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

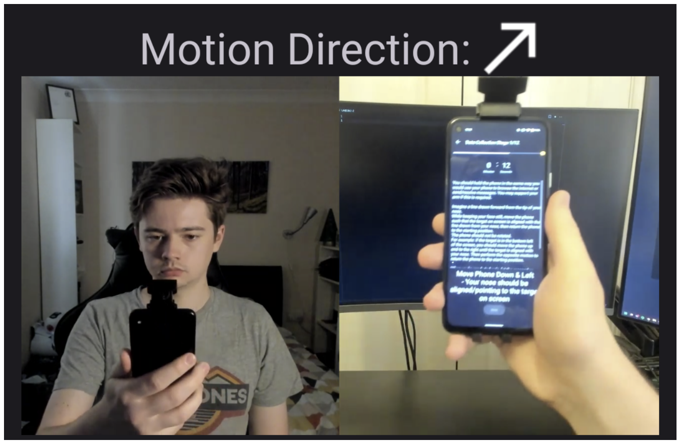

Figure 1: ***TODO: Replace with Image showing diff between head vs phone moving resulting in similar photo (at least head pose)***

ABSTRACT

TODO

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, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

There are attempts to introduce an additional modal of interaction with smart devices utilising the user's face. These exist on a spectrum with regards to interaction techniques: Using the face as a pointer, typically based on the movement/position of the user's nose; detecting gestures based on the movement/pose of the user's face; and a combination of the two.

A common issue that afflicts many of these systems/approaches is that they don't distinguish between the movement of the phone or the movement of the user's head. For example, the user moving their head to the left, will be treated the same as the user moving the phone to the right, since from the front-facing camera's perspective it looks like the head is moving in the same wayThis reduces the number of 'recognisable' gestures.

We look to explore whether such a system could distinguish between the user moving their head vs the phone being moved.

In order to develop a proof-of-concept, a data driven approach was taken. As such a study was undertaken to collect the camera feed and IMU/Gyro of the smart device, an IMU within an earbud worn by the user, and 3D positioning of the user's head and the smart device via a motion capture stage. With the motion capture data being synced to the IMU/Gyro data and photos, a system could be trained to recognise several gestures and learn to distinguish between whether an observed gesture was due to the phone or the user's head moving.

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

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

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

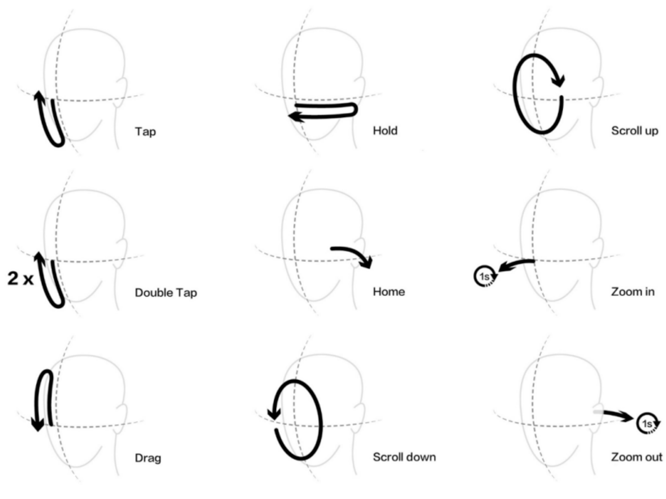

Figure 2: Proposed head gestures and their corresponding actions[20].

at existing literature that outline the head and phone gestures you would expect to use while interfacing with a smartphone.

Gestures can be classified into 5 classes[6]:

- **Dietic** These are 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.

Of these gesture classes only Dietic, Manipulative, and Semaphoric can be applied to head gestures.

An example of this can be seen in the work of Yan et al. who propose 9 gestures (Figure 2) and utilise the IMU embedded within the Hololens[20].

The gestures they propose were derived from a study wherein they asked participants to suggest head movements that they believed corresponded to the action taken. These were then collated by manually into 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.

