Real-Time First-View Hand Gestures Recognition

Yipeng Lin and Xinyang Chen
Carnegie Mellon University
{yipengli, xinyangc}@andrew.cmu.edu

June 23, 2022

Contents

1 Introduction

Hand gesture recognition is one of the most popular applications in computer vision, which provides a simple communication method in human and computer interaction (HCI). We present a hand gesture recognition system dedicated to the designated application scenario that aims of providing a fast, light-weight, scalable, and user-friendly service. The presented system is composed of multiple highest-precision detection or recognition models, including dynamic hand gesture recognition model, static hand poses recognition model, and gesture recognition with gloves model. In this document, we briefly introduce the general design of the *Hand Gesture Recognition* system and the trail of tests we have done.

The proposed system takes in a video stream and deals with two recognition tasks, recognizing *static poses* and *dynamic gestures*. The general working procedure of the system is depicted in Fig. 1. Static poses refer to poses of hands that can be recognized using a single image, e.g., hand poses for numbers and “ok”. Dynamic gestures refer to gestures that contain meaningful information in multiple consecutive frames, e.g., waving hands. Static pose recognition is achieved by using MediaPipe [1]. Dynamic gesture recognition is achieved via a customized deep Recurrent Neural Network (RNN) which will be described in Sec. 2.

In general, recognizing static poses is achieved by defining the range of angles for each key points of a human hand, which is named as “Hand Landmarks Degree Calculation Algorith” in Fig. 1. Static images are first fed to MediaPipe to perform a key-point recognition. Each hand will be represented by 20 hand landmarks. A list of landmarks and their corresponding positions in a hand is depicted in Fig. 2. Static poses are then recognized by searching for a target pose whose required range of angles or distances between several landmarks match the input image. For instance, the pose "one" or "point" can be classified when only landmark 5, 6, 7, 8 are in a line. Scaling up the system to support a new pose can be achieved by mathematically defining the relative positions (angle or distance) of each landmark of interest. In this project, the recognition of fifteen poses are supported, which includes number counting (i.e. 1 - 9), thumb up, thumb down, yeah, gun, rock & roll, and point.

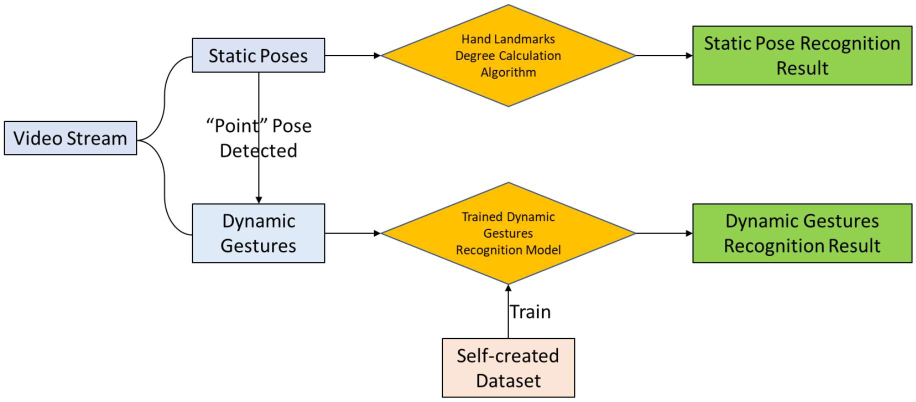

Video Stream
Static Poses
Hand Landmarks Degree Calculation Algorithm
Static Pose Recognition Result
“Point” Pose Detected
Dynamic Gestures
Trained Dynamic Gestures Recognition Model
Dynamic Gestures Recognition Result
Train
Self-created Dataset

Figure 1: Structure of the recognition system.

