A Smartphone-based Controller for Virtual Reality Applications

John Heukers
10050027

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisors

Dhr. dr. Robert Belleman

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 107
1098 XG Amsterdam

Dhr. dr. Frank Nack

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 107
1098 XG Amsterdam

June 27th, 2014

Contents

1 Abstract

This research claims that a smartphone can be used to create a controller for virtual reality applications. Such a controller would require motion tracking to provide a mapping of the user's physical body movements to that of the virtual avatar. The validity of this claim is tested by the construction of a prototype that uses the accelerometer, magnetometer and gyroscope of a smartphone to determine its orientation, and the touchscreen to determine the location of the user's thumb. A virtual scene consisting of a virtual arm holding a smartphone uses this information to mimic the rotation of the user's arm and the location of thumb. The performance of these pose estimations by the prototype has been evaluated on a visual basis by overlaying the pose of the physical arm and thumb onto that of the virtual model. The results indicate that a smartphone is capable of accurately tracking the rotation of the lower arm and hand around its three axis, and to relay the position of the user's thumb as long as it maintains contact with the touchscreen. While the virtual representation is capable of reproducing the rotation of the arm and the translation of the thumb, there are several aspects of the pose estimation that cannot be tracked due to a lack of data. These limitations include the manner in which the user is holding the smartphone and a number of unnatural thumb pose solutions found by the inverse kinematics algorithm.

2 Introduction

Virtual reality has seen an increase in attention during the past two years. The upcoming Head-Mounted Displays (HMDs) such as the Oculus Rift and Sony's Project Morpheus are likely to provide a very compelling experience for consumers. This surge in interest and innovation in the virtual reality field has given rise to numerous startup companies and research groups with a specific focus on virtual reality technology. Research areas of interest include display technology, motion tracking, interaction techniques, haptic feedback and many more. As the display technology in HMDs improves to a point where prolonged use remains comfortable, the desire for appropriate interaction techniques grows larger. Most virtual reality applications have the ambition of convincing the users into believing they are no longer in the real world, but rather in what ever virtual space they see in the HMD. This sense of actually being in this virtual space is often referred to as *presence*, and the adequacy of the technological setup to accommodate this feeling is often called *immersion*. There are many factors of a virtual reality experience that have an effect on the level of immersion, and therefore the level of presence. One of these factors involves the user's body. The HMD effectively blocks all view of the outside world, including ones body. Therefore, most virtual reality applications provide the user with a virtual body. This virtual body will generally display most of the actions performed by the user, such as walking around, opening doors and piloting vehicles. However, the majority of current virtual realty applications require the use of a keyboard and mouse or game controller to control the user's avatar. Because the user is unable to view the outside world, they are also unable to look for specific keys on the keyboard or game controller, forcing the user to navigate the controls by touch. Additionally these interaction techniques can all be considered super-human interfaces. A small action on a game controller such as pressing a button or slightly nudging a joystick can result in relatively large actions such as turning the user's body and pushing obstacles aside. This can cause a disconnect between the actions perceived by the users, as they observe the actions being performed by their avatar, and the gestures they command their physical body to make. As will become apparent in the following section, various studies have shown that a greater mapping of a user's physical gestures to those of the virtual avatar can greatly benefit immersion and therefore presence. Achieving an accurate mapping of gestures will require motion capture technology. This technology generally uses either sensors such as accelerometers and gyroscopes, optical systems such as RGB cameras and infrared depth cameras or a combinations thereof. These systems are often bulky and expensive, and they sometimes even require a dedicated room. This greatly affects the accessibility of these solutions for the average consumer, forcing them to stick with an interaction technique based on a keyboard or game controller resulting in a poor experience for most virtual reality applications. A sensor device that is already in the possession of a great majority of the consumers is the smartphone. Any modern smartphone has access to accelerometer, magnetometer and gyroscope sensor data as well as having a touchscreen. A combination of these components could be used to create a rudimentary motion tacking system to form the basis for a virtual reality controller. This project will focus on tracking the movement of a user's arm. The users place a smartphone in their hand and put on an HMD. In the virtual environment the users will be presented with a virtual body holding a virtual smartphone in its hand. The accelerometer, magnetometer and gyroscope will be used to determine the ori-

entation of the physical smartphone. This orientation also dictates the orientation of the virtual smartphone, and the virtual arm holding the smartphone will adjust its pose accordingly. If the user's thumb touches the smartphone's touchscreen, those coordinates will be sent to the virtual environment. The thumb of the avatar will then be placed on the corresponding coordinates on the virtual smartphone, effectively mimicking the movement of the thumb as long as it maintains contact with the screen. This would result in a system that can track the rotation of a single lower arm and hand and the translation of the thumb across a smartphone screen. This system could provide numerous virtual reality experiences with an immersive and intuitive controller for various interaction techniques. For example, the user could rotate the smartphone to point at a virtual object and manipulate it by selecting it. Additionally the virtual smartphone screen could be used to display various menus and options, allowing the users to navigate and interact with it much like they would with a physical smartphone. In summary this project aims to answer the following research question: *Can a handheld smartphone be used to create a controller for virtual reality applications?* To answer this question a prototype of such a system will be constructed. This prototype will then be evaluated on the range and accuracy of its motion mapping to the virtual environment in addition to a review of its latency.

3 Literature review

Literature most relevant to this project involves algorithms to determine the orientation of a device via accelerometer, magnetometer and gyroscope sensor data, inverse kinematic algorithms to determine the proper position of a kinematic chain given a target location and the relationship between interaction methods and immersion in virtual reality applications.

3.1 inertial measurement unit pose estimation

Siegling et al. (2011) [7] describes the challenges of gathering sensor data using a smartphone. It appears that smartphones are not able to gather sensor data at a consistent interval, this is larger due to the multitasking behaviour of smartphone operating systems. The primary processor switches between tasks on a timer, even if the current task is not yet completed. This behavior can result in occasional sensor measurements being skipped in favor of another process. They propose interpolation methods as a possible solution. Lawitzki (2012) [2] uses the sensors of a smartphone attached to a headset to track the orientation of the user's head. This information is then used to calculate the location of a bineural audio source. Using smartphone sensor data to track its orientation is not a trivial matter. Each of the sensors in a smartphone has strengths and weaknesses: The accelerometer and magnetometer have poor short-term accuracy, yet robustly maintain their orientation over the long-term, This is in contrast to the gyroscope which is highly accurate yet loses its orientation of the long-term. It would appear that the best method to construct an accurate and responsive tracking system is to combine these three sensors. Such an implementation is generally referred to as *sensor fusion*. Lawitzki's master's thesis includes a well-documented implementation of sensor fusion that has been invaluable to the prototype stage of this project.

