Finger People: a hand-character model for natural video game input

Noah Murad
University of Waterloo
Waterloo, Canada
nbmurad@uwaterloo.ca

ABSTRACT
People can represent a character's movement by letting their index finger and middle finger represent the character's legs, and the rest of their hand represent the character's body. Finger People is a system that implements this paradigm as a video game input controller for 2D side-scrolling games. Our system's hardware consists of a single camera. Finger People is implemented by analyzing the blob behavior and optical flow of the player's hand to determine which gesture they are performing. Our system is advantageous to other gesture recognition systems because it only requires one hand to operate, it has simple hardware requirements, and it implements a unique paradigm.

Author Keywords
human-computer interaction; computer vision; video games; input systems; hand detection; gesture recognition; optical flow.

Video https://www.youtube.com/watch?v=HhO2iqlgZCQ&feature=youtu.be

INTRODUCTION
Humans can naturally use their hands to represent story characters. Specifically, the index and middle fingers serve as the character's legs, and the rest of the hand acts as a large upper-body (Figure 1a). Video games traditionally use game controllers as input, but there have been recent emerging natural gaming interfaces such as the Microsoft Kinect and the WiiMote. In our project, we use the hand-character model to act as a natural input system for video games. This interface should provide the player with a more interactive, immersive, and intimate experience.

Most of the available natural gaming interfaces require the player's entire body for input. Our system only uses one of the player's hands to operate which requires significantly less space and energy from the player. Gaming input systems almost always require the use of handheld hardware. Our system is more natural since, like the Kinect, it only uses a camera. But unlike the Kinect, our system can have a seamless hardware setup since cameras are usually already embedded in the gaming device (laptop or smartphone).

We call our system Finger People. Finger People allows video game players to use their hands as controllers by using a camera to gather gesture data from the user's bare hand. The player's hand acts as the game character which the player is controlling, where the character's legs are represented by the player's index and middle fingers. At its core, Finger People is a hand gesture recognition system. The different gestures that Finger People recognizes are mapped to different game inputs. Finger People's basic gestures are idle, run, and jump (Figure 1bcd). These three basic gestures allow Finger People to act as a controller with a joystick and at least one button. This makes Finger People suitable for many 2D side-scrolling platformers such as Sonic the Hedgehog and Super Mario Bros. Overall, Finger People aims to create a natural and playful input system for video games.

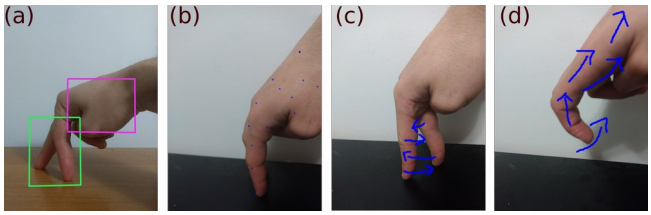

Figure 1: (a) is the hand-character model, with the Finger Person's legs highlighted in green, and the upper-body highlighted in magenta. (b), (c), and (d) is Finger People's fundamental gestures which are idle, run, and jump, respectively. There are approximate motion vectors (in blue) to help visualize motion.

Finger People is the first video game controller to use hand gestures which directly reflect the game character's actions; specifically, Finger People brings the hand-character model to life.

RELATED WORK

Hand Gesture Recognition
Finger People's core problem is to perform hand gesture recognition. [11,15] present a comprehensive overview of hand gesture recognition techniques. Vision-based hand gesture recognition consists of three steps: detection, tracking, which is optional if detection is fast enough, and recognition [11]. Typically, detection is performed with color, shape, texture, and motion features [15], and recognition passes these features though a classifier such as a Support Vector Machine (SVM) for static gestures or a Hidden Markov Model (HMM) for dynamic gestures [11].