[10] utilise a form of Manipulative gestures, wherein the pose of the head is used to either adjust the view of what is on screen, emulating a change in perspective, or by changing the UI (such as opening th ebookmarks bar while looking at a webpage)

Kong et al. outline several gestures to extend user interaction when user is forced to interact with phone single handed (flicking and zooming). These would be Manipulative and Semaphoric (some map to actual actions, e.g. flicking between items, others are less derivative and have less of a connection to the desired effect (moving phone closer/further from face.))

2.2 Head Feature Extraction and 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.

2.2.1 *HAAR Cascades / The Viola Jones Algorithm.*

(Viola Jones Algorithm[18])

[7] Utilise HAAR-like features with an Adaboost classifier to detect 4 features: left eye, right eye, nose-tip, and mouth.

Use Greyscale image, and use integral image (summed area table) to apply the features. Determine head pitch and yaw (not roll).

[14] Also utilise HAAR-like features in a classifier, however they simply use this to identify the region of the image that contains a face, and then utilise a histogram of pixel intensity (amount of variation in a given slice of the image) to identify the eyes. This is done once during calibration to then extract the eyes (left eye 30% along the line, right is 70%) and nose (if the eyes are d far apart, the nose is $d * 0.45$ below the midpoint between the eyes), assuming head is upright and user is facing camera.

A downside to the V-J algo is "Informal observation suggests that the face detector can detect faces that are tilted up to about ±15 degrees in plane and about ±45 degrees out of plane (toward a profile view). The detector becomes unreliable with more rotation than this."[18]

2.2.2 *Convolutional Neural Network (CNN).*

[19]

2.2.3 *Colour Segmentation.*

Either removing all colours that aren't within the defined colour space, to then perform additional processing, such as edge detection, or using histograms to find regions in the image of given colours to detect features?

2.3 Phone Localisation and Tracking

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[1] or the use of a Head Mounted Display with a mounted tracking marker[12]. These may be suitable in specific environments, but are not reasonable in meeting our goal.

More reasonable, yet still in-feasible, involve localising the smartphone relative to the 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. 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 (cameras that capture an array representing the distance to something in the environment) or LiDaR (which provides a point cloud of the environment ahead of the phone), and tracking the smartphone's movement through the observed 3D space. Unfortunately these require special hardware that isn't available on most smartphones, in particular rear-facing; most LiDar/depth cameras that exist on modern smartphones are front-facing***ref samsung and ios face filters/emoji crap***.

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)[3, 9, 11, 13]. An IMU provides the acceleration experienced in the 6 Degrees of Freedom (DoF)[1] the smartphone can be manoeuvred through.

Preprocessing (low-pass filtering, rolling average, kalman-filters, gesture segmentation).

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.

2.4.1 Relative Positioning.
One solution employed by papers proposing systems that tracked Manipulative (***or dietic, or both?***) pointing gestures was to simply use the raw data (or a function of the data) to map detected facial features directly alter the UI.

A common approach was to take the position of the nose extracted from an image from the front-facing camera and map it to a point on the screen[4, 15, 17].

2.4.2 Recurrent Neural Network (RNN).
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[16]. 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[2]. 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.

2.4.3 Hidden Markov Model (HMM).
One way that a gesture can be described is via a set of possible states (e.g. head poses or movement) and a set of rules which describe how these states can change. However you may not be able to directly observe the Elmezain et al. utilise such a method through the use of a Hidden Markov Model to [2] Neelasagar and Suresh took a similar approach, however they were trying to interpret arabic numerals 'drawn' with the movement of the phone[13]. As such IMU data was fed into a HMM to predict the number drawn.

After preprocessing / cleaning the data, the systems reviewed would the use HMMs[13] to classify the gesture performed, possibly in conjunction with additional input such as gaze[8].

A benefit of HMMs over RNNs is that they hold state between input, and as such you can process each frame of data as it is available, rather than needing to provide a sequence.

