Distinguishing Between Head and Phone Gestures On a Smartphone With Front-Facing Camera and IMU

James Whiffing
jw204@bath.ac.uk
University of Bath - Department of Computer Science
Bath, England

Figure 1: Demonstration of how changes in head or phone orientation can result in the same head orientation being observed by a smartphone's front-facing camera

ABSTRACT

Head gesture recognition is a common approach taken in active research for increasing the available modes of input for mobile devices. Head tracking techniques that utilise the front-facing camera can incorrectly detect head movement caused by the phone being moved. In this work we review the mobile device head gesture recognition space to understand the techniques currently in-use. We also perform a study with to collect head and phone movement data when performing semaphoric head and phone gestures. We also propose a model/approach with which semaphoric head and phone gestures can be distinguished.

ACM Reference Format:
James Whiffing. 2022. Distinguishing Between Head and Phone Gestures On a Smartphone With Front-Facing Camera and IMU. In *Proceedings of (Conference acronym 'XX)*. ACM, New York, NY, USA, 19 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, N/A, N/A

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Touchscreens have been the primary interface with which users interact with modern smartphones, either through directly touching UI elements (such as buttons or an on-screen keyboard) or through the use of touch-gestures. Over recent years there has been a trend of smartphone touchscreens increasing in size [32], which does not afford optimal reachability of the full screen for the thumb for interactions [19]. This reduces usability for one-handed interaction, which is a common mode of use [11].

An emerging solution to this is to include head gestures as an additional mode of input, by tracking the head via the smartphone's front-facing camera [5, 7, 9, 10, 26, 31]. A problem with tracking something via a camera that can have a moving Point-of-View (PoV) is that changing the camera's PoV can *look* like movement of the object being tracked. For example, the user moving their phone to the left, will be treated the same as the user moving their head to the right, since the positioning and movement of the head from the front-facing camera's point of view will look the same, see Figure 1. Some papers knew of this issue and accept it as a feature [10], others note it as a known fault to be aware of [7, 29], while others don't indicate whether this has been accounted for [5, 9, 26, 31].

In this paper we look to propose a system that can distinguish between the head or smartphone being moved. In being able to

differentiate the two, it should also be able to recognise gestures based on the smartphone movement.

In order to develop a proof-of-concept, a data driven approach was taken. As such a study was performed to collect data: image sequences from the front-facing camera of the smartphone, IMU data from the smartphone, and 3D positioning of the user's head and the smart device via a motion capture system (to provide a ground truth).

With the motion capture data being synced to the IMU data and images, a system could be trained to recognise several gestures and learn to distinguish between whether an observed gesture was due to the smartphone or the user's head moving.

Pending confirmation of system's performance (what was actually delivered/produced) and how it could be extended upon.

2 LITERATURE REVIEW

In this section we will review existing literature to build an understanding of: the gestures we can expect to process, how they may be used and what they mean; Methods with which to obtain data pertaining to the pose of a head; and finally the means with which we can track movement and determine the gesture being performed.

2.1 Gesture Classifications and Usage

Given our goal is to develop a means to distinguish head and phone gestures on smartphone devices, we first need to understand the gesture's we want to recognise and distinguish. Here we will look at existing literature that outline the head and phone gestures you would expect to use while interfacing with a smartphone.

After a review of gestures utilised within Human Computer Interaction literature Karam et al. define five distinct classes with which we can differentiate between types of gestures utilised by the systems proposed in the literature [15]:

Deictic Gestures that involve pointing, and mapping this to either a specific object or location within the interface.

Manipulative A gesture which indicates intent to manipulate some object.

Semaphoric Gestures which map to a specific action or intent.

Gesticulative Typically accompany speech, but not required to transfer meaning.

Language A substitute to written or verbal language.

In our review of head/phone gesture systems we found that none utilised the Language or Gesticulative gesture styles, which is to be expected as we were focusing on gestures for control and interaction rather than for communication. Of the 3 remaining gesture styles, we noted that systems rarely utilised a single gesture style. Either due to the gestures themselves being viewable as multiple gesture styles, being both semaphoric and manipulative, or by actively including different styles of gestures, such as pointing to a region, then using a sempahoric gesture to trigger an action.

An example of this can be seen in the work of Yan et al. [34] who propose nine head gestures (Figure 2), the majority of which are purely Semaphoric, however several (such as scrolling, dragging, and zooming) could also be seen as Manipulative through the mapped action physically moving the content on screen. Yan

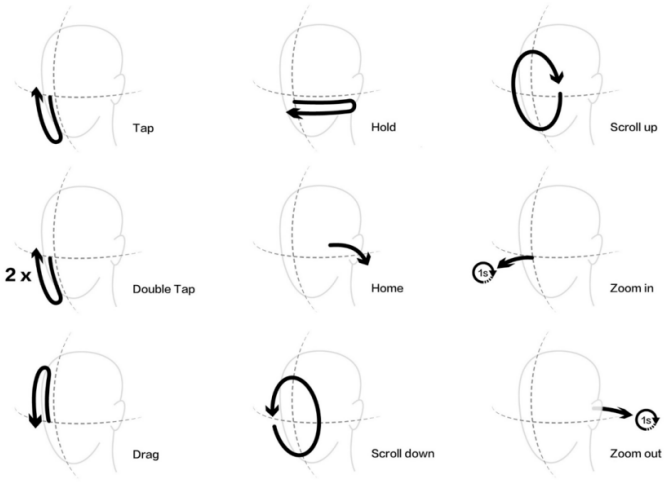

Figure 2: Proposed head gestures and their corresponding actions [34].

et al. derived these gestures through a study wherein participants where asked to proposed a set of head gestures, without being given an associated action. These gestures were then collated manually into a set of 80 gestures, which were then effectively voted upon by the participants for their respective actions. The gestures with the most votes for a given action were selected, with some minor adjustments to ensure there were no clashes between actions.

Another system that utilised multiple gestures was EyeMu [17], which outlines several gestures that are performed by physically moving the smartphone, to improve user interaction when the user is forced to interact with phone single handed. As with Yan et al.'s gestures, most are Semaphoric, but can be viewed as manipulative. Some map to actual actions, e.g. flicking between items, others are less derivative and have less of a connection to the desired effect, e.g. moving phone closer/further from face to select an item / open a page.

Two systems that were purely Deictic are Nouse [9] and a system developed by Varona et al. [29], both of which map the position of the user's nose, within images from the front-facing camera, to a location on the screen. Neither system recognises sequences of motions of the head as gestures, other than recognising blinking and winking, which were recognised as an action to select what was under the cursor. You could also argue that these systems are also manipulative given they show a cursor and as such the head gestures are in an attempt to move the cursor to the relevant location.

One system which could be said to combine all three gesture styles would be the virtual 3D display proposed by López et al. [20]. Their system treats the smartphone screen as a window into a 3D box, visualised in Figure 3, where the region of the interior rendered is controlled via the user adjusting the position of their head with respect to the screen. The tracking of relative positioning of the head to adjust the perspective of what is rendered can be seen as a form of both Deictic and Manipulative gesturing, as the user is looking

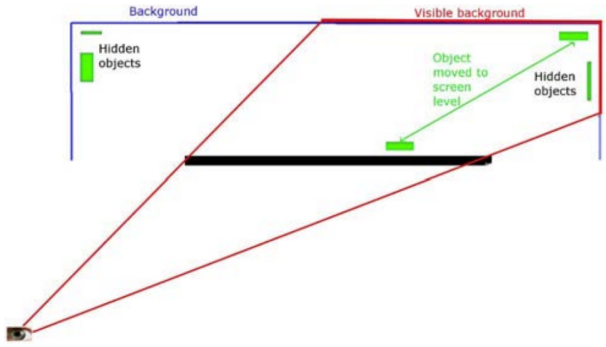

Figure 3: How user's perspective alters content that is rendered [20].

at different regions within the interior of the virtual box by simply changing where they are looking, but with the intent to adjust the visible interior of the virtual box. The system also provides Semaphoric gestures when interacting with specific programs; in one example they use a browser, with which they can look to the edge of the page to reveal the bookmarks bar.

2.2 Head Localisation

Before being able to distinguish between head and phone gestures, we first need to extract them. To start with we will be reviewing the methods in surrounding literature to extract relevant data required to track head gestures through the use of a smartphone.

The naive approach often taken for finding a face, or more generally a person, within an image is to perform colour segmentation [2, 4, 12], which involves taking an image and filtering the pixels based on a range of colour values that have been chosen as representing skin-tones.

While simple, and given favourable conditions, effective, this approach has several drawbacks:

(1) Detection of objects which have colours that have similar colour and chrominance levels, as noted bybin Abdul Rahman et al. [2].
(2) Determining the values with which to segment the image, i.e. what colours will we accept as skin-tone? During their system evaluation Chan and Abu-Bakar [4] used participants with similar skin-tones to improve the system's robustness.
(3) Dependence on environment lighting.
All three of the papers above [2, 4, 12] do make use of the HSV/yCbCr colour spaces, which make them more robust to changes in lighting intensity, however these systems can still be susceptible to changes in lighting temperature, colour, or shadows.

A less naive approach is the Viola-Jones algorithm proposed in 2004 [30], used in digital cameras, smartphone camera apps, and several head gesture systems [7, 16, 24]. Rather than looking for skin-tone to find faces, it uses the difference of intensity between regions of pixels, and checks if they match a set of templates, Haar-features. These features compare the relative intensity of 2, 3, or 4 neighbouring regions, e.g. is the centre of a region brighter than the regions to the left and right. The algorithm proposed by Viola and Jones uses a degrading cascading classifier[1] to apply these Haar-features on an integral image[2] and will return a bounding box for each face found.

Kim et al. build upon the Viola-Jones algorithm, still utilising Haar features but building their classifier to return the locations of four facial features: left eye, right eye, nose-tip, and mouth, in place of bounding boxes [16].

A more typical approach however is to use the Viola-Jones algorithm to retrieve the bounding box of faces within the image, and then perform further processing to extract facial features [7, 16, 24]. One downside with the algorithm that Viola and Jones note is that it cannot reliably detect faces that are rotated ±15°, while the person is still facing the camera, or ±45°, where the person is facing off to the side of the camera.

