Assistive Technology for Navigation, Selection, Pointing, and Clicking in a Mouse-free Environment

Shiqi Wu, Sihao Chen, Weili Liu, Frank Cai, Yizhou Wang, Xuantong Liu, Brian Barsky
Department of Electrical Engineering and Computer Science, University of California, Berkeley

***Abstract*—We devise a new solution to computer mouse controlling system using a webcam with computer vision-based methods. By applying computer vision algorithms, we are able to track the hands, recognize the gestures as well as use gesture information to control the mouse. The entire process can operate automatically and free users from physical mouse interaction.**

I. Introduction

Nowadays, the mouse has become a standard input device for computers. We must use mouses or trackpads to interact with computers and send commands. However, some people have trouble controlling a physical mouse. Conditions that can cause difficulties in using computer mouses include impaired sensation from nerve damage, Type II diabetes, spinal cord or head injury, and some other illnesses.

People with conditions such as severe motion impairments have the demand to use computers but may have trouble working with a mouse or a keyboard to control the computer [1]. Unfortunately, most of the very few existing feasible alternative solutions out there are not applicable with consumer-grade computers. Furthermore, supplemental devices might be needed for some of them to work. It's also suggested that individuals with motion impairments tend to prefer camera-based communication interfaces because of the customizable and comfortable nature. Most importantly, those approaches do not require the use of additional accessories that could emphasize users' disability [2].

To help people with difficulties using a traditional physical computer mouse, we have developed a computer vision-based input system with a camera as its input device.

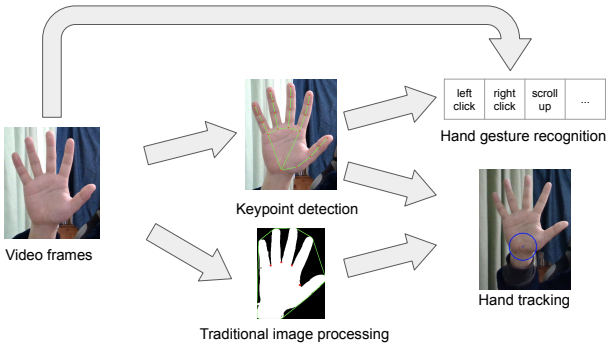

Fig. 1: Breakdown of our methods. 1 is Hand keypoints detection; 2 is Skeleton-based hand gesture classification; 3 is Image-based hand gesture classification; 4 is Image processing; 5 is Hand tracking.

II. Related Works

A. Image Processing

Traditional image processing is a common tool for hand tracking and gesture recognition. To segment hand area from original frame, there are several standard procedures. As in [3] and [4], background subtraction and skin color segmentation can be the first major process. Based on skin color modeling, several methods [5], [6], [7], [8], [9] use skin color to locate related human body parts. With adaptive technology [9], the precision of skin detection could increase by a certain degree. Besides skin segmentation, another important process is face removal. Works like [10], [11] remove face region using Haar-like features, which requires decent background light intensity and computing power. After the above operations, the contour and defects of hands could be found using the OpenCV library [12] since convexity defects [13] are a simple but efficient way to mark hand gestures.

Generally, we would like to find defects and inner circle of given hand so that it becomes much explicit for classifiers to figure out gestures, as shown in figure 2. The capital A to H areas are our convexity defects, while the red circle is our inner circle.

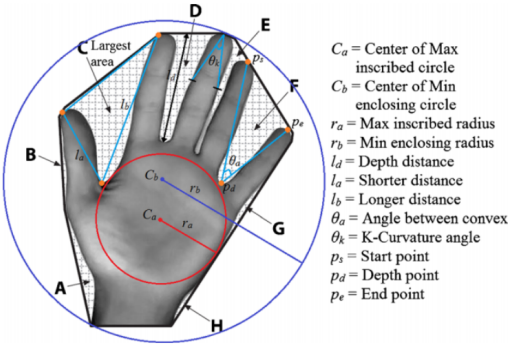

Fig. 2: Hand Contour and Related Properties from Image Processing from [3]

B. Image-Based Hand Gesture Recognition

Recognizing the gestures represented by a sequence of image frames requires large amounts of computations. Thanks to the advance of deep convolutional neural networks (CNNs) [14][15][16] and recurrent neural networks (RNNs) such as Long Short-Term Memory (LSTM)[17] and Gated Recurrent Unit (GRU)[18], new techniques based on large-sized CNN

models tend to achieve high accuracy in gesture classification tasks. Most approaches use either 3D-CNNs or RNNs (LSTMs) to extract temporal information.

3D-CNN-based models perform convolutional operations along the time axis, which enables them to learn rich temporal information. [19] achieved state-of-the-art performance by using a lightweight CNN model to detect gestures and a deep 3D-CNN model to classify the detected gestures. The ResNeXt-101 model, which is used as a classifier, achieved the state-of-the-art offline classification accuracy of 94.04% on the EgoGesture dataset[20].

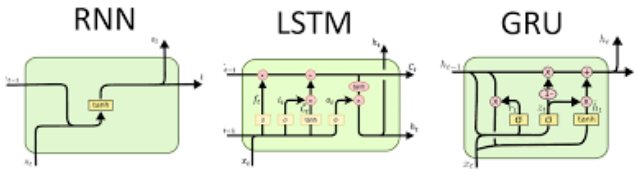

Fig. 3: Sequential models. Based on classic RNNs (on the left), long-short term memory (LSTM, on the middle) and gate recurrent units (GRU, on the right) are developed to improve models' performance on more complicated sequences.

Videos are basically data with sequential structure. Therefore, sequential models can be applied to multi-frame gesture recognition. Recurrent neural networks are models specially designed for sequential data analysis. Long short term memory networks (LSTM) designed based on traditional RNN significantly improves models' long time feature extraction and can be applied into gesture recognition tasks. LSTM based model performs well when the related information locates relatively far in the gesture sequence. [21] introduces LSTM to the CNN structure and significantly improves the model's ability to analyze dynamic gesture's information, as [22] introduce a new way of training models of gesture recognition, which sample the frame sequences in different time scales. For instance, they downsample the video by 2, 4, 5, and 10 times. this helps the model learn features varying in different ranges of time. Plus, this improves the model's ability to infer gestures' categories with only a few frames.