Dynamic gestures recognition model is built on top of static pose recognition. Particularly, we perform the recognition of dynamic gesture only when the user is posing “point” and the coordinates of the index finger tip is collected. The recognition task is achieved by building a Recurrent Neural Network (RNN). We fed the trace of the finger tip positions in a consecutive 45 frames into a pre-trained RNN model, and a "Trained Dynamic Gestures Recognition Model" in Fig. 1 will perform

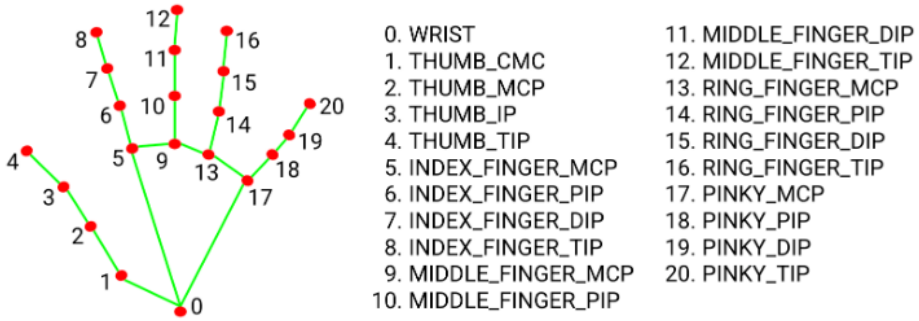
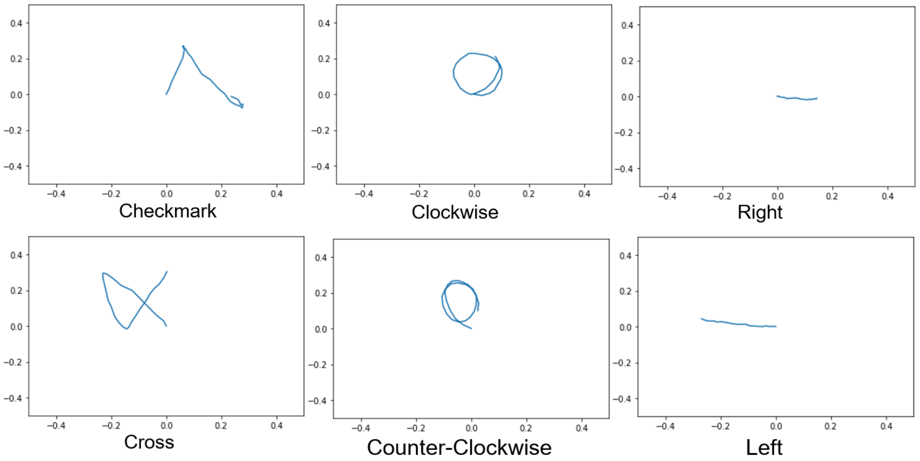


Figure 2: Hand landmarks [2].

the recognition task and provide a prediction result. Limited by the amount of data we can collect, we adopted Long-Short-Term-Memory (LSTM) [3] model to balance the computation complexity and recognition accuracy. Due to the adoption of a learning-based model, we built a gesture dataset, which will be elaborated in Sec. 2.1 and Sec. 2.2. In this project, we focused on the recognition of six dynamic gestures, which are listed in Table 1.



Checkmark

Clockwise

Right

Cross

Counter-Clockwise

Left

Figure 3: Trajectories of dynamic hand gestures.

2 Dynamic Gesture Recognition

In this section, we describe the proposed dynamic gesture recognition system. We first describe how we build and clean a gesture dataset. Then, we discuss methods to augment data points. Finally, we sequentially describe different attempted models and evaluate their performances.

Table 1: Supported hand gestures.

Gesture Id	Gesture Description
0	Rotate clockwise
1	Rotate counter-clockwise
2	Checkmark
3	Cross
4	Move left
5	Move right
6	None

2.1 Dataset Creation and Cleaning

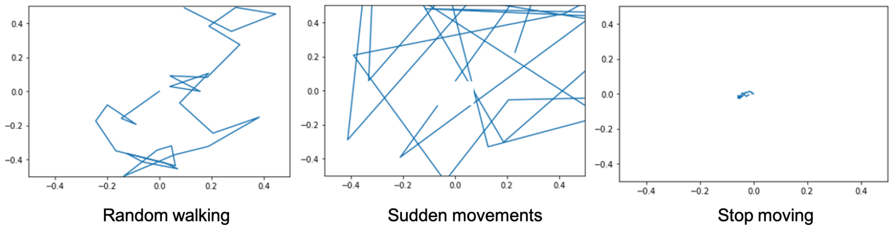
We built a dataset that contains 400 data points for each of the six gestures. For data point, we use a web-camera to record a 30-frame (1 second) video slice and then use MediaPipe to get the x-y position of index finger tip in each of the 30 frame. Thus, each data point is a vector of 60 real numbers, i.e., 30 pairs of x and y coordinates. For most of the gestures in Table 1, a 30-frame video slices is adequate to capture the whole gesture.

