University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

ECE 498B

Final Report

(Virtual Reality Assisted Interior Design System)

Prepared by

Group 52

Group Members:

Du, Zhechen (20340480)

Duanmu, Zhengfang (20388152)

Li, Debin (20389489)

Li, Lingyun (20369079)

Pan, Rui (20383546)

Consultant: Roehl, Bernie

13 February 2015

Table of Content

Abstract

Home renovation can be a headache for homeowners. It is hard for the home owners and renovators to virtualize the entire 3D space. The Virtual Reality Assisted Interior Design system can solve this problem by rendering the room in a three-dimensional virtual reality world. The system allows user to interact with objects within the virtual environment. Leap Motion controller and a computer are integrated within the system. The Leap Motion controller is used to capture user's hand movements. Then, machine learning knowledge is extended to help recognize the pushing and dragging gestures, which are used to interact with the virtual objects. In addition, head movements of the user can be tracked by a webcam on the computer. The computer display changes dynamically as the user's head position changes. This creates the illusion of three-dimensional space. User can view the room in different angles and positions by simply moving their head around. Image processing and real time programming knowledge are used in this sub-system. The system can offer both the designers and the homeowners an accurate representation of room after renovation. The Virtual Reality Assisted Interior Design System brings a whole new experience into interior design.

Acknowledgement

We would like to express our appreciation to Bernie Roehl, who suggested ideas and helped the team throughout the final year design project. Without his guidance and persistent help, we would be wasting time on doing trail and errors on different solution to our problem.

We would also like to thank Professor Daniel Davidson, who marked our reports and provided us feedbacks. Without his constructive feedbacks, this report would not have been in the quality it is now.

List of Figures

List of Tables

1. High-Level Description of Project

1.1 Motivation

In today's interior design industry, it is important for both the home designers and the homeowners to observe 3D models of the room that they are trying to renovate. Current home renovation software packages provide 3D modeling, however, most of them requires the user to be sitting in front of a computer in order to change objects within the modeling environment. Users have to interact with the virtual environment only using mouse and keyboard [1]. This can be inconvenient for some users. In some locations, like the lobby of a home renovator firm, not all users can have access to the computer.

1.2 Project Objective

The project for the Final Year Design Project group is called "Virtual Reality Assisted Interior Design System". The goal is to provide users a new way of modeling interior design and home renovation.

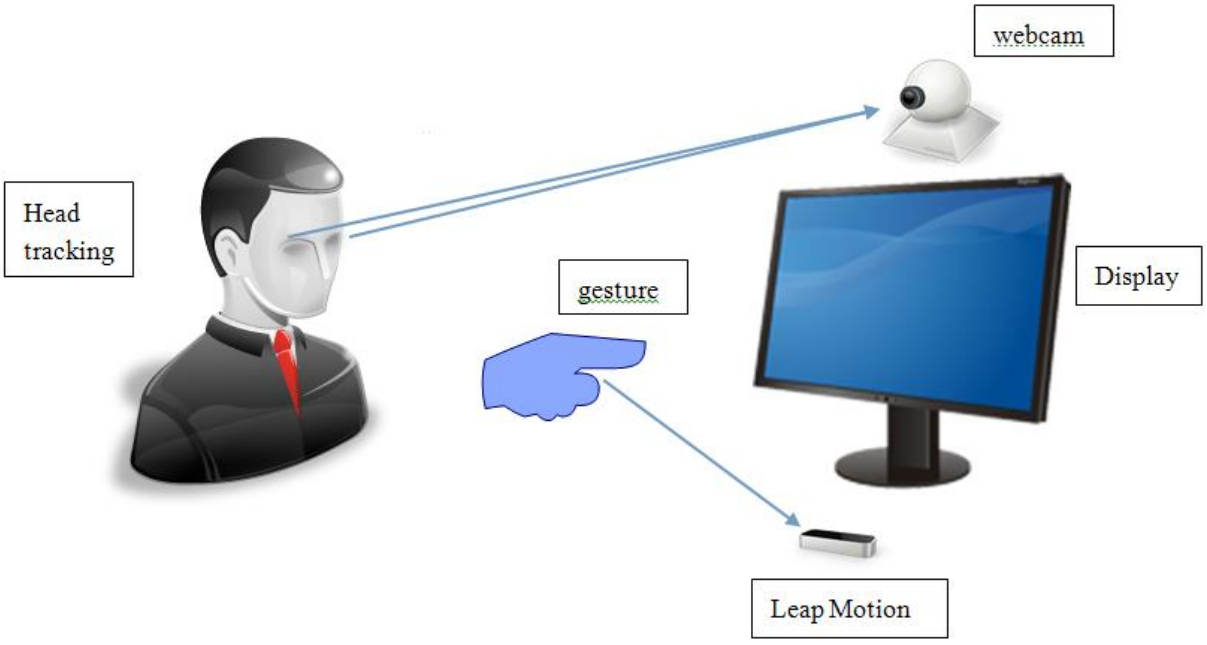

Figure 1. Virtual Reality Assisted Interior Design System Illustration

The goal of the project is to offer home owners a more natural way of visualizing their home renovation project, utilizing dynamic perspective. In order for interior designers to successfully design and remodel a home, three dimensional modeling is an essential part of the project [2].

The Virtual Reality Assisted Interior Design System allows user to control virtual objects inside the screen using gesture movements. The gesture is detected using the Leap Motion controller and further processed in the computer. In parallel, a webcam is used to track user's head movement. By knowing user's position, the perspective on the display will change dynamically. The dynamic perspective should allow user to see an illusion of three dimensional space. By combining the hand and head movements, the user can interact with the computer without using their mouse and keyboard.

1.3 Block Diagram

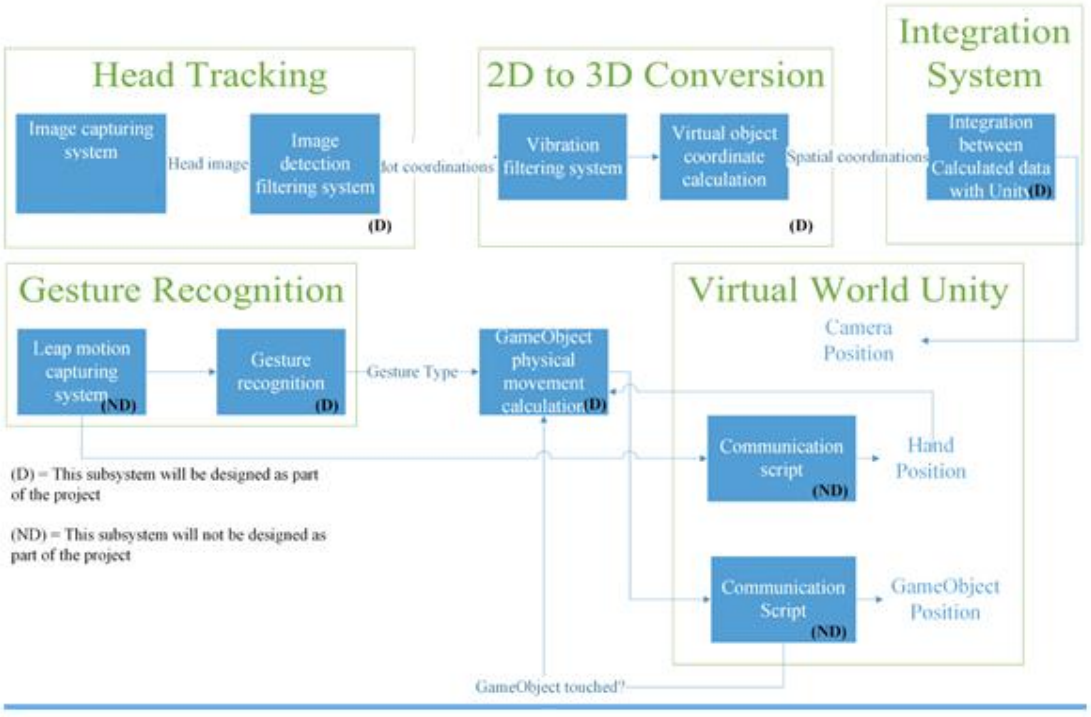

Figure 2. Block diagram of the project design

There are two main subsystems to be considered in the design: gesture recognition and head tracking. The software tool Unity will combine the outputs to update the image dynamically.

The first subsystem, gesture recognition system, performs data collection and gesture pattern recognition. To begin with, a Leap Motion controller, a finger recognition hardware, captures human hand movement, and the Leap Motion API provides the interface to process the captured image into analyzable data. Then, sufficient training dataset is utilized to generate a model for each predefined gesture. According to the identified gesture types, the system will determine the physical movement of the virtual objects being touched.

The user's head movement is captured by webcam using user's eye as position indicators. The image frames are first filtered by the application to get the 2D coordinate of each eye. Afterward, the 2D coordinate is the again filtered to remove any noise or vibration. Furthermore, the 2D coordinate will be calculated and transformed into 3D space. Finally, the 3D coordinate is fetched to Unity for further processing.

Unity will be used to integrate all subsystems together. It will take in processed and calculated input data using Dynamic Link Library (DLL) script. The DLL will be passing processed data from the Head Tracking system to Unity, telling it where to move its Camera position in order to reflect the user's head movement. This will emulate the dynamic perspective needed to be achieved by the project.

2. Project Specifications

2.1 Functional Specifications

Table 1. Functional Specifications

Essentials
The webcam should be able to detect human face movement against the interferences of the environment, for example, the ambient light
The webcam should be able to track human faces from at least 30 cm away from the remote, and able to track the in a fan shaped area radius of 1.5m, Diagonal Field of View 60 °.
The webcam and system should be able to process the image at 24fps

The head tracking system is design for only one person use, the tracking accuracy should be more than 95% using formula
The virtual scenario should be updated in real-time according to hand and head movement
The delay of the display should be less than 0.1 seconds. Delays are introduced in each subsystems of the project. The summation of each delay should not exceed two frame in the Webcam, which is 0.066 seconds.
The system must be able to track hand in an area dimension of 25cm * 25cm * 25cm
The system should be able to process hand movement at a speed of 0.2m/s
The virtual reality system should show a room
The system should convert hand spatial information into xyz-coordinate data
In Unity, the furniture grabbed by the virtual hand can be moved by the operator using hand gestures
The operator should be able to control the speed of furniture movement
The gesture recognition model in the system should have a high recognition rate. We currently support three gestures: push, drag, drop and restart. A successful model should have a recognition rate greater than 90%
The program should require less than 1 GB memory
Non-essentials
Leap Motion controller should be able to detect all five fingers
The system shall only recognize one hand and ignore all other hands
The system should factor in all the external forces such as the gravity and friction
The system should have a headphone to perform audio output, Audio should be in surround sound format
Operator should be able to hear changes in audio as operator's head changes position in relative to the audio source inside Unity scene

2.2 Non-functional Specifications

Table 2. Non-Functional Specifications

Essentials
Cost: The total cost of the system should be lower than 500 Canadian dollars.
The equipment used in the process are two LED lights, Leap Motion Controller, webcam, headphone, and computer
Non-essentials
Usability: User should take no more than 3 minutes to install all components of the system each time.
Size: All hardware components of the system except computer and display should be able to fit in a 20cm by 20cm by 20cm box.
The webcam and Leap Motion controllers should be powered by USB port
The system should have a good expansibility

3. Detailed Design

3.1 Subsystem Gesture Recognition Design

In the Gesture Recognition Subsystem, we designed a model that automatically recognizes a set of predefined gestures to be used to perform the physics movement calculation of the virtual object in Unity. In order to record the human hands movement, a Leap Motion controller [3] was used. The Leap Motion controller was able to track both hands and all 10 fingers with pinpoint precision. The advantages of Leap Motion compared to the other motion capture devices, such as Kinect motion sensor [4] and Thalmic Labs Myo armband [5], were precise, cheap and compatible. The Leap Motion Application Programming Interface (API) provided the interface for Unity game engine to retrieve the spatial information of hands and fingers, as well as the virtual object being touched by the virtual hand in Unity. This functionality allowed us to apply the gesture recognition technique in the project without worrying about inadequate information.