C. Skeleton-Based Hand Gesture Recognition

Extracting human pose information from images requires a large amount of computation. By contrast, coordinates of body keypoints are a more compact and meaningful representation. However, extracting keypoints from camera outputs is non-trivial and often imprecise. Thanks to the advance of depth sensors like Microsoft Kinect and Intel RealSense and modern keypoint detection approaches [23], large amounts of precise keypoint data is available to researchers nowadays. Skeleton-based hand gesture recognition approaches [24][25][26] use coordinates of pre-defined hand keypoints as input and output labels of hand gestures. These approaches have much smaller model sizes and require much less computation compared to image-based approaches. De Smedt et al.'s approach [24] used a temporal pyramid to obtain the feature vectors from a multi-level representation of Fisher Vectors and other skeleton-based geometric features. It then completed the classification task using a linear SVM classifier. [25] proposed a model that combines a Convolutional Neural Network (CNN) and a Long-Short Term Memory (LSTM) recurrent network for handling time series of 3D coordinates of skeleton keypoints. [26] proposed a Double-feature Double-motion Network (DD-Net) for skeleton-based action recognition, which uses a Joint Collection Distances (JCD) feature and a two-scale global motion feature.

[27] presented a similar method. First, the system detects the presence of a human body in the webcam video and extracts certain keypoints from it using [28]'s OpenPose framework, as shown in Figure 4. After preprocessing and normalizing these keypoints, the system then performs a 1-Nearest Neighbor classifier with dynamic time warping as distance metrics to recognize what gestures the user is performing. The overall accuracy that the paper managed to achieve was about 77%. This is by no means impressive, but the main advantage of it is that this system is highly flexible and can run on almost any machine, and it requires very little data to set up.

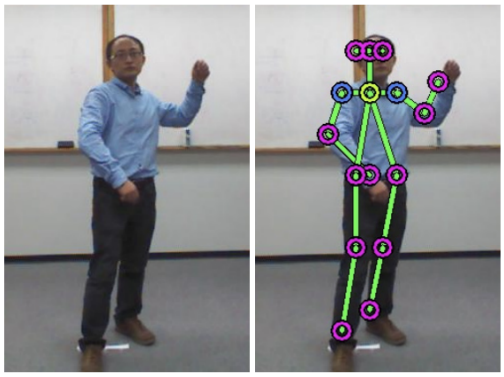

Fig. 4: Openpose Body Keypoints Extraction (Image source: [28])

III. Methods

The functionality of our project could be divided into two parts: gesture classification and hand tracking. We have tried different techniques for each part. As in figure 1, our techniques can be classified into five categories: hand keypoints detection, skeleton-based hand gesture classification, image-based hand gesture classification, image processing, and hand tracking.

After obtaining the video stream from a camera, we use a hand keypoint generation tool, such as Mediapipe [29], to extract pre-defined keypoints (connections) of user's hand. Then, we use the coordinates of keypoints as the input to our hand gesture classifier, which outputs the label of the gesture shown in the video, e.g. left click. An alternative way to do this is using image-based hand gesture classification, which almost always includes a Convolutional Neural Network (CNN) [30], to classify gesture from the video stream without extracting keypoints.

Meantime, we perform image processing techniques, such as hand segmentation, on the video stream. This step will output the location of the palm. We finally use a cursor controller to move the cursor based on the location of the hand.

Portability is the main factor we consider in our project. The program should be able to run on a regular computer without a GPU at a high frame rate. So, we avoid complex deep neural networks and evaluate our models by both accuracy and runtime.

A. Hand Keypoint Detection

Since our system design relies heavily on hand keypoints, we paid particular attention to the choice of keypoint detection framework. Two commonly used ones were investigated, namely Google's Mediapipe [29] and CMU's Openpose [28]. Table I outlines the major differences between them.

We tested both frameworks on a computer with an Intel 6th generation duo-core processor and 16 gb of memory. This computer is chosen because we want to imitate the average device that our target audiences have and test the two framework's performances on this specific condition. We conclude that Mediapipe is the clear winner in this case. It managed to achieve about 30 frames per second with some occasional drop in frame rate, while Openpose only managed to get as high as 5 frames per second. Mediapipe is also much easier to use as it comes with a highly optimized and constantly updated python package so that we do not need to manually load the model as we did with Openpose. While both frameworks output 21 keypoints on the hand, Mediapipe offers additional depth information on the keypoints, which is beneficial to our gesture recognition model since some gestures rely on this information.

	Mediapipe	Openpose
FPS	High	Low
Ease of Use	High (Python Package)	Low
Support	High	Low
Academic	No	Yes
Keypoints	21	21
Dimension	3D	2D

TABLE I: Comparison of Mediapipe and Openpose on Hand Keypoint Detection

B. Skeleton-Based Hand Gesture Recognition

Gesture recognition is the bottleneck of our system's performance. Image-based hand gesture recognition methods often use deep neural networks such as ResNet [16], which contains millions of parameters and cannot run on a CPU at a high frame rate. Skeleton-based hand gesture recognition has two advantages over image-based approaches: First, the location of hand could be computed from keypoints which could be used for hand tracking, and thus we do not need a separate palm detector. Second, the skeleton data has much fewer dimensions than image data, and its information can be extracted by shallow neural networks with lower computational cost compared to CNN on images.

The procedure of skeleton-based hand gesture recognition is as follows: First, we use a hand keypoint detector to extract the skeleton of the user's hand from a video stream. Then, we pre-process the coordinates of keypoints and generate features. Finally, we use the feature as the input to a classifier which maps skeleton data to the probability of each output label, i.e., gesture.

Our hand keypoint detector has the same structure as MediaPipe Hands, a real-time on-device hand tracking tool [23]. It includes a palm detector and a hand landmark model. The former operates on a full input image and locates palms via an oriented hand bounding box, while the latter operates on the cropped hand bounding box provided by the palm detector and returns 21 landmarks consisting of x, y, and relative depth.

Although neural networks complex enough are able to learn latent representations in the coordinates for gesture classification tasks, past research finds that handmade features are useful for gesture classification. In our experiments, we found that introducing certain features could effectively improve the classification accuracy without increasing the inference time. For each video frame i, we extract feature vector f_i, which consists of:

1) *distances* between every pair of keypoints
2) *angles* between every pair of keypoint connections

For each pair of keypoints a and b, we calculate their distance using the norm of the vector connecting them,

$$distance_{ab} = \left\| \overrightarrow{l}_{ab} \right\|_2 . \tag{1}$$

We define a keypoint connection as a vector pointing from one keypoint to another, where the keypoints are connected in our graph. For every two connections $\overrightarrow{l}_{ab}$ and $\overrightarrow{l}_{cd}$, we then calculate the angle between them as follows,

$$angle_{\overrightarrow{l}_{ab}, \overrightarrow{l}_{cd}} = \arccos \frac{\overrightarrow{l}_{ab}^{\mathsf{T}} \overrightarrow{l}_{cd}}{\left\| \overrightarrow{l}_{ab} \right\|_2 \left\| \overrightarrow{l}_{cd} \right\|_2} \tag{2}$$

In order to keep the inference time short on CPUs, we use a simple RNN-based neural network to classify hand gestures using these features. We designed and tested two approaches to use the feature sequences $f_0, f_1, \cdots, f_n$: sliding window and stateful RNN. In the sliding window approach, as shown in Figure 5, we simply use the features in the last k frames as the input and reset the hidden states for every frame. In the stateful RNN approach, as shown in Figure 6, we keep the hidden states and only feed the features in the last frame into the network. The sliding window approach has better accuracy in general, but its cost is higher by *sequence length* times. We found that the model consisting of a single GRU cell and a softmax layer gives decent and robust classification accuracy in both settings.