Another solution present in the literature, that is invariant to face pose, is the use of depth cameras. In particular we found the use of Apple's ARKit framework used alongside the front-facing depth camera (on supported iOS devices) [5, 13, 31]. ARKit provides a reliable 3D representation/positioning of the face and its features which could be used for tracking the head's movement. The only downside of this approach seems to be the requirement of the hardware and OS in order to use the ARKit framework.

The final solution we reviewed was the use of Convolutional Neural Networks (CNNs). YuNet [35] for example outputs the bounding box along with the positions of the eyes, nose tip, and the corners of the mouth. YuNet is in fact the replacement suggested by OpenCV for detecting faces, which previously used and recommended the Viola-Jones algorithm.

Yan et al. propose a CNN which instead predicts the roll, pitch, and yaw of a face provided within an input image [33]. With their CNN they were able to observe reduced error in their predictions compared to existing tools.

The benefit of a CNN is that they should be invariant of head rotation, given the training data includes samples of heads at different rotations. A potential downside is the need for sufficient processing power. However there do now exist mobile variants of popular Neural Network frameworks, such as TensorFlow, with TFLite, and PyTorch, with PyTorch Mobile, which make running CNNs feasible on mobile devices.

2.3 Phone Localisation

During our review of related works we came across several means of localising a smartphone / tracking a smartphone's movement, each with varying degrees of feasibility.

The least reasonable methods do not bare reviewing due to unrealistic expectations of the population of smartphone users, such as the need for a Motion Capture (MoCap) system [3] or the use of a Head Mounted Display with a mounted tracking marker [22]. These may be suitable in specific environments, but are not reasonable in meeting our goal.

[1]Where a traditional cascading classifier will have possibly have 2 branches at each node, a degrading one will always exit, returning nothing, on one of the branches of each node.
[2]A representation of the input image that permits an efficient means to calculate the sum of a rectangular region of the image with just the for corner points.

Figure 4: Figure showing points being tracked [1].

A more reasonable set of methods involve localising the smartphone's position relative to its environment. One method is 'camera tracking', wherein the movement of the camera is estimated through analysis of an image stream from the rear-facing camera. This is a technique common-place in VFX to recreate the path taken by a camera in 3D [1], Figure 4 shows extracted 3D movement being used to track 3D cones into a video. However this technique has also been extended for use in Augmented Reality applications [14]. Unfortunately this isn't reasonable to use on current modern mobile phones as they don't all support the ability to capture images from multiple cameras (some via software, others due to hardware limitations). As such we will not be able to utilise the rear camera as the front-camera will be required to track the user's face in our proposed system.

Another solution is the use of either Depth-Cameras or LiDaR and tracking the smartphone's movement through the observed 3D space. Unfortunately these require special hardware that isn't available on most smartphones; most depth cameras that exist on modern smartphones are front-facing and the only current mainstream phones to provide a LiDaR on the rear of the phone are the iPhone 12 and 13 Pro series.

The only method we found to be reasonable and feasible was to record the linear and angular acceleration of a smartphone's Inertial Measurement Unit (IMU) [8, 18, 21, 23]. An IMU provides the acceleration experienced in the 6 Degrees of Freedom (DoF)[3] the smartphone can be manoeuvred through. A common issue however with processing IMU output is noise, as noted by Neelasagar and Suresh. To address potential noise they utilised low and high pass filters on the acceleration data.

2.4 Gesture Recognition

Knowing how we can obtain facial features and the 'pose' of the user's head through a front-facing camera, and the localisation of the smartphone itself, the next step is to be able to recognise gestures performed by the user with either their head or the smartphone.

One solution employed by papers proposing systems that tracked Deictic pointing gestures (and possibly Manipulative pointing gestures), was to simply use the raw data, or a function of the data, to map detected facial features to a location within the UI.

A common approach was to take the position of the nose and map it to a point on the screen. This could either be used to manipulate a cursor [9, 25, 29], allowing the user to move their head to highlight specific places on the screen, or to highlight the region of the phone the user is looking [13, 26, 31].

For semaphoric gestures you need to be able to be able to identify the gesture within a sequence of input. An RNN is a Neural Network that takes a sequence of elements and has an internal state that is updated by some function of the current element being processed and the current state. Sharma et al. proposed the use of an RNN in order to recognise head gestures, wherein the RNN input was a sequence of facial landmarks extracted from a sequence of images [27]. An advantage of using an RNN is that you don't strictly need to know exactly when the in the sequence the gesture was performed, just that it is present within the sequence. A downside however is that internal state isn't maintained between predictions, as such you *must* provide a sequence and the input sequence *must* always be of a fixed length[4]. Input must there for be broken-up to fixed lengths, either requiring padding prior to/after the gesture recorded (if you do not have enough elements for the required sequence). To break-up the input you need to either run the model each time-step, providing a rolling window representing the last x frames of state, or to have another means to segment your recorded input to then pass into the RNN.

Another method for predicting Semaphoric gestures from a sequence, is to use Hidden Markov Models (HMMs) [6, 28]. A HMM describes the possible hidden states a system can be in, the probabilities/rules for transitioning from one state to another, the sates that can be observed, and the probability that a given observation arises from each hidden state. For example, the HMM employed by Elmezain et al. [6] has the gestures as its hidden states, in this case arabic numbers, with the possible observations being a quantised direction[5] that the user's hand travelled, captured via a camera. The probabilities of the sequence of observations would be observed for a given number drawn by the user can then be trained.

3 DATA COLLECTION

This section details the process undertaken to develop the system which can meet the goal (outlined in section 1): distinguishing between head and phone based gestures on a smartphone.

In order to develop the aforementioned system we opted to take a data-driven approach.

For a data-driven approach to work we need to first determine what data we need to collect in order to train our system. We have identified the following types of data:

[3] 3 Linear Axis: X, Y, Z, 3 Angular Axis: Yaw, Pitch, Roll

[4] This is to the best of our knowledge using common neural network frameworks such as TensorFlow and PyTorch

[5] To reduce the possible observation space, the angles from 0°to 360°are bucketed into a range of 0 to 18

Images From The Front Facing Camera
Though a depth camera would likely be more reliable and accurate in extracting the shape/pose of the user's face, it does require hardware that is not yet standard on all smartphones. To enable our system to be run on as many smartphones as possible we will be opting to detect and track the user's head via the front-facing camera, for which we found many techniques from which to extract the user's head movement [7, 16, 20, 24, 29, 30, 33].

Smartphone IMU Data
To understand whether the smartphone's PoV is changing we need to know how it is moving. As noted in our review of surrounding literature, the most feasible means to do this is with the IMU present in modern smartphones [8, 18, 21, 23], which will allow us to track the phone's linear and angular acceleration.

Ground-Truth Data
In order to accurately train a model that can achieve our goal, we will need some Ground-Truth data. This will include the gesture the data is associated with, so that the system can classify gestures. It will also be helpful to capture the actual head and phone poses/motion during given gestures such that we can aim to train a model that can predict the movement of the head and phone.

3.1 Apparatus and Techniques

3.1.1 Pixel 4a.

To collect images and IMU data we opted to use a Pixel 4a smartphone. The operating system is Android 12, and the phone's dimensions are 144mm x 69mm x 8, with a screen size of 5.6 inches. This was chosen as it we believe it to be fairly representative of modern smartphones (at least for the android market share), with reasonable dimensions that should not be too difficult to hold for our participants, a front-facing camera capable taking up to 1080p resolution images, an IMU, and a sufficiently powerful processor such that we should not need to worry about an application that can record photos and capture IMU data stuttering and missing data.

3.1.2 Data Collection Application.

Since the Pixel 4a runs on Android, we opted to develop our data collection app using Kotlin and the Android framework.

The application was designed to instruct the participants on how to perform a series of gestures and to provide the ability to record data from the IMU and images from the front-facing camera as participants performed the gestures. To instruct the participants on how to perform the gestures, we included three different mediums for describing the gesture: Text, Images, and Video. Before a participant could record a gesture, they would first be shown a diagram that outlines what they should be looking at on the screen, and how the head or phone should be moved. This would be accompanied by some text that would describe the same motion in more detail, in order to try and resolve any confusion with the participant's understanding of the diagram. Once the participant has observed the diagram and text, they are then shown a video containing the viewpoint of both the phone and the head during the gesture. This is accompanied with a symbol indicating the direction the phone or head should be moved or turned, and a further textual description of the direction the phone or head should be moved.

To begin recording the gesture shown, the participant presses a record button, placed at the bottom of the screen to be easily reached with their thumb. At this stage the record button will be frozen, to prevent the recording be cancelled prematurely from a double tap, and the application will begin recording images and IMU data for a minimum of 2 seconds. After the 2 seconds have elapsed, the record button is unfrozen and the participant can chose to stop the gesture recording if they have finished. If the recording is not stopped, the participant is given an additional 8 seconds to perform the gesture, after which the recording will be automatically stopped, and the next gesture shown to the user. We employ a limit for the time to perform a given gesture as none of the gestures we intend to capture should take more than a few seconds to perform, however we did not want to risk cutting off a gesture mid-performance. This is then repeated for each gesture we wish to capture and then each gesture is shown again 2 more times, such that each gesture should be performed 3 times. We include repeats of each gesture to include additional variance in the recorded data.

The recorded data for each gesture is saved into device storage, under a directory path starting with the participant id, followed by the gesture take number, then the gesture name, and finally the gesture direction. For example `"4/attempt_0/ROTATE_HEAD/RIGHT"`

Requesting data from the IMU was a simple task, you simply need to register a `SensorListener` for the type of data you want. In the interest of collecting as much data as possible, in order to give us more options with regards to which data to use for our model, we setup six listeners: three to retrieve the linear acceleration, and three to retrieve the angular acceleration. The first linear acceleration sensor provides the acceleration along each axis (X, Y, and Z), but this includes gravity and any bias that may exist in the sensor, the second excludes the bias, and the third excludes bias and gravity. The first angular acceleration sensor provides the acceleration about each axis (roll, pitch, and yaw), but may include bias, the second removes this bias, and the third provides a quaternion representing the phone's orientation with respect to magnetic north and gravity.

Unfortunately it was not possible to request all the data with a single listener. To ensure the IMU data recorded was synchronised, we had the listeners store the latest values in a set of arrays, and had a timer that would trigger at $50Hz$ to pull the latest values from the arrays and store them with the current timestamp in a csv for the current take, gesture, and gesture direction.

