**Design Report**

***Abstract:***

**Background**Hand and body motion is becoming an ever more common form of human computer interaction. With applications in gaming, security, education, as well as general computer usage, it provides a natural, intuitive and easy way for users to interact with what they are seeing.

**Aims**The main aim of the project is to showcase a real-time, interactive application that allows a user to control objects and menus in both 2D and 3D space using purely gestures and motions made by their hands. It uses readily available hardware and with places only a few restrictions on the user. As such this will show the capabilities and feasibility of using human motion as a method of computer interaction. Within this I will study the effectiveness of using depth images generated from a Microsoft Kinect as the input data used to detect the hand position, as well as the effectiveness of using machine learning to analyse and interpret hand poses and gestures.

**Method**After initial research to determine the best methods, the three main aspects of the application (hand detection, pose estimation, and interactive environment) will be developed and tested separately, then integrated to create the final product. All development will be done in C++. Hand detection will utilise the Kinect SDK and OpenCV, pose estimation will utilise OpenCV, the visual and interactive environment will utilise OpenGL and Bullet Physics Engine.

**Proposed solution**

A fully interactive application where the user can interact with objects in a 3D environment as well as with 2D menus. The functionality of the application will show off the capabilities of the system with allowed gestures such as pushing a button, picking up an object, throwing an object, pulling a lever, etc. The user will be able to use both their hands at the same time. It will require them to be stood within a certain distance from the Kinect camera and with a relatively clear area with nothing in between the user and the camera.

**Keywords –** Human Computer Interaction, Microsoft Kinect, OpenCV, OpenGL, Bullet, C++, hand motion, hand tracking, gesture recognition, pose recognition

**I. Introduction**

***1. Background***

Hand oriented motion control has a wide range of uses but the most notable of those is gaming, in particular one of the fastest developing areas of the gaming industry, virtual reality (VR). VR is a natural fit for a system like this and using just your hands as a control method will provide an even more immersive experience to pressing buttons on a controller. Besides VR, normal games can utilise this method of control as well. 2D menus naturally lend themselves to be manipulated by a user’s hands, as well as any game or application that involves interaction with objects in 3D space being suitable for hand motion controls.

Systems that use hand motion controls already exist, most notably Microsoft’s Xbox Kinect games and the Leap Motion controller.

Xbox Kinect enables games tend to focus on whole body movements and very few use specific hand gestures as a method of control. When they do, it is normally for controlling a 2D menu and only the position of the hand is taken into account. The Xbox Kinect v2 does have gesture recognition capabilities[1], however it does not seem to be used in any games, and was in fact removed as a method of control for navigating the systems menus due to low usage numbers.

The Leap Motion controller is a small piece of hardware that sits below a user’s hands and accurately tracks the exact pose allowing for very precise interaction. However the hands must stay within a small radius of the device.

***2. Project Aim***

The aim of this project is to test the feasibility of creating a fully functioning, responsive, simple, and easy to use application controlled only by hand movements made by the user. By demonstrating that such an application works well, it shows the possibility of using hand motion as an alternative method of computer control. It also investigates the possibility of incorporating the use of two hands in a scenario such as this one, something the literature often overlooks, focussing almost always on a single hand.

As mentioned earlier, the hand control methods that the Xbox One provided were never really adopted by users. I think the main reason behind this is that the applications that used the technology were not specifically designed with this method of interaction in mind, or an alternative method was easier to use. As such I will ensure that my final application is created with hand control in mind.

To test the feasibility and success of the project, the final application will be tested on users who have never seen the system before. This will judge how reliable it is as a method of interaction, its ease of use, and its accuracy. It will also see how well the system copes with a range of people all with different hand shapes and sizes.

***3. Hand Tracking***

The initial part of the project is hand tracking which involves identifying the location of the user’s hands from a single input stream. I have chosen to use depth imagery as the input data as it is widely considered to be superior to standard RGB images for the task of hand position detection[1]. As such I have chosen the Microsoft Kinect v2 to be the input device[2] as it produces a high quality 640x480 pixel depth input stream in real time by processing up to 2 gigabits of data per second using its infra-red depth camera. It is also now a fairly cheap and widely used piece of kit. Depth images have many benefits over standard colour images including light invariance and background and user colour invariance. They of course also provide very useful depth information which is helpful for identifying the hand location, and makes it very easy to incorporate with a 3D application as the depth value can easily be obtained.