In additional to deep-learning approaches, we also experimented with a few traditional machine learning algorithms. The intuition behind this is that machine learning algorithms generally require much less computational power than deep learning, which is a huge consideration in our system design,

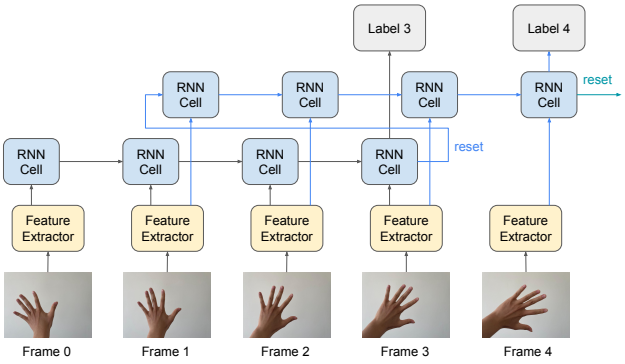

Fig. 5: The sliding window approach processing 5 frames (sequence length is 4)

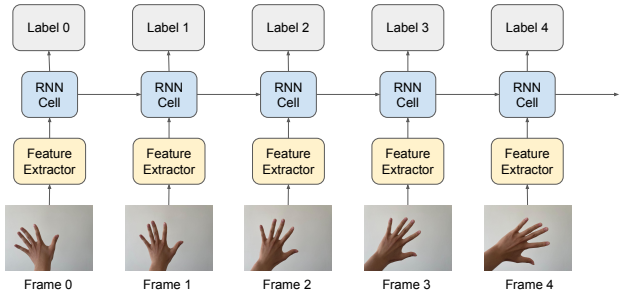

Fig. 6: The stateful RNN approach processing 5 frames

and it offers higher interpretability. Table II shows the tradeoffs among the three machine learning algorithms we tested. We only looked at supervised learning algorithms because labels are readily available in the Egogesture dataset and unsupervised learning algorithms generally have much poorer performance than supervised algorithms in gesture classification.

k-Nearest Neighbor(kNN) is a simple and intuitive algorithm with very high interpretability. It finds the k most similar training instances to the input in terms of a distance metric and returns the most popular label in these k instances. The advantage of using such an algorithm is that it does not involve any actual training. Therefore, it usually serves as a good starting point for a classification problem and is used as a baseline for more complex algorithms. We believe kNN is suitable for our use case because same gestures have similar spatial arrangement in an image and in turn low distances between corresponding keypoints after proper normalization. However, kNN's inference time scales with the size of the dataset because it needs to find all k neighbors and is expected to offer poor real-time performance.

Random Forest(RF) offers good performance in classification problems involving categorical data and provides additional regularization over decision trees. The algorithm combines a set of decision trees and trains each tree with different sets of data. For each tree, the algorithm builds the tree by doing the split that provides the highest information gain. In other words, random forest allows us to see which features are more important than others in making predictions. However, having to build numerous decision trees means that it requires a long training time compared to other algorithms tested in this paper.

Unlike kNN and RF, support vector machine(SVM) is a linear classification algorithm and can only provide a linear decision boundary without further tweaking. Essentially, SVM tries to find a hyperplane with equation 3 and determines the w and b that will minimize the hinge loss. Gradient descent is the technique we employed in order to achieve minimal loss. Although SVM does not seem to offer any clear advantage over other algorithms, inferencing using SVM is extremely time-efficient since it only involves two vector operation. It also does not require much memory since all it needs to store are the w and b vectors, making it effective in real time on low-end computers.

$$w^T x + b = 0 \tag{3}$$

Model	Pros	Cons
k-Nearest Neighbor	High interpretability, easy to implement, no training time	Poor scalability, sensitive to outliers
Random Forest	Regularization (Reduce overfitting), works well with categorical data	Low interpretability, long training time
Support Vector Machine	Fast inference time, memory efficient	poor in distinguishing similar gestures, not suitable for large dataset

TABLE II: Tradeoffs among Different Machine Learning Algorithms

Data preprocessing is performed before each of the above mentioned 3 models is trained. Normalization is required because the detected hands in each data instance can have different translation, scale and rotation. Since we only care about the relative position between each keypoint pairs, we need to eliminate the effect caused by these three transformations. Therefore, we performed the normalization procedure on our data as shown in figure 7 before feeding them into the models. After extracing the 21 3D-keypoint coordinates from the image, we will first perform translation, so that keypoint 0 is at the origin (0, 0). Then, we calculate the vector formed by keypoint 0 and keypoint 9; we will call it $v1$ in subsequent discussions. Then, we formulate the rotation matrix needed to align $v1$ and the y-axis by determining the angle between $v1$ and the y-axis. Lastly, we scale all the keypoints so that the maximum coordinates a point can have is 1.

C. Image-Based Hand Gesture Recognition

Image-based method aims at using an end-to-end model to obtain the accurate category information of a gesture. As the basis of multi-frame gesture recognition, we firstly conduct research in single frame gesture recognition using deep models. In our single-frame gesture recognition part,

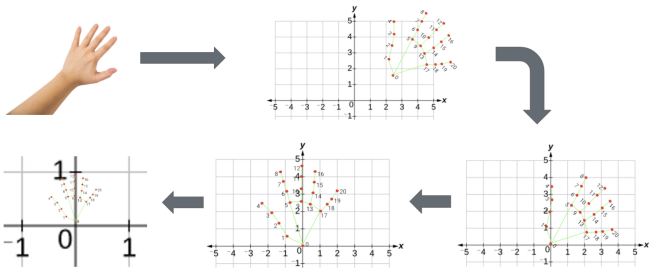

Fig. 7: Data Preprocessing Procedure

we apply single-shot detection (SSD) to detect hands in the scene and then crop the bounding boxes out. SSD network is a popular object detection framework that perform well in both accuracy and speed. It uses the VGG backbone to extract image feature and merge features from different feature map sizes. Thus, it can perform well when objects in the image change in a significant range.

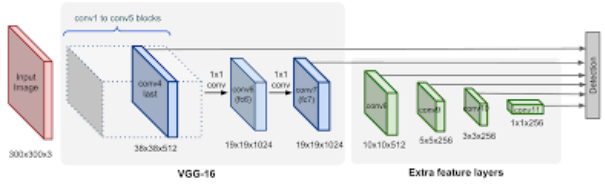

Fig. 8: Single-shot object detection network

Afterward, we design a convolutional neural network decribed in Table III to classify the gesture that the hand is performing in the bounding boxes.