According to the project specification, different gestures should have different impact on the virtual object in the Unity scene. Therefore, the system had to be able to recognize the human hand gestures. As it was already demonstrated by Adaptive System Laboratory in their paper, the position difference and angle between palm and fingers contained discriminative information about the corresponding gesture [6]. Based on this information, different gestures could be accurately distinguished. With the above information in mind, we proposed an algorithm to satisfy all the design specifications. Figure 3 shows the workflow of the algorithm.

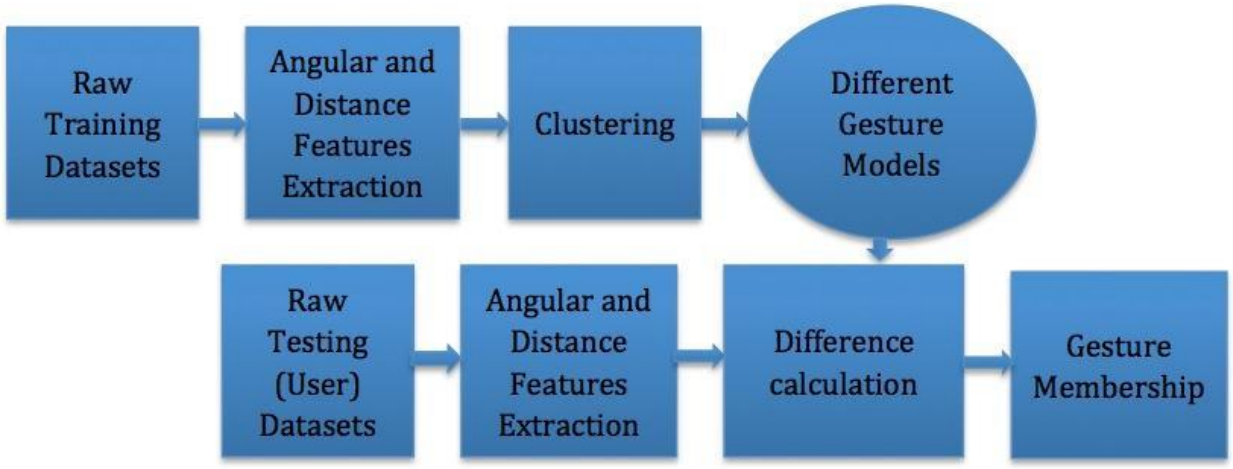

Figure 3. Gesture Recognition Subsystem Design Workflow

As it is shown in the Figure 3, we started with sampling several predefined gestures within the group. We set the sampling duration to be 2 seconds using Leap Motion controller. The obtained datasets were then used to train the gesture recognition model. However, the raw dataset that contained the spatial information of hands and fingers was meaningless to the gesture recognition directly. Therefore, we had to perform gesture feature extraction by calculating the spatial difference of fingers and palms and the angle between each fingertips and palms. A simple featured data had the following form.

(distance1, distance2, distance3, distance4, distance5, angle1, angle2, angle3, angle4, angle5) = (3.4, 5.2, 5.5, 4.8, 3.6, 1.5 degrees, 2.5 degrees, 2.2 degrees, 2.0 degrees, -0.8 degrees)

The first feature data of each gesture was selected as the centre values of each gesture model. Then the rest of training gesture data was compared with each model one by one. The cost function of the system was defined by the formula 1:

$$Cost = \sum_{i=0}^{n-1} \sqrt{(x_i - x_{i\,group\,j})^2} \qquad (formula\ 1)$$

Here *n* was the number of features, x_i was the i-th feature of a testing sample, and $x_{igroupj}$ was the average value of i-th feature in the j-th group. The sample was identified into the group with the minimum cost. Thereafter, the centre values of the gesture model were updated by the average values of the centre values of the model and the incoming feature data. The process would continue until the last data sample was clustered. As a result, the well-trained gesture models were eventually generated. The model would be used to identify the pre-defined gestures. Here is a piece of source code:

```
centre1 = featureData{1};      % pre-known push
clusters{1,1} = featureData {1};   % first 1 represents gesture number, second parameter indicate the number of samples in the gesture model
centre2 = featureData {2};     % pre-known drag
clusters{2,1} = featureData {2};
centres = {centre1, centre2};

for i = 3:length(featureData)
      for j = 1:numCluster                % numCluster is an input parameter, 2 in this case
         Distance = norm(featureData{i}-centre{j});
         if ( Distance < minDistance )
                  minDistance = Distance;
                  clusterNum = j;          % compute the membership
         end
      end

     % calculate the number of samples in the cluster
      nonemptyCells = ~cellfun(@isempty,clusters(clusterNum,:));
      numSample = length(clusters(clusterNum,nonemptyCells));
     % update the centre value
      clusters{clusterNum,numSample+1} = featureData{i};
```

```
    centreVs{clusterNum} = (centreVs{clusterNum}.*(length(clusters{clusterNum} - 1)) +
featureData{i}) ./ length(clusters{clusterNum});
end
```

The similar gesture capturing and feature extraction processes were employed to obtain the feature information of the testing gesture. During each program frame, the Euclidean distance of the sampled data with the centre values of each model was calculated, and the model with the least distance would eventually take the membership of the data sample.

A decision-making matrix was employed to evaluate different aspects of the proposed algorithm, which were Spatial Difference Only approach and another commonly used gesture recognition model, namely Hidden Markov Model (HMM) [7]. The description of an implementation of HMM was beyond the scope of this report. A detailed introduction of HMM could be found in the referenced paper. For the third method mentioned above, it simply computes the positional difference between all fingers to determine which gesture is performed. For example, when all fingers have a spatial difference under a certain threshold, the gesture is treated as drag.

Table 3. Gesture Recognition Model Decision Making Matrix

Criteria	Alternative (m=3)								
(n=3)	Proposed Algorithm			HMM			Spatial Difference Only		
	Ci1	Pi1	Si1	Ci2	Pi2	Si2	Ci3	Pi3	Si3
Accuracy	90	40%	36	100	40%	40	50	40%	20
Speed	90	40%	36	60	40%	24	100	40%	40
Ease to Implement	80	20%	16	50	20%	10	100	20%	20
Total Score	88			74			80		

As it is demonstrated in the Table 3, the proposed gesture recognition model fitted the application best because it had the highest score. Regarding to the accuracy criterion, the HMM got the highest score because its excellent recognition rate (about 99.78% with 100 training samples per gesture) according to the paper by J. Yang and Y. Xu in CMU [8]. In addition, it has been proved that the Spatial Difference Only algorithm has a poor performance in gesture recognition by Darrel T. [9]. The accuracy of this algorithm will dramatically decrease as the number of gesture types included increases because some gestures had similar spatial difference of fingers. The proposed algorithm has been shown to be effective with a gesture recognition rate greater than 95%. Detailed analysis can be found in the section 4. In terms of speed criterion, the proposed algorithm has a complexity

of $O(N^K)$, where N is the number of features and K is the number of pre-defined gestures. the complexity of HMM in the clustering stage is $O(N^{K+1}KT)$, which is demonstrated in the referenced paper, where N is the number of states, K is the number of Markov Chains (pre-defined gestures in this case), and T is the length of a testing sample. Due to the strict speed constraint of the program, HMM should take a low score on this section. Although the speed of the proposed algorithm is slower than the Spatial Difference Only method whose complexity of O(n), the algorithm was still able to identify different gestures in real-time in our test. Therefore, both algorithms should still obtain a high score. The speed analysis is demonstrated in details in section 4. Last but not least, the HMM was most difficult to implement, whereas the Spatial Difference Only algorithm was clearly the easiest one. However, this criterion was not as important as accuracy and speed.

Overall, the proposed algorithm was adopted because it satisfied all project specification closely.

Table 4. Pre-defined Gesture and Expected Reaction

Gesture	Expected Reaction
Victory (two fingers of the same hand out with close distance)	Can take an object and drag it
One hand out	Push
Two hands out	Restart
Two fingers close (after victory)	Drop

Table 4 describes a set of pre-defined gesture in the system and its corresponding function in the program. A set of sample gesture images can be found in the appendix.

3.2 Unity GameObject Physical Movement Design

In the virtual world designed by Unity, all the objects should be able to be moved around by the virtual hand (Leap hand), which was shown when human hands were tracked by the Leap Motion controller. All the entities in Unity scenes, included Leap hand and fingers, were of type GameObject [10]. Three types of interactions between Leap hand and other game objects designed in this project are dragging, dropping, and pushing. When the Leap hand was at the same position

of one or more non-static GameObject in the Unity scene, one of the three interactions was triggered. Unity could automatically detect whether or not the hand was overlapping with other objects. The exact interaction to trigger was dependent on the gesture of virtual hand.

Rigidbody and collider component could be added to each GameObject in Unity scene. With Rigidbody property, a game objects were under control of Unity's physics engine. Then those objects could be pulled down by gravity and react to collision if those functions were enabled [11]. Collision was detected when a (game object with) Rigidbody hit a collider. Then OnCollisionEnter() would be called for both game objects. If no collision was detected when two objects overlap, OnTriggerEnter() would be called.

The movement of hand detected by Leap Motion controller was scaled into Unity scene space. The default scaling factor was 1 cm : 2 Unity unit scale, which meant 1 cm movement of hand detected by Leap Motion Controller would result in the virtual hand moving by 2 Unity unit scale. If the room size in Unity scene was greater than 100 square Unity unit, the scaling factor was increased proportionally. However, the scaling factor could not exceed 1 cm : 4 Unity unit. Else the virtual hand and any objects being dragged would be moving at the speed greater than 2 Unity unit per second when the human hand was moving at 0.5m/s, which meant the specification could be violated.

Dragging would be triggered if the virtual hand had the corresponding gesture (see 3.1) and was holding on an object. During dragging, the object being dragged should move together with the fingers. It was tempting to update the position of the objects being dragged directly in OnCollisionEnter() or OnTriggerEnter(). However, those event handlers were only called at the instance of overlapping. Unless the interaction between virtual hand and the object being touched was stopped and restarted, the event handlers were not called again. There must be a method to update the positions of the objects being dragged continuously. This was done by calling DoMovement() and DoRotation() function in Update(), which was automatically executed once per frame [12], when necessary. DoMovement() and DoRotation() were functions provided by Leap Physics Sandbox sample application available at [13]. Since they perform well in moving objects with Leap hand, the functions were reused in our program. We instead designed the logic of deciding when should an object be changed to dragging mode, and made sure the object stay in dragging mode when the corresponding gesture of virtual hand was presented. To produce realistic

behaviours for objects being pushed by fingers or hit by other objects, Rigidbody was added on the objects and the fingers. However, when dragging an object, the physics engine could produce undesired behaviours on objects being dragged, such as moving the objects randomly. The problem could be solved by setting isKinematic to True. After that, collisions and forces (e.g. gravity), would not affect the object, thus it could be dragged around freely. The object being dragged could also be used to push other objects in the scene since their Rigidbodies still had isKinematics set to default value (False). Below is the pseudocode of dragging objects. Some of the code and fuctions, such as removing disabled fingers, DoMovement(), SetFocus(), were provided in sample applications available at [13]. They are shown in bold.