3.2 Inverse Kinematics

Aristidou et al.(2009) [1] wrote a comprehensive paper comparing various inverse kinematic algorithms and suggests a novel solution that is computationally cheap, supports movement constraints and allows for natural motion. This algorithm is called Forward And Backward Reaching Inverse Kinematics (FABRIK). Instead of using rotational angles it transforms the positon of a kinematic chain's joints via lines connected between neighbouring joints. This implementation was then compared to various other algorithms on computational cost and quality of found solutions. The inverse kinematic algorithms included in the evaluations were CCD, Jacobian Transpose, Jaconbian DLS, Jacobian SV-DLS and Triangulation. The experimental results indicate that the FABRIK algorithm is computationally cheaper than the other algorithms, requiring the fewest iterations to reach the desired target. While in motion the solutions generated by the FABRIK algorithm produce the most natural movements compared to the other algorithms.

3.3 Immersion

The first paper of interest, McMahan et al.(2012) [5], concerns the effect of varying levels of both display fidelity and interaction fidelity on the level of immersion. Test subjects were placed in a CAVE system with either a low or high fidelity display, and a low or high fidelity interaction technique. The subjects were then asked to complete various tasks while their performance was being measured. Their perceived sense of immersion was also measured based on a questionnaire. The results indicated that both a high fidelity display and high fidelity interaction technique offered the greatest sense of immersion. Slater et al. (1995) [8] did a comprehensive study on different types of interaction techniques for locomotion in a virtual space. These methods included pointing in the desired direction to move, or walking-in-place to move towards the faced direction. It also goes in-depth to define immersion as the extent to which the virtual world is extensive, surrounding, inclusive, vivid and matching while arguing that immersion requires a good match between the user's physical movements and those displayed in the virtual world. Thus, the greater the precision of physical movement mapping to the virtual movement, the greater the immersion. These articles are very useful because they reveal the important features of interaction techniques that have an effect on the level of immersion and support the claim that motion tracking is a requirement for a well performing virtual reality controller.

4 Approach

The first stage of this project involves the construction of a prototype. This prototype will use the accelerometer, magnetometer and gyroscope of a smartphone to determine its orientation, and the coordinates of the user's thumb contact with the touchscreen to determine the location of the user's thumb. This data will then be send to a remote desktop computer via a wifi connection. A game engine running on the desktop computer will receive and process the sensor data. The virtual scene of the game engine will consist of an avatar holding a virtual smartphone in its left hand. The orientation of the virtual smartphone will be dictated by the orientation data received from the physical smartphone. The virtual arm connected to the smartphone will adjust its pose accordingly by either rotating at the wrist or around the lower arm. The thumb of the virtual hand

will received its target location via the coordinates of the touch event of the physical smartphone. An inverse kinematics script will ensure a proper position of the thumb's kinematic chain. The combinations of these components will result in a prototype capable of tracking rotations of the lower arm and hand, as well as track the location of the thumb as long as contact with the touchscreen is maintained. The performance of these components will be evaluated on a visual basis by comparing the pose of the physical arm and thumb with their virtual representations. The evaluation will be concluded with a measurement of the prototype's latency.

5 Theoretical foundation

This section will provide the theoretical knowledge behind the sensor fusion algorithm, which is used to track the rotation of the user's arm, and inverse kinematics algorithms which is used to determine the pose of the virtual thumb.

5.1 Sensor fusion

Determining the orientation of a smartphone is a common problem with various solutions ranging in complexity and performance. The prototype of the virtual reality controller requires responsive and accurate tracking of the changes in orientation. Significant motion-to-photon lag or inaccurate orientation estimates would undoubtedly have a negative impact on the practicality of the smartphone as a controller. Currently, the sheer majority of smartphone games and applications that utilize the orientation of a smartphone in any meaningful manner rely on data from the accelerometer and the magnetometer. The accelerometer measures, as the name suggestions, the acceleration of the smartphone along three axis (see figure 1). To get an intuition into the workings of an accelerometer one could imagine it as a cube suspended by a spring on each side, and each of those springs is attached to the body of the smartphone. In an environment without gravitational forces (or during free-fall), the cube would not apply force to any of the strings, and the accelerometer would therefore return 0 for each of the axis. If the smartphone is resting on a stationary surface, it will measure an acceleration of $9.8m/s^2$ away from the center of gravity. This is because the cube is compressing the string below it, towards the center of gravity. The force applied the phone to keep it from falling towards the center of gravity is the value measured by the sensor, and therefore points away from the center of gravity.

The magnetometer measures the magnetic field in the environment and is often used as a compass in smartphones. However, this sensor does not only measure the magnetic field of the planet's poles. It also senses the magnetic fields generated by nearby electronic devices and even other components in the same smartphone. This sensitivity to other sources of magnetic fields causes the sensor to be vulnerable to noise. [6] The magnetometer also has three axis, because of this the magnetometer cannot function as a compass by itself because it lacks the information to know which axis should function as the pivot point or in other words, what the horizontal plane is. To address this problem it needs knowledge of the orientation of the smartphone, specifically it needs to know what is up. Fortunately the accelerometer can provide this information. Recall that the accelerometer can measure the direction of gravity. The direction pointing towards the center of gravity can be used to determine the pivot point of the magnetometer, effectively

forming a compass.

These two sensor can be combined to determine the orientation of a smartphone. The accelerometer can measure the pitch and roll of the smartphone. [3] The magnetometer aided by the data of the accelerometer can form a compass which can determine the yaw of the smartphone. This is sufficient information to determine the orientation the device.

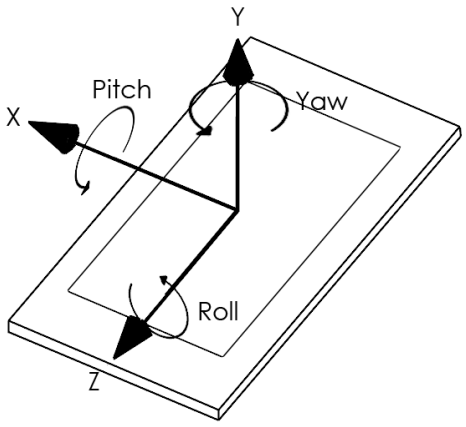


Figure 1: Axis of the smartphone

However, this solution is not optimal and does not meet the criteria of a fast and accurate motion tracker. The magnetometer is unable to react quickly to large changes in orientation, resulting is visible lagging. This problem can be addressed by adding information of the gyroscope to the system. The gyroscope is a sensor which has recently been a common addition to modern smartphones. The gyroscope used in smartphones is that of the Micro Electro Mechanical Systems (MEMS) variant. This type of gyroscope consists of a vibrating element inside the sensor that is affected by the Coriolis effect as its frame changes its orientation. This measures the angular velocity of the smartphone, which can be defined as the change of rotation per unit of time. The orientation of the smartphone can then be determined by integrating the angular velocity. The gyroscope has the great advantages of being very responsive to sudden changes in orientation and is capable of giving robust measurements even during large rotations. However during the calculation of the orientation small errors accumulate which result in increasingly inaccurate orientation results. This phenomenon is ofter referred to as $gyro\ drift$. This unfortunate characteristic of MEMS gyroscopes prevents it from being a practical solution for any application designed for prolonged use by the user.