Capturing the images was a more difficult challenge. The easiest option would be to record a video with a fixed frame-rate, however this would result in the captured frames of images being compressed, and in a format that would not achievable in deployment of the system we aim to propose, i.e. we would not intend to record and decode a video in realtime to capture images for head gesture recognition. Instead we setup a repeating camera request with the front-facing camera, which when activated would try to retrieve an image from the camera image buffer as often as possible. This does come with 2 downsides however:

(1) The capture rate is not fixed. This could be a positive, by providing variance to the data collected, but could reduce the learning rate of the system we aim to propose as it will

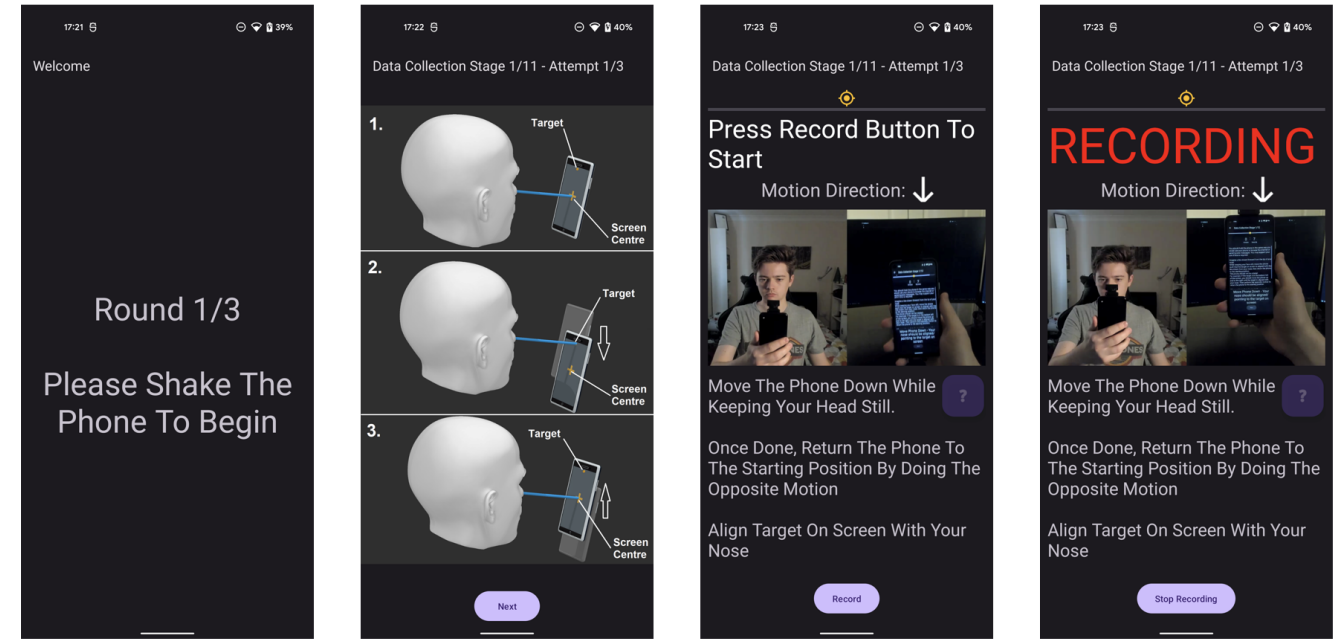

Figure 5: Various stages of the data collection app UI

also need to learn how to adapt to images that can occur inconsistently.

(2) The captures can be halted by Garbage Collection. Though this didn't seem to be an issue with the IMU data collection, we observed some deltas between images being taken jumping sporadically. We believe this to be down to GC events, as they appear within the profiler when the deltas spike.

Images were saved as raw YUV bytes, the image format provided by the camera, into an `images` directory for the current gesture recording, with the timestamp at the time of capture being used as the file name. We did originally try to convert the YUV image into RGB, which we would expect to feed into the system we will propose, however this required significant processing time and memory, resulting in a low sample rate and images still being converted and saved after the gesture recording had finished. If the participant was recording gestures at a reasonable rate this would result in an Out of Memory Error and crash the application. We did try to utilise OpenCV for android, however we were unable to package it into the application such that the required native binaries were available at runtime. By saving the raw YUV bytes we were able to leave the RGB conversion to be performed after the data was collected, and increased our sample rate for images. This does provide a bit of disparity between the collected data and the data we will require for the system we will propose, primarily the delta between samples, however this does allow use to treat the collected data as being taken from a smartphone with better hardware, which could achieve higher sample rates and conversion to RGB. We can also aggregate collected data to lower sample rates to be more realistic with what the Pixel 4a can achieve.

To allow us to synchronise the data collected from the smartphone and the ground-truth data we will also be collecting, we have the application ask the participant to shake the phone prior to beginning a round of gestures. For the shake to be recognised it must exceed a total magnitude of $2g$. If a shake is detected as exceeding $2g$ the current timestamp and the acceleration of the phone, taken from the IMU, are recorded to a csv. We can then hopefully find the shake in the ground-truth data to be able to line-them-up and generate a csv with both the data from the phone and the ground-truth.

3.1.3 *Motion Capture System.*

To track the actual positions of a participant's head and the smartphone, we will be using a the Motion Capture (MoCap) system managed by the CAMERA team on the University of Bath campus.

Camera count and software

To track both the head and the smartphone a set of ten markers, that can be tracked by the MoCap system, will be used (five markers each). One marker will only allow you to track the location of an object. Two markers will allow you top derive the pitch and yaw of the object with just the use of some trigonometry and the vector defined by the two points. A third marker will allow you to also extract the yaw of the object using the same trigonometry, however the third point must be placed such that a right angled triangle is formed by connecting the points. The additional 2 markers used are to allow us to distinguish which set of markers are associated with each object being tracked, being placed in unique positions around the three main points.

To affix the tracking markers to a participant's head we attach them onto a rigid square of foam, in order to prevent the relative positions of the markers to each other being distorted, and then attach this to a skull-cap that the participant can place upon their head. The markers are attached to the skull-cap such that the markers should be at the rear and centre of the participant's head. We chose to track the rear of the head as having tracking markers present on a participant's face could result in the markers being picked-up in the images collected from the front-facing camera, which could impact the ability to detect faces or extract the participant's head pose.

A slight issue with affixing the markers with a skull-cap is that they won't always be in the same place with respect to the participant's face, due to everyone having slightly differently shaped faces. For example a person with a taller head will have the markers higher-up from the base of their neck, someone with a deeper face will have the markers further away from their nose. We could take measurements of each participant's head and determine the exact relative position of the trackers to a set of facial landmarks, however we do not believe the this to be significant issue as the data will still allow us to track the relative movement of the head, which should be enough to train a model to recognise the gestures. This will also introduce a small amount of variance into the training data which should aid reducing the impact of over-fitting.

To affix the markers to the smartphone we 3D printed a mount that the phone could be snapped into and the markers attached. To ensure the markers would not impact the participant's ability to hold and manipulate the phone one-handed, we had the mount extend from the top of the phone. To ensure the markers would be visible to the MoCap system through all the motions the smartphone would be moved through for the collection of gesture data, and to ensure the markers and mount would not be visible to the front facing camera, it was mounted at 22.6°away from the front of the phone. Due to the mount being printed out of plastic, it was not particularly grippy, and as such only friction was preventing it from moving side-to-side on the phone. This should not be a significant issue, as with the participant's grip on the phone, it should not move significantly while performing gestures, and may only slip as a participant adjusts their grip. As with the markers for the head, this should still allow use to get the relative movement of the phone during the recording of gestures.

Image of the 3D printed mount

The output provided by the MoCap system was an fbx file which contains the positions of each of the markers, measured cm, captured at a sample rate of $60Hz$.

3.2 Procedure

Upon attending the study, the participant will be provided an information sheet which will provide details on the study, such as the purpose, what data will be collected and managed, what they will be asked to do. Once read they will read through and sign the consent form if they are happy to continue.

If consent is obtained, the participant will be asked to enter the Motion Capture stage (the space that is viewable to the MoCap system) and asked to wear the skull cap with the motion trackers attached. The MoCap system will then begin recording. and the participant will be provided with the smartphone, which will already have the data collection app running, and asked to read through the disclaimer, which reiterates what data is captured. Once they have read the disclaimer, the participant will shake the smartphone to start with their first take of gestures.

Within the data collection app, the participant will be shown a diagram and text describing the gesture they need to perform. Once understood they are able to move on to show a video of the gesture, from both the phone's Point of View, and the participant's, along with additional brief text and a symbol describing the direction of motion for the gesture. If the participant has any questions regarding the motion they need to perform, they were welcome to ask the researcher present to further explain the motion, or to provide a demonstration. The participant will additionally be instructed to hold the phone as they typically would when using their smartphone, to say browse the internet or read messages.

Once the participant is ready to perform they gesture they will tap the record button and perform the gesture. The recording will then be stopped when they press the button again after completing the gesture, or it will automatically stop after 10 seconds. The next gesture will then be shown to the participant.

This is repeated until the participant has completed each gesture three times. Between each round of gestures the participant will be asked to shake the smartphone before being shown the next round of gestures. Once all the gestures have been completed, the participant will remove the skull cap, hand-in the smartphone, and read-through a debrief sheet, containing an additional request for consent to use the data.

Regarding the gestures participants were asked to perform, we decided upon 11 gestures, each with 2-8 variations (effectively directions the gesture could be performed in), resulting in a total of 44 distinct motions to obtain samples of. A breakdown of the gestures and variations can be found under Appendix A.

Table 1: Breakdown of Participants

Participant ID	Age	Gender
0	23	Female
1	25	Male
2	21	Female
3	24	Male
4	26	Male
5	24	Male
6	27	Male
7	23	Female

3.3 Participants

For our study we recruited 8 participants from the University of Bath campus who were between the ages of 23-27. 37.5% of the participants identified as female, the remaining 62.5% identified as male. A full breakdown can be see in Table 1.