Fang et al. [6] created a real-time gesture recognition system with Adaboost to detect hands, optical flow to track, and an automatically adjusting color model to segment the hands. Detection is performed on the segmented hand through blob (palm) and ridge (finger) detectors. This limits gestures to being defined by finger count and orientation. Cyclops [2] uses Motion History Images (MHI) in order to speed up computation by only analyzing one image per frame instead of a history of frames for each frame. A few projects use neural networks as classifiers. Molchanov et al. [10] train a convolutional neural network (CNN) on 885 samples to achieve 75.5% accuracy on 19 dynamic gestures. The network consists of RGB-D image gradients which pass though high-resolution and low-resolution sub-networks – similar to how optical flow operates on multiple resolutions. Each network layer is three dimensional with x,y, and time dimensions. Tsironi et al. [14] go a step further and build a 2D recurrent neural network instead of a 3D network. The architecture consists of small (64x48) MHI images which are fed into a CNN, which are then passed though a Long-Short Term Memory (LSTM) recurrent network. The network achieved over 90% accuracy, precision, and recall on 9 gestures by training on 543 samples. This displays that, with proper architecture, a robust detector can be trained with few training samples and small input resolutions. Looking at many different hand gesture recognition systems, we observed that there is success across a wide variety of algorithms as long as the detector-and-recognizer architecture is used. We also observed that various input preprocessing techniques, such as MHI and gradients, simplify the classification problem.

Dynamic Gesture Recognition

At their highest level, hand gestures can be classified as static or dynamic [11]. Since most of our base gestures are dynamic (all except idle), we are interested with dynamic gesture recognition. Rautaray and Agrawal [11] presented HMMs and Dynamic Time Warping as solutions to dynamic gesture recognition. In neural network gesture recognition approaches, we have seen Molchanov et al. use a 3D network [10] to accommodate for the time dimension among the sequence of 2D images. This technique does not allow for variable-length gestures. Tsironi et al.'s newer neural network-based approach uses a recurrent LSTM network [14], allowing for variable-length gestures. Cutler and Turk [3] develop a real-time optical flow algorithm which is used to pre-process images. Blobs are formed based on pixels with similar optical flow movement. The blobs are then classified as gestures. We have also seen projects use MHIs [2,10] to store the motion values of previous frames into one frame, reducing dimensionality and increasing processing speeds.

Gesture Recognition for Games

There is a vast selection of gesture recognition techniques, but when designing for games, researchers mention that it is important that the algorithms operate at high speeds with low hardware costs [7,8]. Freeman et al. [7] use custom hardware to quickly calculate image moments and orientation histograms in order to approximate position and orientation of hands or bodies. The parameters obtained from the user's hands or body are then passed as input to steer vehicles and balance on snowboards in video games, respectively. Brehme et al. [1] built a dance-pad controller which is printed and placed on the floor. Shoes with markers must be worn to track feet, and two cameras are used in order to obtain depth information to determine whether a player's foot in grounded or hovering. Cyclops [2] recognizes arm and leg motions with one wearable device. These capabilities are used to drive cars and throw snowballs in video games. Dong et al. [5] implement a gaming input system for full-body gestures using adaptive background subtraction and trained pose recognition. Many recent projects [9,12,13] use a depth camera (specifically, Microsoft's Kinect) to simplify the segmentation step of gesture recognition. While this technique is more effective than using an color [4], it comes as the cost of expensive hardware and does not simplify the entire recognition process. No available vision-based gaming input system uses dynamic finger movements, or more specifically, the natural hand-character model which Finger People implements. Finger People also has the cheapest hardware setup (a single, regular camera), and the required fast response times for gaming applications.

Figure 2: Screenshot from Sonic the Hedgehog, a 2D side-scrolling platformer. In this game, the character must travel to the end of the level (to the right), collecting rings and defeating enemies.

SYSTEM OVERVIEW

Finger People is a video game input system which uses a hand-character model where the player's index and middle fingers serve as the character's legs, and the rest of the hand acts as a large upper-body. The Finger Person can run, jump, and stand still to control a video game character (Figure 1). The hardware requirements for using Finger

People are simple: a camera – no gloves or bright colored bands required for the user to wear. To make setup even easier, Finger People offers the user the option to use their smartphone's camera as the input device. This is convenient because smartphones can easily be mounted on stands or leaned against objects like glasses cases or water bottles to point at the player's Finger Person.