Since MediaPipe may fail to get the finger tip positions in some frames, after collecting the data, we proposed to clean the dataset as follows

- Define a tolerant moving distance of the finger tip in consecutive two frames. If the moving distance of the finger tip in the two frames exceeds the tolerance, we use the position in the previous frame to estimate and replace the position in the current frame.
- We wrote a small program to visualize the trace of the finger tip in each 30-frame data point and manually decide whether the trace matches the corresponding label (gesture type).

In addition to collected gestures, we add gestures for label “None”, which represents gestures that are not categorized as one of the six gestures. We consider three cases for the “None” gesture: random walking, sudden movements, and stop moving. Examples of the trajectories are depicted in Figure 4.

- **Random walking** mimics the situation where a user moves the hand subconsciously and does not mean to express any meaning.
- **Sudden movements** mimics the situation where a user makes some sudden movements which may exceed the detection range.
- **Stop moving** mimics the situation that a user stops doing gestures and hovers the hand at a random position.



Random walking

Sudden movements

Stop moving

Figure 4: Three examples of trajectories of “None” gestures.

Table 2: Applied augmentation operations.

Gesture Description	Rotation	Translation	Scaling
Rotate clockwise	✓	✓	✓
Rotate counter-clockwise	✓	✓	✓
Checkmark		✓	✓
Cross	✓	✓	✓
Move left		✓	✓
Move right		✓	✓

2.2 Data Augmentation

The cost to collect data for a large dataset is high. We proposed to conduct data augmentation in the following ways:

- **Rotation** The input data point is rotated by a randomly selected degree while its associated gesture type is unchanged.
- **Translation** The input data point is shifted in x and y axes at random distance.
- **Scaling** The input data point is scaled up/down at a random ratio.

Each data point is applied sequentially with multiple augmentation operations, and which operation can be applied is shown in Table 2.

3 Recognition Models

In this section, we describe several dynamic gesture recognition models we have attempted and their performance evaluations.

3.1 Fully-Connected Model

In this model, we proposed to use a Fully-connected neural network [4] to perform the prediction. The x-y coordinates of the index finger tip in a consecutive 30-frames will be formulated as a 60 × 1 vector and then be fed to a fully-connected neural

Table 3: Fully-connected model results before augmenting data.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.44	0.38	0.41	92
Counter Clockwise	0.67	0.55	0.61	112
Checkmark	0.70	0.76	0.73	97
Cross	0.58	0.64	0.61	110
Right	0.52	0.47	0.49	111
Left	0.72	0.78	0.75	85
None	0.75	0.89	0.81	96
Accuracy			0.63	703

Table 4: Simple LSTM network model results before augmenting data.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.49	0.58	0.53	92
Counter Clockwise	0.80	0.65	0.72	112
Checkmark	0.80	0.82	0.81	97
Cross	0.69	0.58	0.63	110
Right	0.77	0.59	0.67	111
Left	0.65	0.86	0.74	85
None	0.81	0.95	0.88	96
Accuracy			0.71	703

network. The output of the network is the prediction confidence of each label. The final prediction result is by taking the label that is of the highest confidence. The evaluation results on the original dataset that did not perform data augmentation are listed in Table 3.

The simple fully-connected model is able to achieve a prediction accuracy of 63%. However, fully-connected model does not have the ability to extract the temporal information between each coordinates in a sequence of frames.

3.2 Single LSTM Network Model

We proposed to use an LSTM network to capture the temporal information. The testing results on the original dataset are listed in Table 4. Compared with results in Table 3, LSTM model has a better ability to capture the temporal information and achieves a higher prediction accuracy. We also conducted another experiment on augmented dataset, and the results are listed in Table 5. Augmentation helps to improve the prediction accuracy by 23%.