For each finger object in Unity:

```
void OnCollisionEnter(Collision collision){
        //"Touchable" means that object can be pushed and moved by virtual hand
        if (collision.collider.tag == "Touchable")
                LeapUnitySelectionController.Get().OnTouched(gameObject,
                collision.collider);
}

void OnTriggerEnter(Collider other){
        if( other.tag == "Touchable" )
                LeapUnitySelectionController.Get().OnTouched(gameObject, other);
}

public void OnTouched(GameObject finger, Collider other)
{
        //m_Touching: fingers that are touching objects, their current and last positions are used to for moving and
rotating the objects
        if (!m_Touching.Contains(finger)){
                m_Touching.Add(finger);
                m_LastPos.Add(finger.transform.position);
        }

        //m_ObjectTouched: recording which finger is touching which object
        //if the object just get touched is not in m_ObjectTouched, add it
        if (!m_ObjectTouched.Exists (obj => obj.finger == finger)){
                objectTouched obj = new objectTouched();
                obj.finger = finger;
                obj.gameObject = other.gameObject;
        m_ObjectTouched.Add(obj);
        }
        //else if the finger was touching another object, but the object was not being dragged, change
m_ObjectTouched to indicate that the finger is touching the object passed in as parameter
        else if (!m_FocusedObject){
                int index = m_ObjectTouched.FindIndex(obj => obj.finger == finger);
        if (index != -1){
                        objectTouched touched = m_ObjectTouched.Find(obj => obj.finger == finger);
                touched.gameObject = other.gameObject;
```

```
                m_ObjectTouched.RemoveAt(index);
                m_ObjectTouched.Add(touched);
        }
        }

        //if there are exactly two fingers on the screen, and the two fingers are of the same hand, and they are both touching the same object, change the object to dragged mode by calling SetFocus()
        if (thereAreExactlyTwoFingersOnScreen){

                List<objectTouched> l = m_ObjectTouched.FindAll(obj => obj.gameObject == other.gameObject);

       if (l.Count == 2 && twoFingersAreOnTheSameHands){
                if (!m_Selected && other.gameObject != m_FocusedObject)
                        SetFocus(other.gameObject);
        }
        }
}

public void SetFocus(GameObject focus)
{
        m_FocusedObject = focus;
        m_LastMovedTime = Time.time;
        HighLightTheObject();
}
```

Update function called for each frame:

```
void Update(){
        //Remove fingers which have been disabled
        int index;
        while((index = m_Touching.FindIndex(i => i.collider && i.collider.enabled == false)) != -1)
        {
                m_Touching[index].collider.isTrigger = false;
        m_Touching.RemoveAt(index);
        m_LastPos.RemoveAt(index);
        }

        while( (index = m_ObjectTouched.FindIndex(i => i.finger.collider && i.finger.collider.enabled == false)) != -1 )
        {
                m_ObjectTouched.RemoveAt(index);
        }
        if(ObjectMoved)
                m_LastMovedTime = Time.time;

        if(inDraggingMode){
        // Check if dragging mode should be ended, see description of dropping mode for more details
                if( CheckEndSelection(thisFrame) )
                        ClearFoucus();
                else {
                        if( CheckShouldMove(thisFrame) )
                                DoMovement(thisFrame);
                        if( CheckShouldRotate(thisFrame) )
                                DoRotation(thisFrame);
                }
```

```
        }
        Update_m_LastPos[]_with_postions_of_m_Toucing[]();
}
```

During dragging mode, if the gesture of dropping was detected, or the dragging gesture was not continued, the objects being hold would be released. The dropping gesture was unique and no other movement of virtual hands could cause the detection of it by accident. Therefore, it was safe to drop the object immediately after the gesture was detected. However, Leap Motion could sometimes erroneously detect the third finger of virtual hand even when the user only had two fingers out. Although the period was brief, it was enough to cause the system to think that there was no dragging gesture in the corresponding frame. Therefore, the object should not be immediately dropped when the system did not detect dragging gesture in a frame. Instead, there should be a time period for redetection of dragging. If dragging gesture was not re-detected in this period of time, the object could be safely dropped. Otherwise the system would remain in dragging mode.

Two options for dropping objects were "Use Gravity" and addForce() function. "Use Gravity" was enabled on all the Rigidbody by default. The objects in the air with this option would fall onto a surface automatically when there was nothing holding them. The only problem of using this option was that the object may not end up standing after falling freely onto certain surface. This could be fixed by limiting the object's rotation ability. After the object's ability of rotating sideways was disabled, it would always end up standing after falling down. As mentioned earlier, addForce() function could also be used to drop down the objects. However, there was no need to test it out since "Use Gravity" option could perfectly achieve what was necessary – dropping objects to any surface without tilting them. It should be noticed that Rigidbody.isKinematic was set to true for the object being dragged. To let gravity have effect on it, its Rigidbody.isKinematic needed to be reset to False.

Below is the pseudo-code of functions that decides if object should be dropped or not. As shown in the pseudo-code of dragging objects, they were called in Update().

```
public bool CheckEndSelection(Frame thisFrame)
{
        if( DroppingGestureDetected )
                return true;
```

```
        // the logic uses m_LastMovedTime variable
        if( noDraggingGesture && timeObjectIsIdle > maxIdleTimeAllowed )
                return true;

        return false;
}

public void ClearFocus(){
        if( m_FocusedObject != null )
        {
                DeHighlightObject(m_FocusedObject);
                m_FocusedObject.rigidbody.isKinematic = false;
        }
        m_FocusedObject = null;
        m_LastMovedTime = 0.0f;
        m_Touching.Clear();
        m_LastPos.Clear();
        m_ObjectTouched.Clear();
}
```

In pushing mode, besides moving in the direction of the palm, the objects should also slide for some extra distance after being pushed to simulate the movement inertia. This required force to be applied to the Rigidbody of the objects being pushed. Rigidbody.addForce() could be used for this purpose. The faster the hand was moving at the instance, the greater the force was applied. There was also an alternative way of pushing objects. In Unity scene, when objects were hit by a Rigidbody with certain velocity, they moved for some distances due to the Rigidbody's velocity. Because of that, the objects they collide with could be pushed forward automatically in pushing mode. Leap Motion controller automatically detected the fingers' velocities when they were moving. Those velocities could then be converted to Unity's scale by using scripts provided in sample applications on Leap Motion developer website [13]. As an object was moving in Unity scene, its velocity was automatically updated. To use Rigidbody.addForce() option, the effect of fingers' velocities must be disabled. This could be done by making the object being pushed kinematic (or assigning its Rigidbody.isKinematic to True). In the end, the method using fingers' velocities was chosen since it produced more realistic movements in experiments.

3.3 Head Tracking Subsystem Design

3.3.1 Image Capture Method

In the head tracking subsystem, we used head position of the user as input, and provided the real-time pixel position on the video captured. The pixel position was then provided to the next subsystem to calculate the 3-dimensional position.

There were three possible options to implement this subsystem, and a decision matrix was used for the design. Since the system was expected to provide a 3-dimensional scenario updated in real-time according to the position of user head, tracking accuracy must be high. Also, if tracking had a large delay, the scenario displayed would also be delayed. In that case, the user would not be able to get an instant feedback. Therefore, large weights were assigned to tracking accuracy and tracking delay. Tracking range was also an important property since we expected user head to move in a range up to 1.5 meters away from the screen. The head tracking system must be robust against the interference of the environment, including interfering background, change of brightness and the appearance of another human head. Therefore, large weights were also assigned to robustness. Another important requirement of the system was that the position of every part of the system must be fixed. Otherwise, the head position detected would be offsetted. Cost and easiness to implement were also taken into our consideration. Another important specification was frame rate. However, since all the three options provided frame rate of 30 fps, it was not included in the decision matrix.

The first option was to use Kinect, which was a motion sensing input device designed by Microsoft. Kinect was used to detect the human body movement. It precision of head movement detection was not relatively low. According to the product specification, the detection range was maximum 4000mm, which was above our requirement. And the delay of detection was also low enough. The price of Kinect was $110 and there was a Kinect software development kit to allow developers to write Kinect apps in different programming languages, making it easy to interface with. The size of kinect was 27cm by 26cm by 7.8cm, so it was difficult to be maintained to a fixed position.

The second option was to use a webcam associated with face detection algorithm. The tracking accuracy was considerably high. This was because we used the size of user face in the video captured to calculate head distance to the camera, and different user had different face sizes. This

resulted in error of distance calculation. Since complex face detection algorithm was to be implemented, the system did lots of calculation for each frame, this led to potential tracking delay. Through an experiment using a 720p webcam at ISO 800 under ambient room light, we found that the longest distance that human eye could be detected was about 2 meters, which was above our requirement. The robustness of face detection was not great because there were interferences of the environment, including interfering background, changing of brightness and the appearance of another human head. These interferences had large negative influence on face detection. Since we only needed a webcam for this option, the cost was low and it was easy to fix position. However, the face detection algorithm took more effort to implement compared to other options.

The third option was to use a webcam and two mini LED lights. In our design, headphones were provided to the user for 3-dimensional audio. Since the mini LED lights were very small, they could be easily fixed on the two sides of the headphones using adhesive tape, and the system would detect the position of the two LED sources to determine user's head position. The estimated tracking accuracy was high and tracking delay was low because the detection algorithm did not involve as much calculation as face detection. The tracking range and robustness was also very great because the mini LED lights used had relatively high light intensity. Further analysis of this method was done in the next part. This option took some extra effort to integrate the LEDs into the system, and cost was around 25 dollars.

Table 5. Head Tracking Subsystem Decision Matrix

	Full Mark	Kinect	Face Detection	LED Detection
Initial estimated tracking accuracy	20	10	15	20
Tracking delay	20	15	15	20
Tracking range	10	10	10	10
Estimated robustness	20	20	15	17

Cost	10	0	10	5
Position fix Ability	10	0	10	5
Easiness to Implement	10	10	5	5
Total	100	65	80	82

According to the decision matrix, the option using webcam to detect LEDs was selected. Since some of these factors were estimation only, further investigation and test was needed to determine if the method would be feasible. Face detection method was set as a backup method in case the LED detection method failed under harsh environment.

3.3.2 Pixel Location Determination

The LED detection method used two mini LEDs together with a 9 VDC-20 mA battery, as it was shown in Figure 4. Red LEDs were used instead of normal white LED because red light is more robust against interferences of environment. The LEDs were attached on headphones using adhesive tape.

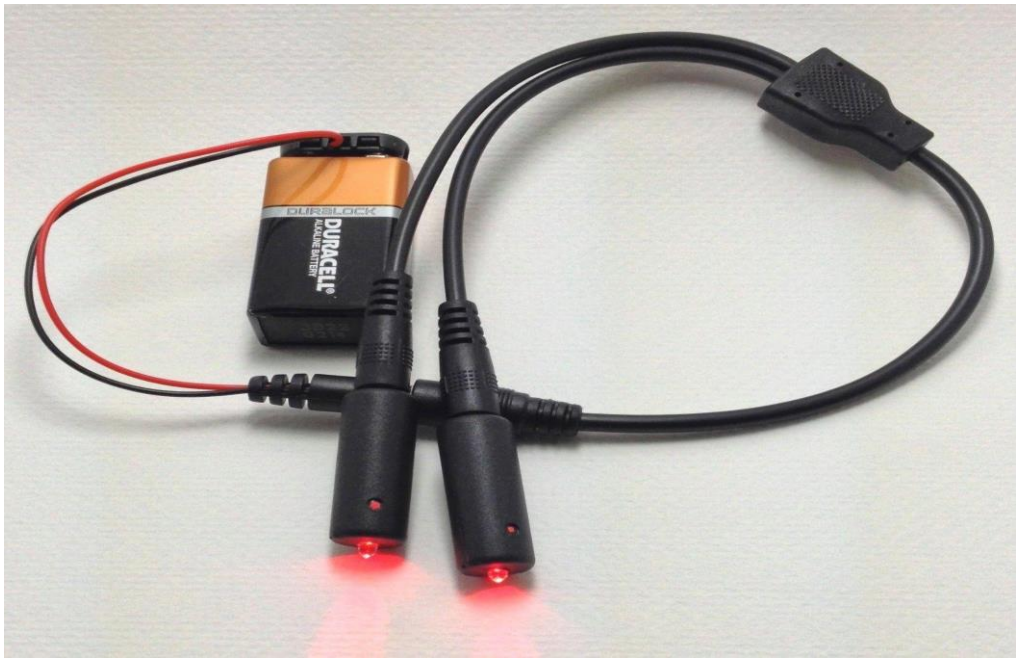

Figure 4. Two mini LEDs together with a 9 VDC-20 mA battery

In the next segment, EmguCV [14], an open source library of programming functions mainly aimed at real-time computer vision, was used for the real-time capture of webcam video. The programming was done in C# and Visual Studio was used as IDE. The captured video would then be processed frame by frame to detect the position of two LED lights by applying colour space filters. The colour space was converted to HSV (hue, saturation and value) instead of RGB (red, green and blue). The reason was that HSV could separate colour information from intensity of lighting. Because values were separated, thresholding rules could be constructed based on both red colour and high intensity of LED lights. This was a nice improvement in practice since HSV colour space provides more robust color threshold over RGB. [18] After filtering, data matrices of images containing only two points were obtained. As shown in Figure 5. The two white points represented the two LEDs. Each matrix represented one frame, and entrances of these matrices were all zeros except of the position of the two LEDs. These matrices were then used to calculate the pixel position of the two points for each frame.