Layer name	Parameters or Settings	Feature map
Conv 1	(3,3,32,0,1)	(198, 198, 32)
Activation 1	ReLU	(198, 198, 32)
Conv 2	(3,3,32,0,1)	(196, 196, 32)
Activation 2	ReLU	(196, 196, 32)
Maxpooling 1	(2,2)	(98, 98, 32)
Dropout 1	0.5	(98, 98, 32)
Flatten 3	None	(98 * 98 * 32)
Dense 1	None	(1024)
Activation 3	ReLU	(1024)
Dropout 2	0.5	(1024)
Dense 2	None	(128)
Activation 4	Softmax	(10)

TABLE III: Self-designed convolutional classifier structure. For convolutional layers, (3,3,32,0,1) refers to kernel size of (3,3), stride 1 and padding 0. For maxpooling layers, (2,2) refers to pooling size. For dropout layers, 0.5 refers to drop probability.

Single frame-based gesture recognition can make sense under some particular circumstances. However, most gestures can only be expressed with a sequence of frames. The single frame model cannot handle this problem. Thus, we need to develop models which can handle multi-frame gestures.

In our multi-frame gesture recognition research, we mainly focus on two models with similar thoughts. They are multi-timescale model devised from temporal relation networks (TRN) [31] and frame-segment models devised from multi-frame fusion models (MFF) [32]. TRN is a network which samples the gesture videos into different time scales. For instance, we sample the video with different frame numbers (2, 5, 10, etc.). Afterward, different sample groups are fed into different convolutional neural networks to generate feature vectors. In the final step, all vectors are fused together to obtain a global representative of the gesture information and a multi-layer preceptron (MLP) is implemented to complete the category regression. The structure of multi-timescale can be shown in figure 9.

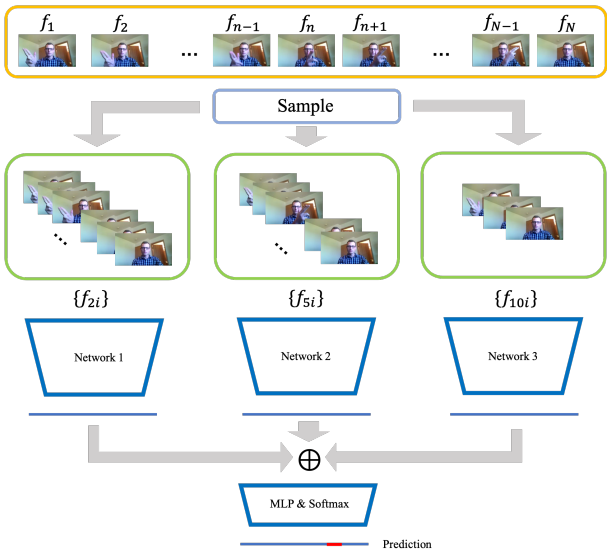

Fig. 9: Multi-timescale gesture recognition model

frame-segmentation model's structure is quite similar to the multi-timescale model. It also samples the different areas of a gesture video. It difference from TRN can be shown in figure 10. It split the video directly into segments with the sequence of the time. After each segment's feature was extracted by convolutional neural networks, these features are concatenated into a whole representation of the original video. It also implemented optical flow computation to help analyze the motion of the gestures.

D. Hand Tracking Based on Image Processing

Since we cannot expect all of our potential users to have a fine GPU to perform deep learning methods, traditional methods based on image processing are as well needed. To combine works together, image processing is used to locate the hand area and provide it as an input to gesture classifiers. In general, we think it would be much easier for classifiers to utilize explicit hand images without any noise or background information.

We divide our implementation into two major parts: Background Subtraction and Contour Finding.

1) Background Subtraction: Our implementation defines background subtraction as a combination of skin-color segmentation, face removal, morphology operations and Gaussian filter.

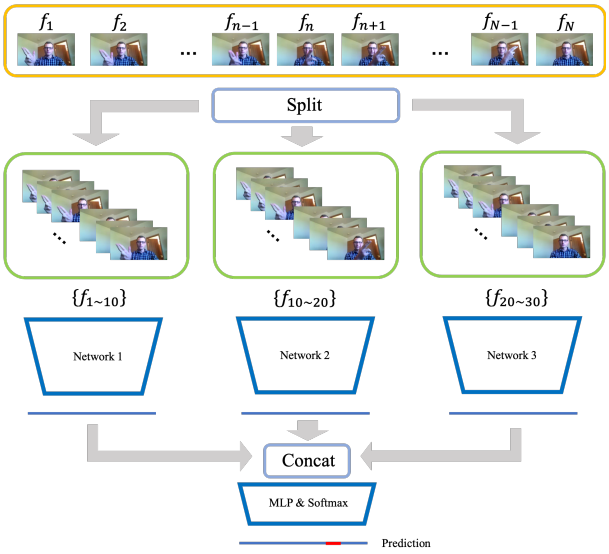

Fig. 10: Frame segmentation gesture recognition model

Skin-Color Segmentation: Unlike RGB channel, researchers find the YCrCb and HSV channels are more efficient to differentiate skin color from various image background colors. Based on our tests, we choose the below ranges for HSV and YCrCb values so that we could best extract skin regions, as in Figure 11.

$$54 \leq Y \leq 163, 133 \leq Cr \leq 173, 77 \leq Cr \leq 127$$

$$0 \leq H \leq 33, 58 \leq S \leq 255, 25 \leq V \leq 255$$

$$H, S, V, Y, Cr, Cb \in [0, 255]$$

However, those ranges could not cover all kinds of skin colors, such as people with darker skin. Another issue is this color space could contain many skin-like background information like furniture or floor. Thus, it requires a high contrast environment which should be different from human skin color. To maintain a more precise controller for HSV and YCrCb color space, we design a track bar for all six values so that users could adjust proper thresholds at the initialization, as shown in Figure 12.

Fig. 11: Skin Segmentation

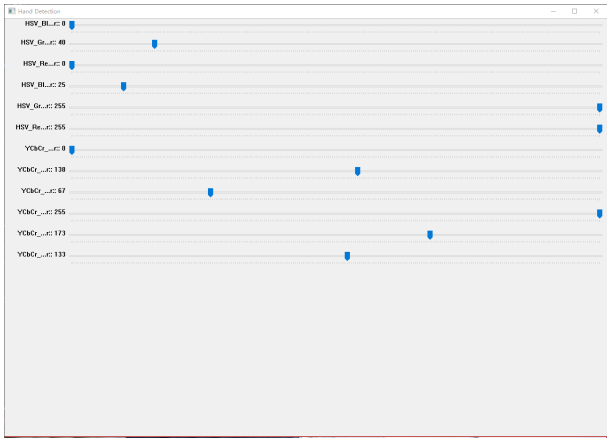

Fig. 12: Track Bars

Face Removal: Since the skin color segmentation extracts both hand and face area, our system needs to differentiate the hand and the face region so that it could further locate hand area. Based on previous researches, we decide to use Haar-like features to figure out which area is the face and therefore remove it by simply blocking out. Fortunately, OpenCV library provides us with a well-prepared Haar-like XML so that we could simply utilize it as our face recognition library. This XML file is efficient and well-performed for other methods to subtract face area in most conditions. Figure 13 shows a clear face blocking effect after face removal.