Table 5: Simple LSTM network model results after augmenting data.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.98	0.95	0.97	979
Counter Clockwise	0.97	0.95	0.96	966
Checkmark	0.90	0.88	0.89	845
Cross	0.87	0.88	0.88	971
Right	0.96	0.97	0.97	778
Left	0.93	0.96	0.94	853
None	0.95	0.98	0.97	997
Accuracy			0.94	6389

3.3 Double-Network Model

After using a simple LSTM network model, we tried a double-network model to increase the prediction accuracy on complex gestures (i.e., $checkmark$ and $cross$). The proposed double-network model contains two independent neural networks, denoted as model 1 and model 2. Model 1 represents the simple LSTM network model described in Sec. 3.2 that recognizes all seven gestures. Model 2 represents a new model designated only for the recognition of $checkmark$, $cross$, and $none$. The $none$ type of gesture means all the other types of gestures except $checkmark$ or $cross$. Model 2 shares a similar network model as model 1, just reduces the output dimension from 7 to 3.

During the training, the two models are trained using the same dataset separately and independently. During the testing, model 1 and model 2 process the same frame simultaneously. The prediction result of both model 1 and model will enter their own counter that counts the frequency of each predicted result in the most recent 30 frames. For model 1, the output of the counter is the most frequent three gesture types. For model 2, only the most frequent gesture type is produced. If the prediction result from model 2's counter is also among the three prediction results from model 1's counter, the final prediction result is the model 2's result since model 2 is more dedicated to the complex gesture. If model 2's result is not in model 1's three prediction results, we take the most frequent result from model 1 as the final prediction. The structure of the double-network model is depicted in Fig. 5.

We evaluate the double-network model in two scenarios. In the first scenario, the inputs are 60×1 vectors that are from the dataset but are not used during the training. The testing performance is listed in Tab. 6. In the second scenario, we captured a short video of 1408 frames and fed the video to the system to do a live prediction. The performance is listed in Tab. 7.

From the two tables, we can see that the proposed double-network model achieves good prediction results when the test inputs are from the dataset while its prediction accuracy decreases dramatically on a more realistic application. We think that this is a problem of time alignment of input data. For data points in the dataset, the 30 frames are all from and for the same gesture. In real applications, it is common to ask a model to predict the gesture in real application while the user is in the transition stage between two gestures.

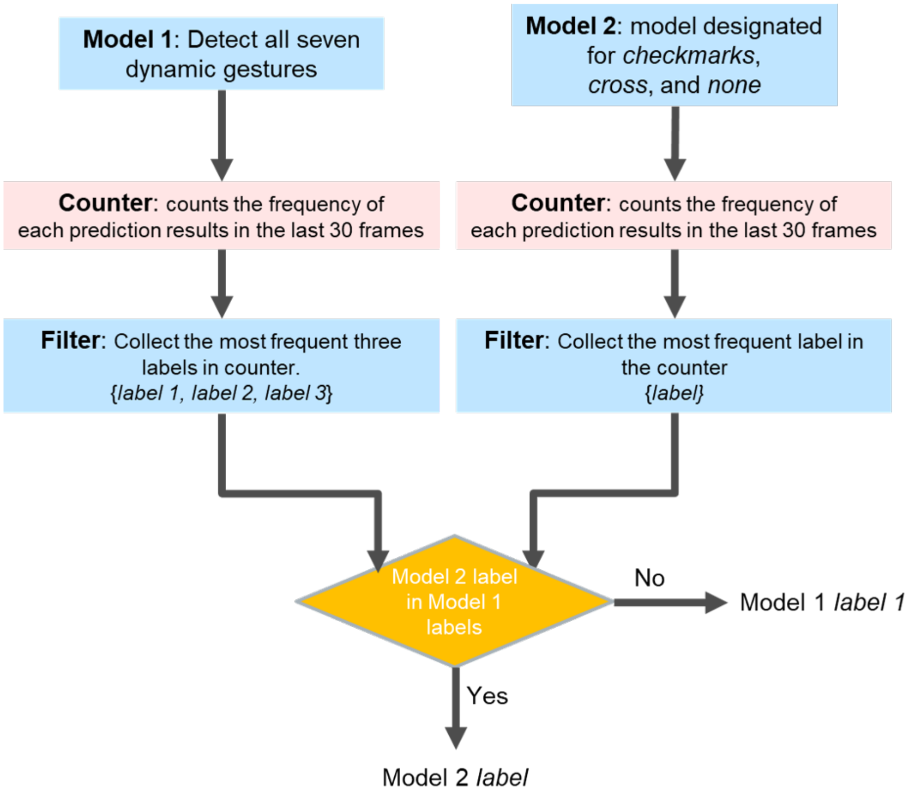

Model 1: Detect all seven dynamic gestures
Model 2: model designated for *checkmarks*, *cross*, and *none*
Counter: counts the frequency of each prediction results in the last 30 frames
Counter: counts the frequency of each prediction results in the last 30 frames
Filter: Collect the most frequent three labels in counter. {*label 1, label 2, label 3*}
Filter: Collect the most frequent label in the counter {*label*}
Model 2 label in Model 1 labels
No
Model 1 *label 1*
Yes
Model 2 *label*

Figure 5: Double network model.