3.4 Results

In total we were able to record 1028 gestures. The sample rate for the images was between 30-33 images per second, and the IMU

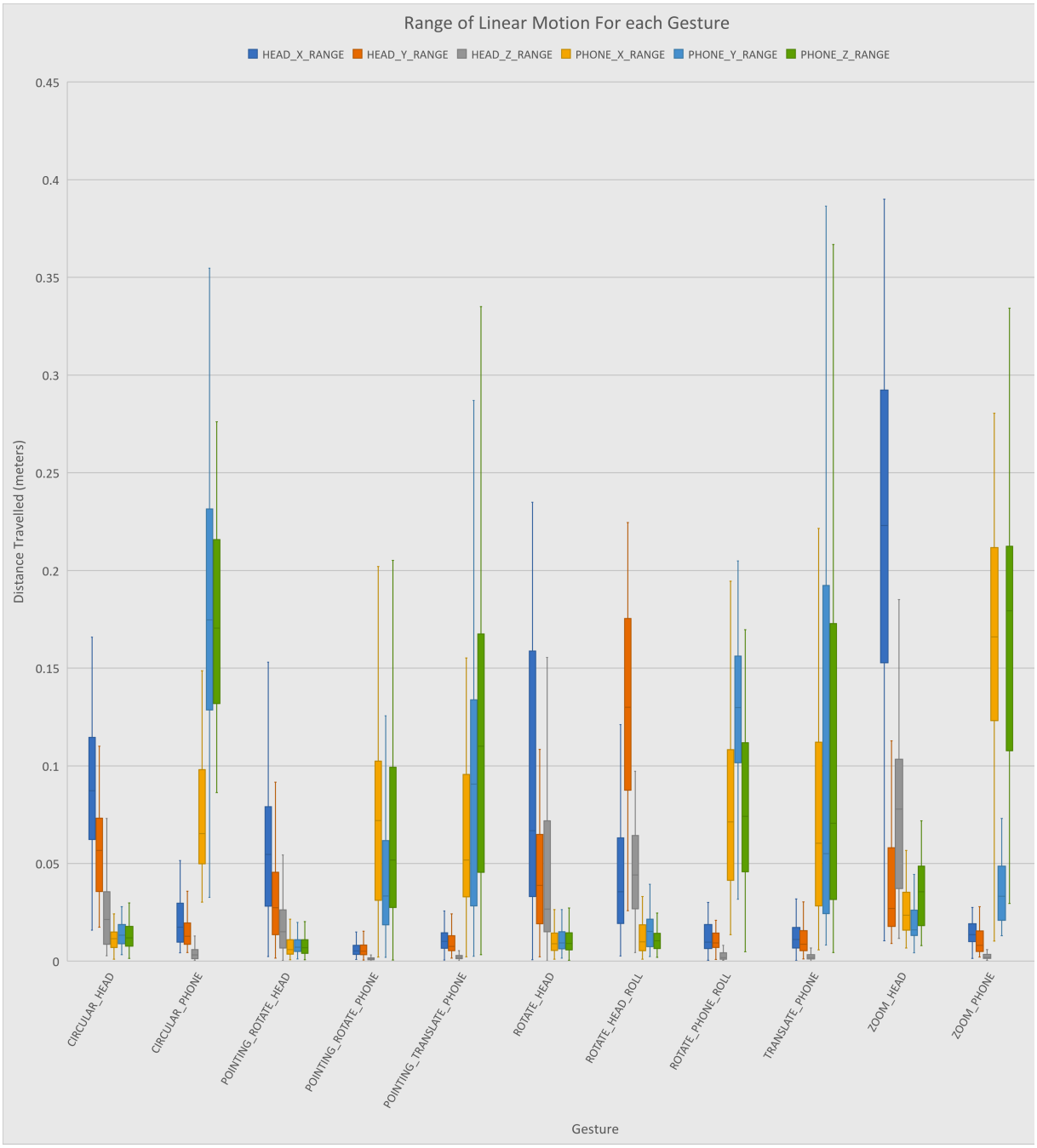

Figure 6: Range of linear motion for each gesture, in meters

sample rate was an expected 50Hz. A breakdown of the range of linear motion can be seen in Figure 6. From this we can see the axis with the greatest movement typically correlate to the object being moved, e.g. the head or the phone, which is to be expected and helps to confirm the MoCap data is accurate. The gesture with the most movement in any given axis was for moving the head towards and away from the phone, with movement typically being between 15-30cm. However the gestures that involved the most movement across multiple axis were the ones that involved the translation of the phone, which typically saw movement across all three axis.

Unfortunately we are unable to accurately breakdown the range of angular motion as we do not record the amount of rotation performed, only the exact rotation observed. As such we have some issues with rotations being calculated as around 360°/ 2π. None

of the gestures performed by the participants actually required a rotation that large, however due to the cyclical nature of angles, you can rotate a small amount and end up with a larger angle if you try to take the difference between frames. This only seems to have impacted the calculation for roll, calculated yaw and pitch appear correct. This is due to the orientation of the participant during recording ensuring that the observed pitch and yaw never exceeding 360°or falling below 0°. A breakdown of the angular motion with can be found under Appendix B, however the calculated roll is unfortunately incorrect. Reviewing the data we do have however does show that the MoCap data is consistent with what we would expect, showing all accurate rotations never exceeding 180°, rotation being very small during linear motion gestures, and that head gestures typically require greater rotation than those of the phone.

By using a face detector [35] with OpenCV we were able to generate a breakdown of the percentage of images within which a face could be detected. The percentage of frames with a face detected for each gesture, and variant of a gesture, is available as an appendix under Appendix B, along with a breakdown of average time elapsed performing a gesture. From the table we can see that some gestures have much lower frequency of frames wherein the head is detected. Most of these are to be expected, such as the gestures that rotate the phone, which can cause the camera's point of view to no longer include the face. A particular example is the Bottom-Centre variant of the Pointing-Rotate-Phone gesture, which involves the phone being tilted such that the top of the phone is tilted away from the participant's face. Given the front-facing camera is mounted on the top, this will result in the camera moving the furthest from the face.

On average all the gestures took between 2.4 - 3.6 seconds, with the circular and zoom motions, for both head and phone, taking the longest to perform.

3.5 Data Synchronisation and Post-Processing

Before being able to use the data for training, we needed to synchronise the data recorded from the smartphone, and the fbx data from the MoCap studio. To do this we derived the acceleration of the phone based on the MoCap data to find where it meets/exceeds the magnitude of the shake recorded by the app. From this we can determine the frame of the fbx data that corresponds to the recorded timestamp. We can determine the frame for any subsequent timestamp based on the known frame-rate of 60fps. To verify the data didn't drift we resync the data based on the other 2 recorded shakes, verifying that they're within 10 frames of the expected frame. In doing this verification, we did not come across any recording wherein subsequent shakes were not found to be at the expected frame. The synchronisation was performed with a Python script that was run within Blender after loading in the fbx. Blender was used as it permitted viewing the fbx data so that we could verify the frames the shake was detected by watching the playback. We could also use Blender to verify the derived location, roll, pitch, and yaw were correct by inserting a plane into the scene and assigning it the derived values, checking that it lined-up through all of the motion trackers. The other reason for using Blender is that it allowed us to programmatically access the fbx file as a data structure and therefore write a script capable of deriving the required head and phone poses from the fbx, and then sync it with the data recorded from the phone. Synchronised data was exported to CSVs for each motion recording, containing a path to the image, the raw IMU data, and the derived MoCap data.

An additional stage of pre-processing that was performed was to generate the RGB images from the YUV data captured by the phone. This was performed with a Python script and the OpenCV library.

4 MODEL DEVELOPMENT

In this section we propose a CNN for encoding the direction of linear and angular movement observed between 2 frames, and a RNN to classify sequences of encoded movement as a gesture. These models are to be used in tandem to classify semaphoric gestures performed, distinguishing between gesture made with the head and those made with a smartphone.

We opted to split the development of our model into two distinct models (the CNN and RNN) for 2 reasons. The first was to reduce the observation space for the RNN, since the acceleration can be any value and the position of the face in the image also any range of values, within the boundaries of the image resolution. To use them as raw inputs to an RNN would require a significant amount of data to ensure we have samples that cover the entire training space. By first encoding the data we can reduce the possible training space. The simplest way to perform this would be to quantise the data. This would involve reducing the resolution of the data, for example mapping all the angles of rotation into a smaller range, as was performed by Elmezain et al. to convert the movement of a hand capture in a sequence of images to the angle of the movement [6], reducing the possible input to their HMM to just 19 observable states. We have decided to instead one-hot encode the motion of the head and phone. This reduces the possible states for two frames down to a 12x3 grid, where the columns are the degree of freedom.

The second reason for the split was to reduce the memory requirements for the system. An RNN requires a sequence of input in order or to predict an output. If we were to utilise a CNN atop the RNN, we would need to hold enough images in memory for the required input length of the RNN. The longer the input sequence is, the more images we would need. 2 Ways to get around this are to keep a short input sequence, reducing the amount of history that the RNN can learn and predict from; or we can reduce the image size, however reducing them too far can result in fewer features being extractable. By splitting into two models, with distinct responsibilities, we only need to hold two images at a time in memory to encode the motion, and then hold onto the encodings for the second model's input sequence.