Fig. 13: Face Removal

Smooth Methods: After skin segmentation and face removal, there are still some noises around our frame. To eliminate them as much as possible, we implement morphology operations and Gaussian Filter to acquire a clean hand area. In this project, we apply Open Operator, which means Erosion first and then Dilation. The Gaussian Filter helps us to blur our frame a bit so that edge noises such as boundaries could be eliminated.

2) Contour Finding: From our first major step, we could acquire a binary image frame containing hand area with white color and black color for background information. That large white area, which is supposed as our hand area, should be our ultimate goal. We use *contour* from the OpenCV library, an array of points representing a hand curve, to locate hand region. Since it is common to finalize multiple contours from

the previous binary frame. We compute the weighted average of contour center and area to determine which one is hand center. Obviously, the largest area is the ideal hand location.

Figure 14 shows a related hand contour we find. We use lines between the star and far points of each defect to include the whole hand area, while those red points are the valley area of each defect.

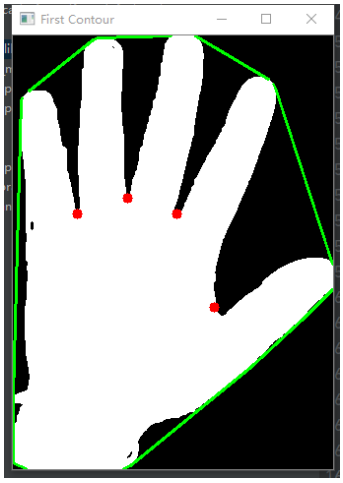

Fig. 14: Hand Contour and Defects from Image Processing

Due to the hand contour, we could simply detect hand gestures based on the defects of that contour. For example, we consider two gestures: open and closed. If those defects of the hand contour are well-organized, which means endpoints are behind start points and far points, we consider that situation as an open gesture. We could as well use the center of the inner circle to move our cursor. Figure 15 shows a demo of gesture recognition and the center of the inner circle.

Fig. 15: Gesture Recognition and Center of Inner Circle

E. Mouse Control

1) Basic Cursor Control Theory: Our mouse control takes information from the sequences of frames from MediaPipe, performs calculations on hand key points, and then moves the mouse to a corresponding position. We have tried three different mouse control methods, including absolute mouse control, relative mouse control, and joystick mouse control.

The absolute mouse control is the simplest and most intuitive one among the three methods. It aggregates the hand key point info to a single coordinate point on the screen. It gets the width and length of the screen, and the x, y coordinates the hand's center and moves the mouse to the proportional position. The cursor is at the place of the hand, and it moves with the hand movements. Please find an example of the absolute mouse control method attached below. In Figure 16, we can see that the cursor is around the center of the hand. In order to move the cursor, we just change the position of our hands. Although this method is easy to get started with for beginners, it has some shortcomings. For example, it is susceptible to noise. The user's hand might have some unconscious shakes during the usage of the assistive mouse. Then, the cursor will also move because of these shakes. In addition, this method may include many hand movements. Suppose the user wants to move the cursor from the left to the right of the screen; he or she may act a large wave of the hand. Besides, the result of the mouse control is also affected by the camera resolution. The conversion of the position of hand and cursor may be inaccurate if the resolution of the camera is low.

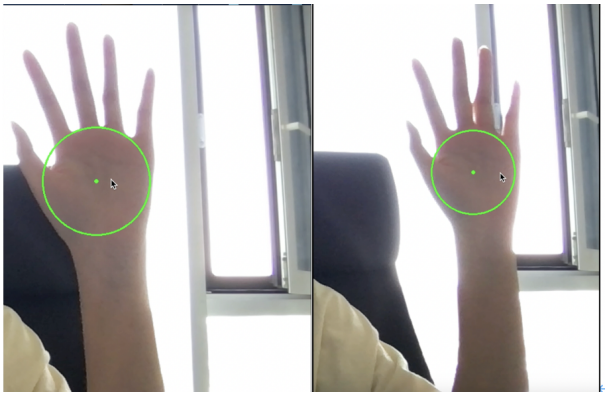

Fig. 16: Example of Absolute Mouse Control

The relative control method gets motivation from the current mouses and trackpads. It gets inspiration from the current mouses and trackpads. Unlike the absolute mouse control, the mouse cursor's position is not necessarily the position of the hand on the screen. The mouse moves faster if the hand's position changes a lot. The relative mouse control keeps track of the previous position of the hand and calculates the distance between the last center and the current center for the direction and acceleration of hand movement. Then, we move the cursor to the relative position to its previous position. We use a deque with length 5 to keep track of the center position and for noise reduction. We push the new center to the deque after every cursor movement. The relative mouse control is less sensitive to noise than the absolute mouse control. It is the approach that is being used in similar commercial products, such as the built-in accessibility features of Mac OS and Windows. In addition, this method requires additional parameter tuning compared with the absolute mouse control.

Our third method is the joystick mouse control. It works like a real joystick. We control the mouse with a joystick by moving our hand in different directions with different distances (acting as forces) from the center. There is a fixed point we set on the screen used as the center of the joystick. This method calculates the hand position's angle and magnitude with the center to decide the mouse movement's direction and speed. Suppose if our hand is on the right-hand side of the center, then the mouse cursor will move to the right. In Figure 17, the center we set is the starting point of the blue line. The cursor can keep moving right in the same direction if the user's hand stays in the same place on the right of the center. If we want the cursor to move faster but in the same direction, we just move our hand farther from the joystick center. Figure 18 is an example showing the mouse moving in the upper-left direction. When the user's hand is in the upper-left position of the center, then the mouse will start to move in the same direction on its current location; the movement speed depends on the user's hand's distance from the center. In addition, for noise reduction, there is a dead zone area around the center we set. If the resulting coordinate is within the area, then there will be no movement of the cursor. This method is less susceptible to noise. In addition, we do not need many hand movements by using this method if we want to move the cursor in one direction continuously. However, the joystick mouse control is not very intuitive for users to get started with.