3.4 Single-Network Model with Random Prefix and Suffix

To deal with the alignment issue found in Sec. 3.3, we generate a new data point that may contain transitions of two other gestures. For each data point, two more data points are randomly selected while one is treated a prefix data point and the other is treated as a suffix data point. We first concatenate the three data points one after another and then truncate it to the designated length. The padding visualization can be seen in Figure 6, where the orange $Gesture$ is the target gesture for model to predict and two other gestures are randomly selected as its prefix (L) and suffix (R). In the truncate step, we keep a constant length (45 frames) and make sure that the target gesture always occupies the most number of frames, i.e., >= 23 frames. Figure 6 shows the maximum length of prefix (i.e. $L = 22$) in 6a, balanced prefix and suffix in 6b, and the maximum length of suffix (i.e. $R = 22$) in 6c. By controlling the maximum number of allowed frames, we control the difficulty of the training task.

Table 6: Model 2 Testing Validation Results.

Gesture Description	Precision	Recall	f1-score	Support
Charkmark	0.92	0.98	0.95	894
Cross	0.97	0.92	0.94	956
None	0.98	0.97	0.98	997
Accuracy			0.96	2847

Table 7: Model 2 Video (1408 frames) Testing Results.

Gesture Description	Precision	Recall	f1-score	Support
Charkmark	0.54	0.80	0.64	374
Cross counter-clockwise	0.57	0.29	0.39	388
None	0.94	0.10	0.18	646
Accrucy			0.34	1408

For instance, the difficulty of case 6b is the smallest because the target gesture takes up the longest time.

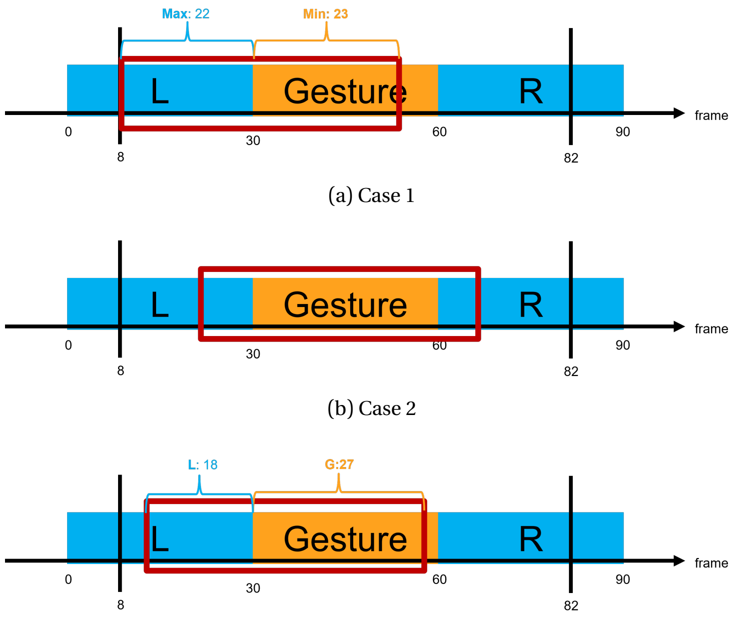
We also make experiments of the three cases in Figure 6 and the results can be seen in Table 8, 9, and 10 separately. From these three tables, we can find that the Table 8 of case 6b has the highest accuracy. This may because the target gesture is complete and with lowest distraction from its prefix and suffix gestures.