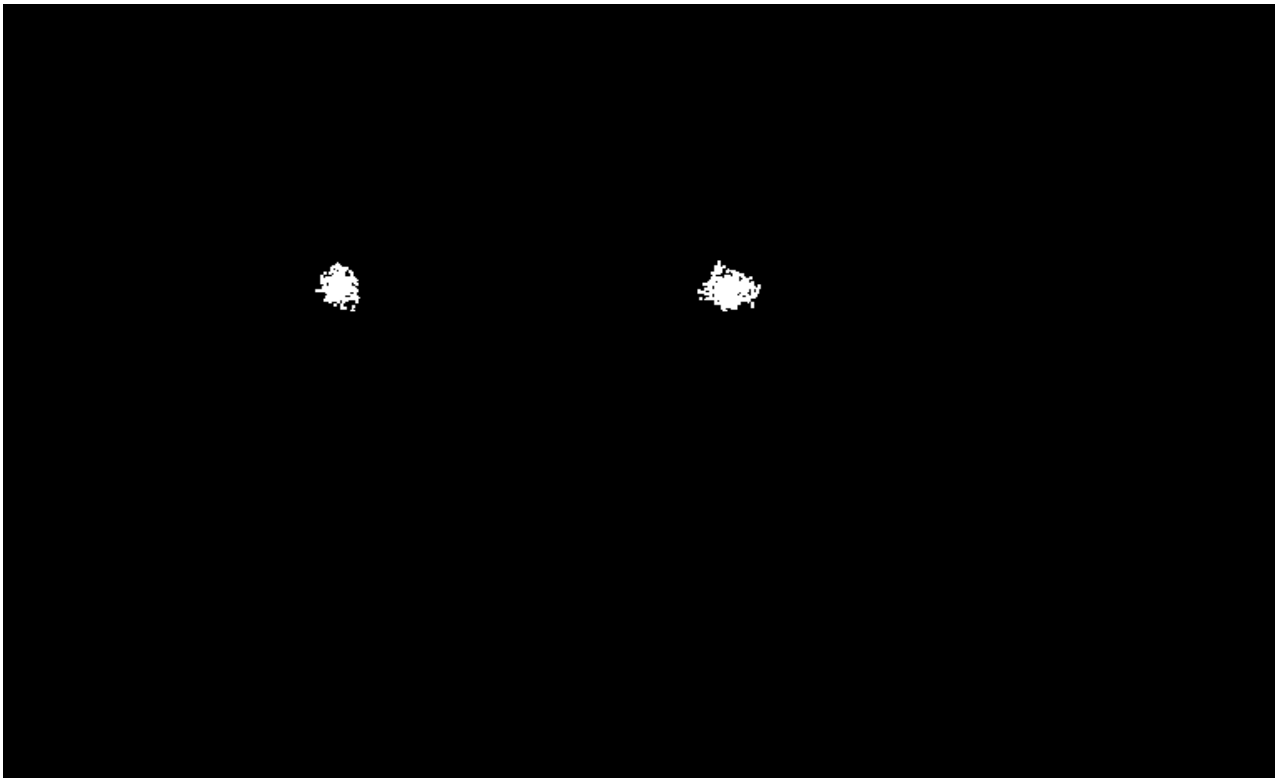

Figure 5. Output image in HSV colour space

During implementation stage of this subsystem, we found that the accuracy of webcam and LED lights was not as good as expected. When there were people wearing clothes of the same color as LED light passing by, the head tracking subsystem would consider the clothes as a source of LED light. If the threshold was changed to limit background colour completely, sometimes the LED would not appear correctly in the image. Thus, the robustness of the LED light detection would be affected significantly. During our testing, the head tracking system failed about 20% of the time when the person was moving at 10 cm/s. This was below our functional specification of 95% success rate. Therefore, this method was determined as a failure.

Because of the failure of the webcam and LED lights, we implemented the option with the second highest score, face detection. EmguCV was used for the real-time capture of webcam video. Again, the programming was done in C# and Visual Studio was used as IDE. The captured video was then processed frame by frame to detect the position of two human eyes by applying filters. The original 720p video was first compressed to 480p video by applying averaging filter. The tracking delay was decreased greatly by decreasing the frame resolution. Also, when using 720p frames, the

subsystem sometimes stopped to respond and the software would crash after timeout. By compressing to 480p, this problem was solved completely. After the compressed was passed to face detection filter, data matrices of images containing only two points were obtained. The two points represented the two human eyes. To be noted, only the closest human face will be detected, so the user could be distinguished from the crowd. Each matrix represented one frame, and entrances of these matrices were all zeros except of the position of the two LEDs. These matrices would then be used to calculate the pixel position of the two points for each frame. The calculated pixel positions were finally provided to the next subsystem to calculate the 3-dimensional position of user head.

3.3.3 Noise Filter

Human movement could hardly be controlled precisely. Therefore, vibrations always existed in the video we captured from the webcam. In order to reduce the vibration, we designed a moving average filter to remove the no. In this method, we convoluted the input signal with a set of impulses, and then used summations to add them together [15], the following is the Matlab source code:

```
MA = ones(M,1)/M;
y = filter(MA,1,x);
%x is the input coordinate, M as the number of impulses, MA as a set of impulse and y is the output
```

The application of this filter is shown in Figure 6. The cyan line illustrated the x-coordinate data points collected from user motion. As shown, there are lots of noise due to the inconsistent measurement. Filter was then applied to this line, and the output would be the filtered value.

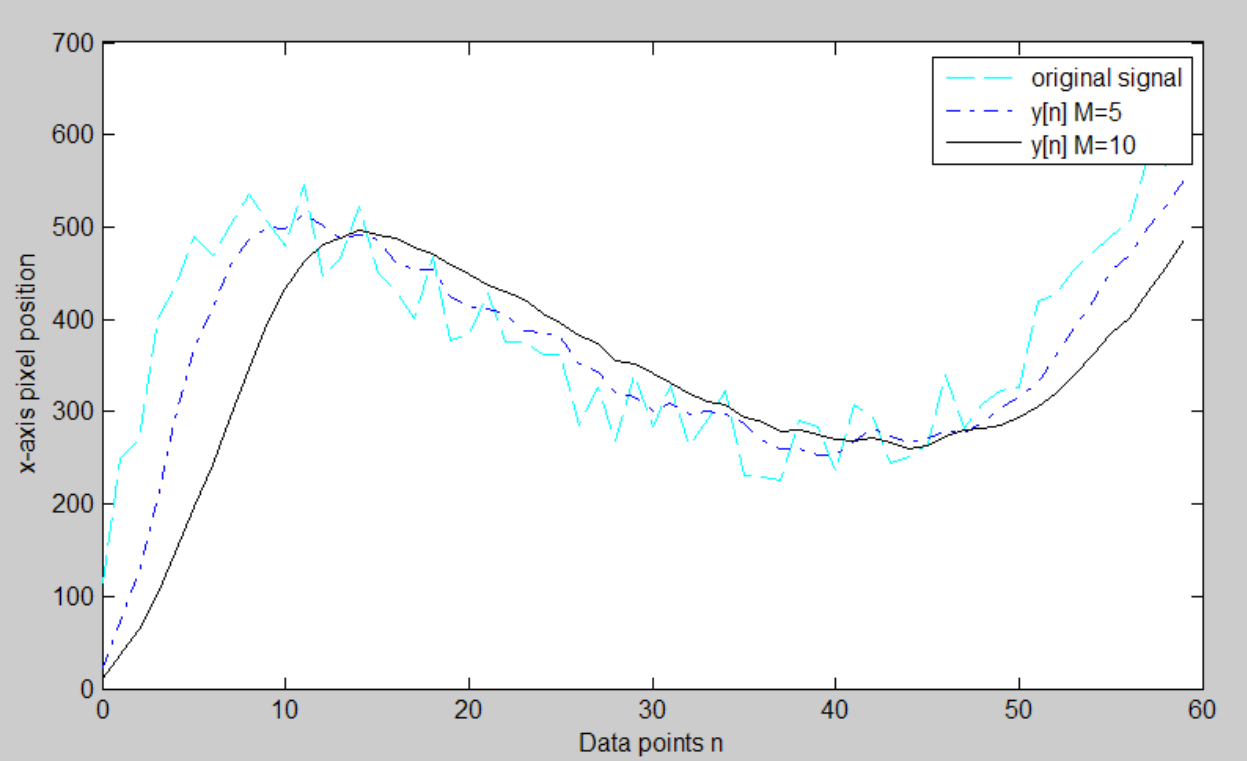

Figure 6. Moving average filter result

For comparison, we applied two different M values into the filter, M=5 and M=10. After comparing the two different outputs, we found that the signal at M=10 was much smoother than that at M=5. This meant that signal at M=10 had better noise reduction. Both of the reductions should be competent for our design. Furthermore, when M=5, output's amplitude resembled the original signal more closely. In addition, it had less than 2 sample periods delay compare to M=10, which sometimes had a delay of 5 sample periods. Since we were operating in a real time application and the webcam would capture the image at 30fps, 1/6 sec delay was certainly unacceptable. Therefore, M=5 was a better choice for moving average filter.

3.3.4 3D Coordinate Conversion

This sub-system was aimed to convert 2D image coordinate into virtual 3D coordinate. The input of the system was the 2D X-Y coordinates of the two points in the filtered image, and the output was a head position vector in 3D space. the output data would be fetched into Unity for further processing.