4.1 Motion Encoding CNN Model

First we propose a model for extracting the motion performed by the head and the phone. Since this model requires extracting features from images, in order to learn and predict the motion of the head within the images, we elected to use a CNN. To save on computation time and effort we employed transfer learning, a technique for extending upon an existing model with pre-initialised weights. For this we chose to use the MobileNetV2 network available within

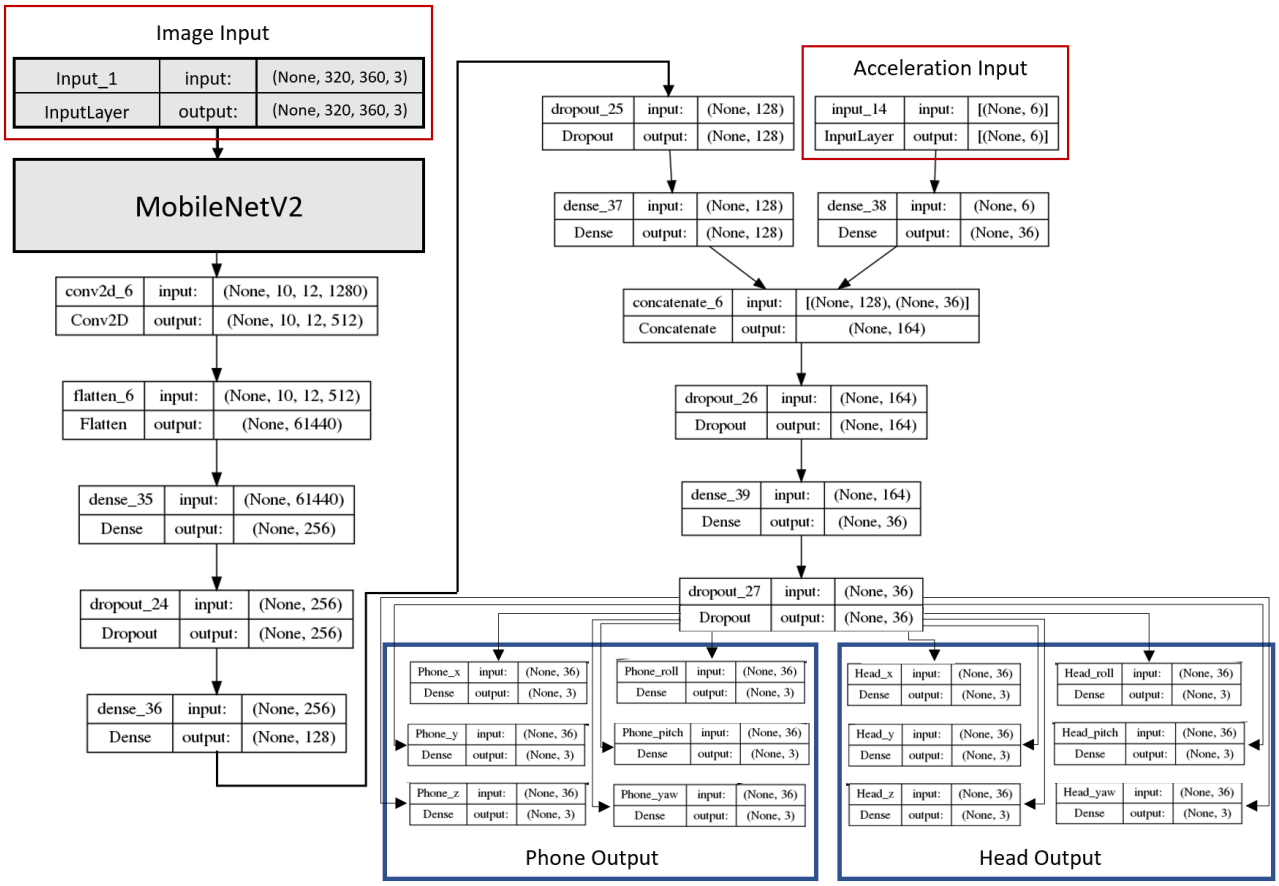

Figure 7: Motion Encoder Model

TensorFlow's Keras prebuilt models. We chose MobileNetV2 as, of the models available within Keras[6], it strides a balance between size (14MB), accuracy (71.3%), and CPU step inference time (25.9ms). MobileNetV2 was trained on the ImageNet dataset[7], a dataset of millions of labelled images. These contain various classes, amongst them are people and faces. By using just the CNN component of MobileNetV2 and freezing the weights, such that they are not updated during training, we should be able to utilise the features they have already learnt in order to process our image inputs, without needing to train a CNN from scratch.

Given we need the model to extract the motion of the head between two images, you may wonder why the input of the model accepts only a single RGB image with the dimensions $320x360x3$, as can be seen in the model graph Figure 7. This is due to a limitation of TensorFlow/Keras, wherein using 2 of the same model for transfer learning fails due to layers from each model having the same name, resulting in the model graph being unable to compile. As such we were unable to have two branches utilising the MobileNetV2 to process the images individually. So to get around this issue we first stitch the images side-by-side, such that the older image is

Figure 8: Concatenated Image Input For the Motion Encoder

on the left, and the later image on the right, as shown in Figure 8. This should not be an issue, as given the nature of CNNs, learned features are not dependent on location within the image, and as

[6]A list of available models included within Keras for transfer learning can be found here: https://keras.io/api/applications/
[7]https://www.image-net.org/

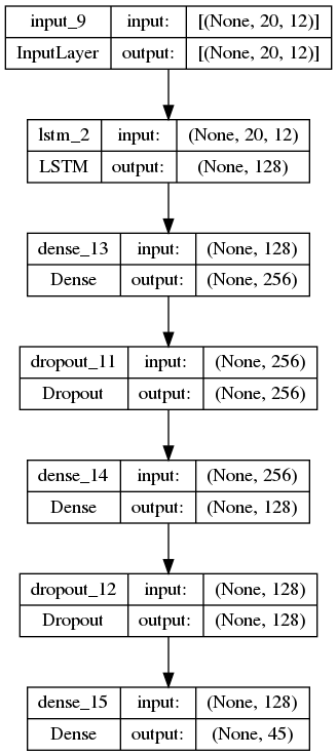

Figure 9: Gesture Classifier Model

such features should still be extracted for both images. This also reduces the size of the model, given it does not need to have two copies of the same CNN.

Following the MobileNetV2 CNN, we add an additional couple of convolution layers, batch normalisation, and max pooling to reduce the parameters in the model, that need to be trained, before flattening the into a 1D tensor that will be concatenated with the acceleration input, starting the first layer of our Fully Connected Network (FCN). The FCN layers are responsible for performing the classification based on the features extracted in the CNN. We opted to provide the acceleration as input into the model in order to allow the model to learn how the head pose changes with respect to acceleration. We also have the model try to encode the motion of the phone, since we are providing the phone acceleration as input. The output for this model is a total of 12 layers, each with 3 units which are encoded via a softmax activation function, which normalises the 3 values for each output, such that the most prediction that the model believes is most likely is the value closer to 1.

The model was implemented using TensorFlow's Keras Api. The full model layers, excluding the MobileNetV2 CNN, can be seen in Figure 7.

4.2 Gesture Classification RNN Model

Once we have our encoded motions, we can now look to classifying the sequence of these observations as gestures. We elected to use an RNN for this task as, unlike a traditional Neural Network, it can hold onto state between elements of a sequence, allowing it to learn to recognise features within sequences. This is ideal as gesture is a sequence of motions, as long as the motions are correctly encoded and the sequence is long enough to envelop the motion duration, or at least the most significant motions within the gesture.

Our proposed gesture classification model, which can bee seen in Figure 9, uses a single Long Short Term Memory (LSTM) layer to process the input sequence. An LSTM has two internal states: the hidden state, and the cell state. Both are passed through the chain of cells within the LSTM, with the hidden state being used to represent the prediction of the cell, and the cell state being a means to retain important features from further back in the chain of LSTM cells. After the LSTM layer the model continues with an FCN with only 3 layers, the layer last being the output of 45 units (1 for each possible gesture from the study, plus no gesture), normalised via a softmax function to one-hot encode the gesture.

As with the first model, this was built using Keras.

4.3 Training

Prior to training the first half of proposed model, we wanted to increase the amount of effective data we have for training we performed some fps scaling of the data. This involved iterating through our collected data and only extracting frames if they *would* have been available at a lower sample rate. For example, if we had 60 images sampled at 30fps and wanted to treat the data like it was at 10fps, we would wind-up with a sequence of 20 images sampled from the original 60.

For the images and the MoCap data we simply took the last available datum for the current new timestamp, but for the accelerations we calculated the average based on the time elapsed, as using just the last value would not be representative of the acceleration of the phone during the period. Additionally if a second image were to occur before the expected, for the given frame-rate, we would create a new file starting at this image, with the intention that we could generate an additional set of data from the same recording, but offset slightly, ensuring the lower sample rates are still processing all of the images captured during the study. In deployment of the model this would require that the accelerations are averaged in between images being captured, but this should provide greater resolution on how the phone is moving.

As a result of this resampling of the collected data, we were able to generate the following:

Raw - 1029 gesture recordings without resampling, recorded at about 32fps.
5 fps - 7997 gesture recordings sampled.
10 fps - 4971 gesture recordings sampled.
15 fps - 4451 gesture recordings sampled.
20 fps - 4184 gesture recordings sampled.

During our generation of the resampled data, we also generated the expected motion encoding for each pair of frames for a given sample rate by quantising the mocap data, reducing the rotation range from 360°to -9 to 9 and converting the positional data from metres to centimetres and rounding to the nearest int. We then generated encodings such that `[1,0,0]` indicates a negative delta, `[0,1,0]` no delta, and `[0,0,1]` indicating a positive delta.

Both models were trained on the same machine, which was equipped with an Nvidia Geforce RTX 3070 GPU with 8GB of dedicated video memory, 32GB of RAM, and an Intel i5-10600K CPU. The Motion Encoder model was trained using the GPU, unfortunately the Gesture Classification model was run with only the CPU due to an issue that developed between training the models which resulted in a failure to load required CUDA libraries.

The Motion Encoder was trained on only a subset of the full training data generated as the initial epoch runtime estimate was over 48hrs. As such we chose to instead use the 10 fps resampled data, as it still contained over 170k motion encodings to learn, within the 4971 resamples.

The Gesture Classifier was trained on the same subset of data, however the only data from the files extracted were the motion encodings as input, and the file names used as the gesture class. The gesture class was also one-hot encoded. We chose to train the second model using the encoded motion data generated during the resampling, rather than using data generated from the output of the first model, to ensure that it was trained on ideal and expected data. This way if the first model was unable to accurately predict motion, we would still be able to verify whether using an RNN to learning from a sequence of encoded motion could still be effective at classifying gestures. An further difference to the input for our second model is that it takes a sequence of input. To generate these we read-in all the gesture recordings, shuffle the order of the recordings. Once we have our shuffled recordings we read from the first frame for the required length of the input sequence and extract this as a single input sequence before starting from the second frame and repeat, resulting in over-lapping sequences. If the length of a recording, from the current frame, is shorter than the required sequence length, we capture until the end of the that recording and begin reading frames from the next recording.

We elected to merge together different recordings into sequences as we did not want to expect the input of the model to be a clean sample of motion from a single gesture, as this would require additional pre-processing to segment input. instead If a sequence contained input from more than one recording, we set the class to be whichever gesture had the most frames within the recording. We did originally try to derive the average encoding, however this would not work with Categorical Cross Entropy loss calculations, and would prevent the model from learning.