This leaves the final option of combing the data of all three sensors, also known as sensor fusion. As previously mentioned, estimating the orientation of the smartphone via magnetometer and accelerometer has significant limitations including slow reaction time during large movements and noise, but it does maintain an accurate estimation of the orientation over the long-term. This is in contrast to the gyroscope which is responsive and accurate, however over time it suffers from gyro drift. The robust orientation estimation of the magnetometer and accelerometer can be combined with the orientation estimation of the gyroscope to cancel the gyro

drift while the gyroscope reacts to short-term changes in orientation. Combining the orientation estimates of the magnetometer, accelerometer and gyroscope is done using a complementary filter. The resulting orientation replaces the current orientation estimate of the gyroscope to remove any of the errors that have accumulated in that estimate due to gyro drift. The specific implementation of this will be discussed in the following implementation section.

5.2 Inverse Kinematics

The inverse kinematics algorithm is required to be computationally efficient, allow for smooth, natural motion and should support contraints to movement. Computational performance is very important for virtual reality application since it can greatly affect the frame rate. The movement of the thumb should also mimmic the movements of a real thumb as closely as possible, involving accurate range of motion and smooth movements. Applying constraints to the movements of the virtual thumb will ensure that the possible solutions of the inverse kinematics script remain within the natural range of motion of a thumb. As discussed in the literature review, the FABRIK algorithm appears to meet these requirements. The algorithm for FABRIK can be found below.

Algorithm 1: A full iteration of the FABRIK algorithm.

Input: The joint positions $\mathbf{p}_i$ for $i = 1, ..., n.$, the target position $\mathbf{t}$ and the distances between each joint $d_i = |\mathbf{p}_{i+1} - \mathbf{p}_i|$ for $i = 1, ..., n-1$.

Output: The new joint positions $\mathbf{p}_i$ for $i = 1, ..., n$.

```
% The distance between root and target
dist = |p_1 − t|
% Check whether the target is within reach
if dist > d_1 + d_2 + ... + d_{n−1} then
    % The target is unreachable
    for i = 1, ..., n − 1 do
        % Find the distance r_i between the target t and the joint position p_i
        r_i = |t − p_i|
        λ_i = d_i / r_i
        % Find the new joint positions p_i.
        p_{i+1} = (1 − λ_i) p_i + λ_i t
    end
else
    % The target is reachable; thus, set as b the initial position of the joint p_1
    b = p_1
    % Check whether the distance between the end effector p_n and the target t is greater than a
    tolerance.
    dif_A = |p_n − t|
    while dif_A > tol do
        % STAGE 1: FORWARD REACHING
        % Set the end effector p_n as target t
        p_n = t
        for i = n − 1, ..., 1 do
            % Find the distance r_i between the new joint position p_{i+1} and the joint p_i
            r_i = |p_{i+1} − p_i|
            λ_i = d_i / r_i
            % Find the new joint positions p_i.
            p_i = (1 − λ_i) p_{i+1} + λ_i p_i
        end
        % STAGE 2: BACKWARD REACHING
        % Set the root p_1 its initial position.
        p_1 = b
        for i = 1, ..., n − 1 do
            % Find the distance r_i between the new joint position p_i and the joint p_{i+1}
            r_i = |p_{i+1} − p_i|
            λ_i = d_i / r_i
            % Find the new joint positions p_i.
            p_{i+1} = (1 − λ_i) p_i + λ_i p_{i+1}
        end
        dif_A = |p_n − t|
    end
end
```

Figure 2: The FABRIK algorithm [1]

This algorithm does not apply rotational changes to the joints of the kinematic chain, as is common amongst inverse kinematic algorithms, instead it translates the position of the joints to orient the segments of the chain into their proper position. The first step of this algorithm is to determine whether the target location for the thumb in is range by calculating the total length of the thumb. This is done by taking the sum of the length of each segment and then comparing it to the distance of the root to the target location. If the target location is not in reach then the calculation is terminated. Otherwise the algorithm enters the forward reaching stage. During this stage the end effector P_n is placed on the location of the target t. Between the new position of the end effector $P'n$ and the position of the next joint P_{n-1} the line l_{n-1} can be defined with a length of d_{n-1}, the end of this line is the new position of P_{n-1}. These same steps are then repeated for all of the remaining joints. However, this would cause the root, P_1 to change position as it moves into its place on the l_1 line. To address this problem the algorithm executes the backward reaching stage. During this stage the steps of the forward reaching stage are repeated from the top of the chain (P_1) to the bottom (P_n). This results in a kinematic chain that has remained stationary on its root position, but is now closer to reaching the target location. These steps form a single iteration. Within the FABRIK algorithm a limit is set to the maximum number of allowed iterations, if the limit is reached then the algorithm will terminate.

6 Implementation

The implementation of the prototype can be divided into four parts: The android application to gather, process and send sensor data, the Unity scene which contains the virtual environment including the virtual avatar, the sensor fusion algoritm to determine the orientation of the smartphone and the inverse kinematics script to determine the movement of the thumb.

6.1 Hardware setup

The hardware used to implement and evaluate the prototype is listed below. Unity 4.4.3 was used on the desktop computer to create the virtual environment.

Dekstop computer	
CPU	Intel Core 2 Quad Q6600 2.40Ghz
GPU	ATI Radeon HD 5770
Memory	4 GB DDR2 800MHz
Operating System	Microsoft Windows 7 64bit

Smartphone	
Name	Samsung Galaxy S4
CPU	1.6Ghz octa-core
Memory	2GB
Operating System	Android 4.4.2
Sensors	Accelerometer, magnetometer, gyroscope, RGB light, barometer, proximity, gesture, temperature, humidity, hall sensor