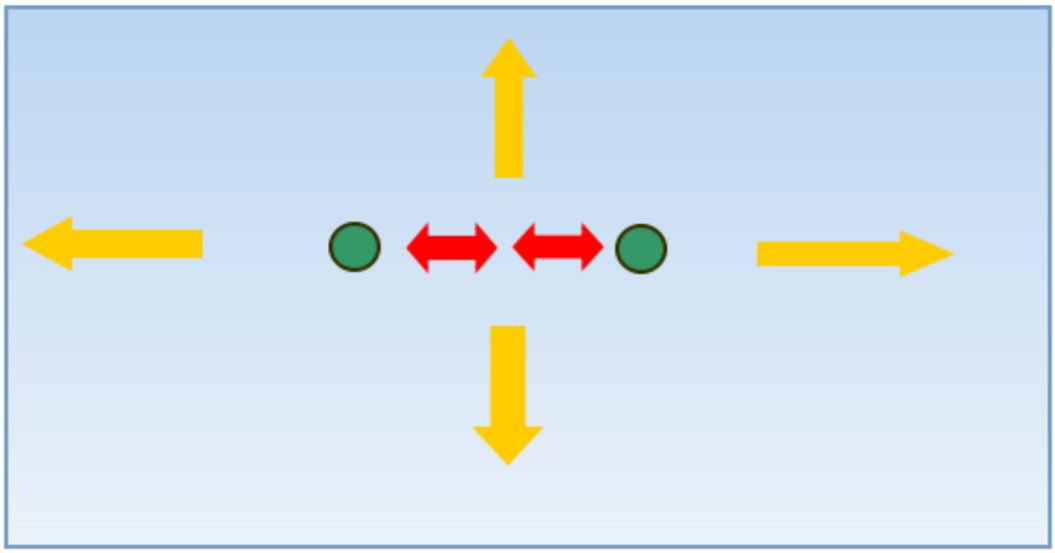

Figure 7. Eye light movement

By inspecting the location of the two eyes, it could be seen that there were three kind of movement inside the video frame. The first movement was when the user's eye move away or toward each other. The second one was the coherent movement of the user's eyes. And the third movement was the rotation of eyes. Due to the limited ability to determine head rotation from 2D images with two points, [17] the third motion was ignored in our system. By finding the travelled distance of these two movements, we could make perform calculations to possibly get the x-y-z coordinate of the person's head with respect to the camera.

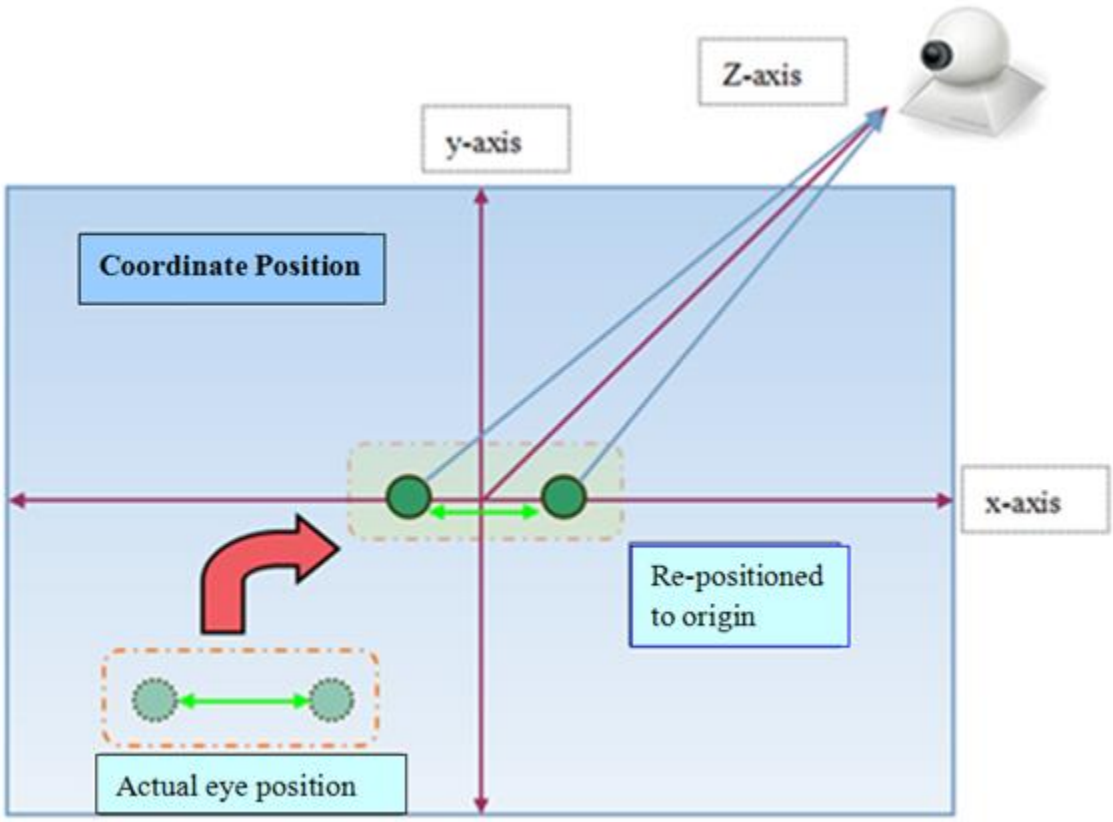

Figure 8. 3D illustration of the positioning process

The first method was to set up a reference point as the origin. Camera always had some distortions due to lens' physical limitation. Since the image of webcam was symmetric with respect to the center of the image. The distortion due to any two points symmetric to the origin would be the same. Hence, the distortion compensation with respect to the origin could be ignored. Furthermore, the distortion could cause error in measurements between the two eyes. We used an experiment to measure our webcam's distortion. By positioning a 10cm long ruler 50cm away from the webcam, we found that the length of the ruler at the centre of the image was 5% shorter than the same ruler at the corner of the image. This was a small error that ordinary users would never notice. Therefore, it was ignored in this method.

By position the person at the centre of the screen, the centre of user's eyes would overlap with the origin, we could calculate how far the person was away from the webcam by finding the distance between the two eyes. As the person moved closer to the webcam, the two eyes on the captured frame would move further apart, and the distance between the two would increase. As the person

moved away from the webcam, his or her eyes would be closer together on the image, the distance would decrease. The equation to calculate the distance was

z-direction_coordinate = (distance_between_eyes/2)*distance_per_pixel

The distance between eyes divided by 2 was the distance from the eye to the origin, the distance per pixel was a reference number we set up by experiment. In this case, we assumed that the distance was linear with respect to pixels. We could simply use one value for distance per pixel for the calculation. In this case, our reference number was 1000 pixel/meter.

Since the above case only worked when the x and y direction was fixed around the origin, we had to implement a method to find x and y axis coordination. This was done by measuring the x and y distance from the midpoint of the LEDs to the origin. The coordinate was the distance in pixels multiply by the distance per pixel. Distortion factor also had to be multiplied. The value of it was acquired through experiments. The result was the 3D coordinate of the person's head.

The first method used origin as the only reference point. It may be inaccurate due to camera distortion. In order to eliminate this problem, we could use fuzzy logic to train the head tracker. The first step in this method was to pick N points on the 3D coordinate. By using accurate measurement, we could find the 2D image coordinate of these points. These points were then set as reference points. The area between the points was set to fuzzy region. After the pixel coordinates of the eye became available, the approximate location of the person could be found using genetic algorithm. Five random points is set as the parent gene, the fitness function can be calculated by the following formula.

$$Fitness = \sum (referenc\_point\_location - measured\_location)$$

After every iteration, a new set of reference point was randomly generated by mutation, the fitness function was recalculated. The stop condition was when there was no better result produced in 10 iterations. The average of the five reference points will be used as the 3D coordinate of the user. Using this algorithm, it was easy to see that as number of N increases, the calculation would become more accurate. As N approaches infinity, we would have an error free estimation. However, this method had obvious drawbacks, such as inflexible and hard to implement. For example, it

could take significant time to measure the reference points. Next, changing webcam could potentially make all the data useless. Furthermore, this method also required longer run time. To support this, the run time was measured in the initial development with 100 random reference points. The result shown that it could take up to 0.1 second to run this algorithm once. This can cause significant delay since we are required to run this algorithm 30 times per second.

Each of the two methods had pros and cons. To evaluate these two methods, we set up decision making matrix in Table 6.

Table 6. Coordinate calculation decision making matrix

Criterion	Weight	First method	Second method
Simple of design	20	18	10
Flexibility of design	20	15	5
Accuracy	20	10	18
Response time	20	18	10
Defect possibility	20	15	15
Total Score	100	76	58

By using decision making matrix, we could see that the first method was clearly better. We chose the first method as it satisfied with all the specifications. The calculation did not take a long time. The delay was almost zero in this part. This design was also more flexible with different cameras. As an extra note, further experiments had to be done in order to see how distortion would affect the calculation.

3.4 Integration of the Head Tracking and 3D-Environment

In order to display the dynamic perspective 3D view, the head tracking system would need to be integrated with a 3D Modeling platform, which in turn would translate numerical data gathered by the head tracking system to visual outputs for the user. It was decided that C# scripts would be used to interact with Unity, which would be used to display the virtual 3D environment.

Raw data from the camera would be passed to the head tracking C# code, which would detect the reference points in the frames. Spatial data gathered by the camera would be passed to the calculation section of the code. The main task for this subsystem was to integrate the calculation performed in the head tracking system, to Unity. The basic idea was to track user's head movement in real time, then reflect the movement in Unity's virtual environment by adjusting the Camera View's position and rotation within the Unity scene. The calculation subroutine from previous section would provide all the position adjustments for the Camera View within Unity. To integrate the calculated data with Unity, the integration subsystem would utilize Unity Scripting API, which was compatible with main program's C# code [16]. The Unity API would provide the functionalities of move and manipulate objects within the scene.

This design satisfied the project requirement, as it facilitated the integration of the data gathered from the head tracking system with Unity 3D environment. The data gathered would be used to interact with objects within the Unity scene. In the case of this project, the primary mean of interactions would be moving the Camera View within Unity around. This combination was needed in order to achieve the dynamic perspective that the project was planned to deliver.

A top-down approach was followed in the designing of this project. The end goal was to display a 3D virtual environment with dynamic perspective. With the end goal in mind, each component was chose to implement towards that goal. Priorities of the components were assigned, and the components were decided in the order of their priorities.

Some of the major design decisions in the project that affect this subsystem was the design of the Head Tracking system, and the design of the 3D Environment system. This was because the goal for this subsystem was to combine the two systems together, so the design of the two systems would affect this component.

One of the major decisions in the project was to pick a 3D Graphics platform. A list of possible options was examined, which included Blender, CryEngine, and Unity. Many factors would need to be considered when picking this component for the project, as it was one of the major components for it. A decision making matrix was used to help on making the decision of which 3D Environment software should be used for the project, as seen in Table 1.

Table 7. Decision making matrix for 3D platform

Criterion	Weighting (out of 10)	Blender	CryEngine	Unity
Cost	5	10	7	10
Relative Experiences	8	2	0	7
Scripting Language Support	8	5	5	8
Ease of Use	7	6	8	6
Total Score	28	14.8	13.1	21.2

As concluded from the decision making matrix, Unity would serve the project better than the alternatives.

With the primary components chosen, it was needed to design the integration system, which would make interactions between the major two components within the project. In Unity, the animation script could be scripted using JavaScript, C#, or Boo, which had a Python-inspired syntax. The scripting language was decided based on the groups past expertise, and C# was chosen to be used for scripting. It was concluded that a wrapper DLL, which wraps the C# script functions in a DLL file, would need to be created for interacting with Unity objects. The wrapped DLL file would then be able to import into Unity as an asset, and the Unity scene would be able to be manipulated using the functions.

The virtual scene in Unity was designed using computer-aided design. The scene was developed in 3D space, and had a virtual dimension of 4m*5m*3m. Detail rendering and debugging would take more time in the future. The perspective view and object movement was provided in Unity.

The output of Unity would be displayed on the computer screen in real time in order to meet specifications. This required a fast computer to process the program.

4. Prototype Data

In this section, the experimental data is divided into three subsections. The first subsection discusses the accuracy of the system respectively. It is followed by the second subsection where the computational analysis is performed.

4.1 Accuracy

4.1.1 Gesture Recognition Rate

Since gesture recognition was served as the core tool for users to interact with the virtual object in the program, the gesture recognition rate was the most essential component in the system accuracy analysis.