Using this depth image stream with computer vision and image processing techniques will, with high accuracy, identify the location (x, y, and z coordinates) of the two hands of the user.

***4. Pose Identification***

The second part of the project is to identify the pose that is being made by each of the hands at any one moment. Taking the region of the hand as identified in the previous step, a number of data features can be extracted from that region and input into a classifier trained using machine learning. [3] reviews numerous methods for pose estimation. It classifies two main approaches for doing this, “partial pose estimation” and “full pose estimation”. Partial pose estimation involves identifying a hand to be in one of a set discrete hand poses. Full pose estimation effectively involves creating a full model of the hand such that any single pose can be recreated by this model. I decided to go for a partial pose estimation method as this is more suitable to the required functionality of my application since only a few discrete gestures/poses will be needed to interact with the application.

I decided to just have a few recognised poses in my application as I think recognition accuracy is preferable to more functionality. The poses that I will allow include closed fist, open palm, 1 finger up, and 2 fingers up. Slight occlusions will also be tolerated, but in general the hand must be not be rotated by any more than about 45O in any direction.

The poses will be determined by machine learning. The classifier will be trained on a large training set with a wide range of different users providing the input data. The classifier can then be run each frame of the application using the obtained feature data and a pose can be estimated in real time. Because the classification uses only a small amount of data as the feature data, this can be done quickly which allows it to be done per frame. A machine learning method also allows for easy addition of more poses as more data can be provided and the classifier re-trained.

***5. Application***

The application is a way to show the workings and capabilities of the previous two stages. There will be two main parts to the application, the 2D menu and the 3D environment. The 2D menu is where the user can perform actions such as cycling through options by grabbing and moving, or selecting an option by pressing their open hand forward as if they were pressing a button. It is also where they can change settings for the whole application such as which 3D environment they want to use, for example. An initialisation step will be required before using any of the features so that the user does not accidentally do something they do not want to by moving over a feature. This initialisation will be in the form of the user hovering their hand in front of a feature for about a second before they perform the gesture.

The 3D environment is of the form of an interactive space with a number of objects that can be manipulated by the user. An example of the kind of thing that can be found is a cube that can be picked up by the user hovering their hand (represented by a 3D model on the screen) over the object and making the grabbing gesture (going from open palm to closed fist). Then will then be able to pick up and move around the cube, drop it, or throw it by moving and dropping the cube. The cube will interact with other objects thanks to the Bullet physics engine that will be incorporated with the application.

***6. Deliverables***

*Basic –*

* Locate and track a user’s hands in real time using the input data from the Kinect
* Estimate the pose of a hand with machine learning using samples obtained from the identified hand location
* Create a simple 3D environment with physics incorporated, no interaction necessary

*Intermediate –*

* Identification of the pose made by a single hand to occur in real time as hand tracking
* Enable interaction with the 3D environment

*Advanced –*

* Include the 2D interactive menu
* Allow application to work with two hands at a time
* Evaluate the systems usability with a group of test users

**II. Design**

***1. System Architecture***

The whole system will be developed in C++. I chose this language because it will allows easy integration between the three main parts, and there is a wide range of well supported tools and libraries to help accomplish the tasks required at each stage of the project. It is also one of the languages supported by the Microsoft Kinect SDK.

***1.1 Hand Detection***

The first stage of detecting the hands is to generate the depth data from the Kinect. Using SDK (software development kit) provided free by Microsoft, the Kinect can be used similar to a normal webcam and produce a constant stream of input data frames of depth images. Mapping the depth value to a colour value can provide a visual representation of depth, see figure 1. To make sure there is a good colour gradient, and pixel that is beyond a certain depth is cut off. The range I use is between 500mm (the minimum) and 1500mm, thus the user has quite a wide range to move around in. Having a cut-off distance also ensures that there is no cluttered background, although because this is depth information and not colour information a cluttered background has less of an impact. Using OpenCV, the raw depth data can then be converted into a standard OpenCV image and thus processed to find the important information.

Most of the academic literature related to hand detection focusses on just a single hand. As such the basic method of finding the hand is to find the closest object to the Kinect. This is a slight limitation of the system, but it is unlikely that there would be anything in between the user and the camera anyway.