6.2 Android application

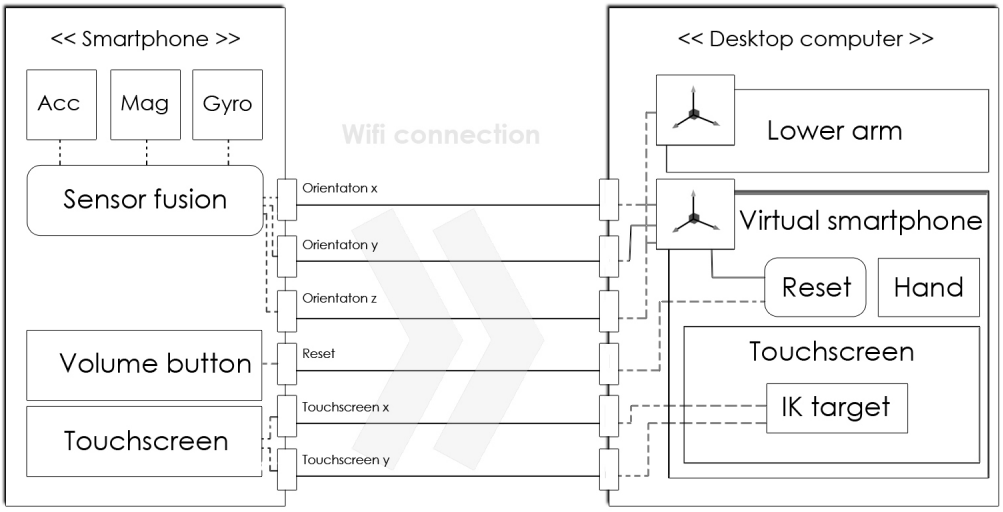


Figure 3: Diagram of the system

The Android Application gathers the sensor data from the accelerometer, magnetometer and gyroscope and processes these into a single estimate of the orientation represented as 3D coordinates. It also gathers the location of the user's thumb on the touchscreen, contact with the touchscreen is often called a *touch event*. Touch events are represented by a set of 2D coordinates, these coordinates are defined as the number of pixels in both width and height from the top-left of the screen. Occasionally the user's position changes after the first initialization of the application, for example due to movement in a rotating chair. Any changes in the user's orientation will be considered rotational changes of the smartphone itself, since there is no frame of reference to determine the orientation of the user and smartphone independently. In such a case it is necessary to hold the physical smartphone straight and to then reset the position of the virtual smartphone to its default, straight position. This is done in the Android application by

listening for any interaction with the volume button on the smartphone. If the volume button is pressed, then a boolean is sent that will trigger the reset command in the Unity engine. The smartphone gathers the orientation data, touchscreen coordinates and reset value into a single packet and transmits this to the server running on the desktop via wifi. To achieve this the Open Sound Control (OSC) java library is used [9].

6.3 Unity scene

The virtual scene in the Unity engine consists of an avatar holding a virtual smartphone in its left hand. The smartphone's dimensions of both the case and the screen have been modeled after the dimensions of the Samsung Galaxy S4 smartphone to accommodate an accurate evaluation of the pose estimation and the inverse kinematics performance.

The orientation data received from the smartphone is used to rotate the smartphone model, however rather than rotating the smartphone around its center, the pivot point has been moved the the location of the virtual wrist. In addition the virtual hand is attached to the smartphone, so any rotation made by the smartphone model will also apply to the hand. This simple design effectively mimics the rotating the wrist while holding a smartphone. To ensure proper behavior of the arm model, the rotation data of the roll movement is also passed on to the model of the lower arm to rotate along with the wrist. The rotation values are applied using the quaternion system to avoid the risk of gimbal lock.

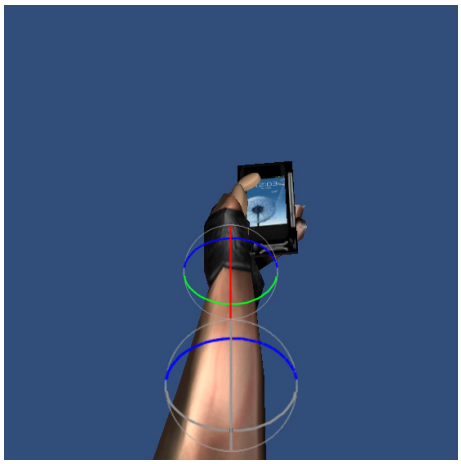


Figure 4: Unity scene

The red circle represents pitch
The blue circle represents roll
The green circle represents yaw

The values of ths touch event coordinates are based on the number of pixels of the smartphone's screen. Therefore a single set of coordinates from different smartphone models with varying screen resolutions does not translate to the same position on the touchscreen. To address this problem the received touch event coordinates are translated into relative coordinates, taking

the smartphone's screen resolution into account. This calculation is done by the following code snippet:

```
touchX = (inputX / physcialResolutionX) * virtualResolutionX
touchY = (inputY / physcialResolutionY) * virtualResolutionY
```

The resulting set of coordinates corresponds to a location on the virtual touchscreen. This set of coordinates also marks the target location for the inverse kinematics script of the virtual thumb.

6.4 Sensor Fusion

The algorithm of this sensor fusion implementation is based on the implementation of Lawitzki (2012) [2], The raw data from the smartphone sensors require preprocessing before they can be used to estimate the orientation of the smartphone. Fortunately the Android API provides various methods to simplify this process. First the sensor data from the gyroscope is retrieved and its rotation vector is calculated. This is done by integrating the gyroscope data over time for each of its three axis. The simplified calculation for one of the three axis is show below:

```
gyroOrientation = gyroOrientation + (gyroValue * deltaTime)
```

The *gyroOrientation* value will suffer from gyro drift over time as the additional gyroscope values introduce small errors. To cancel the gyro drift the orientation estimate of the accelerometer and magnetometer is required. Android provides a helpful method for this calculation which takes the rotation vectors from both the accelerometer and the magnetometer and returns a rotation matrix. This rotation matrix can then be used to calculate the orientation matrix which is an estimation of the orientation of the smartphone.

```
SensorManager.getRotationMatrix(rotationMatrix,null,accValue,magValue)
SensorManager.getOrientation(rotationMatrix, accMagOrientation)
```

To effectively combine the orientation estimate of both the gyroscope and the accMagOrientation a complementary filter will be implemented. This filtering process needs to take place at a constant and frequent interval, therefore the implementation of the complementary filter is executed in a separate thread on command of a timer. At a sampling rate of 33Hz the following code is executed once every 30ms.

```
FILTER_COEFFICIENT = 0.98

fusedOrientation = FILTER_COEFFICIENT * gyroOrientation +
                   (1-FILTER_COEFFICIENT)* accMagOrientation

gyroOrientation = fusedOrientation
```

This snippet of code gives a weight to the gyroscope orientation and the accMag orientation. The combined values of the gyroscope orientation and the accMag orientation results in a fused orientation estimate. This new orientation estimate then replaces the current orientation estimate of the gyro orientation. Over time the robust long-term orientation estimate of the accelerometer and the magnetometer will correct the orientation estimated by the gyroscope. Since the gyroscope orientation estimate is replaced by the fused estimated, the gyro drift errors that would normally be introduced into the gyroscope orientation estimate are canceled. This implementation will result in an accurate estimation of the smartphone's orientation that is both responsive to large, short-term changes in rotation and well as maintaining a robust orientation over a long term, unaffected by gyro drift. These estimated orientation values are then passed on to the virtual smartphone to dictate its orientation and in turn the pose of the virtual arm that it is connected to.

6.5 Thumb pose estimation

The FABRIK algorithm is applied to a simple model of a thumb consisting of three capsule-shaped segments. The base of the thumb is rooted onto the virtual hand and the target of the thumb's end effector is determined by the coordinates of the touch event given by the physical smartphone. If the user's thumb is not making contact with the touchscreen, then the coordinates of the inverse kinematic target is adjusted to hover 1 cm above the last known touch coordinates on the virtual screen. This will allow the inverse kinematic algorithm to position the kinematic chain of the thumb in a hovering pose over the screen. This elevated resting position of the virtual thumb allows for the simulation of tapping behaviour. As the user briefly makes contact with the touchscreen for a tapping guesture, the virtual thumb will snap back into position on the touch event coordinates, after which it will return to the hovering position. In order to avoid unnatural poses of the thumb a number of constraints are placed on the inverse kinematics script. These constraints are on based on the range of motion of an average human thumb. [4].