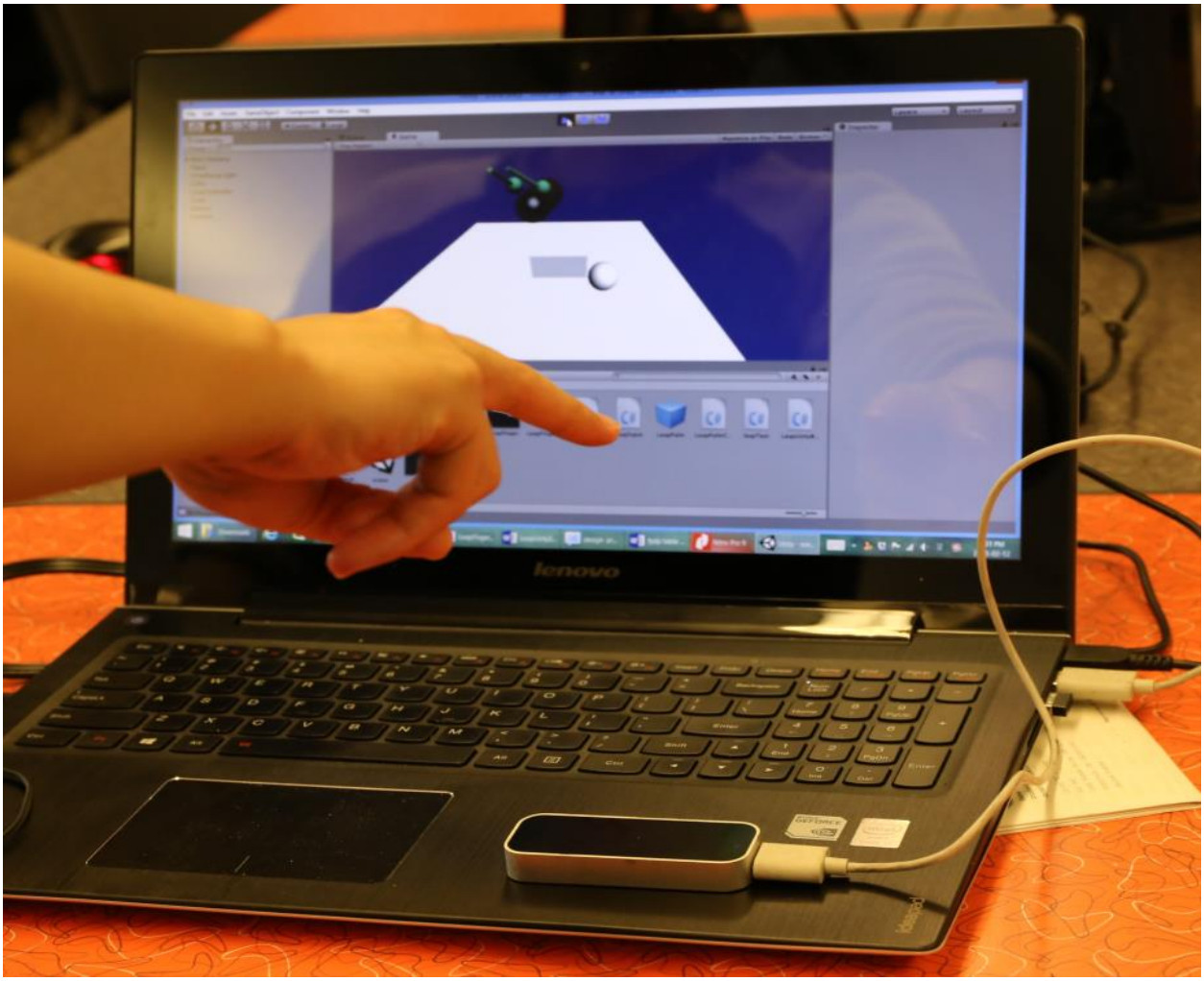

Figure 9. Gesture recognition test

To evaluate the performance of the gesture recognition model, Unity is used as the testing platform. The tester was asked to perform each claiming supported gesture 20 times. The gestures were captured by Leap Motion and then analyzed by Unity to control a cube object in the testing program. The reaction of the cube was recorded. If the behavior of the cube was identical to the expectation, the gesture recognition of the run was identified as success. Thereafter, the number of successful run was divided by the total number of trials to give the gesture recognition rate. In addition, 4 gestures that are similar to the pre-defined gestures were also recorded with the procedure described above in order to test the robustness of the system. If the system identified a non-defined gesture as a pre-defined gesture, it was ruled as a failure. During the test, the distances between fingers were strictly monitored by a ruler positioned beside the hand of the subject. Finally,

both of the results were combined to give out a overall gesture recognition rate. The metric is shown in the following formula.

$$Gesture\ Recognition\ Rate = 0.5 \times \frac{number\ of\ successful\ trials_{in\ test\ 1}}{total\ number\ of\ trials_{in\ test\ 1}} + 0.5 * \frac{number\ of\ non-identified\ trials_{in\ test\ 2}}{total\ number\ of\ trials_{in\ test\ 2}}$$

Table 8. Gesture Recognition Test

Gesture	Recognition Rate
Victory	20/20
One hand out	19/20
Two hands out	20/20
Two fingers close	20/20
Two fingers out with a distance greater than 2 cm (similar to victory)	2/20 (identified as victory)
Four fingers out (similar to one hand out)	1/20 (identified as one hand out)
Nine fingers out (similar to two hand out)	0/20 (identified as two hand out)
One finger out (similar to two fingers close)	0/20 (identified as two fingers close)

Table 7 shows the result of the gesture recognition test. It can be seen from the table that all the pre-defined gesture had a recognition rate higher than 95%. The only failure case in the first test was that one of the one-hand-out trials was not identified as push gesture. Regarding to the second test; however, two fingers out with a distance greater than 2 cm had a 10 percent of chance to be identified as victory. All the other gestures in the second test had an at least 95 percent of change to be ignored as expected.

According to the formula described above in this section, the overall gesture recognition rate was 97.5%. Since we were aiming at a gesture recognition rate greater than 90 percents, the system clearly satisfied the functional requirement of the project.

4.1.2 Head Tracking Accuracy Analysis

Head tracking accuracy test was another important measurement of our product's user experience. In order to increase diversity and compensate human error, five testers had conducted the accuracy test. The test set up is illustrated in Figure A. First, a white wall was marked with coordinate. The origin was mapped with a X. The tester was asked to move vertically and horizontally with respect to the origin. The tester's final location was stopped by a tapped paper shown in figure A. The computer was set 80cm away from the origin. To measure the z-direction accuracy, the tester was asked to move forward and backward against the wall.

Each tester was asked to perform the same test 5 times in order to limit human error. Due to the lack of equipment to track the actual human movement, it was not feasible to obtain exact measurement. The only suitable solution to define the tracking failure was by tester's judgement. Normally, this would be observed when there was a significant latency in the reorganization.

From daily testing and usage, it was assumed that the system will only have to deal with small movements of the head. Therefore, the main moving distance of the test case was set to 10cm and 20cm. The weighting factor was set to 0.5 for the 10cm movement and 0.4 for the 20cm movement. The other main accuracy test was when the person's head moves into the camera's field of view. This provided an important use case that could not be measured in the previous two cases. The weighting factor for this test case was set to 0.1. The overall head tracking accuracy rate formula was the following.

$$Accuracy\ rate = \sum_{i=1}^{3} weight\ \ factor_i * \frac{numerber of\ success_i}{number\ of\ trials_i}$$

Table 9. Head tracking accuracy

Distance from origin	10cm	20cm	Move into camera's field of view	Accuracy rate
Weighting factor	0.5	0.4	0.1	1
Horizontal Accuracy	25/25	25/25	23/25	99.2%
Vertical Accuracy	25/25	25/25	23/25	99.2%
Depth Accuracy	23/25	24/25	20/25	92.8%
Total	74/75	74/75	66/75	97.1%

Table 9 shows the results of head tracking accuracy test. Under normal movements within the camera's field of view, the head tracking was very accurate. The only failure happened when moving front and back. However, the head tracking system was not really sufficient when the tester came into camera's field of view from outside. The system's main deficiency happened when the person moved backward into the camera's view. This showed the system's inability to track in the z-direction. By using the formula above, the overall system success rate was about 97.1%. This was better than the 95% accuracy rate set as functional specification.

4.2 Computational Analysis

4.2.1 Algorithm Complexity Analysis

Using knowledge obtained from ECE 406, Algorithm Design and Analysis, the code for the prototype of the project was analyzed. The C# code utilized event handlers to listen to events within the application. The event that was of interest was the one where the image for each video frame was generated by the camera API. Whenever the ImageGrabbed event was fired, the method for processing frame would be called.

Within the ProcessFrame, faces within the frame were detected by calling OpenCV's DetectFace() function call. It was assumed that the DetectFace() function call would run in a linear time, despite

the number of faces within the video frame. It was also assumed that the DetectFace() function call's runtime would linearly increase or decrease with the number of pixels within each frame. This was tested by using the "detectionTime" parameter that the function call returned, which indicated the time that was used for facial detection in the given frame. The function call would return two lists of Rectangles, which represented the location of faces and eyes. Once the relative position of the faces and eyes had being determined, a foreach loop was used to loop through all the faces detected within the frame. The foreach loop could be simplified and counted as a method of linear execution time. This was because in most cases, the project was aiming for only tracking one person's head at a time. Therefore, the processing time for each frame could be seen as having a linear algorithm with O(1) runtime, as the runtime complexity would stay constant no matter how many faces are in the frame, as the face detection algorithm would have to scan through the entire image no matter how many faces are in it.

This was also confirmed experimentally. It was found that number of faces does not affect the speed of the face detection algorithm significantly. It was found that on the laptop, when no face were found, the algorithm would take 139 ms. This number is lower when there are faces present, and can be caused by the fact that the algorithm don't have to process and package return information to the calling method. When there's one face found, the method would take 248 ms, and when two or more faces were found, it would take 245 ms. It was observed and concluded that number of faces would not affect the performance of the algorithm.

4.2.2 Memory Complexity Analysis

The memory complexity of the program at runtime could also be analyzed. If each frame was counted to be having a constant runtime, the entire head tracking algorithm would depend on the resolution of the video frame. As the goal of the project was to track one head at a time, the number of faces would not be considered to have any effect on the memory usage of the algorithm. Each of the video frames would need to be stored temporarily, which accounted for the majority portion of the algorithm's memory usage. The frame rate of the video did not affect the memory complexity of the application, as each frame would be discarded after processing.

The memory analysis data was also collected experimentally. The application's memory usage can be tracked easily using Window's built in Resource Monitor. It was observed that the camera

capture functionality uses around 50 to 60 MB of RAM when running at 320 by 240 pixels. It was observed that the OpenCV does not use a lot of memory for videos with lower resolution. This analysis helped us to optimize performance while use less memory.

As the goal for the algorithm was to provide head tracking with minimum delay, it was essential that the linear runtime for each frame processed to be low enough so that the application did not lag too much. The resolution of the video frame could be adjusted so that the processing time for each frame did not take too long to process. Adjusting the video feed's resolution could also help reduce unnecessary memory usage.

5. Discussion and Conclusion

5.1 Evaluation of Final Design

Table 10. Evaluation of Final Design

	Specification	Outcome	Score
Gesture Recognition Rate	90%	97.5%	97.5%
Head Tracking Accuracy	95%	97.1%	97.1%
Speed	10 fps		
Memory	1 GB		
Overall			

Table 12 shows the overall performance of the virtual reality assisted interior design system. The gesture recognition rate, head tracking accuracy, speed and memory usage all satisfied design specification with a considerable safety margin. The integrated system performed as expected. Accurate head tracking and smooth gesture recognition provided users with excellent design experience.

The integrated system only contained a laptop and a Leap Motion Controller, which was affordable and small enough for everyday carry. The only hardware setup user needed to do was to plug in the Leap Motion Controller. Also, Unity game engine was used for display of the virtual scene. It provided great expendability to the integrated system, and additional game objects could be added to the system easily.