2.4.4 Dynamic Time Warping (DTW). regarding phone gestures...Others however utilised a technique called Dynamic Time Warping (DTW), sometimes used in conjunction with a HMM. ***refs***

3 METHODOLOGY

This section details the process undertaken to develop the system which can meet the goal (outlined in the Introduction): 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. The benefit of taking a data-driven approach is that we can leverage Machine Learning, and train the system with exemplar data, rather than needing to manually determine the features and derive the algorithm needed to accomplish our goal.

3.1 Taking 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:

Images From The Front Facing Camera
Given the majority of papers we reviewed ***Citations again?*** utilise a camera to track the user's face, from which they can derive the gestures, we feel it necessary to do the same.

Smartphone Acceleration Data
To understand whether the smartphone's PoV is changing we need to know how it is moving.

Head Acceleration Data
If we can determine the movement of the smartphone via acceleration data, it is reasonable to see if we can also do the same with the user's head.

Actual Head and Phone Pose (Ground-Truth)
In order to accurately train one models we propose below, we will need some Ground-Truth data.

With these data-types we can then build-up a dataset by recording each of the data-types during the performance of a series of gestures. Knowing the gesture associated to the recorded data will allow us to then train our system to recognise the gestures based on the data.

To create the required dataset 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

[1] 3 Linear Axis: X, Y, Z, 3 Angular Axis: Yaw, Pitch, Roll (though this can also be expressed as a Quaternion to avoid gimble lock***ref here, or just drop Quaternion note?***)
[2] ***metion that best to our knowledge this isn't supported***

samples of. A table of the gestures and variations can be found under Appendix A.

3.2 Data Collection Study

3.2.1 Apparatus and Techniques.
Given the data types listed above, we decided to use the following tools:

Smartphone
Pixel 4a An Android Smartphone with Bluetooth, a front-facing camera, and an IMU
eSense Earable - A Bluetooth Earbud with an IMU
CAMERA Motion Capture Studio
- A Motion Capture (MoCap) studio found on campus within the University of Bath

The smartphone and earbud (when paired via bluetooth) will be able to provide the first three data-types defined above. While the Ground-Truth data can then be supplied by the MoCap studio.

To collect the data we developed an application to run on the smartphone. This was developed in Kotlin[3] and the Android SDK. The application was designed to show participants a motion (a gesture and direction/variation) to perform. This is detailed in text, images, and a video. ***Include figures to show this (spread accross columns at top of page?)*** The participant would then be asked to perform this motion after pressing a record button. While recording the app would do the following:

- Capture images as frequently as possible from the front-facing camera, saving them as raw YUV bytes, with the UTC timestamp as the name.
- Record the smartphone IMU data (linear and angular accelerations), saving them to a csv with the UTC timestamp.
- Record the earbud IMU data (linear and angular accelerations), saving them to a csv with the UTC timestamp.

Once the participant has finished with the motion they could press the same button to stop the recording. Otherwise the recording will automatically terminate after 10 seconds, since the gestures shouldn't take more than a couple seconds to perform and the phone has limited RAM and storage with which to save data. To prevent accidentally stopping the recording too soon, say by accidentally double-tapping the screen, we disable the button for 2 seconds.

Once a motion has been recorded, the app shows the participant the next motion to perform. When the participant completes the final motion to perform, the app returns to the first motion. This repeats two times, such that each motion is captured 3 times. This is to collect variance in each motion for each participant.

In order to collect the Ground-Truth data, the study was performed within the MoCap studio. The participant was asked to wear hat that had a motion-tracker attached, such that the tracker was placed around the middle of the back of their head. An exact position wasn't important as we only needed to determine the relative movement of their head, rather than the exact position. The smartphone was then tracked via a motion tracker attached to a 3D-printed mount, such that the tracker would not affect the participant's grip on the phone, or interfere with the images captured from the front-facing camera.***Include figure to show this*** Each tracker was composed of 5 points. 3 were positioned such that they formed a right-angle triangle, allowing the orientation of the tracker to be derived. The other 2 points were there to improve tracking accuracy, and help make the trackers unique and distinguishable. The MoCap system would track each of the 10 points at 60 fps and export the data as an fbx file.