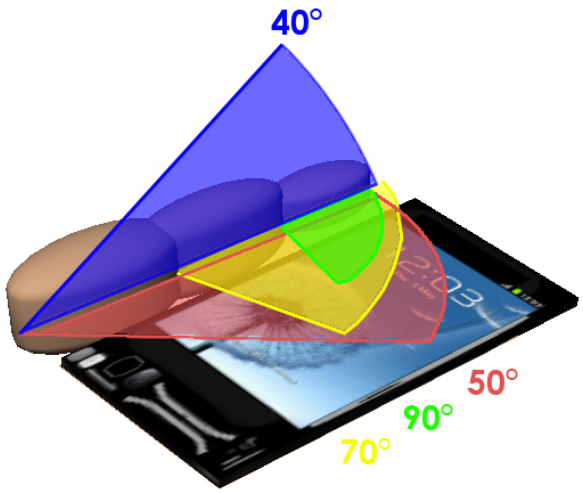


Figure 5: Range of motion of the thumb

As shown in figure 5, the first segment has an adduction range (blue) of 40°and an abduction range (red) of 50°. The second segment has a flex range (yellow) of 70°, and the final segment as a flex range (green) of 90°.

7 Evaluation and results

The purpose of this project is to determine whether a smartphone can be used as a virtual reality controller. The success of such a controller is largely based on aspects that determine the practicality. Users with an HMD are effectively blocked from all outside view, therefore having the ability to perceive the movements of ones body in the virtual environment becomes paramount. The evaluation of the controller implementation will focus on this aspect. The first factor to be evaluated is the accuracy of the pose estimation. If this virtual reality controller does not convey the proper pose of the lower arm to the user then this can lead to a very poor user experience. Take for example the interaction task of pointing the smartphone at a virtual object, if the pose estimation is off by a significant amount, then the user is forced to be mindful of this limitation during its use, this prevents the controller from being a natural form of interaction. In another example where the user is controlling a vehicle by using the smartphone as a steering wheel, any error in the pose estimation can result in a situation where the rotation of the smartphone no longer results in a properly corresponding steering angle of the car. The poses most relevant to this project concern the rotation of the lower arm, hand and movement of the thumb. Because the accuracy of the visual representation of the arm is most important, the model of the arm and thumb will be evaluated on a visual basis. Several images will be taken of both the physical and virtual arm in various poses. The image of the physical arm will be given an outline which will be overlaid onto the image of the virtual arm. This should give an indication as to the accuracy of the model since any significant differences in pose will be visible. The final part of the evaluation concerns the latency of the smartphone controller. Latency is very important in any

video game, but especially so for virtual reality applications. A good user experience requires the image of the virtual environment displayed inside the HMD to quickly and accurately reflect the actions of the user. In the case of motion controllers a high latency forces the users to plan actions and movements ahead of time and having to wait for their effects to become visible in the virtual environment on the screen. When using an HMD this is worsened as the users cannot see their physical body, and must therefore estimate the pose of their physical body while waiting for the virtual body to catch up with them.

7.1 Pose estimation

The pose estimation of the lower arm and hand is evaluated visually, comparing the visual representation of the virtual arm to that of the physical arm. This is done by recording both the physical and virtual arm and overlapping these two images. Accurately quantifying the angular difference between the pose of the physical arm and that of the virtual model using these images has proven to be difficult, instead a limit is set on the error allowed by the virtual pose estimation. The outline of the physical arm and thumb is used to define this limit. If the estimated pose treads outside of this boundary, then the estimated pose is considered to have a significant fault. Note that it is assumed that the user will hold the smartphone in a similar manner to that of the virtual model, since the sensor data do not provide enough information to accurately detect different grip location on the smartphone. The first step to this measurement is to suspend a downwards facing video camera over a flat surface. This camera is used to record the movements of the physical lower arm, hand and smartphone. On a remote computer the Unity engine is running the virtual environment in which the virtual camera is placed over the virtual arm is the same manner. Once the physical arm is straightened and flat against the table, the position reset command is given to the smartphone to reset the rotation values of the virtual smartphone, resulting in a virtual arm holding the smartphone straight. A number of wrist movements on each of the axis are then performed. These two videos will then be overlapped to determine any error in the pose estimation. The image of the first, straight pose of the virtual arm is overlapped onto the initial straight pose of the physical arm, performing a series of scaling and translation operations. The operations to synchronize the image of the virtual and physical arm will then be repeated for each subsequent frame of the video that is evaluated. A number of the poses in the recorded video are selected and overlaid onto each other. An outline of the image of the physical arm is overlaid onto the virtual arm to enable the visual comparison of the estimated pose. The images below show the overlaid images of a pose for each of the rotation axis of the arm and movement of the thumb.