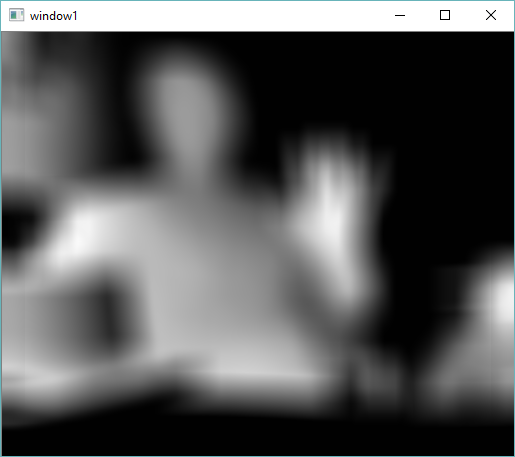
Due to the nature of the hand, the closest individual pixel normally lies on a fingertip. This is not particularly useful, and finding a standardised point on the hand each time. To do this I use a technique described in Dominio et al.[5] where a Gaussian blur with a high standard deviation and large kernel size is applied to the image. I found that a kernel size of 45x45 and standard deviation of 100 works well. The Gaussian blur makes each pixel have a value equal to the weighted sum of its surrounding pixels. The kernel size determines the range of the pixels included surrounding pixels, the standard deviation determines how distance of a surrounding pixel corresponds to the weight it is assigned. Since the centre of the hand has the highest density of high intensity pixels, after the blur this will be the location of the highest intensity point (see figure 2).

To find the hand centre the image can be scanned through and the points of highest intensity can be found. It may be the case that one hand is further away than another, so to ensure that the two highest intensity points do not come from the same hand, the two points must be a certain distance away. To speed up the process, only every 2nd pixel is checked.

Obviously the hands need to be located in each frame, but to prevent searching the entire image each time, only the area around the position of the hands in the previous frame needs to be checked. Since the hand will not suddenly move a great distance between one frame and the next, it is fair to assume that the hand will be within a small radius of its position in the previous frame. As such, the search space is greatly reduced and thus the time searching for the hands each frame is also greatly reduced. I have found that a search area of 120x120 around the centre pixel of the hand from the previous frame works well. This area is scaled depending on the distance from the Kinect of the hand in the previous frame.

The system is suitably robust. The hands can move at high speeds and not be lost, they can be detected at any angle or pose, they can be close and sill be distinctly detected, and they can move off screen (i.e. out of the range of the Kinect) and still be detected as soon as they move back into the frame (this is because the arm is detected instead of the hand). Perhaps the biggest limitation is that no hand crossover can occur. However, should the hands fail to be tracked correctly, all that is required is a quick re-initialisation that involves the user holding their hands out in front of them and the system can quickly find the hands once again.

Apart from the use of Gaussian blur, this method is one of my own creation. As mentioned earlier, few studies address the scenario of having two hands so there are not many methods described to deal with such a problem.



*Figure 1 Figure 2*

***1.2 Pose Identification***

Once the positions of the hands are found, the pose they make must be determined. A range of methods have been proposed to perform such as task. I decided that a machine learning based approach would be a suitable way forward since the number of allowed poses will be relatively low, and there will be a big discrepancy between the poses.

In order to have a high classification accuracy, I decided that a lower number of poses is better. By having fewer poses, there are fewer options for the classifier to choose from meaning it will be incorrect less of the time. Also with just a few different poses, a lot of functionality will still be available, especially when combined with movements to make a gesture.

The poses I have decided to use are open palm, closed fist, one finger up (includes thumb in the in and out position) and two fingers up (also includes thumb in and out). These poses are orientation invariant, meaning that an open palm facing to the right is the same classified pose as an open palm facing to the left. Intermediate poses will also be classified as one of the distinct poses. For example when the fingers are in a state of closing they will be classified as either fully open or fully closed. This is appropriate as very rarely will the hand be in a state where it is in a half closed position intentionally.

In order to carry out the machine learning, the image of each hand will need to be represented in a certain data format. I decided that raw pixel values would be too much data so decided to use a method that is commonly used to represent hand poses that uses the concept of a convex hull. A convex hull is a minimal area around set of points X such that a line between any two points in X is also within the area. The points that make up the outside of that area, i.e. any point on the edge of the convex hull of the hand, are the points that are given to the classifier.