In order to later synchronise the data collected we required the user to shake the smartphone, with a force of at least 2G, prior to beginning each round of 44 motions. The app would record the shake magnitude (in the X, Y, and Z axis) and the UTC timestamp of when it happened.

The full study protocol can be found under Appendix C

3.2.2 Study Results. Our study was run with 8 participants.

Unfortunately due to an issue with the application, the earbud IMU data was not recorded, despite the earbud being on and paired with the phone. This was not caught until after the study was completed.

Due to a late start, and overrunning into the next participant's slot, participant 0 was unable to complete their 3rd round of motions.

Some participants didn't initially stop recording upon completing a motion, as such their initial motions have superfluous frames that don't contain data relevant to the motion they're recorded for.

Stats from the data, including figs and tables? Range of motion per gesture. Time taken by gesture and participant, Average sample rate for IMU and Images, Average Number of frames containing face by participant and gesture

3.3 Data 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.

Synchronisation and post-processing of the fbx data was performed with Blender and Python. Blender was used as permitted viewing the fbx data and verify derived location, roll, pitch, and yaw were correct. Also only way I was able to access the fbx data programmatically. 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.

To increase the amount of effective data we have for training we shall do some fps scaling, such that we copy the data, but assuming we're only capturing images every Xfps. We will find the photo

[3] A programming language that runs on the JVM and is used to develop applications for Android.

closest to the new frame where it would have been captured, and average appropriate data (such as acceleration).

We will also slice the data in overlapping chunks (at least for the RNN model).

3.4 The Proposed Models

To achieve our goal we opted to train 2 models with which we could evaluate and compare performance. The first is a cascading classifier which predicts the motion being performed given a sequence of data. It first identifies if a face is present in an image via a CNN which returns a bounding box of the face[21]. If a bounding box is present the pixels within the box are passed to another CNN which extracts the position of 68 landmarks of the face[5]. These landmarks, the bounding box location, the average IMU data since the last image, and the last X frames are then passed into an RNN we trained to classify the gesture. If no bounding box or landmarks are found for a given image, zeros are provided ***or previous data?***. ***is input going to be padded with zeros for first frames / last frames, or require certain number of frames before attempting classification?***

The second model is 2 models which will be trained to predict the direction of movement in each of the 6 DoF for the head and phone (the head model will also take the landmarks and bounding box as input). It will output as a 2d one-hot encoded array, each row being the Degree of Freedom, the column being the direction (0 = stationary, -1 = negative, +1 = positive). The output of the 2 can then be fed into a HMM trained to predict the gesture performed based on the derived motion. (possibly an RNN if easier)

3.4.1 *Training.*

3.4.2 *Model Evaluation.*

3.5 Model Deployment

4 SYSTEM EVALUATION

4.1 Results

4.2 Discussion

5 LIMITATIONS AND FURTHER WORK

Collect more data (to improve accuracy)

Collect the earbud data (if model unsuccessful)

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

6 CONCLUSION

REFERENCES

[1] 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.

[2] Mahmoud Elmezain, Ayoub Al-Hamadi, Jorg Appenrodt, and Bernd Michaelis. 2008. A hidden markov model-based continuous gesture recognition system for hand motion trajectory. In *2008 19th international conference on pattern recognition*. IEEE, 1–4.

[3] 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.

[4] 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.

[5] Yin Guobing. 2021. Head Pose Estimation By TensorFlow and OpenCV. https://github.com/yinguobing/head-pose-estimation

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

A TABLE OF GESTURES AND VARIATIONS

B DATA COLLECTION APP CLASS DIAGRAM

C STUDY PROTOCOL DIAGRAM

D STUDY DATA ANALYSIS