Gesture + Play Exploring Full-Body Navigation forVirtual Environments," Proceedings of the 2003 CVPRW. pp. 1-8, 2003.

[9] Y. Tanaka and M. Hirakawa, "Efficient Strength Training in a Virtual World". pp. 1-2, 2016 International Conference on Consumer Electronics-Taiwan, IEEE, 2016.

[10] H. Sharp, Y. Rogers, and J. Preece, "Interaction Design: Beyond Human-Computer Interaction". Wiley, 2007.

[11] J. Shotton, A. Fitzgibbon, M. Cook, et. al., "Real-Time Human Pose Recognition in Parts from Single Depth Images," pp. 1297-1304, CVPR, 2011.

[12] F. Multon in F. Steinicke et al. (eds), "Human Walking in Virtual Environments" (Chapter 8, "Sensing Human Walking: Algorithms and

Techniques for Extracting and Modeling Locomotion), Springer Science+Business Media New York, 2013.

[13] Schellenbach M., Lovden M., Verrel J., Jruger A., Lindenberger U., "Adult Age differences in familiarization to treadmill walking within virtual environments". Gait Posture Vol 31 pp. 259-299, Elsevier, 2013.

[14] G. Bruder and F. Steinicke in F. Steinicke et al. (eds), "Human Walking in Virtual Environments" (Chapter 10, "Implementing Walking in Virtual Environments"), Springer Science+Business Media New York, 2013.

[15] L.H. Sloot, M.M. van der Krogt, J. Harlaar, "Effects of adding a virtual environment to different modes of treadmill walking," Gait & Posture Vol. 39 pp. 939-945, Elsevier, 2014.

[16] Bolibar, J. "Kinect Audio-Runner: Audio Feedback for Improving Performance in Long-Distance Running," Master of Science Thesis Stockholm, Sweden, 2012.

[17] Taylor B., Birk M., Mandryk R. L., Ivkovic Z. "Posture Training with Real-time Visual Feedback". Proceedings CHI '13 Extended Abstracts on Human Factors in Computing Systems, pp. 3135-3138, 2013.

[18] "From horror fest to shoot-'em ups, here are the 20 best Oculus Rift Game". https://www.digitaltrends.com/gaming/best-oculus-rift-games/. Accessed 22nd January 2018.