5.2 Use of Advanced Knowledge

As the project was mainly a software project, many of the software knowledge learned in the upper years were applied to the project. Some of the knowledge used on the project were learned from ECE 406 (Algorithm Design and Analysis), and ECE 459 (Programming for Performance). In ECE 406, the knowledge of how algorithms are designed, and how to analyze them for correctness and efficiency were learned. This knowledge served an important role in the project, as the algorithms needed to be designed efficiently, as well as logically correct, for the optimal performance. Apart from designing a correct and efficient algorithm, Professor Stephen Smith had also taught the knowledge of how to analyze methods for computational and memory complexity. The algorithm's computational complexity could be analyzed, and different methods that are less computational or memory complex and can achieve the same output could be found. Some of the techniques included dynamic programming, which breaks a big problem into smaller, but similar sub-problems. Each sub-problem can then be computed individually, and the output can be stored and reused for similar sub-problems. This could reduce the total amount of computational power required for a complex problem. Memory requirement could also be optimized by strategically using shared resources and temporary storages throughout the application.

Knowledge learned from ECE 459, Programming for Performance, had also helped the project greatly. As the project attempted to product output in real time or with minimum delay, performance was very important to the success of the project. The knowledge of how to parallelize codes to make them run more efficiently on modern multi-core CPU were learned in the course.

Using this knowledge, the algorithms could be analyzed and optimized, which could help in obtaining the optimal output and performance.

5.3 Creativity, Novelty, Elegance

The project involved many creativity designs that were not seen, or rarely seen before. The idea of integrating Leap Motion controllers with a video camera for perspective change was a rather creative idea. As Leap Motion controller is a new technology, there had not been many innovations involving it.

Since the technologies involved are quite new, it required creative ideas to put the technologies together to work with each other. The project combined camera head tracking system with Leap Motion controllers to interact objects within a virtual reality. The project was prototyped to aid interior design, but it could be scaled easily to perform other functions too, such as virtual autonomy, construction design, and various simulation modeling. The project also did not use special equipment to aid with the head tracking, unlike other implementation of similar head tracking projects. This was achieved by comparing the size and relative distance of chosen point that represent facial features. This was a new approach to solve the head tracking problem, which was not observed in the previous similar implementation of the project.

Combining Leap Motion controllers with the head tracking system is also a novel idea. As Leap Motion controller is a relatively new technology, not much has been done by the others to incorporate it into projects. By combining Leap Motion controllers with video camera head tracking system, much new possibility could be created. User could interact with the virtual reality world in ways that was not available before, as the user could move the perspective, as well as interact with the virtual objects, which give user a more immersive virtual reality experience.

The project was designed to be operated hands/equipment free for the operator. In the earlier prototypes of this project, users were required to wear a hat with LEDs on it to assist with the camera with head tracking. In the later iteration of the project, it was refined so that the users would not have to wear anything special for the system to work. The elegance of this design was that it was more user friendly and convenient. In an environment such as a hospital, the system could perform really well, as it did not involve patients touching any equipment, therefore, limiting cross contamination between patients. The lack of special equipment required for the users to operate the system has vastly increased the elegance of the system.

5.4 Quality of Risk Assessment

Two possible risks have been identified in the Virtual Reality Assisted Home Design System. The first risk could be caused by improper management of the cable wires. The system is made of a computer, two Leap Motion Controllers, and a webcam. Without proper management of these components, the system could become a safety hazard. It is very easy to get trip over the cable wires. To prevent this, it is suggested to use a computer with build in webcam. Furthermore, the Leap Motion Controller cables should be bundled together.

Another possible risk could be resulted from the improper gesture which controlled the scene flushing. Since the "two hand out" gesture was designed to restart the program, if people other than the user occasionally put their hands into the range of the Leap Motion, the system may be restarted unexpectedly. To overcome the problem, a timer was set to count the duration that two hands stay in the range of the hand capture hardware. The "two hand out" gesture that was not kept for 2 seconds would be ignored.

5.5 Student Workload

Zhengfang and Rui worked closely to design the gesture recognition model and Unity gameObject physical movement subsystem. The two students performed technical analysis on different gesture recognition model, implemented the initial model and different physical movement mode to the object in Unity, and testing scene in Unity together in ECE498A. Besides that, Zhengfang helped with the integration of subsystems. Rui contributed more to programming and debugging. In ECE498B, the two students worked together to improve the accuracy and robustness of the gesture recognition model, which included parameter tuning and debugging. They also prepared and conducted test cases to the model. Each of the student contributed 20% in the project.

Zhechen, Lingyun and Debin worked closely on head tracking implementation. The three students split the tasks into three sub-sections. Lingyun worked on converting image on 2D coordinate, Zhechen worked on converting the 2D coordinate to 3D coordinate, and Debin worked on system integration, testing and debugging. The three students worked closely together and cooperated on all the tasks. All major components were developed during the spring term, small modification and tuning was completed during the winter term. Furthermore, system evaluation and data

collection were also performed during the spring term. Overall, every student contributed evenly in the head tracking part of the report. In total, it counted as 60% of the project.

The report were divided evenly, each person was responsible to their assigned subsystem. The other parts of the report were done in group and every student provided significant contribution to the report.

References

[1] Chief Architect Software, Chief Architect Software, 2014. [Online]. Available: http://www.homedesignersoftware.com/homedesign/. [Accessed 30 May 2014].

[2] Autodesk, "Home design and decorating ideas to get inspired and get expert tips," Autodesk, June 2014. [Online]. Available: http://www.homestyler.com/home. [Accessed 1 June 2014].

[3] "Leap Motion", Leap Motion Inc., 2014. [Online]. Available: https://www.leapmotion.com. [Accessed 20 June 2014].

[4] "Kinect - Xbox", Microsoft, 2014. [Online]. Available: https://www.xbox.com/en-CA/Kinect. [Accessed 21 June 2014].

[5] "Myo", Thalmic labs, 2014. [Online]. Available: https://www.thalmic.com/en/myo/. [Accessed 25 June 2014].

[6] A. Samadani and D. Kulic, "Hand Gesture Recognition Based on Surface Electromyography", *IEEE Engineering in Medicine and Biology Conference*, vol. pp.4196-4199, 2014.

[7] Lawrence R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceeding of the IEEE*, vol. 77, no. 2, 1989.

[8] Jie Yang, Yangsheng Xu, "Hidden Markov Model for Gesture Recognition", CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ., Pittsburgh, PA, May 1994

[9] T. Darrell and A. Pentland, "Space-time gestures," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 1993, pp. 335–340

[10] Unity Technologies, "GameObject" [Unity Documentation], 2014. [Online]. Available: http://docs.unity3d.com/ScriptReference/GameObject.html. [Accessed 28 June 2014].

[11] Unity Technologies, "Rigidbody" [Unity Documentation], 2014. [Online]. Available: http://docs.unity3d.com/ScriptReference/Rigidbody.htmll. [Accessed 28 June 2014].

[12] Unity Technologies, "Execution of Event Functions" [Unity Documentation], 2014. [Online]. Available: http://docs.unity3d.com/Manual/ExecutionOrder.html. [Accessed 29 June 2014].

[13] Leap Motion, "C# Examples for Unity3D", n.d. [Online]. Available : https://developer.leapmotion.com/documentation/csharp/examples/Unity_Examples.html

[14] "EmguCV" [Online]. Available: http://www.emgu.com/wiki/index.php/Main_Page [Accessed 28 June 2014].e

[15] H. yeganeh, "ECE 413-Assignment 1," 2014. [Online]. Available: https://learn.uwaterloo.ca/. [Accessed 22 May 2014].

[16] Unity Documentation, "Welcome to the Unity Scripting Reference!," 2014. [Online]. Available: http://docs.unity3d.com/ScriptReference/. [Accessed 22 June 2014].

[17] F Caballero, I Maza, “A Robust Head Tracking System Based on Monocular Vision

and Planar Templates” Sensors, 2009. [Online].

Available: http://grvc.us.es/publica/revistas/documentos/caballero_sensors09_web.pdf

[18] “Information on the HSV color space ” University of California, Irvine, [Online]. Available: http://dcssrv1.oit.uci.edu/~wiedeman/cspace/me/infohsv.html

Appendix A

ECE498B: Prototype Hazard Disclosure Form* Group number: 2015. 52

Instructions: Answer all the following questions by putting an *X* in either the yes or no column. If unsure, answer yes. If you answer yes to any question, set up an appointment with the Lab Instructor to ensure your prototype is safe for the symposium. Include this completed form in your Final Report (Appendix A) even if you answer no to all questions.

Question: Does your prototype…	**yes**	**no**
1. include any circuitry that you designed or built by yourself?		X
2. include any circuitry that is not enclosed in an approved plastic or metal electrical box?		X
3. involve any 120V AC circuitry/device that is not approved by CSA or ESA?		X
4. involve circuitry that is not connected to its power supply with a fuse and a switch?		X
5. use high-capacity or high-density (e.g., lithium-ion) batteries?	X	
6. emit non-trivial amounts of RF radiation?		X
7. involve x-rays or radioactive materials?		X
8. use unprotected lasers of any class?		X
9. involve strobe lights?		X
10. have exposed moving parts that may pinch, hit, or crush a person?		X
11. involve projectiles or any part that can fly?		X
12. have exposed sharp edges or points?		X
13. use high-pressure gases or liquids?		X
14. involve irritating or dangerous chemicals (assume all nano-materials are dangerous)?		X
15. involve any biological materials (dead or alive) or any food/drink?		X
16. emit dangerously loud sounds?		X
17. eject gas, particles, or fluids into the environment?		X
18. involve accessible components that reach temperatures above 40°C or below 0°C?		X
19. involve any other hazard? Describe:		X

Appendix B

ECE498B: Symposium Floor Plan Request Form* Group number: 52

Instructions: Answer all the following questions by putting an *X* in either the yes or no column. Provide additional information in the far right column as requested. Include the completed form in Appendix B of your Final Report, even if you answer no to all questions. Requests for equipment (e.g., an oscilloscope, a power supply, a large monitor, a computer, or a lab stool) should not be included on this form; instead, for such equipment requests use the on-line reservation system as described in the *Symposium Checklist and Schedule* document.

Question:	yes	no	if you answered "yes"…
1. Does your project require one or more hardwire internet connections at the symposium? **Note:** We provide Ethernet connections only, via a male RJ-45 connector. The connection comprises a static IP address in the uwaterloo.ca domain on a shared 100Mbps link. Do not count on the DC building wireless system being available for your project.		√	If yes, state how many connections you require:
2. Do you desire to use your own wireless router at the symposium? If yes, you will need to have your setup approved well before the symposium date. Unapproved routers are strictly prohibited.		√	If yes, contact Paul Ludwig for details as soon as possible.
3. Each booth has a 7.5-amp 6-outlet power bar. Does your project require more than 7.5 amps of mains (120 V AC) power?		√	If yes, state how many amps you require in total:
4. Each booth has a 7.5-amp 6-outlet power bar. Does your project require more than 6 outlets?		√	If yes, we will supply your booth with two power bars.
5. Do you intend to put any electronics/computers on the floor under your booth?	√		If yes, we will ensure you can do this safely.
6. Does your project involve projectiles or flying parts? (We have a large "cage" at the symposium for projects that involve projectiles or flying parts.)		√	If yes, state the nature of the projectile or flying part:
7. Does your project require special lighting conditions (e.g., dim light, bright light)? We will do our best to accommodate lighting requests, but no guarantees are made.	√		If yes, state the nature of the desired lighting conditions:
8. The default booth consists of a 5' wide by 2.5' deep table. The table is 2.54' tall. Does your project require extra table space (giving you a total table area of 7.5' wide by 2.5' deep)? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints.		√	If yes, explain specifically why you are requesting the extra table space:
9. The default floor space, including the table and area for you/visitors to stand, is around 6' by 6'. Does your project require extra floor space? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints. **Note:** Due to the extra-large class size this year, very few requests for extra floor space will be granted.		√	If yes, explain why you need the extra floor space: State how much extra space you are requesting:
10. Do you have any other special floor plan requests?		√	State your request:

Our demo needs a laptop. We also need uniform and bright light (if possible) for our webcam.