7.1.1 Wrist pose

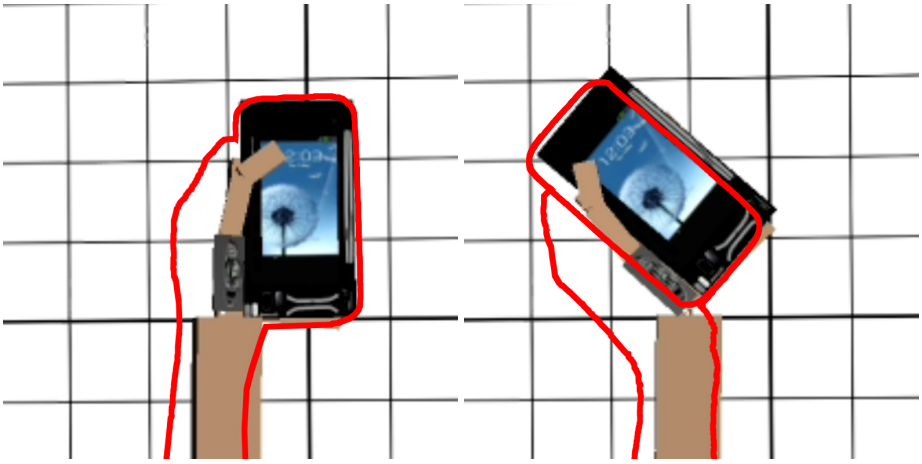


Figure 6: Pose comparison of the default and yaw

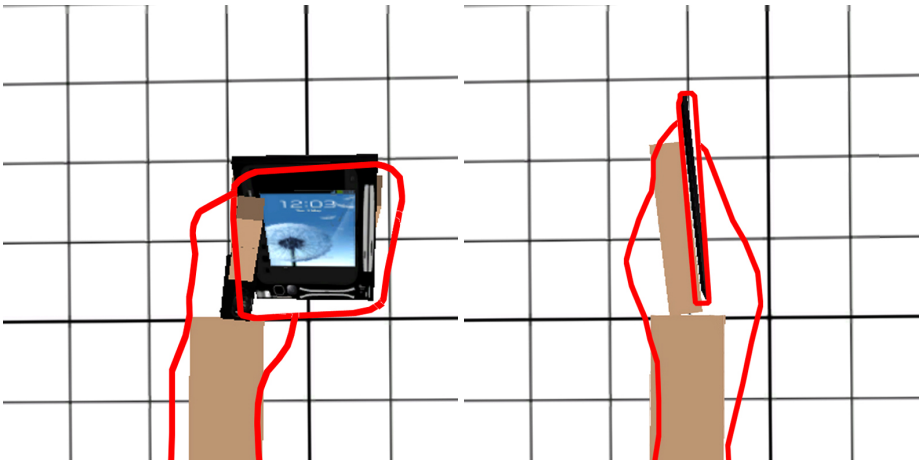


Figure 7: Pose comparison of the pitch and roll

These images appear to confirm that the sensor fusion implementation is accurate, as any difference in orientation of the physical and virtual smartphone is not measurable in an image of this quality. It does however reveal a limitation of this model: The left image of figure 7 shows a slight translation of the virtual smartphone, while the orientation remains similar. This translation is due to the distance of the physical smartphone to the pivot point in the physical wrist being different than that of the virtual model. Unfortunately the sensor data from the smartphone is unable to provide enough information to address this problem leaving manual adjustment as the only solution.

7.1.2 Thumb pose

The pose evaluation of the virtual thumb has been evaluated using the same method of the arm pose evaluation. An outline of the physical smartphone and thumb is overlaid onto the image of the virtual model to determine any faults in the pose estimation. A set of these images are shown below. The green dot on the images indicate the location of the touch event on the physical touchscreen, in other words this dot is the target location for the inverse kinematics algorithm.

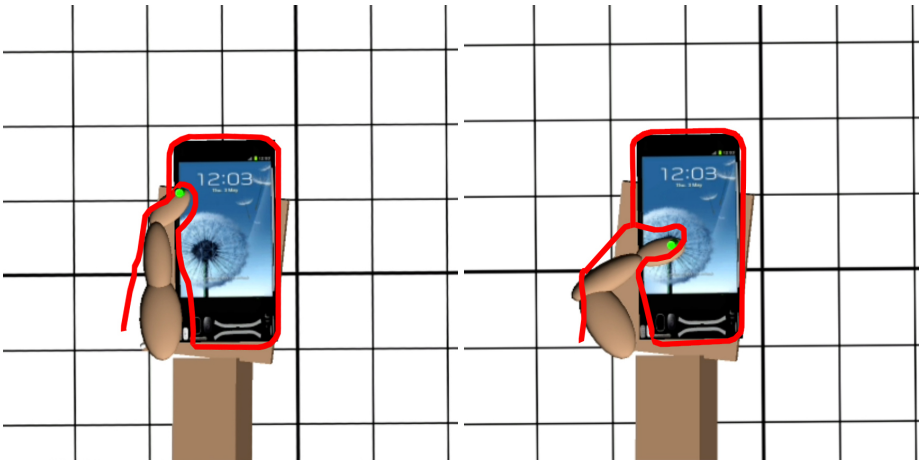


Figure 8: Pose comparison of the thumb

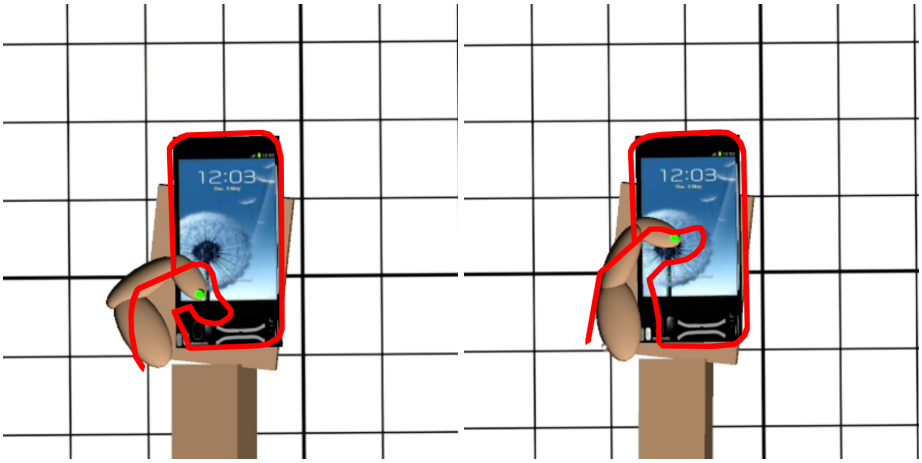


Figure 9: Pose comparison of the thumb

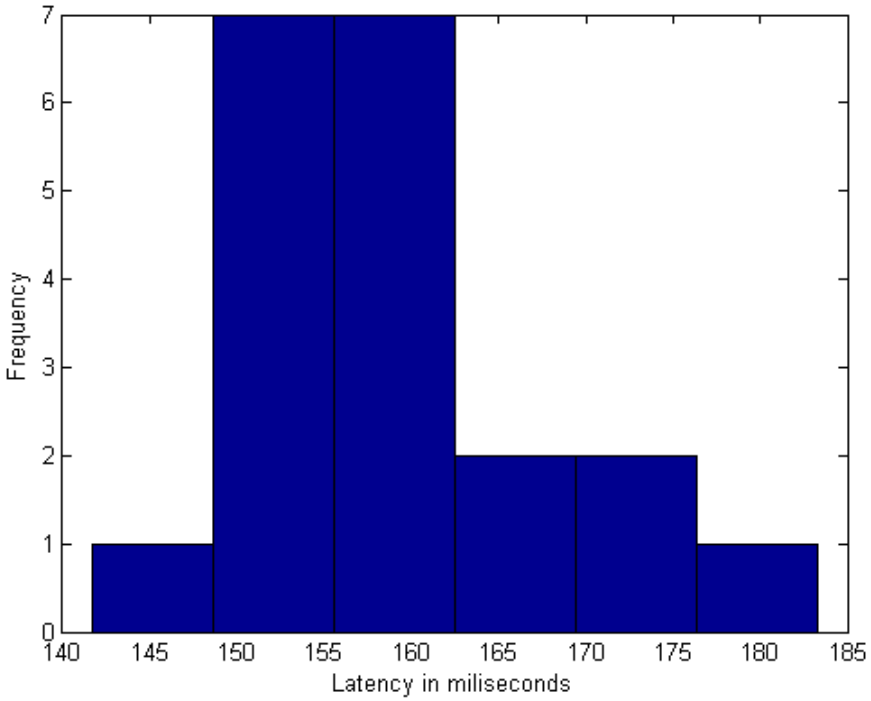
Notice that the location of the end effector of the virtual thumb in the images of figure 9 is slightly off compared to the pose of the physical thumb. The location of the touchscreen touch event is accurately represented as a set of pixel coordinates which are then translated to the relative coordinate system of the virtual touchscreen, the indicated location of the touch even is correct. However the contact area of the thumb with the touchscreen is not always in the same