4 Glove Preprocessing Method

Mediapipe's hand tracking solution is based on deep learning models trained with a hand dataset. Therefore, it's sensitive to not only the hand's shape but also the skin color. Gloves have little influence on the hand's shape but change the color

Table 8: Prefix and suffix case 1 Results.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.94	0.90	0.92	996
Counter Clockwise	0.93	0.86	0.89	963
Checkmark	0.87	0.77	0.82	864
Cross	0.75	0.79	0.77	1009
Right	0.89	0.90	0.89	749
Left	0.91	0.85	0.88	832
None	0.81	0.95	0.87	1101
Accuracy			0.86	6514



(a) Case 1

(b) Case 2

(c) Case 3

Figure 6: Prefix and suffix visualization.

Table 9: Prefix and suffix case 2 Results.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.98	0.96	0.97	996
Counter Clockwise	0.98	0.94	0.96	963
Checkmark	0.92	0.88	0.90	864
Cross	0.89	0.85	0.87	1009
Right	0.93	0.95	0.94	749
Left	0.90	0.92	0.91	832
None	0.88	0.97	0.93	1101
Accuracy			0.93	6514

Table 10: Prefix and suffix case 3 Results.

Gesture Description	Precision	Recall	f1-score	Support
Clockwise	0.97	0.94	0.95	996
Counter Clockwise	0.95	0.91	0.93	963
Checkmark	0.85	0.82	0.83	864
Cross	0.83	0.77	0.80	1009
Right	0.90	0.94	0.92	749
Left	0.87	0.90	0.88	832
None	0.86	0.94	0.90	1101
Accuracy			0.89	6514

which significantly decreases the tracking success rate and accuracy. We develop a preprocessing method for gloves that can fit them into Mediapipe's suitable color range.

We show the one-frame preprocessing pipeline in Figure 7. To begin with, we convert the raw input from RGB color space into CIELAB color space and get a target mask based on the value of *A* and *B*. On the construction site, the lightning condition can be various. In CIELAB color space, *L* is corresponding to light intensity while *A* and *B* are stable with the different lighting conditions. Thus, we only use the value of *A* and *B* to get the mask which can avoid the influence of lighting conditions. After getting the mask, we add or minus a specific value to both *L*, *A*, and *B* of pixels inside the mask. For *A* and *B*, the value is corresponding to the gloves' color and we can determine it by checking the distance between the suitable range and the gloves' color range. For *L*, we run an adjusting module to find the suitable value under a specific lighting condition. We introduce the module in Figure 8. Finally, we convert back to RGB color space and send the output to the Mediapipe hand detector to get the tracking result.

We show the complete pipeline of the glove preprocessing method in Figure 8. At the very beginning, we run the parameter initialization module to determine *L*'s shifting value until the module successfully finds a suitable shifting value. For each frame after determining *L*'s shifting value, the process is the same as in Figure 7. In the parameter initialization module, the first step is to initialize the shifting value as the difference between the maximum *L* value inside the mask and the maximum possible *L* value. Mediapipe is more likely to successfully track object with higher *L* value, so we attempt to set the pixels' *L* value as high as possible inside the region of interest. After initialization, we run a loop to gradually increase the shifting value and check whether the Mediapipe can track the hand successfully. When the module succeeds, we set *L*'s shifting value and stop running the parameter initialization module. Otherwise, we keep on trying until succeed.

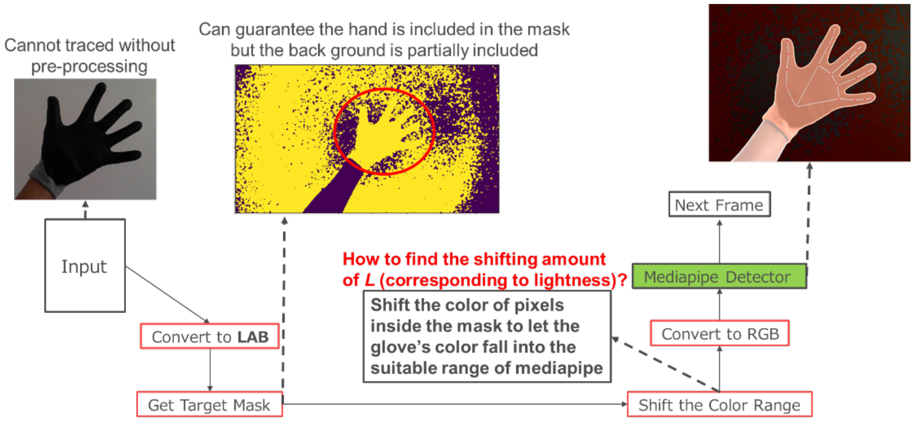

Cannot traced without pre-processing
Can guarantee the hand is included in the mask but the back ground is partially included
Next Frame
Input
How to find the shifting amount of *L* (corresponding to lightness)?
Mediapipe Detector
Shift the color of pixels inside the mask to let the glove's color fall into the suitable range of mediapipe
Convert to LAB
Convert to RGB
Get Target Mask
Shift the Color Range

Figure 7: Preprocess in one-frame.