Appendix C

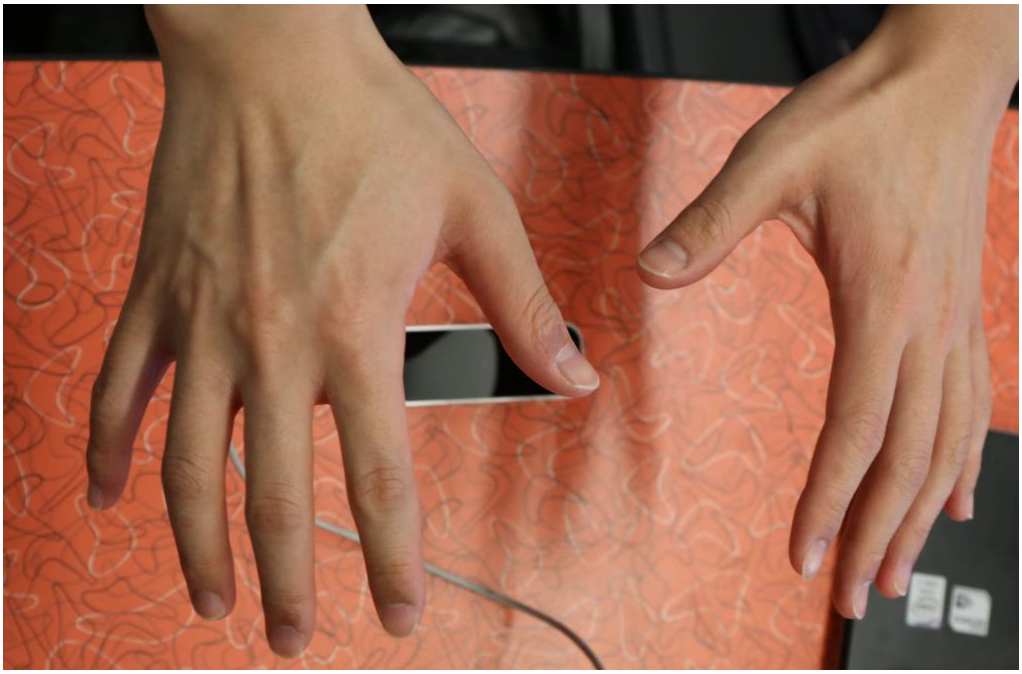

Figure A1. Gesture Restart (two hand out)

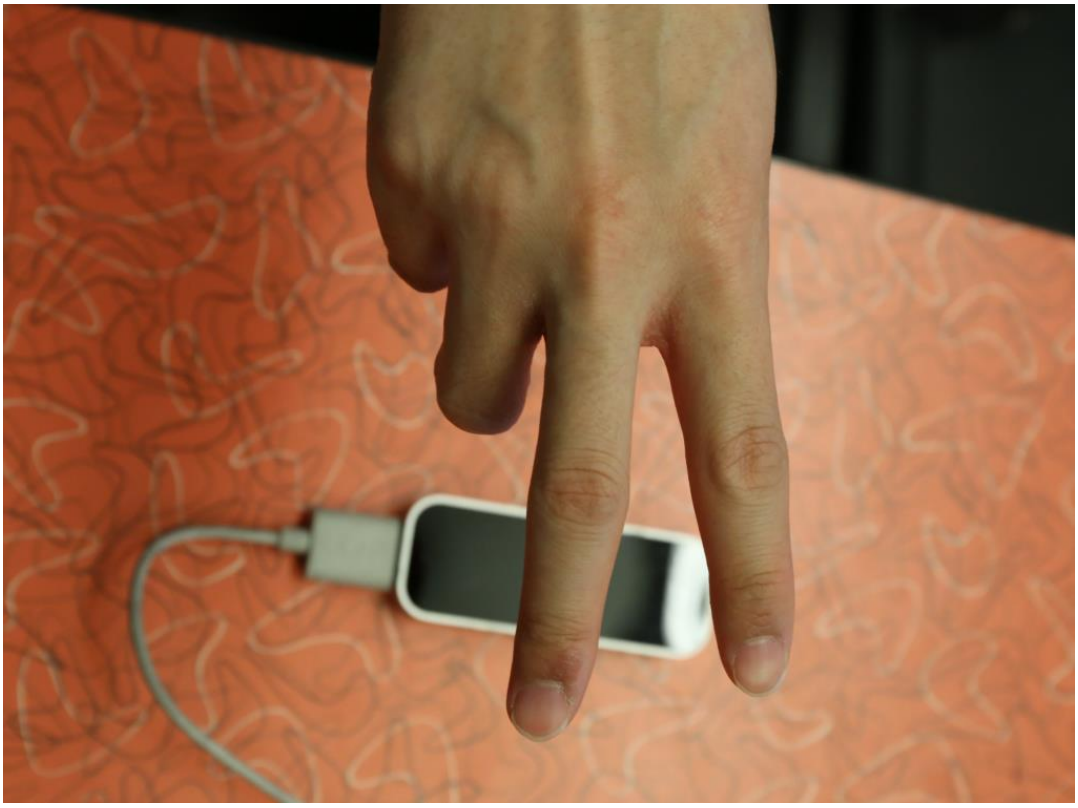

Figure A2. Gesture pick and drag (victory)

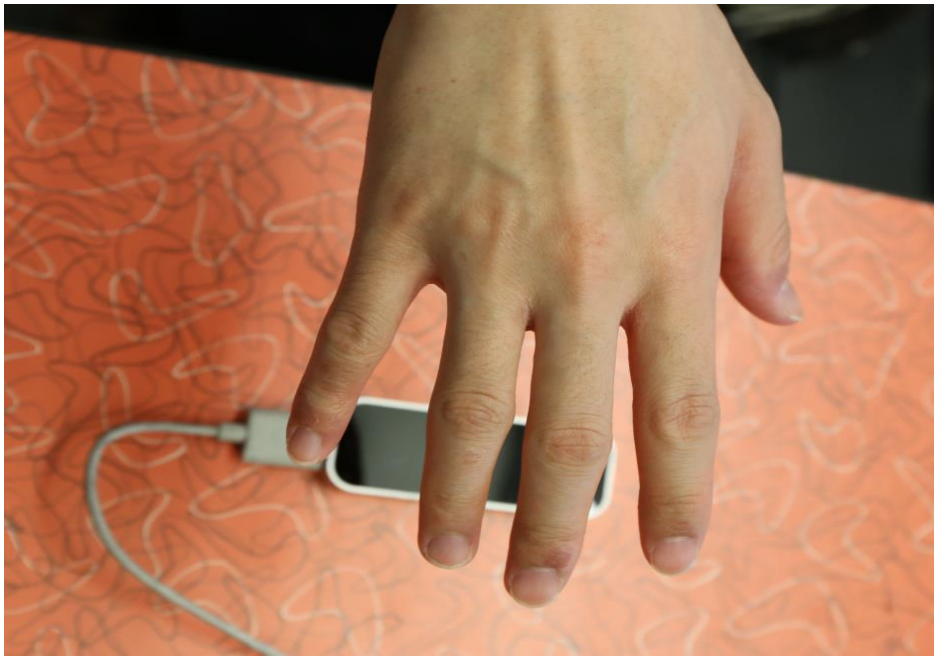

Figure A3. Gesture Push (one hand out)

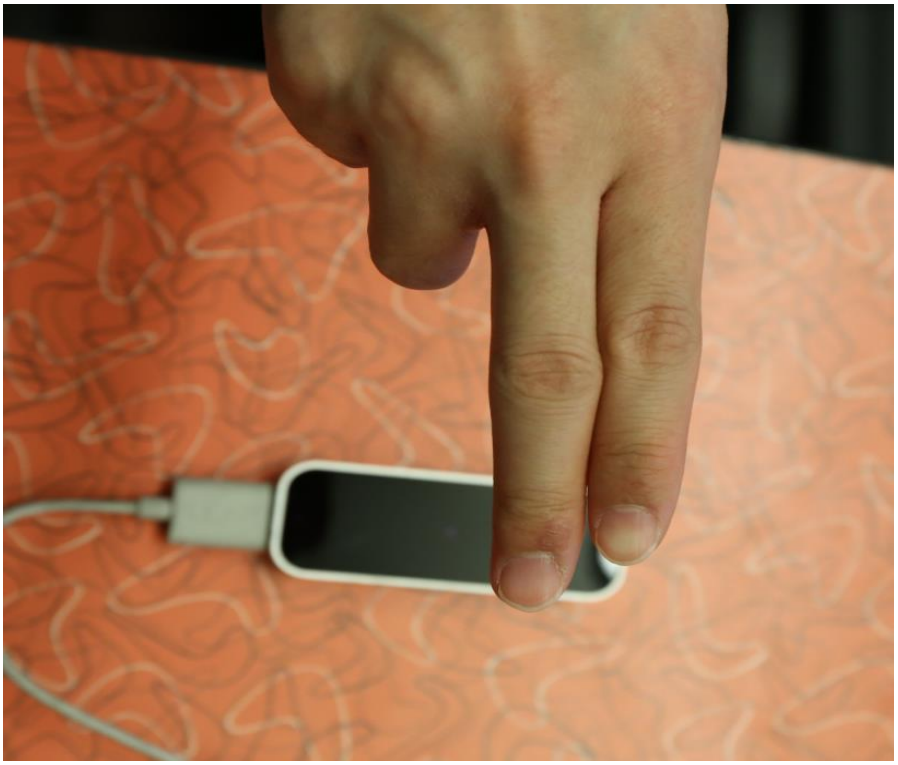

Figure A4. Gesture Drop (two finger close)

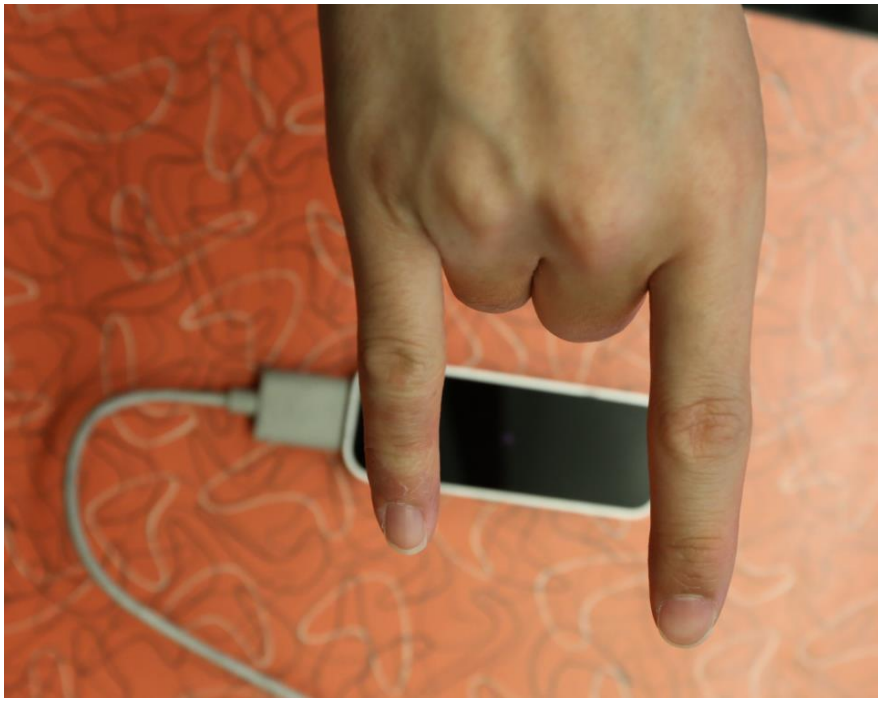

Figure A5. Gesture experimental use only (two finger open wide)