During the generation of the input sequences we also extracted sequences motions encodings which had all degrees of motion encoded as being stationary. This was because we unfortunately failed to include a stationary gesture in our study, e.g. a recording of the participants not performing any gesture. A model learns to only predict gestures is thusly going to be incorrect whenever a gesture is not being performed. This also removed the stationary sequences prior to or after participants performed the gesture during recordings.

5 MODEL EVALUATION

In this section we display the results of our model evaluation. These results include the training and validation loss, along with the confusion matrices for the models when run against the testing set that they were not trained on.

5.1 Motion Encoder

During training and validation of the motion encoder, we observed only slight improvement in the encoders accuracy for all the linear axes, and rotational axes showing greater than 70% accuracy from the first epoch, with very little change as training progresses, see Figure 10.

A confusion matrix created by comparing model predictions against our testing set, shown in Figure 11, shows that for the majority of outputs the encoding predicted is for no movement. Of the 12 outputs only Phone X, Y, & Z, and Head X made predictions for non-stationary movement, and of those only Phone Y showed predictions of negative movement.

5.2 Gesture Classifier

During training and validation of the gesture classifier, we observe accuracy an loss plateauing from the tenth epoch, as seen in Figure 12.

Reviewing the confusion matrix for the gesture classifier (Figure 13) shows a high number of No Gesture samples and predictions, with a clearly defined set of true positives down the diagonal, with only minor noise throughout the rest of the matrix. The low values for the noise could however be slightly distorted by the high number of No Gestures sampled.

6 DISCUSSION

Reviewing the results from the evaluation of our gesture classifier we observe under-fitting. We can see from the loss and accuracy graphs (Figure 10) that the model did not appear to learn, besides a small increase in accuracy for classifying the motion observed in the linear axes for the head and phone. It also started with a very high accuracy for predicting the rotational motion, and the height of the head in the Z axis. This was of course great to see, until we noticed the lack of learning. Viewing the confusion matrix for the model (Figure 11), we can see the why: the model did not need to learn as guessing stationary for the motion in a given axis was correct most of the time. As such the calculated loss was heavily skewed by the rotational predictions. In an attempt to combat this retrained the model with loss weights to prioritise the loss of the linear axis 50-fold. unfortunately this did not lead to negligable improvement.

The gesture classifier on the other hand actually performed relatively well, though it also seemed to have some under-fitting. The confusion matrix (Figure 11) best shows what we believe to be the cause of the under-fitting, the presence of many stationary

Comparing the results for the two models, you can clearly observe the gesture classifier having greater performance than the motion encoder. This is most clear within the confusion matrices for the two models (Figure 11, Figure 13), wherein you can observe a much higher true negative and positive sample rate in the gesture classifier. This suggests that although the motion encoder was ineffective, the gesture classifier could be promising for distinguishing between head and phone based semaphoric gestures if given correct input.

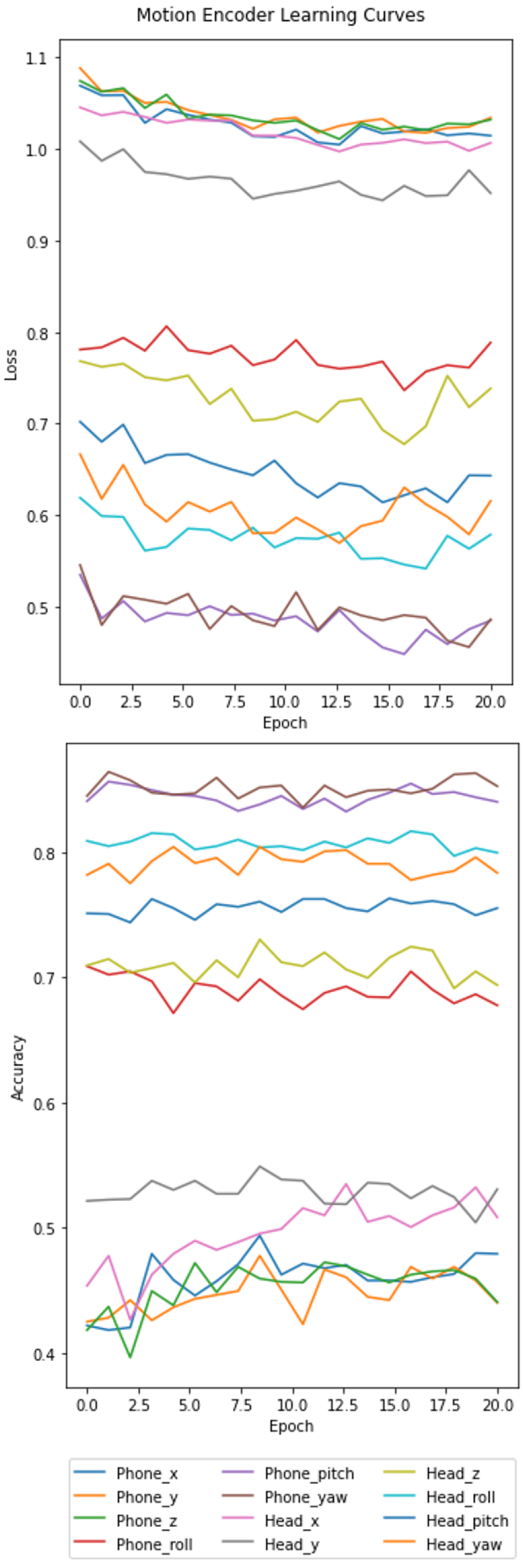

Figure 10: Motion Encoder Validation Loss and Accuracy

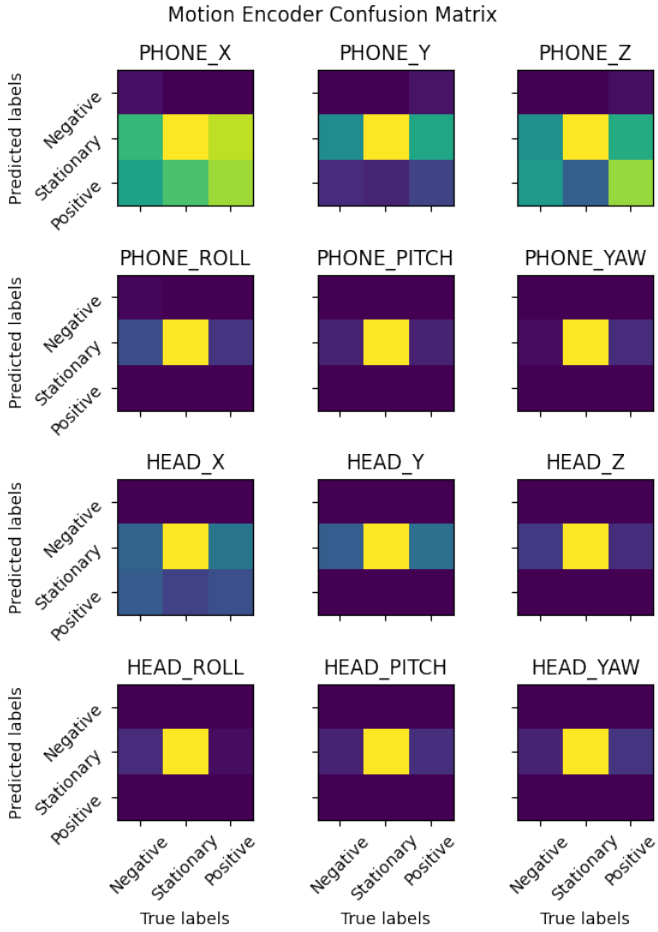

Figure 11: Motion Encoder Confusion Matrix From Testing Set Evaluation

6.1 Limitations and Further Work

Depending on the model limited by only recognising gestures, not pointing / cursor manipulation

7 CONCLUSION

REFERENCES

[1] Alastair Barber, Darren Cosker, Oliver James, Ted Waine, and Radhika Patel. 2016. Camera tracking in visual effects an industry perspective of structure from motion. In *Proceedings of the 2016 Symposium on Digital Production*. 45–54.

[2] Nusirwan Anwar bin Abdul Rahman, Kit Chong Wei, and John See. 2007. Rgb-h-cbcr skin colour model for human face detection. *Faculty of Information Technology, Multimedia University* 4 (2007).

[3] Wolfgang Büschel, Patrick Reipschläger, Ricardo Langner, and Raimund Dachselt. 2017. Investigating the use of spatial interaction for 3D data visualization on mobile devices. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*. 62–71.

[4] YH Chan and SAR Abu-Bakar. 2004. Face detection system based on feature-based chrominance colour information. In *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.* IEEE, 153–158.

[5] Chatsopon Deepateep and Pongsagon Vichitvejpaisal. 2020. Facial Movement Interface for Mobile Devices Using Depth-sensing Camera. In *2020 12th International Conference on Knowledge and Smart Technology (KST)*. IEEE, 115–120.

[6] Mahmoud Elmezain, Ayoub Al-Hamadi, Jorg Appenrodt, and Bernd Michaelis. 2008. A hidden markov model-based continuous gesture recognition system for

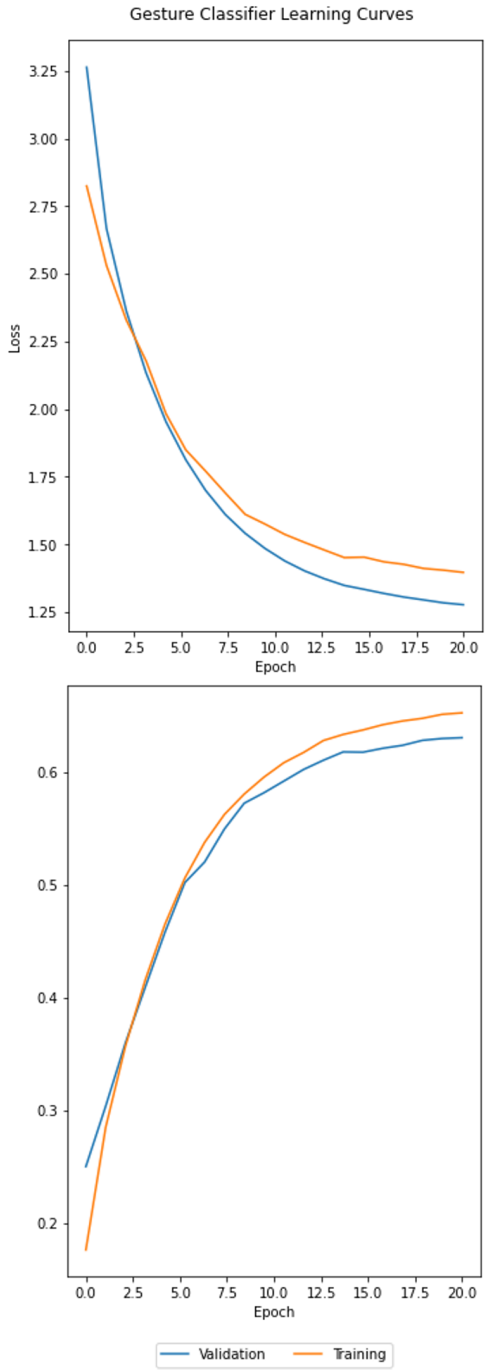

Figure 12: Gesture Classifier Training and Validation Loss and Accuracy

hand motion trajectory. In *2008 19th international conference on pattern recognition*. IEEE, 1–4.

