GameboyAir: A Camera-based User Interface for Cursor Control and Gaming

Zhibo Fan
zb1439@berkeley.edu

Tzu-Chuan Lin
tzu-chuan_lin@berkeley.edu

Abstract

This paper describes GameboyAir, a mouse-free and keyboard-free user interface library for laptop which supports cursor control and gaming. GameboyAir could run on any laptop with a camera and controls mouse cursor and keyboard inputs based on detected hand gestures and movements. We utilized MediaPipe [11] for hand keypoint detection and tracking, and create higher level features and build classifiers upon the detected keypoint coordinates. With gesture sequence and keypoint coordinate sequence, we can move the cursor, raise mouse events, and simulate keyboard input for various applications. It is worth mentioning that GameboyAir is a framework rather than an application that developers can easily create new applications based on it.

1. Introduction

Hand keypoint detection [1, 3, 11] is an active research field due to its potential for the next-generation human computer interaction. For example, given the detected hand keypoints, gesture recognition and VR/AR algorithms can be applied to control the device and render interesting visual effects. In this project, we focus on applying hand keypoint detection for a novel mouse-free and keyboard-free user interface on laptops with cameras, where applications could be built upon the interface. Since the project is more like a contact-less platform for different games and applications, we name it as GameboyAir.

There are several literature on cursor control by hand [2, 6], however, they focus mainly on gesture recognition and could not be easily deployed on laptops. Horatiu-Stefan et al. [2] requires special environment to detect hands and Okan et al. [6] utilized a heavy 3D-CNN [10] with ResNet101 [4], resulting in low frames per second (FPS) and bad user experience. In contrast, we propose to make an easily accessible user interface which only requires a webcam and can be run real-time without special hardware (e.g., GPU). Fortunately, Google's open-source library MediaPipe [11] offers a hand landmark detection solution which can be run real-time with a promising accuracy. Given the 3D keypoint locations, we are able to build simple yet efficient classifiers and controls the device accordingly.

Specifically, we build a feature descriptor indicating whether a finger is bent or straight based on the angles and distances. Then, we implement static gesture classifiers based on the finger feature descriptor and the original keypoint coordinates, and dynamic gestures are viewed as special sequences of static gestures. For cursor control and other applications involving position tracking, however, we found that MediaPipe [11] outputs are not stable enough due to input noise. Thus, we add a Kalman filter [5] to eliminate the noise and make movements smoother. In addition, we carefully design a camera-based user interface pipeline with interfaces between different modules such that developers are able to customize and plug in their own modules based on GameboyAir with minor amount of code. We build a cursor controller and games including Mario, Greedy Snake, NSShaft, and Flappy Bird based on the GameboyAir library.

2. Method

In this section, we focus on introducing the methodologies we used in this project. We start with a brief introduction of MediaPipe Hand [11], then detail our original implementation based on MediaPipe.

2.1. A Brief Introduction of MediaPipe Hand

Google's MediaPipe Hand [11] solution is divided into palm detection and landmark detection stages. Palm detection utilized an hourglass structure backbone [7] to support low-resolution input with considerable amount of anchors. The detector head resembles RPN [8] to predict the bounding box of hand palms, with an extra head predicting coarse-scale keypoints for finger base knuckles. Then the bounding box is rotated based on the direction from the wrist keypoint and the middle finger knuckle and scaled 2.6x to possibly include the full hand. The original image is cropped by the scaled box and fed into the landmark detector to produce 21 hand landmarks as shown in Figure1. MediaPipe Hand [11] also includes a hand existence classifier and a keypoint tracker to prevent unnecessary computation when no hands exist or re-computing the keypoints if the move-

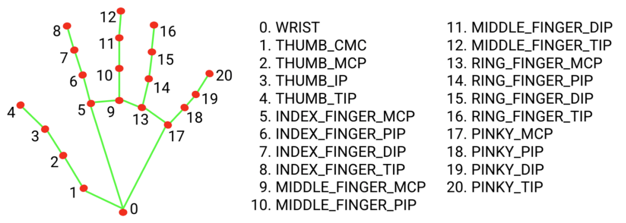


Figure 1. MediaPipe hand landmarks [11].

ment is small. However, the original predictions from MediaPipe is not smooth enough for cursor control.