A video game's renderings and interface is what is ultimately responsible for providing feedback to the player from their inputs. A player using Finger People will not see any of the inner-workings of Finger People; they will simply see the game which they are playing, with the game's feedback to their gestures being the only indication that Finger People is running. Representing user inputs as video game responses is an essential feature of video game control systems which breaks the barrier between a player clicking buttons, and a player controlling a character. Finger People aims to have the in-game character's movements be represented by the player's hand movements. If a player's Finger Person slowly moves their legs, their in-game character will walk. If the player speeds up their Finger Person's leg movement, the in-game character's speed will increase as they begin to run. Similarly for performing a jump by lifting their hand, or idling by making their Finger Person stand around, the in-game character will mimic those actions. Finger People aims to make the player feel like the in-game character is an extension of themselves in order to create a more immersive experience.

Finger People is a non-general purpose video game controller. Like any specialized video game controller such as a steering wheel, a dance pad, or a guitar controller, Finger People is most suitably used on a subset of video games. Finger People aims to be used with 2D side-scrolling platformers (Figure 2); games which require the player to control a character in 2D space, from a side view (as opposed to a top-down view), to reach a goal point by jumping on platforms, fighting enemies, collecting items such as power-ups and currency, and solving puzzles. Finger People can currently allow a player to idle, walk or run left and right, and jump. While this is not many controls, these controls are sufficient to play popular games such as Sonic the Hedgehog and Super Mario Bros. Finger People is still in early development, see the Future Work section for information on how we plan to support more controls and a wider range of 2D side-scrolling platformers.

Finger People is an interesting concept because there does not yet exist a natural interface for the popular genre of platformers. As noted, this is a genre where the player controls a character; moving the control from clicking buttons to using one's hand as a direct representation of the character is a big step in immersion. To add to the immersive experience, the user is not required to wear any special equipment. Also, unlike most natural interfaces, Finger People requires a relatively small physical space and

Figure 3: A screenshot of Finger People's debug screen (left) and hand mask (right). The green text in the debug screen is (from top to bottom) the mean magnitude of the optical flow of the Finger Person's legs, the x- and y-velocity of the hand, and the gestures being recognized by Finger People (running, not jumping, not kicking, moving forwards). The red box should surround the Finger Person's legs. The red box is formed starting from the center of the hand blob (blue dot), extending to the farthest left point (magenta dot), and extending down to the bottom of the frame. The green dot is the lowest point of the hand blob.

amount of energy from the player to use as it only requires the player to use one hand. This contrasts with other systems such as dance pads, steering wheels, and the Kinect and is similar to the WiiMote. Finger People is also very affordable as it operates with only a single camera, making it much cheaper than most natural gaming interfaces. Additionally, while the camera does have a frame-rate requirements of at least 30 frames per second, (a standard frame-rate), the camera has virtually no resolution requirements as Finger People operates with 174x144 video.

IMPLEMENTATION

Finger People is implemented in four modules: hardware, hand detection, gesture recognition, and video game input. It is important that Finger People can run in real-time, so we designed the vision pipeline with real-time algorithms. All of Finger People's software components are implemented using Python and OpenCV.

Hardware

Finger People uses a simple hardware setup: one camera. The camera must face towards a surface where the player can rest their hand to have their Finger Person standing. Because it uses the optical flow of the user's fingers, Finger People operates more accurately if it has a side-view of the user's Finger Person. This angle could most easily be achieved through the placement of an external camera. We initially configured our system to run with a USB camera. Because smartphones are more ubiquitous than USB cameras, we decided to also enable the use of Android smartphone cameras. We used an Android application called IP Webcam [16] to stream 174x144 video to our computer through Wi-Fi. IP Webcam provides an address

from which we can load a JPEG image of the most recently streamed frame. We use Python's networking module to read this image and then convert it to be used by OpenCV. We use low-resolution video because it provides sufficient information to execute our algorithms with high accuracy and because it ensures real-time processing speeds. While we are able to achieve real-time processing speeds, we encountered difficulties streaming video in real-time under certain Wi-Fi conditions.