Fig. 17: Example of Joystick Mouse Control

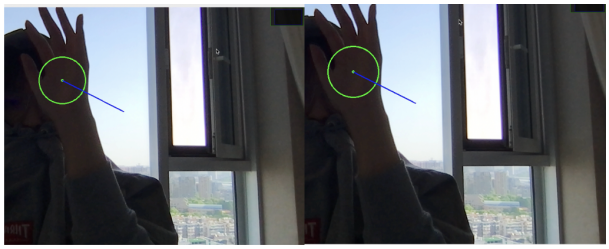

Fig. 18: Example of Joystick Mouse Control

Besides, we use some extra steps to reduce the effects of handshakes on mouse movement further. We have applied two filters on the center of the palms, the sliding window and the Kalman filter. In the sliding window method, we keep track of the past five frames in a queue and return the average of them as the center used for the next hand movement. In the Kalman filtering method, we track and predict hand movements according to their previous locations. This method makes use of many linear algebra techniques. The equation in Kalman filtering is divided into prediction and correction equations. It calculates the state, computes its noise, and uses them to predict later hand movements.

2) Pinch Gesture: Inspired from [33] and MediaPipe key-point, pinch gesture is another procedure we apply to maintain a stable cursor control. Similar to daily human gestures, pinch gesture is one of humans' common pickup operations. By bringing the thumb and index finger together, we apply a method for small range viewing situations to avoid complex and fragile hand gesture classifiers. By detecting the distance between two fingertips, we could simply find out if the pinch gesture is formed or not. We maintain a state machine to transform from pinch to no pinch or no pinch to pinch. If two fingertips are closer enough, as in Figure 19, our system would decide it as a pinch gesture. While in a pinch gesture shown in Figure 20, when two fingertips are far away, the system would consider it as a gesture transform from pinch to none pinch. Therefore, when forming a pinch gesture, the system would start cursor control. As shown in Figure 19, the user forms a pinch gesture so that the system start to trace the centroid of that hole. To prevent unable to visit the boundaries of screen, we form a blue rectangle in the frame to represent the screen area. When the centroid point is in the right upper of the blue rectangle, the cursor appears as well in the right upper location of the computer screen.

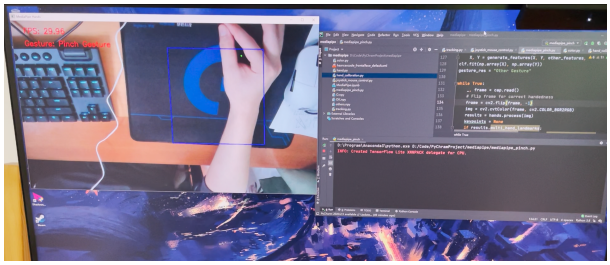

Fig. 19: Pinch Gesture

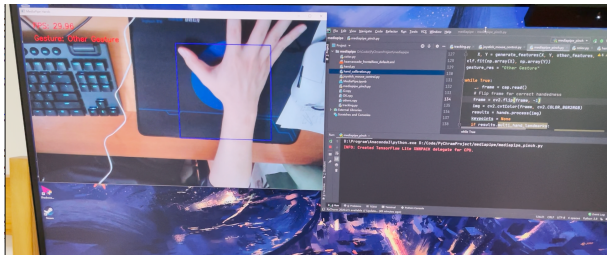

Fig. 20: Pinch Gesture

The pinch grasp is simple and precise. Therefore, re-opening and re-closing could be accurately detected by our state machine. When the system detects a re-opening and re-closing operations from hand, it would trigger a mouse-down event, as shown in Figure 21. When triggering a mouse-down event,

once users open their thumb and index finger again, the system would trigger a mouse-up event. Hence, it is succinct for us to apply the click operations (click and double-click) and the dragging operation.

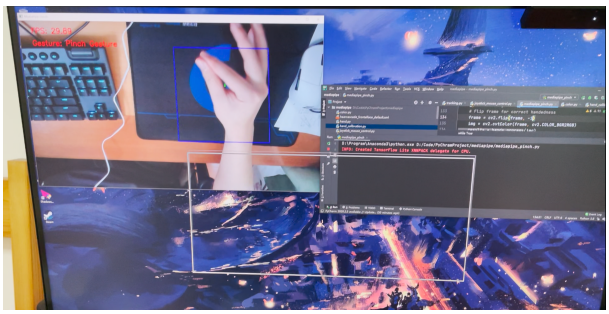

Fig. 21: Dragging Operation

IV. Experiment Results

A. Hand Tracking Based on Image Processing

Based on image processing, the cursor control system could use the palm center or center of the inner circle as the cursor position to move the mouse. Unlike official mouse control testing, we test this traditional OpenCV works in a related simple but succinct way.

1) Benchmark: Since in this project we hold up our hands in mid-air and use them to control our mouses, we cannot avoid the hand shaking and other body trembles. To best test this part, we decide to draw two vertical and horizontal lines in drawing applications in Windows so that we could find how much the reference image is covered by our hand movements. In detail, the benchmark is shown in Figure 22, while the drawing result is shown in Figure 23 .

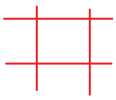

Fig. 22: Reference Benchmark

2) Test Results: Comparing basic physic mouse drawing and our hand tracking drawing, we generate Table IV to present our tracking accuracy. As mentioned above, we need more time to move to a precise location due to the hand shaking. The time consuming is relatively much higher than a common computer mouse.

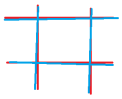

Fig. 23: Drawing Result

Test Type	Image Processing Method	Physic Mouse
Accuracy	99.16%	94.28%
Time	6.47s	19.89s

TABLE IV: Test for accuracy and time consuming for normal mouse and image processing method

B. Skeleton-Based Hand Gesture Recognition

To train such a classifier, we use EgoGesture, a dataset containing more than 24,000 gesture samples and 3,000,000 frames for both color and depth modalities from 50 distinct subjects [20]. To make the data format fit our problem setting, we use a subset of videos performing 11 static gestures and only use the RGB frames of the videos. The gestures we selected are fist and ten gestures representing digits from 0 to 9.

Using the coordinates as well as generated features as input, we train classification models using machine learning. We then evaluate our model on a validation set which we sliced from EgoGesture dataset. We tested single-layer networks, including Recurrent Neural Network, Long Short-Term Memory, Gated Recurrent Unit, and Multilayer Perceptron, that processes concatenated features along the time dimension. We also did ablation tests on our generated features by adding or removing individual features to show that using these features could improve the overall accuracy. We choose the sequence length to be 5 and the frame rate of input to be 10 FPS. The results of the sliding window models are shown in Table V, while the results of the stateful RNN models are shown in Table VI. Recurrent neural network using a single layer of GRU reaches the highest accuracy of 83.25% with the sequence to sequence models and 77.66% with the stateful RNN models. Note that although LSTM and GRU have similar accuracy in multiple experiments, GRU has a simpler structure and gives much smaller latency. So, we choose GRU as our final model. We also found that most false predictions happen when our keypoint detector cannot detect keypoints precisely.