2.2. Heuristic Finger Descriptor

We build two different heuristic finger descriptors, based on distance and 3D angles, respectively. The former one is robust to gesture transition, which is applied on cursor control where mouse events are triggered by more complex gestures. Generally, the latter one is more stable but fails to ignore the transition between certain gestures and yields false positive.

The distance-based descriptor consists of stacked conditions for each finger. For each finger, the distance from the finger tip and the finger knuckle to the wrist and the ratio between the two is computed, and we threshold on some of these values to decide whether the finger is straight or bent. Additionally, since the thumb is shorter than the other four fingers, the distance criteria may fail occasionally, thus we add an extra condition thresholding the 2D angle of the vector from thumb tip to thumb knuckle and the vector from thumb knuckle to the wrist to justify the classification.

The angle-based descriptor is inspired by Google's latest paper building real-time gesture recognizer based on MediaPipe [9] (implementation and API not released yet). Specifically, we divide each finger into **four** segments: from tip to finger DIP (or thumb IP), from finger DIP(or thumb IP) to finger PIP (or thumb MCP), from finger PIP (or thumb MCP) to finger MCP (or thumb CMC), and from finger MCP (or thumb CMC) to wrist (see Figure1). We compute the 3D angles between the first mentioned segment and the other three, and take the maximum angle. Then we apply different thresholds on this angle to decide different fingers being straight or bent. Generally speaking, this descriptor is more stable in most cases, but suffers failures from certain gestures. For example, when pinching (thumb and index finger meet), the gesture transition yields false positive classification, and results in the cursor to move away before mouse clicking is triggered (which is the corresponding mouse event for pinching). Thus, we utilize the distance based descriptor in cursor control, and the angle based one for other applications.

2.3. Gesture Recognition

In this project, we assume only one hand appears in the camera and only deals with single-hand gestures. To ensure real-time classification, the classifier takes the current keypoint coordinates and the sequence of finger descriptors as input features instead of raw pixels. Final finger description is obtained by voting from the finger descriptors of the last 5 frames. Static gestures are recognized based on a stack of if-else conditions upon the given features, and we assume dynamic gestures are sequences of static gestures with simple patterns, which can be classified based on static gesture sequences. Without using machine learning or deep learning models, the carefully justified heuristic based classifiers serve well in the whole pipeline. Different applications may use different gestures and therefore different classifiers, we will detail the gestures and classifiers for different applications in Section 3.2 and Section 3.3.

2.4. Cursor Control and Kalman Filtering

Keypoint tracking is required for cursor control applications and needs highly stable landmark detection. Since the output from MediaPipe [11] is not stable enough to provide a smooth user experience on cursor control, we apply a Kalman filter to remove the measurement noise from the keypoint predictions. We use position-based relative control to move the cursor, which scales the movement of the index finger tip when the gesture "point" is detected.

Kalman filter [5] is a classical recursive noise filter under linear system assumptions. Here we define the observation states as the x-y coordinates, and assume that the system dynamic follows Newtonian kinematics and is linear w.r.t to position, velocity, and acceleration along x and y axis. The system at time step k is formulated as follows:

$$X(k) = AX(k-1) + W(k-1) \tag{1}$$

$$Z(k) = HX(k) + V(k) \tag{2}$$

where X is system states (position, velocity, and acceleration along x and y axis) and Z is observation states (x-y position). W and V are process noise and measurement noise and are initialized as $\sigma_p I$ and $\sigma_m I$, respectively. We set $\sigma_p = 0.003$ and $\sigma_m = 1$ since the output from MediaPipe can be quite noisy. As we assume the system dynamic follows Newtonian kinematics, A and H is defined as:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