Hand Detection

Before recognizing gestures, Finger People first locates the player's hand. This step is necessary to isolate the data of interest. To detect hands, Finger People combines color-, position-, and shape-based hand detection techniques [11]. To increase the difference in color of the player's hand from similar-colored objects (such as wood, or their face) we enable the smartphone's front-facing LED. The user first calibrates the skin color histogram by taking 5 sample snippets of their hand, preferably in different positions. A mean histogram of the samples' hue and saturation is then calculated, and each frame is back-projected onto this histogram and thresholded at a value of 25 in order to get a rough mask of the player's hand. To clean up this mask, the frame is blurred with a 7x7 Gaussian filter before performing back-projection. After back-projection the binary mask is and passed through one iteration of opening morphology, and two iterations of closing. To differentiate the hand from any remaining blobs, we assume the hand is the largest blob. Alternatively, we have also used the blob with the lowest position, with an area greater than some threshold, but because the hand is always very close to the camera, simply using the largest blob has proven to be more robust.

Gesture Recognition

With a rough mask of the player's hand, Finger People can classify hand gestures. Finger People currently supports *idle*, *jump*, and *run* gestures. The **idle** gesture is implied – it is always assumed that the Finger Person is idling, unless another gesture is occurring. Because of this design, it is important that there is a low false positive rate for any other gestures. To detect a **jump**, we calculate the hand's velocity using its blob's center as its position. In effort to reduce errors while maintaining real-time updates, we use exponential smoothing with α=0.5 to update the hand's velocity. If the hand's y-velocity is greater than 1.5 times its x-velocity, and if its y-velocity exceeds 7.5 pixels per frame, a jump is triggered. Similarly, if the former condition is met, and the hand's y-velocity exceeds -4.0 pixels per frame, the jump is terminated. To avoid jump-lock, if the jump-termination condition is not met for 0.8 seconds, the jump is terminated. To detect **run** speed, we use Farneback optical flow. Before performing optical flow, we need to define a region where the Finger Peron's legs are located. Because the Finger Peron's legs are blurred from motion when running, relying solely on the hand

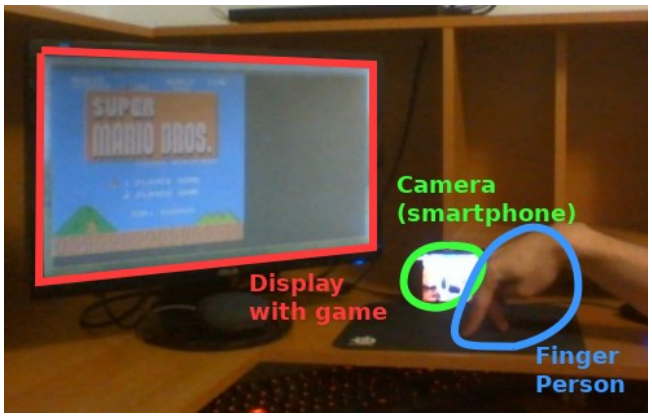

Figure 4: Finger People's hardware setup.

detection algorithm is not robust. Instead, we define the Finger Person's leg region as a rectangle which starts at the hand blob's center, extends as wide as the hand blob's left-most point (or right most point for right-facing finger people), and down to the bottom of the frame (Figure 3). We extend to the bottom of the frame because there is very little space below the Finger Peron's legs in frame which is only filled with desk-space which has no movement. To calculate the Finger Person's run speed, we take the average magnitude of the Farneback optical flow in the leg region. We disregard the direction because the optical flow algorithm is not robust at detecting the direction of the blurry movement of the Finger Person's legs. If the run speed exceeds 1.0 pixels per frame, the Finger Person is walking, if it exceeds 3.5 pixels per frame, the Finger Person is running, otherwise, the Finger Person is standing still. The run speed can also just be interpreted as a float value to emulate a joystick. To determine run direction, we use the x-position of the hand's blob center. If this position is in the front 7/9 of the frame ('front' is determined by the Finger Person's facing position, configured by a boolean), the Finger Person runs forward (typically right), otherwise, they run backward. By being able to detect the Finger Person's action, we can use these actions as input into a game.