location. These results indicate that various poses of the thumb have different contact areas with the surface of the physical thumb. The model of the virtua; thumb assumes that the end effector of the kinematics chains is always on the tip of thumb, and thus the tip of the virtual thumb will always seek to reach the location of the touch event. However in the case of the images of figure 9 the contact area appears to have shifted away from the tip of the thumb. While the contact area of the physical thumb shifts positions, the end effector of the virtual thumb remains in the same location. This results in numerous thumb poses that cannot be accurately mimicked by the inverse kinematics script, because the end effector of the kinematic chain is the wrong position. The left image of figure 8 reveals another limitation of the estimated thumb poses. While in this case the end effector of the kinematic chain is in correct position resulting in a properly positioned thumb location, the solution found by FABRIK algorithm is not a natural pose of the thumb and does not resemble the outline of the physical thumb. Unfortunately the issue of a mismatch of the contact area of the physical thumb with the location of the end effector on the virtual thumb cannot be solved using the sensor data. The data does not provide enough information to hint at shift in the contact area of the thumb. However, some areas of the touchscreen may be more likely to have a shift in the contact area than others. If such a pattern could be identified then this information could potentially be used to adjust the end effector to more accurately resemble to location of the contact area of the physical thumb via heuristic rules. The second problem of unnatural inverse kinematic solutions could potentially be solved by a more accurate model of the thumb. A model that accurately simulates the flexing and relaxating of muscles and the pull of tendons may provide the inverse kinematics algorithm with enough information to determine the most natural pose in a given situation.

7.2 Latency

The latency of the smartphone controller is measured as motion to the last photon. This is defined as the time between the initiation of the motion with the physical smartphone and the first frame on the monitor that displays this motion. In practical terms this entails a camera to record both the smartphone and the display. The smartphone's orientation is adjusted multiple times while being recorded. This video is then used to find the start frame of the physical smartphone's motion and subsequently to count the number of frames before that motion is first visible in the virtual environment on the display in the background of the video. Ideally this would involve a video camera capable of recording a high frame rate video, and a monitor with an equally high refresh rate to display a new frame of the virtual environment on the background display in every frame of the recorded video. These measurements were performed using a 120 frames per second recording. Unfortunately the monitor used during these measurements had a maximum refresh rate of 60Hz, meaning that only 60 new frames can be displayed on the monitor per second. The video recorded at 120 frames per second will therefore take two frames before an updated frame is visible on the background monitor. This unfortunate circumstance during the experiments may have resulted in a occasional miscounts of one frame during the measurements. The process of motion to photon measurement is highly time consuming, therefore each evaluation has been limited to 20 measurements. Such a small data set does impact the reliability of the data. In order to determine whether the resulting average is statistically significant a t-test is used to calculate the p value.

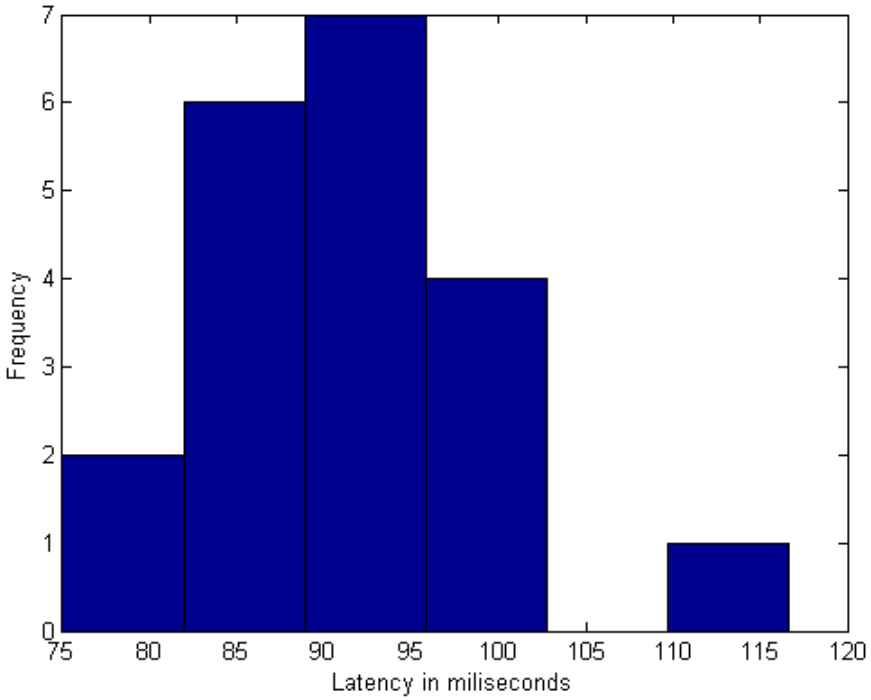
The first measurement relates to the latency of changes in orientation. The smartphone is placed on a flat surface in front of a monitor on which the Unity scene is displayed. A 120 frames per second recording is made of this setup. During the recording the smartphone was tilted upwards to roughly 45 degrees around the x-axis (see figure 1) multiple times. The video was then used to count the number of frames between the first frame to display motion of the physical smartphone, and the first frame to display that same motion on the background display. Since the recording is 120 frames per second each frame spans 8.33ms. The amount of counted frames is then multiplied by 8.33 to determine the latency of the system in milliseconds. The histogram below shows the results of these measurements.



Mean: 158.33 ($p < 0.05$)

Figure 10: Latency of the orientation sensors in ms

Note that while the results are assumed to be statistically significant based on the t-test ($p < 0.05$), it cannot be stated with certainty since these results are based on a small dataset (n=20). The average latency of the changes in orientation is 158.33ms, which is fairly high. In order to determine the cause of this, the touchscreen latency was also measured. The same measurement methodology was used, however instead of tilting the smartphone, the touchscreen was tapped with the index finger multiple times. The results of these measurements are shown in the histogram below.

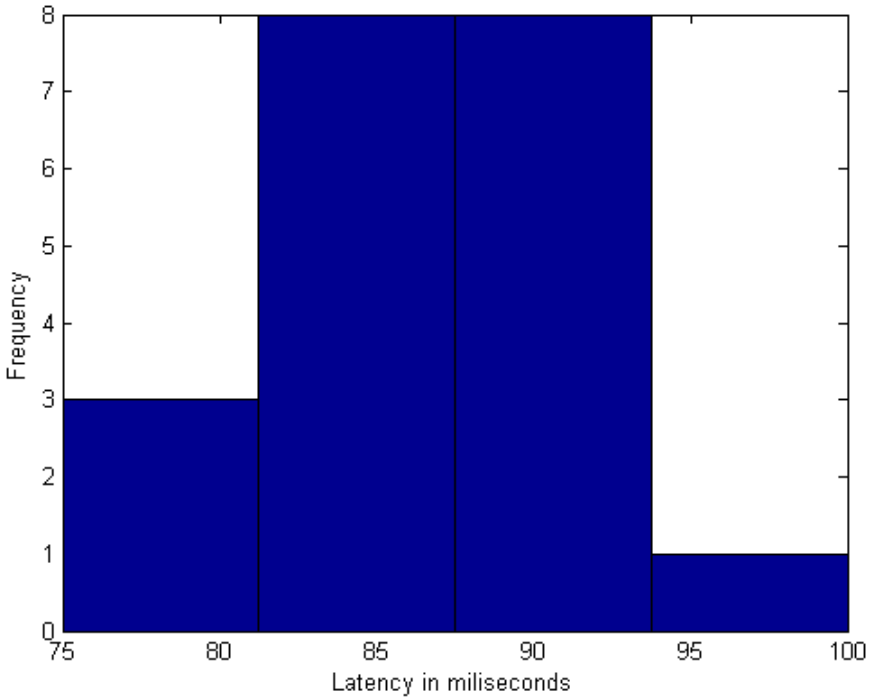


Mean: 90.42 ($p < 0.05$)