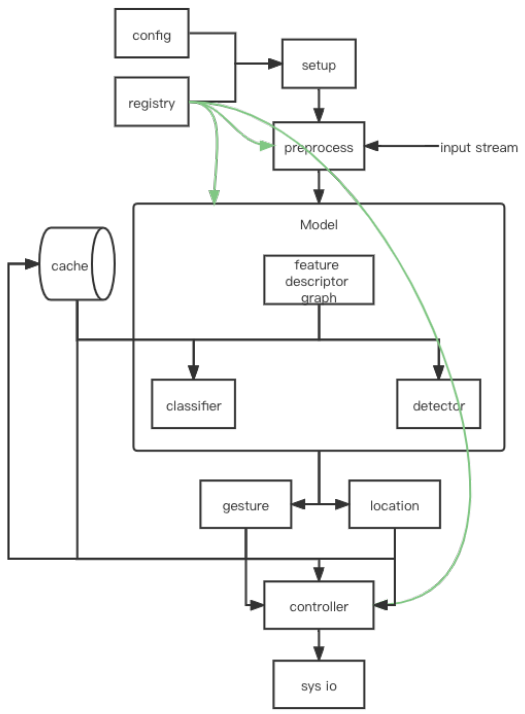


Figure 2. GameboyAir pipeline illustration. Black arrows indicate information flow and green arrows indicate module plug-ins.

3. Results

In this section, we first detail the design of GameboyAir and its features, then we go through the applications built upon GameboyAir, including cursor control and four games.

3.1. Framework Design

An illustration of GameboyAir pipeline is shown in Figure 2. First, the pipeline is initialized based on the configuration file and registered modules, then the input video frames are preprocessed and fed into the model part. Model consists of a feature descriptor graph, where successive feature descriptors may rely on other feature descriptors, for example, the default feature graph consists of the MediaPipe [11] API and the finger descriptor, where the latter one depends on outputs from MediaPipe. Every output from the nodes in the feature descriptor graph is fed into the classifier and the detector, which yields the gestures and locations to control the cursor and raise mouse events. Kalman filter is applied in the detector module, which perceives the keypoints as measurement and predicts the final location. In addition, an external memory cache stores a fixed-length queue of the output from the modules, which facilitates sequence-based modules like the dynamic gesture classifier and relative control of the cursor.

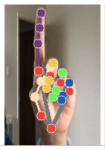
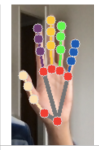
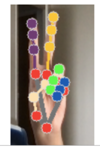
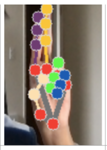
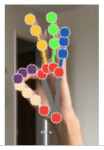
Sample					
Gesture	Point	Open palm	Victory	Victory(close finger)	Pinch
Mouse event	Move cursor	Right click	Swipe	Click	Left Click & Hold

Figure 3. Gestures and their corresponding mouse events for cursor control.

Note that the above mentioned pipeline is flexible and can be customized using different configuration files and registered modules. For example, to build a new application using new gestures, the developer only needs to derive from the base classifier with a customized `predict` method and modify the configuration file accordingly. We will show applications build upon this framework in Section 3.2 and Section 3.3. Please refer to our implementation for more details.[1]

3.2. Cursor Control

Gestures Recognition. We support gestures as shown in Figure3 to control the cursor and raise mouse events. In 3, the first row corresponds to sample images of hand gestures, the second row corresponds to the names of the gestures and the last row corresponds to the mouse events to be raised. The gestures are classified based on which fingers are bent or straight, as well as specific distance constraints. For example, gestures for swipe and click use the same fingers, but can be discriminated by the minimum distance between the keypoints on index finger and middle finger.