Game Input

The final module of Finger People sends inputs into a video game based on the Finger Person's actions. Specific inputs are game- and platform- dependent, but the modular design of Finger People's input system allows developers to easily configure the system for any platform by simply defining functions which emulate an arbitrary key press, and an arbitrary key release. We will describe our implementation of this module for Super Mario Bros., emulated on a Linux system through FCEUX [17]. Super Mario Bros. is a game originally designed on the Nintendo Entertainment System (NES), to be played with a controller with an A button, a B button, and a directional pad. Finger People's jump gesture will press down on the A button. The walk gesture will press down on the corresponding direction which the Finger

Person is walking. The run gesture is similar to the walk gesture, but also holds down on the B button. Because this is emulated on a Linux system, the NES controller buttons are mapped to keyboard keys. To emulate keyboard key presses on Linux, we have our Python code spawn a thread to run Linux's `xte` [18] command. This effectively translates the user's gestures into video game inputs through Finger People's natural model.

DISCUSSION

We implemented a system which uses a single camera to recognize idle, run, and jump gestures in the hand-character model, and used this system to trigger inputs in a video game. This natural and immersive input system is designed to be used with 2D side-scrolling platformers. Although we only demonstrate our system working with Super Mario Bros. through an NES emulator on a Linux system, the modular architecture of Finger People allows us to easily configure the inputs for any game on any platform.

Mixed results were obtained from our preliminary user testing sessions. Overall, while contributing to a more immersive experience for the player, Finger People makes controlling the character more difficult than using traditional tactile input. While this provides a fun and new challenge to experienced players, it can be frustrating to new players and non-gamers. However, these observations are inconclusive, because much of our user testing was performed under suboptimal conditions; our WiFi connections were not perfect, which caused an input lag of approximately 1 second.

This raises other concerns. Although using smartphones does have the advantage of portability and omnipresence, if we cannot guarantee real-time input via smartphone connections, other mediums must be used for capturing video. These issues, however, do not need to be of concern early in the development process unless using a specific input device is essential to the system's design (which it is not, in our case).

When designing an input system, it is most important to consider its limitations, and to design around them. In our case, a major limitation of our system is the idea itself; while unique and interesting, the hand-character model has its limitations. Rotating a Finger Person so that they can walk in their non-natural direction is difficult and uncomfortable to physically perform. To solve this limitation, we initially experimented with having players slowly move their Finger Person in the opposite direction of their steps (effectively making the Finger Person perform a "moonwalk"). This didn't allow players to move their in-game character in the backwards direction for long enough. We also attempted to let the player switch directions by having their hand exceed a certain horizontal velocity threshold. This resulted in an unnatural and unreliable

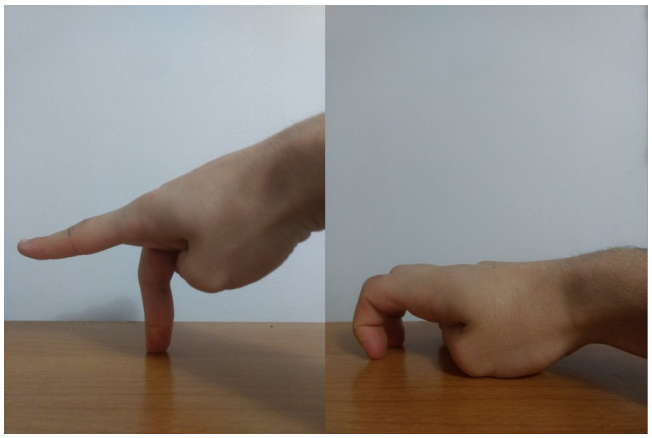

Figure 5: Proposed kick (left) and duck (right) gestures.

motion. We ended up implementing direction change based on the horizontal position of the player's hand. This solution is the more natural than the first-attempted method because the Finger Person walks both forward and backward using in-place walking. This method is more robust than the second method we attempted because it does not require an additional gesture from the user.