[7] Jérémie Francone and Laurence Nigay. 2011. Using the user's point of view for interaction on mobile devices. In *Proceedings of the 23rd Conference on l'Interaction Homme-Machine*. 1–8.

[8] Enrique Garcia-Ceja, Ramon Brena, and Carlos E Galván-Tejada. 2014. Contextualized hand gesture recognition with smartphones. In *Mexican Conference on Pattern Recognition*. Springer, 122–131.

[9] Dmitry O Gorodnichy and Gerhard Roth. 2004. Nouse 'use your nose as a mouse' perceptual vision technology for hands-free games and interfaces. *Image and Vision Computing* 22, 12 (2004), 931–942.

[10] Thomas Riisgaard Hansen, Eva Eriksson, and Andreas Lykke-Olesen. 2006. Use your head: exploring face tracking for mobile interaction. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*. 845–850.

[11] Steven Hoober. 2013. How do users really hold mobile devices. *Uxmatters (http://www. uxmatter. com). Published: Feburary* 18 (2013), 2327–4662.

[12] Kohsia S Huang and Mohan M Trivedi. 2004. Robust real-time detection, tracking, and pose estimation of faces in video streams. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 3. IEEE, 965–968.

[13] Sebastian Hueber, Christian Cherek, Philipp Wacker, Jan Borchers, and Simon Voelker. 2020. Headbang: Using head gestures to trigger discrete actions on mobile devices. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*. 1–10.

[14] Bolan Jiang, Suya You, and Ulrich Neumann. 2000. Camera tracking for augmented reality media. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No. 00TH8532)*, Vol. 3. IEEE, 1637–1640.

[15] Maria Karam et al. 2005. A taxonomy of gestures in human computer interactions. (2005).

[16] Jin Kim, Gyun Hyuk Lee, Jason J Jung, and Kwang Nam Choi. 2017. Real-time head pose estimation framework for mobile devices. *Mobile Networks and Applications* 22, 4 (2017), 634–641.

[17] Andy Kong, Karan Ahuja, Mayank Goel, and Chris Harrison. 2021. EyeMU Interactions: Gaze+ IMU Gestures on Mobile Devices. In *Proceedings of the 2021 International Conference on Multimodal Interaction*. 577–585.

[18] Sven Kratz, Michael Rohs, and Georg Essl. 2013. Combining acceleration and gyroscope data for motion gesture recognition using classifiers with dimensionality constraints. In *Proceedings of the 2013 international conference on Intelligent user interfaces*. 173–178.

[19] Huy Viet Le, Sven Mayer, Patrick Bader, and Niels Henze. 2018. Fingers' Range and Comfortable Area for One-Handed Smartphone Interaction Beyond the Touchscreen. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[20] Miguel Bordallo López, Jari Hannuksela, Olli Silvén, and Lixin Fan. 2012. Head-tracking virtual 3-D display for mobile devices. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 27–34.

[21] V-M Mantyla, J Mantyjarvi, T Seppanen, and Esa Tuulari. 2000. Hand gesture recognition of a mobile device user. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No. 00TH8532)*, Vol. 1. IEEE, 281–284.

[22] Peter Mohr, Markus Tatzgern, Tobias Langlotz, Andreas Lang, Dieter Schmalstieg, and Denis Kalkofen. 2019. Trackcap: Enabling smartphones for 3d interaction on mobile head-mounted displays. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–11.

[23] K Neelasagar and K Suresh. 2015. Real time 3D-handwritten character and gesture recognition for smartphone. *International Journal of Computer Applications* 123, 13 (2015).

[24] Euclides N Arcoverde Neto, Rafael M Barreto, Rafael M Duarte, Joao Paulo Magalhaes, Carlos ACM Bastos, Tsang Ing Ren, and George DC Cavalcanti. 2012. Real-time head pose estimation for mobile devices. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 467–474.

[25] Yoshikazu Onuki and Itsuo Kumazawa. 2016. Combined use of rear touch gestures and facial feature detection to achieve single-handed navigation of mobile devices. *IEEE Transactions on Human-Machine Systems* 46, 5 (2016), 684–693.

[26] Maria Francesca Roig-Maimó, Javier Varona Gómez, and Cristina Manresa-Yee. 2015. Face Me! Head-tracker interface evaluation on mobile devices. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. 1573–1578.

[27] Mohit Sharma, Dragan Ahmetovic, László A Jeni, and Kris M Kitani. 2018. Recognizing visual signatures of spontaneous head gestures. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 400–408.

[28] Juan R Terven, Joaquin Salas, and Bogdan Raducanu. 2014. Robust head gestures recognition for assistive technology. In *Mexican Conference on Pattern Recognition*. Springer, 152–161.

[29] Javier Varona, Cristina Manresa-Yee, and Francisco J Perales. 2008. Hands-free vision-based interface for computer accessibility. *Journal of Network and Computer Applications* 31, 4 (2008), 357–374.

[30] Paul Viola and Michael J Jones. 2004. Robust real-time face detection. *International journal of computer vision* 57, 2 (2004), 137–154.

[31] Simon Voelker, Sebastian Hueber, Christian Corsten, and Christian Remy. 2020. HeadReach: Using Head Tracking to Increase Reachability on Mobile Touch Devices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing*

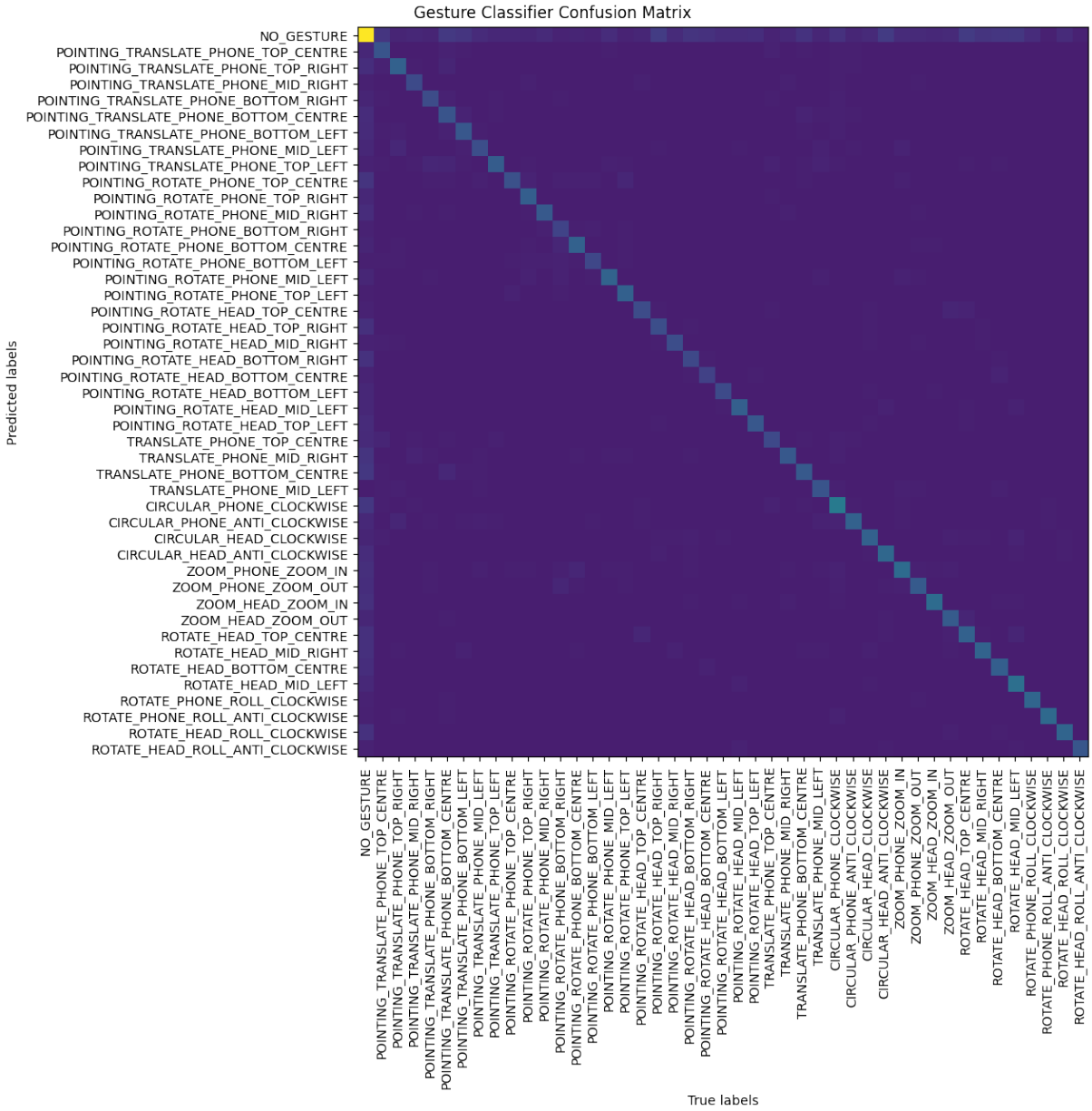

Figure 13: Gesture Classifier Confusion Matrix From Testing Set Evaluation

Systems. 1–12.