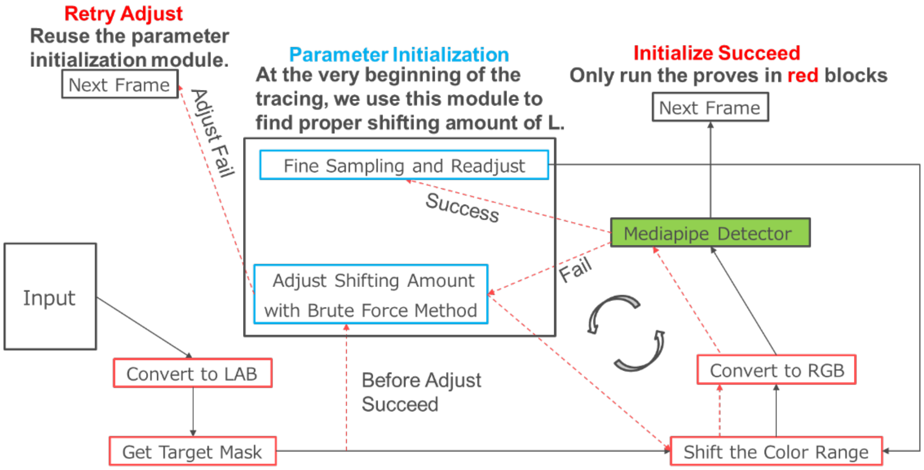
5 Conclusion

Our project's goal is to recognize static and dynamic gestures with a vision-based method. Considering the demand on the construction site, we also develop a preprocessing method to adapt the recognition model to the situation of wearing gloves. We do a series of testing on the gesture recognition model, proving its reliability and applicability. For the performance of the glove preprocessing method, we haven't built a corresponding testing dataset, which is part of our future work. However, we have some sample demos on our project's GitHub page for fundamental reference [5]. We also include source code and instruction in the GitHub page. We will update the page as we step further in this project.

6 Future Plan

For the future plan we have two directions. The first one is refining the dataset. Even though using the padding prefix and suffix method in subsection 3.4 largely improves the performance in real-time video testing, the transition of different gestures in real world cannot be fully mimicked. Thus, we will using recording videos to create the dataset in the next step.

To test the glove preprocessing method scientifically, we are working on building a glove dataset including various validation samples. Given the complicated background in construction sites, we are scheduled to improve our method's robustness. Meanwhile, we are also interested in advancing the method's generalization ability for different kinds of gloves. In general, we will spend effort on enhancing the system's applicability and stability.



Retry Adjust
Reuse the parameter initialization module.
Next Frame
Adjust Fail
Parameter Initialization
At the very beginning of the tracing, we use this module to find proper shifting amount of L.
Initialize Succeed
Only run the proves in red blocks
Next Frame
Fine Sampling and Readjust
Success
Mediapipe Detector
Fail
Input
Adjust Shifting Amount with Brute Force Method
Convert to LAB
Before Adjust Succeed
Convert to RGB
Get Target Mask
Shift the Color Range

Figure 8: Glove preprocessing method pipeline.

References

[1] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," 2019, arXiv:2006.10214.

[2] G. LLC, "Mediapipe hands," google.github.io. https://google.github.io/mediapipe/solutions/hands (accessed May. 22, 2022).

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] J. A. Freeman and D. M. Skapura, *Neural networks: algorithms, applications, and programming techniques.* Addison Wesley Longman Publishing Co., Inc., 1991.

[5] Y. Lin and X. Chen, "Cerlab realtime handgesture recognition," https://github.com/Flora9978/CerLab_RealTime_HandGesture_Recognition/tree/glove (accessed May. 25, 2022).