FUTURE WORK

Different Hardware Setups

As discussed, the lag caused by the WiFi connection is not feasible for a video game input system. We would like the keep the advantages that using a smartphone offer, so we will continue with our efforts in attempt to use a USB connection to transmit video to Finger People. Another hardware alternative is available in a laptop setup. The user can clip a mirror above the built-in camera, so that the

camera can see the the space to the left or right of the laptop. This will guarantee real-time video transmission and does not require the user to purchase any additional hardware aside from a simple mirror clip. Finally, another alternative input system is to use a webcam. This should also provide real-time transmission speeds, but depending on the shape of the camera, the webcam may be awkward to position so that it can easily see the Finger Person. Because of our system's modular design, all of these alternative hardware setups will be presented as different options to the user, allowing them to choose what best suits their situation.

Additional Gestures

The first extension which Finger People's gesture recognition system can use is the kick gesture. This gesture involves one of the Finger Peron's legs to be raised to be perpendicular to the ground, creating a kick-like motion. Most video game characters in platformers have an attack. The kick gesture would be used to trigger the character's attack – a control which is typically mapped to the B button on a game controller. A final gesture which we find natural

for the hand-character model is the duck gesture. To perform a duck, the Finger Person's legs bend and their upper body (the player's hand) moves towards the ground (Figure 5). This gesture can be used for in-game character to duck, or to navigate into objects such as pipes in Super Mario Bros. The duck gesture would be mapped to what is typically the down arrow on a game controller.

Multi-Hand Gestures

By performing only the basic movement gestures, the user's second hand is free and can be used to emulate more actions such as using a power-up. For example, basic gestures like holding up a number of fingers can emulate less-often used buttons which appear on modern gaming controllers. This would require segmenting one hand from another, which is not too difficult. The difficulty lies is the camera angle – Finger People uses the side-view of the player's hand to observe the Finger Person's leg motion. This angle will naturally have the Finger Person obscure the player's other hand. Setting the camera at a 45-degree angle

Figure 6: Finger People set up with a 45-degree camera angle, showing that you can easily distinguish between the player's Finger Person and other hand.

(Figure 6) or using a second camera are both potential solutions. The former is more desirable because it maintains Finger People's easy-setup and low-cost designs.

User Testing

We have only conducted informal user testing in suboptimal settings. In a setting with no input lag, we would like to further test Finger People and gather empirical data on its performance. Important data would be recordings of instances where Finger People fails to match the user intents. These false negatives could be analyzed to design improvements to Finger People's gesture recognition algorithms. User-tests on the naturalness of Finger People could also be performed by having players sit down with nothing but a screen displaying a 2D side-scrolling platformer, and a camera. The players can be told about the hand-character model, or could even be told nothing, and qualitative data could gathered via observation.

CLOSING NOTES

In our research, we designed and implemented a novel natural gaming input system for 2D side-scrolling platformers. The system currently emulates a controller with an a button and a bi-directional joystick through jump and movement gestures performed in the hand-character model. The system has not yet been tested heavily, but gamers are interested by the novelty, naturalness, and immersion which the input system offers.