[32] Pei Xuesheng and Wang Yang. 2018. Research on the Development Law of Smart Phone Screen based on User Experience. In *MATEC Web of Conferences*, Vol. 176. EDP Sciences, 04006.

[33] Peng Yan, Binbin Wu, Xiaoqian Nie, Bingqi Wang, Xianliang Wang, and Ziwei Liu. 2021. A Fast and Efficient Face Pose Estimation Algorithm for Mobile Device. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE, 970–973.

[34] Yukang Yan, Chun Yu, Xin Yi, and Yuanchun Shi. 2018. Headgesture: hands-free input approach leveraging head movements for hmd devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–23.

[35] Shiqi Yu and Yuantao Feng. 2022. YuNet. https://github.com/ShiqiYu/libfacedetection

A TABLE OF GESTURES AND VARIATIONS

Head Gestures

Pointing-Rotate
Turn your head such that the desired content is in the centre of your vision. E.g. if the target is in the bottom right corner of the screen, turn your head down and to the right
Directions: Up, Up-Right, Right, Down-Right, Down, Down-Left, Left, Up-Left

Circular
Turn your head such that the path of motion of your nose draws a circle.
Directions: Clockwise, Anti-Clockwise

Zoom
Move your head directly towards or away from the phone screen.
Directions: Zoom-In, Zoom-Out

Rotate
Turn your head such that you are looking beyond the edge of the screen.
Directions: Up, Right, Down, Left

Roll
Lean your head to the left or right, bringing it towards your shoulder.
Directions: Clockwise (Right), Anti-Clockwise (Left)

Phone Gestures

POINTING-TRANSLATE
Move the phone such that the desired content is in the centre of your vision. E.g. if the target is in the bottom right corner of the screen, move the phone up and to the left.
Directions: Up, Up-Right, Right, Down-Right, Down, Down-Left, Left, Up-Left

POINTING-ROTATE
Turn the phone such that the desired content is in the centre of your vision. E.g. if the target is in the bottom right corner of the screen, twist the phone such that the bottom right corner is brought towards your face.
Directions: Up, Up-Right, Right, Down-Right, Down, Down-Left, Left, Up-Left

Translate
Move the phone such that the phone is off to the side of your head.
Directions: Up, Right, Down, Left

Circular
Move the phone such that its path of motion draws a circle.
Directions: Clockwise, Anti-Clockwise

Zoom
Move the phone directly towards or away from your head.
Directions: Zoom-In, Zoom-Out

Roll
Twist the phone such that the phone centre is still the same location, but the corners of the phone are moving clockwise or anti-clockwise.
Directions: Clockwise, Anti-Clockwise

B STUDY DATA ANALYSIS

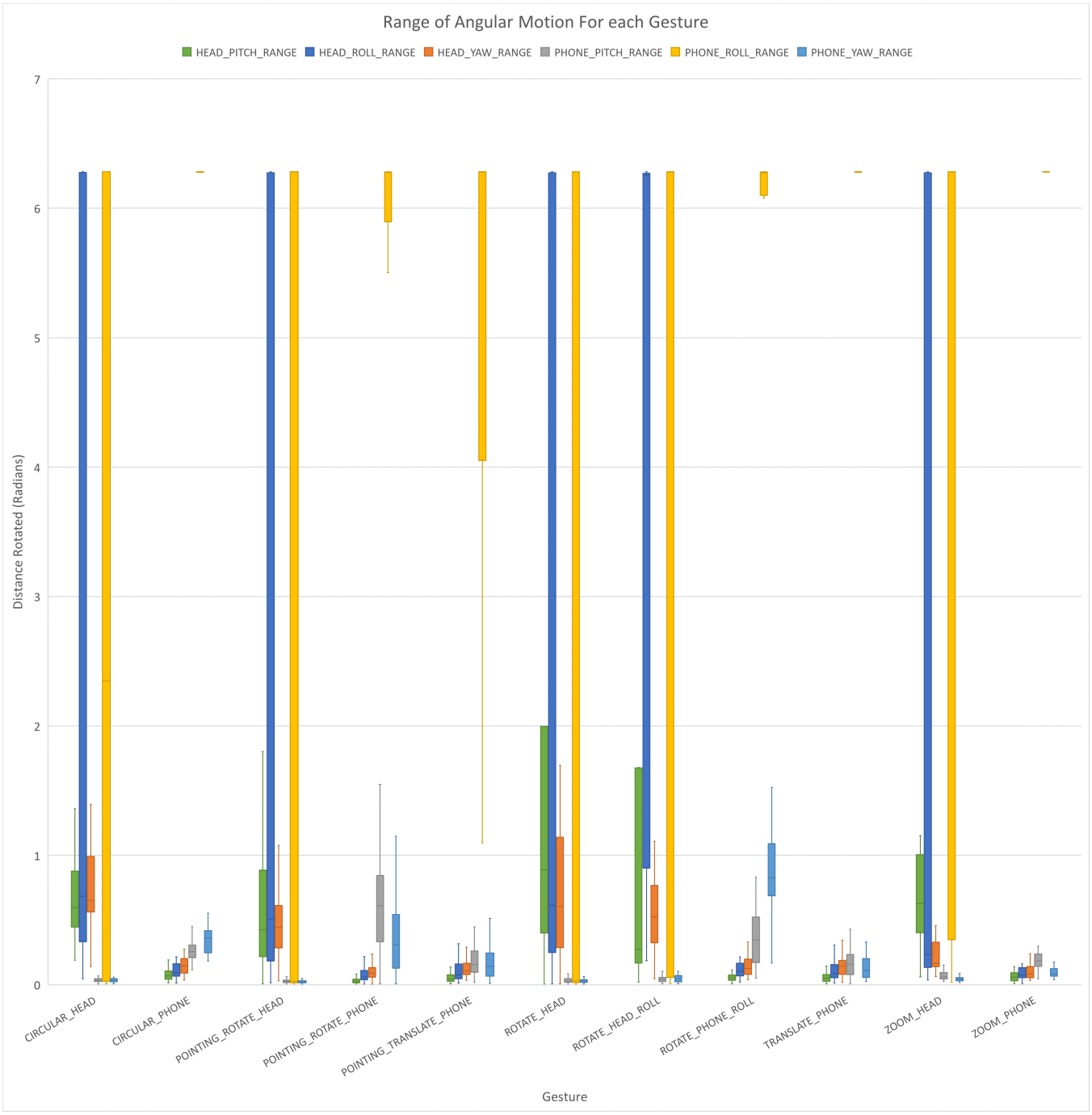

Figure 14: Range of angular motion for each gesture, in meters

Table 2: Frames with head visible, and total time elapsed per gesture and direction

Gesture And Direction	Percentage of Frames Where Head Is Detected	Time Elapsed (seconds)
CIRCULAR-HEAD	***97.4%***	***3.208***
ANTI-CLOCKWISE	97.1%	3.156
CLOCKWISE	97.7%	3.261
CIRCULAR-PHONE	***96.1%***	***3.618***
ANTI-CLOCKWISE	95.9%	3.451
CLOCKWISE	96.4%	3.785
POINTING-ROTATE-HEAD	***97.2%***	***2.466***
BOTTOM-CENTRE	95.3%	2.489
BOTTOM-LEFT	99.9%	2.522
BOTTOM-RIGHT	97.4%	2.463
MID-LEFT	98.9%	2.460
MID-RIGHT	100%	2.392
TOP-CENTRE	92.0%	2.552
TOP-LEFT	96.0%	2.319
TOP-RIGHT	97.8%	2.530
POINTING-ROTATE-PHONE	***64.6%***	***2.537***
BOTTOM-CENTRE	61.9%	2.580
BOTTOM-LEFT	61.7%	2.511
BOTTOM-RIGHT	68.6%	2.362
MID-LEFT	56.1%	2.480
MID-RIGHT	57.4%	2.604
TOP-CENTRE	90.2%	2.641
TOP-LEFT	60.3%	2.563
TOP-RIGHT	60.3%	2.554
POINTING-TRANSLATE-PHONE	***95.6%***	***2.865***
BOTTOM-CENTRE	98.6%	2.987
BOTTOM-LEFT	96.2%	2.639
BOTTOM-RIGHT	94.2%	2.705
MID-LEFT	89.0%	2.505
MID-RIGHT	94.1%	2.621
TOP-CENTRE	99.7%	3.568
TOP-LEFT	95.0%	2.828
TOP-RIGHT	98.1%	3.051
ROTATE-HEAD	***85.9%***	***2.843***
BOTTOM-CENTRE	84.6%	2.841
MID-LEFT	94.1%	2.723
MID-RIGHT	92.7%	2.837
TOP-CENTRE	71.6%	2.975
ROTATE-HEAD-ROLL	***96.7%***	***2.996***
ANTI-CLOCKWISE	96.2%	2.985
CLOCKWISE	97.2%	3.007
ROTATE-PHONE-ROLL	***87.8%***	***2.799***
ANTI-CLOCKWISE	78.7%	2.763
CLOCKWISE	96.9%	2.835
TRANSLATE-PHONE	***80.9%***	***2.898***
BOTTOM-CENTRE	85.0%	2.902
MID-LEFT	71.3%	2.918
MID-RIGHT	70.3%	2.847
TOP-CENTRE	96.4%	2.925
ZOOM-HEAD	***88.1%***	***3.072***
ZOOM-IN	79.9%	3.146
ZOOM-OUT	96.3%	2.998
ZOOM-PHONE	***89.6%***	***3.246***
ZOOM-IN	82.6%	3.247
ZOOM-OUT	96.6%	3.245

C CODE

C.1 Data Collection Application

https://github.com/whattheforkbomb/datacollection

C.2 Data Synchronisation

https://github.com/whattheforkbomb/dissertation_code/blob/trunk/datacollection/dataSynchronisation.py

C.3 YUV -> RGB Image Converter

https://github.com/whattheforkbomb/dissertation_code/blob/trunk/datacollection/imageGenerator.py

C.4 Data Resampling

https://github.com/whattheforkbomb/dissertation_code/blob/trunk/dataprocessing/data_analysis.ipynb

C.5 Model Development

https://github.com/whattheforkbomb/dissertation_code/blob/trunk/training/training.ipynb