Single-time Activation. Inspired by [6], we apply single-time activation for right click since open palm is sensitive to gesture transition. Empirically, undefined gestures may be recognized during right-click due to noise or slight change in hand, which is the limitation of distance-based finger descriptor as stated in Section2.2. Therefore, we do single-time activation instead of activating immediately after the gesture is detected. Specifically, we only raise the mouse event of the most recent gesture of category g if the gesture is the only g in the history gestures sequence. Demo video is available on YouTube.[2]

3.3. Gaming

For currently supported games, we raise keyboard events from gestures to control the agents inside the game. Games run asynchronously and communicates with the main process via the keyboard events, and once the main process or the game process terminates, the other one terminates. Video demos for the games are available.[3]

[1]https://github.com/zb1439/finger_cursor
[2]https://youtu.be/EURRLRz1Bgo
[3]https://youtu.be/wicQDNr-hX8

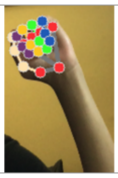
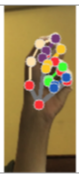
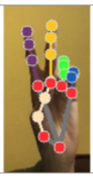
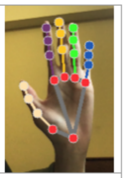
Sample				
Gesture	Fist lean left	Fist lean right	Victory	Open palm
Event	left	right	jump	jump (press longer)

Figure 4. Gestures and their corresponding events for Super Mario.

3.3.1 Greedy Snake

We detect the direction from the wrist to the mean knuckle of the index finger, middle finger, and pinky finger to control the movements of the snake. If the direction falls in a local range of 0, 90, 180, 270 degrees, the snake will move upwards, leftwards, downwards, and rightwards, respectively.

3.3.2 Flappy Bird

Flappy Bird only involves one button to play. We define the action from fist to open palm as the bird's "flapping". Similar to Section3.2, single-time activation is utilized to classify this dynamic gesture and avoid duplicate "flapping".

3.3.3 Super Mario

We designed four gestures for this game, shown in Figure 4. For the left and right events, we used the same method described in section 3.3.1 to compute the angles and then decide whether it is a left-leaning or right-leaning fist. In addition, because the height of the agent - Mario's jump depends on how long the player press the jump button. We invented two more gestures to raise two types of the jumps (a short-press and a long-press jumps).

3.3.4 NS Shaft

For this game, we adapted the idea from how 3.3.3 controls left and right event to move the agent.

Acknowledgements

We thank Prof. Alexei (Alyosha) Efros and Prof. Angjoo Kanazawa for delivering this wonderful lecture and offering advices regarding this project. The idea for this project is inspired by the M.Eng capstone project, Mouse-free Assistive Technology, supervised by Prof. Brian Barsky, especially the cursor control part. We conducted the design and coding for this project independently. We borrowed design patterns and some codes from Detectron2[4] and cvpods[5], and the games are modified upon open source implementations listed in the footnote [6].

References

[1] Oculus connect 6: Introducing hand tracking on oculus quest, facebook horizon, and more. https://www.oculus.com/blog/oculus-connect-6-introducing-hand-tracking-on-oculus-quest-facebook-horizon-and-more/. 1

[2] Horatiu-Stefan Grif and Cornel Cristian Farcas. Mouse cursor control system based on hand gesture. *Procedia Technology*, 22:657–661, 2016. 1

[3] Tomasz Grzejszczak, Michal Kawulok, and Adam Galuszka. Hand landmarks detection and localization in color images. *Multimedia Tools and Applications*, 75(23):16363–16387, 2016. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[5] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960. 1, 2

[6] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, and Gerhard Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–8. IEEE, 2019. 1, 3

[7] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016. 1

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015. 1

[9] George Sung, Kanstantsin Sokal, Esha Uboweja, Valentin Bazarevsky, Jonathan Baccash, Eduard Gabriel Bazavan, Chuo-Ling Chang, and Matthias Grundmann. On-device real-time hand gesture recognition. *arXiv preprint arXiv:2111.00038*, 2021. 2

[10] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015. 1

[11] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *CoRR*, abs/2006.10214, 2020. 1, 2, 3

https://youtu.be/KT7G452Gedg
https://youtu.be/Zvl704EexbI

[4]https://github.com/facebookresearch/detectron2
[5]https://github.com/Megvii-BaseDetection/cvpods
[6]Greedy Snake: https://github.com/GMfatcat/greedySnake,
Flappy Bird: https://github.com/sourabhv/FlapPyBird,
NS Shaft: https://github.com/iPel/NS-SHAFT,
Nes-py: https://github.com/Kautenja/nes-py