REFERENCES

1. D. Brehme, F. Graf, F. Jochum, l Mihailidis, G. Orchard, D. Droege, and D. Paulus. 2006. A Virtual Dance Floor Game using Computer Vision. In *The 3rd European Conference on Visual Media Production (CVMP 2006) - Part of the 2nd Multimedia Conference 2006*, 71–78. https://doi.org/10.1049/cp:20061974
2. Liwei Chan, Chi-Hao Hsieh, Yi-Ling Chen, Shuo Yang, Da-Yuan Huang, Rong-Hao Liang, and Bing-Yu Chen. 2015. Cyclops: Wearable and Single-Piece Full-Body Gesture Input Devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15), 3001–3009. https://doi.org/10.1145/2702123.2702464
3. R. Cutler and M. Turk. 1998. View-based interpretation of real-time optical flow for gesture recognition. In *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, 416–421. https://doi.org/10.1109/AFGR.1998.670984
4. Paul Doliotis, Alexandra Stefan, Christopher McMurrough, David Eckhard, and Vassilis Athitsos. 2011. Comparing Gesture Recognition Accuracy Using Color and Depth Information. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments* (PETRA '11), 20:1–20:7. https://doi.org/10.1145/2141622.2141647
5. Y. Dong, D. Conrad, and G. N. DeSouza. 2011. #x201C;Wii Using Only #x2018;We #x2019; #x201D;: Using background subtraction and human pose recognition to eliminate game controllers. In *2011 IEEE International Conference on Robotics and Automation*, 3887–3892. https://doi.org/10.1109/ICRA.2011.5980310
6. Y. Fang, K. Wang, J. Cheng, and H. Lu. 2007. A Real-Time Hand Gesture Recognition Method. In *2007 IEEE International Conference on Multimedia and Expo*, 995–998. https://doi.org/10.1109/ICME.2007.4284820

7. W. T. Freeman, K. Tanaka, J. Ohta, and K. Kyuma. 1996. Computer vision for computer games. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, 100–105. https://doi.org/10.1109/AFGR.1996.557250
8. J. L. Bernardes Jr, R. Nakamura, and R. Tori. 2009. Design and Implementation of a Flexible Hand Gesture Command Interface for Games Based on Computer Vision. In *2009 VIII Brazilian Symposium on Games and Digital Entertainment*, 64–73. https://doi.org/10.1109/SBGAMES.2009.16
9. Yi Li. 2012. Hand gesture recognition using Kinect. In *2012 IEEE International Conference on Computer Science and Automation Engineering*, 196–199. https://doi.org/10.1109/ICSESS.2012.6269439
10. P. Molchanov, S. Gupta, K. Kim, and J. Kautz. 2015. Hand gesture recognition with 3D convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1–7. https://doi.org/10.1109/CVPRW.2015.7301342
11. Siddharth S. Rautaray and Anupam Agrawal. 2015. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review* 43, 1: 1–54. https://doi.org/10.1007/s10462-012-9356-9
12. Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. 2011. Robust Hand Gesture Recognition with Kinect Sensor. In *Proceedings of the 19th ACM International Conference on Multimedia* (MM '11), 759–760. https://doi.org/10.1145/2072298.2072443
13. J. Suarez and R. R. Murphy. 2012. Hand gesture recognition with depth images: A review. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, 411–417. https://doi.org/10.1109/ROMAN.2012.6343787
14. E Tsironi, P Barros, and S Wermter. 2016. Gesture Recognition with a Convolutional Long Short-Term Memory Recurrent Neural Network. In *Proc. European Symp. Artificial Neural Network*. Retrieved February 26, 2018 from /paper/Gesture-Recognition-with-a-Convolutional-Long-Shor-Tsironi-Barros/8de8e6ab03f13881e590c44966a827805ebfd0bc
15. Juan Pablo Wachs, Mathias Kölsch, Helman Stern, and Yael Edan. 2011. Vision-based Hand-gesture Applications. *Commun. ACM* 54, 2: 60–71. https://doi.org/10.1145/1897816.1897838
16. IPWebcam. https://play.google.com/store/apps/details?id=com.pas.webcam.
17. FCEUX. http://www.fceux.com/web/home.html.
18. xte. https://linux.die.net/man/1/xte.