Figure 11: Latency of the touchscreen in ms

The average latency of the touchscreen is 90.42ms, which is a significant improvement over the orientation latency. As discussed in the implementation section, the Android applications sends single packets to the Unity engine containing both the orientation data of the smartphone as well as the touchscreen coordinates. That would acquit the complexity sensor fusion implementation and the wireless connection of being to cause of the high latency seen during orientation changes. The likely culprits that remain are the complementary filter, which may reduce the speed at which the new orientation values can be updated, and any inherent latency of the sensor hardware itself. Further experimentation will be required to determine the cause and a possible solution. An experiment of particular interest would be to replace the complementary filter with a Kalman filter to not only determine if the latency is improved, but also the overall accuracy of the sensor fusion algorithm.

In order to get a frame of reference on the amount of latency of these two inputs, the latency of a keyboard was also measured in the same Unity instance as the previous two latency measurements. The measurement methodology remains unchanged. The histogram below shows the results.



Mean = 86.25 ($p < 0.05$)

Figure 12: Latency of a keyboard in ms

With an average of 86.25ms latency, it is similar the touchscreen latency. This is an encouraging result that indicates that the wireless connection and the algorithms used by the prototype do not add a significant amount of latency.

8 Conclusion

The goal of this project is to explore the viability of a smartphone functioning as a controller for virtual reality applications. The success of a virtual reality controller is dependent on the performance of its motion tackers to properly simulate the user's movements and on the practicality of its interaction methods. This project focused on the motion tracking capabilities of the smartphone sensors. The sensor data of the accelerometer, magnetometer and gyroscope have successfully been combined using a sensor fusion algorithm. This algorithm accurately determines the orientation of the physical smartphone. A virtual environment has been constructed in the Unity game engine consisting of a rigged avatar holding a virtual smartphone in its left hand. The orientation of the smartphone has been used to determine the pose, limited to rotation on its three axis. The accuracy and latency of this implementation have been evaluated. The evaluation method for the pose estimation did not yield robust, quantifiable results. However it did reveal a number of limitations of the models. While the rotation of the wrist and lower arm

appears to function as desired, the distance between the smartphone and wrist differs from that of the virtual model. This results in a slight translation of the smartphone as it rotations around the wrist. This causes a minor mismatch between the location of the virtual smartphone and the physical smartphone. The evaluation of the thumb poses revealed that the surface contact area of the physical thumb with the touchscreen does not remain static. The virtual model fo the thumb cannot take the shifts of contact area into account resulting in an occasional mismatch of desired thumb pose. Additionally not every solution found by the FABRIK algorithm is a natural pose. The latency evaluation revealed that the average latency of the orientation sensors is 158.33ms. This is considerably higher than the latency of the touchscreen (90.42ms) and a keyboard (86.25ms). Unfortunately the cause of the higher orientation latency is as of yet unknown. Further research will be required to determine the cause of the higher orientation latency and to determine whether a latency of 158.33ms negatively affects the interaction with the virtual reality application. In summary: This project has resulted in a prototype capable of tracking the rotations of a users lower arm and hand around its three axis.

9 Future work

9.1 Improvements on the current prototype

The resulting implementation of this project is still unpolished and will require additional research and time to bring it to fruition. It is currently unclear whether the latency of the orientation sensors can be improved, or whether the current latency of 158.33ms is harmful to the virtual reality experience. Furthermore it as of yet unclear whether a more accurate model of the thumb can solve the occasional unnatural pose solutions of the FABRIK algorithm. Further research into user experience and whether the unpolished aspects of the prototype are damaging to the virtual reality experience would provide greater clarity to the potential of this prototype as a virtual reality controller.

9.2 Expanding the prototype

The current motion tracking component of the prototype is limited to rotation only. Expanding the tracking capability to a full 6 degrees of freedom can greatly benefit immersion through a greater mapping of the user's movements and the ease of use during interaction with the virtual environment. Unfortunately the sensors of a single smartphone do not provide adequate data to determine the translation of the smartphone. Mounting RGB cameras and infrared depth sensors to the smartphone could potentionally provide enough information to determine the translation. Alternatively an additional smartphone or set of smartphone sensors could be attached to the lower or upper arm to determine its orientation, thus being able to determine the translation of the smartphone. This could even be expanded into a full suit of sensor clusters on each major segment of the user's body to provide a full-body motion tracking system.

References

[1] Andreas Aristidou and Joan Lasenby. Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver. Technical Report CUEDF-INFENG, TR-632, Department of Information Engineering, University of Cambridge, September 2009.

[2] Joachim Lawitzki. Application of dynamic binaural signals in acoustic games. Hochschule der Medien Stuttgart, 2012. Master's thesis.

[3] S. Luczak, W. Oleksiuk, and M. Bodnicki. Sensing tilt with mems accelerometers. *Sensors Journal, IEEE*, 6(6):1669–1675, Dec 2006.

[4] Merck Manuals. [www.merckmanuals.com]. [cited 2014 jun 27]; available from *www.merckmanuals.com/professional/special_subjects/rehabilitation/physical_therapy_pt.html*.

[5] Ryan P. McMahan, Doug A. Bowman, David J. Zielinski, and Rachael B. Brady. Evaluating display fidelity and interaction fidelity in a virtual reality game. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):626–633, April 2012.

[6] Dahai Ren, Lingqi Wu, Meizhi Yan, Mingyang Cui, Zheng You, and Muzhi Hu. Design and analyses of a mems based resonant magnetometer. *Sensors*, 9(9):6951–6966, 2009.

[7] Jared D. Siegling and Jon K. Moon. Performance of smartphone on-board accelerometers for recording activity. Orlando, FL, 2011. Obesity Society Annual Conference.

[8] Mel Slater, Martin Usoh, and Anthony Steed. Taking steps: The influence of a walking technique on presence in virtual reality. *ACM Trans. Comput.-Hum. Interact.*, 2(3):201–219, September 1995.

[9] Open Sound Control. [http://www.illposed.com/]. [cited 2014 jun 27]; available from *http://www.illposed.com/software/javaosc.html*.