To obtain the convex hull of a hand image a number of steps[6] are taken using OpenCV. The input is the 120x120 hand region as determined by the previous part. This image has a threshold applied so that any part of the image not belonging to the hand i.e. has a colour value below a certain value is removed. I use a value of “highest intensity in the image – 40” which nearly always removes all background detail whilst also ensuring that the whole hand is kept in the image. The contours of the image are then found, with each contour being a sequence of points around that enclose an area. As such there are a number of these contours found. The largest of these contours is taken to be the one that encloses the outline of the hand as this is by far the largest thing in the whole image. This contour is then the input for the convex hull which returns a set of points. Due to the nature of the hand the convex hull points almost always include the fingertips and thumb tip, as well as other points that describe a hand shape. This set of points is what I use to represent a hand for the machine learning. INCLUDE IMAGES

The set of points in the convex hull does not always have the same number of points. To combat that issue and ensure that the input to the machine learning has a consistent number of variables, the data is padded out with interpolated points so that the total number of points is 25. Interpolation within machine learning data is generally considered a bad idea, but due to the nature of this data describing a continual shape (the outline of the hand) it will not have a big effect. Once all the points are generated, the absolute distance between the point and the centre of the hand is calculated for each point, and this is the final piece of data used. The distance is used over the absolute point for a number of reasons. Firstly the actual position of the points are meaningless if they just describe the global position in the whole image, whereas relative distances mean the points could be from anywhere in the image. Secondly, it allows the points to be scaled. This is important since the distances for a further away hand are smaller, meaning although the same pose may be made, the difference in the data points means the classifier believes it to be a different pose (most likely the closed fist as this is the pose where the points are the closest). COULD CHANGE THIS AS MAY DO DISTANCE + ANGLE.

25 data points is actually quite a small number for an image descriptor, but testing shows that it produces good accuracy with only about 200 data training samples. I believe that the accuracy is high because of the clear distinctions between the different poses. Having so few data points also means that the classifier is very quick to decide on a pose which is very important in a real time application such as this one.

There are a number of machine learning techniques that can be used. The input data is relatively simple and there are only a few classes, so there is no obvious choice of preferred technique. As such I have decided to go with Support Vector Machine (SVM) as it is known to perform well, and I have had success with it previously.

The library I have chosen to use for machine learning is OpenCV. Despite being mainly an image processing and computer vision library, due to the importance of machine learning in computer vision it provides a wide range of functionality for performing machine learning, including SVM. I chose this library over other, arguably better and faster, machine learning libraries for a couple of reasons. Firstly I have experience using the library, so no time will be wasted learning its basic functionality. Secondly I am already using OpenCV in my project, so it seemed a good idea to keep the number of external dependencies to a minimum. Despite OpenCV being slower compared to other machine learning specific libraries, the task it has to perform is one that is relatively simple and as such the speed improvements gained from using alternatives will be negligible.

The classifier will need to be run twice, once for each hand. To get around the issue of the left and right hands having different appearances, the left hand image can just be flipped in the horizontal direction, thus appearing to have the same shape as the right hand.

Re-initialisation

***1.3 Application***

***2. Testing***

***3. Requirements (F + NF)***

**References**

[1] Toby Sharp, Cem Keskin, Duncan Robertson, Jonathan Taylor, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, Alon Vinnikov, Yichen Wei, Daniel Freedman, Pushmeet Kohli, Eyal Krupka, Andrew Fitzgibbon, Shahram Izadi (2015) “Accurate, Robust, and Flexible Real-time Hand Tracking” *CHI '15 Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* p3633 - 3642

[2] T. D’Orazio, R. Marani, V. Renò, G. Cicirelli (2016) “Recent trends in gesture recognition: how depth data has improved classical approaches” *Image and Vision Computing 52 (2016)* p56 – 72

[3] Lulu Chen, Hong Wei, James Ferryman (2013) “A survey of human motion analysis using depth imagery” *Pattern Recognition Letters 34* p1995 – 2006

[4] Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, Xander Twombly (2007) “Vision-based hand pose estimation: A review” *Computer Vision and Image Understanding 108* p52 - 73

[5] Fabio Dominio, Mauro Donadeo Pietro Zanuttigh (2013) “Combining multiple depth-based descriptors for hand gesture recognition” *Pattern Recognition Letters 50 (2014)* p101 – 111

[6] <http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/hull/hull.html>