Although our model's accuracy is lower than the state-of-the-art approach's [19] (93.75%), the model is much simpler and enables the whole program to run on CPU with an acceptable latency. The frame rate is 10 FPS on a MacBook Pro with an Intel Core i9 CPU, whereas [19]'s approach uses 3D-CNN based on ResNeXt-101, which runs at a frame rate

lower than 3 FPS.

Model	Coordinates only	+distances	+angles	+distances+angles
MLP	78.37%	81.37%	80.94%	81.31%
RNN	77.99%	81.46%	80.45%	81.48%
LSTM	81.18%	82.83%	82.18%	82.93%
GRU	81.73%	82.21%	81.95%	**83.25**%

TABLE V: Cross validation accuracy of the sliding window models on 11 gestures from EgoGesture dataset

Model	Coordinates only	+distances	+angles	+distances+angles
RNN	71.23%	75.03%	74.03%	75.58%
LSTM	73.51%	77.16%	74.77%	77.10%
GRU	73.65%	76.67%	75.63%	**77.66**%

TABLE VI: Cross validation accuracy of the stateful RNN models on 11 gestures from EgoGesture dataset

We also present the results obtained from kNN, RF and SVM in table VIII. In order to obtain the best hyperparameters for each model with the exception of SVM (no hyperparameters to be tuned), we perform a 10-fold cross-validation and keep the model that outputs the highest cross-validated test accuracy. Table VII lists the best parameters for each model with their cross-validated accuracy. The real-time performances of these models are reasonably good, with an average fps of 20 to 30. The model inferences cause only a very minimal slow down of the frame rate.

Model	Parameters	Score
kNN	k=6, euclidean distance	68.7%
RF	max tree depth = 23, number of trees = 200	70.5%

TABLE VII: Best-performing Parameters with Corresponding Cross-validated Accuracy for Each ML Model

Model	Accuracy
K-Nearest Neighbor	69.4%
Random Forest	71.0%
Support Vector Machine	71.7%

TABLE VIII: Test accuracy of three machine learning algorithms on Egogesture dataset

C. Image-Based Hand Gesture Recognition

We initially test our single frame gesture classification performance. The models of SSD and CNN classifier are separately trained and tested jointly. The SSD detection model can have an accuracy of 0.893 on the EgoGesture Dataset, and the classification network can have an accuracy of 0.923 on a static gesture dataset collected by ourselves (including Stop, Punch, OK gestures, totally 10 categories).

Both the multi-timescale model and the frame segmentation model are trained on another dataset named Jester-v1, which contains 140000 videos and in total 27 gesture categories. We train models with two different CNN backbones, ResNet and Inception-V3. The performance can are shown in Table IX and we can see that the model's classification result is acceptable. To be specific, frame-segmentation is generally better than multi-timescale model. In terms of feature extraction backbones, Inception-V3 performs better than ResNet. As for the inference time of the model, when processing a 240-frame dynamic gesture video, we firstly introduce a sample step to obtain a 30-frame video and obtain a 7 second forward time. since 240-frame video take over 7 seconds, it is safe to say that we can achieve real-time recognition.

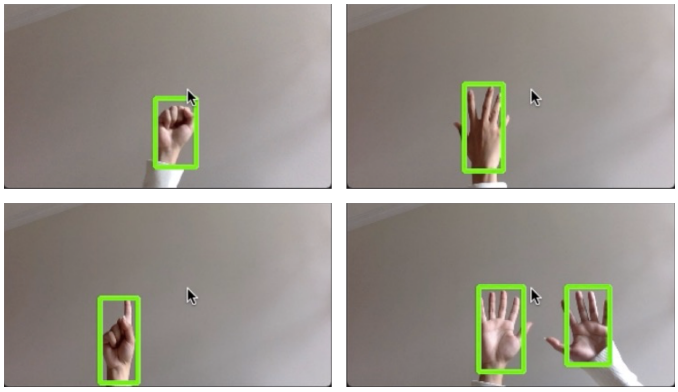

Fig. 24: Single frame gesture recognition. Based on Single-shot detection backbone and self-designed model, the model can handle a range of different gesture categories, and multi-hand gesture recognition is also supported.

Framework	Multi-timescale	Frame segmentation
ResNet	93.67%	94.40%
Inception-V3	94.67%	95.22%

TABLE IX: Accuracy on Jester-v1 dataset of multi-timescale model and frame segmentation model

From the experiments we can find out that the models can perform well on large datasets with an acceptable inference time for real-time recognition requirements.

In conclusion, single-frame-based model can handle some simple gestures (Stop, Punch, OK, etc.). However, it fails to predict multi-frame gestures both theoretically and practically. Therefore, we introduce two models to process gesture videos. When trained with a large dataset, dynamic gesture recognition also reaches good performance. Plus, dynamic gesture models can handle more gesture categories.

D. Mouse Control

1) Basic Cursor Control: We performed a mouse movement and clicking accuracy test on an online mouse accuracy testing tool. The tool randomly generates multiple circles that allow users to click on. It keeps track of the number of clicks and the accuracy rate in the duration of 30 seconds. We experimented on each mouse control method three times and took the average value. In comparison, we also performed the same test on the Mac OS head pointer, which is also an alternate control method that helps those with trouble using a physical mouse. The results of each method are shown in Table X.

2) Pinch Gesture: Since our pinch gesture apples the same cursor control method as the previous part. The accuracy is

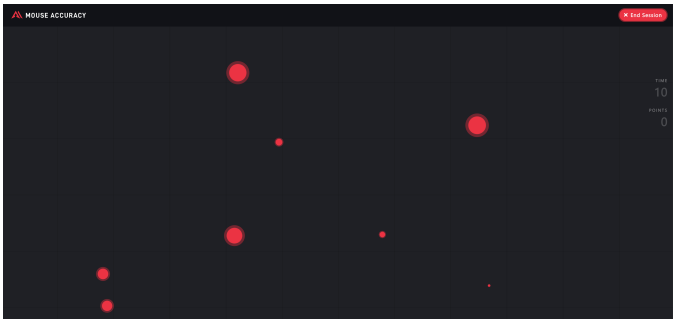

Fig. 25: The randomly generated circles can be clicked on to evaluate the speed and accuracy of each mouse control method

Method	# of Clicks in 30s	Accuracy Rate
Absolute	12	55%
Relative	13	67%
Joystick	9	43%
Mac OS head pointer	10	47%

TABLE X: Number of clicks and cursor accuracy for each mouse control method

similar. For this part, we only test the speed and accuracy of the dragging operation. We find ten users to test our pinch gesture ten times to select their desired application icons in desktop and then move those icons to their destinations, which is shown in Table XI. Generally, the accuracy is satisfied, while the speed needs improvement since users send feedback about speeding too much time aiming at objects.

Test Type	Physic Mouse	Pinch Gesture
Accuracy	100%	96.06%
Time	3.67s	16.03s

TABLE XI: Test for accuracy and speed of dragging operation for normal mouse and pinch gesture

V. Conclusion

In conclusion, our team worked on building a mouse control assistive system that allows users to perform mouse cursor operations without interacting with a physical mouse. In order to use the assistive mouse, the users need to be able to move their hands and manipulate their fingers.

To track the hand and classify gestures, we have implemented several methods:

- Traditional methods based on image processing could satisfy our requirements in low-cost devices.
- An end-to-end deep learning models which process the images directly. Single-shot detection based pipeline can achieve one-image gesture recognition task. For more complex dynamic gestures, we devise multi-timescale models and frame-splitting models, which perform really well on large public datasets.
- An alternative learning model making use of keypoints generated from Mediapipe as input.

We have tried three mouse control methods, including absolute mouse control, relative mouse control, and joystick mouse control. The absolute mouse control is the most intuitive one. The relative mouse control provides us the best accuracy after tests, while the joystick mouse control includes the fewest hand movements.

VI. Future Works

Our implementation still suffers from several limitations. Since our project is composed of several different methods, we aim at different aspects to further improve the system stability and the user experience.

For image processing, ongoing progress would be a more robust extracting method for the hand region. The current method cannot handle the dim light environment. Another problem is about face removal. If the hand overlaps too much over the face, it would cause a tracking false.

The current model to extract keypoints is based on deep learning, which has a high cost and does not work well for certain hand gestures. Future works should consider using other methods to extract hand keypoints in an image.

For mouse control, we will look into improving and stabilizing those mouse control approaches. We will also apply more anti-shake mechanisms.

The gestures we used are from publicly available gesture datasets. Hence, they can feel unintuitive to use when mapped to mouse commands. Future work can expand this by using a custom-generated dataset and make the gestures feel more natural.

Acknowledgement

We would like to acknowledge our faculty advisor, Professor Brian Barsky, for all the feedback and support along the way. We would also like to thank other graduate and undergraduate students in our team, Michael Qi, Yash Baldawa, Raghav Gupta, Rohan Hajela, and Varun Murthy, for their contribution to the assistive mouse project.

References

[1] W. Feng, M. Chen, and M. Betke, "Target reverse crossing: a selection method for camera-based mouse-replacement systems," 2014.

[2] M. Betke, "Camera-based interfaces and assistive software for people with severe motion impairments," 2008.

[3] H.-S. Yeo, B.-G. Lee, and H. Lim, "Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware," *Multimedia Tools and Applications*, vol. 74, no. 8, pp. 2687–2715, 2015.

[4] T. Grzejszczak, M. Kawulok, and A. Galuszka, "Hand landmarks detection and localization in color images," *Multimedia Tools and Applications*, vol. 75, no. 23, pp. 16 363–16 387, 2016.

[5] T. M. Mahmoud *et al.*, "A new fast skin color detection technique," *World Academy of Science, Engineering and Technology*, vol. 43, pp. 501–505, 2008.

[6] Q. Liu and G.-z. Peng, "A robust skin color based face detection algorithm," in *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010)*, vol. 2. IEEE, 2010, pp. 525–528.

[7] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, "A survey of skin-color modeling and detection methods," *Pattern recognition*, vol. 40, no. 3, pp. 1106–1122, 2007.

[8] M. Kawulok, J. Kawulok, J. Nalepa, and B. Smolka, "Self-adaptive algorithm for segmenting skin regions," *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, pp. 1–22, 2014.

[9] M. Kawulok, J. Nalepa, and J. Kawulok, "Skin detection and segmentation in color images," in *Advances in low-level color image processing*. Springer, 2014, pp. 329–366.

[10] A. L. Barczak and F. Dadgostar, "Real-time hand tracking using a set of cooperative classifiers based on haar-like features," 2005.
[11] Q. Chen, N. D. Georganas, and E. M. Petriu, "Real-time vision-based hand gesture recognition using haar-like features," in *2007 IEEE instrumentation & measurement technology conference IMTC 2007*. IEEE, 2007, pp. 1–6.
[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.
[13] C. Manresa, J. Varona, R. Mas, and F. J. Perales, "Hand tracking and gesture recognition for human-computer interaction," *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 5, no. 3, pp. 96–104, 2005.
[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3098997.3065386
[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556
[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/
[17] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, publisher: MIT Press. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735
[18] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv:1412.3555 [cs]*, Dec. 2014, arXiv: 1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555
[19] O. Köpüklü, A. Gunduz, N. Kose, and G. 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)*. IEEE, 2019, pp. 1–8.
[20] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018.
[21] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. A. Shah, and M. Bennamoun, "Attention in convolutional lstm for gesture recognition," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 1957–1966.
[22] B. Zhou, A. Andonian, A. Oliva, and A. Torralba, "Temporal relational reasoning in videos," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 803–818.
[23] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.
[24] Q. De Smedt, H. Wannous, and J.-P. Vandeborre, "Skeleton-Based Dynamic Hand Gesture Recognition," 2016, pp. 1–9.
[25] J. C. Núñez, R. Cabido, J. J. Pantrigo, A. S. Montemayor, and J. F. Vélez, "Convolutional Neural Networks and Long Short-Term Memory for skeleton-based human activity and hand gesture recognition," *Pattern Recognition*, vol. 76, pp. 80–94, Apr. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304405
[26] F. Yang, S. Sakti, Y. Wu, and S. Nakamura, "Make Skeleton-based Action Recognition Model Smaller, Faster and Better," *arXiv:1907.09658 [cs]*, Mar. 2020, arXiv: 1907.09658. [Online]. Available: http://arxiv.org/abs/1907.09658
[27] P. Schneider, R. Memmesheimer, I. Kramer, and D. Paulus, "Gesture recognition in rgb videos using human body keypoints and dynamic time warping," in *RoboCup 2019: Robot World Cup XXIII*, S. Chalup, T. Niemueller, J. Suthakorn, and M.-A. Williams, Eds. Cham: Springer International Publishing, 2019, pp. 281–293.
[28] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.
[29] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "MediaPipe: A Framework for Building Perception Pipelines," *arXiv:1906.08172 [cs]*, Jun. 2019, arXiv: 1906.08172. [Online]. Available: http://arxiv.org/abs/1906.08172
[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989. [Online]. Available: https://direct.mit.edu/neco/article/1/4/541-551/5515
[31] B. Zhou, A. Andonian, A. Oliva, and A. Torralba, "Temporal relational reasoning in videos," *European Conference on Computer Vision*, 2018.
[32] O. Kopuklu, N. Kose, and G. Rigoll, "Motion fused frames: Data level fusion strategy for hand gesture recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.
[33] A. D. Wilson, "Robust computer vision-based detection of pinching for one and two-handed gesture input," in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, 2006, pp. 255–258.
[34] R. Abelson, "Theranos founder elizabeth holmes indicted on fraud charges," *The New York Times*, 2018.

6571 words