Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Hand Gesture Passwords

Shubham Kumar

Instructor: Subrahmanyam Murala

December 21, 2023

A project report submitted in partial fulfilment
of the requirements for course
CS7GV1 computer vision

Declaration

I hereby declare that this project report is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

I consent to the examiner retaining a copy of the project report beyond the examining period, should they so wish (EU GDPR May 2018).

I agree that this thesis will not be publicly available, but will be available to TCD staff and students in the University's open access institutional repository on the Trinity domain only, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Signed: Shubham Kumar Date: 21/12/2023

This must be on a separate page.

1. Abstract

When working in a Virtual Reality and Mixed Reality platforms, a password input becomes a tedious task prone to shoulder tapping attacks as the speed of input decreases significantly with the controller inputs.

This project explores an alternative approach to input passwords that uses concepts from computer vision and can be used in a mixed reality scenario.

There are ways to command a device with different types of hand gestures and it is already being used in conjunction with Raspberry Pi for IoT applications.

This project uses these hand gesture inputs as passwords which can be further be used to authenticate the user for different applications.

The scope of this project is to establish a way to read a series of gestures including repeated gestures, record them, make a simple hash out of them, store them locally and use an input to compare them.

The end result of the project is an example program where the user can capture and enter the password by recording their hand gestures in a particular pattern of their choice. We try to use mediapipe trainer and custom model to train these data and built a basic password entry and validation flow with it. We also explore the feasibility to do hashing on that same.

In the end, we disucss the application of this method and how feasible it is to use it in the general flow.

2. Introduction

The project explores one or more way to record a series of hand gestures as passwords. We already have tons of work done on hand gesture recognition. We can, however, input passwords in two ways: labelled gestures or freestyle gestures.

In the former, we try to provide a limited set of gestures that can be used to set the password. We can consider it to be similar to the letters and numbers in a usual passwords. We get a series of labels which can be then stored as the labels themselves hashed with a hashing function. This approach is easier to store and hash after the capture of the sequence. However, these may be prone to brute force attack since the limited number of gestures increase the attack vector. However, if a large number of gesture is supported, this may be mitigated to an extent.

On the other hand, a freestyle gesture is where the user can capture any gesture as

password. These are then stored in the system. This does improve the attack vector, but it is tough to implement as storing the data and hashing it becomes tedious. There is also need to consider the error when we capture the gesture as a user may not be able to show the same gesture again and again.

In this project, we try to explore both the scenarios with a basic implementation.

3. Related work (Literature review)

To recognize a hand gesture, we first need to recognize the hand(s) in the frame. Once we have detected the hands, we can move on to detect the key-points in hands. The key-points are basically the skeleton of the hands, they highlight the each movable part of the hand structure.

Mediapipe provides a fairly complex and accurate pre-trained model and a library to support detecting the hand key-points. You can further train and import your model into it as well. Mediapipe's hand landmark detection can give all the possible movable joints in the hand starting from the palm. It supports multiple hands too. In total, it detects 21 points in each hand which includes palm, joints and fingertips. Once the key-points are available, they can be normalized based on use case with another option. The median deviation for fingertips has been recorded to be around 1.26 millimeters.

Alphapose is another research done on gesture/pose recognition. This is a complex model that supports detection of body pose alone with hand keypoints. This is implemented with PyTorch. A similar model is OpenPose. These models can be used with a command line or shell interface as well and have a good support for all the major operating systems.

To detect specific hand gestures, we can either use mediapipe's model maker to train our own data set. It's based on Googlenet architecture. These require at least 100 images of each label to train them to a satisfactory accuracy of around 0.7. Mediapipe model maker uses transfer learning wherein it uses the new training dataset on top of the existing model that is supported by the mediapipe. However, it is to be noted that one must use the consistent model with the dataset, i.e. only hand gesture recognition dataset can be used with the hand gesture recognition model and not object classification. It also updates the marker and labels based on the new data.

Alternatively, we can also load a trained tensorflow model and use it to label the gesture data.

Python hashlib can be used to hash string inputs.

4. Proposed work

All the methods are divided in two parts: Store and Input. Store phase is where we create and store the password while Input phase is where we try to get the input from user and try to equate it with the stored password.

4.1 Approach 1: Freestyle with static error values

In this method, we would use mediapipe's hand landmarker task as a pre-trained model and use the model to get the hand keypoints. And that's where we would be done with the landmarking of the hands.

This model would get the hand landmarks which can be stored as localized hand landmarks as these gestures can be present anywhere in the image and should be treated same. We store these localized hand gestures in store phase.

In the input phase, the code compares the landmarks recorded in current phase with the ones recorded in the previous phase, i.e. stored password. These are adjusted with an error parameter which can be either the same for all the landmarks or different for each landmarks.

This approach has a low attack vector as the possibility of gestures is large as it can support almost every type of gesture. However, the static error tuning may not work in most of the scenarios. This can be, though, trained with a large dataset. The biggest issue, with this approach however, is the security of passwords itself.

In most of the modern applications, we store a hashed password. Hashing all the landmark is tedious as we cannot combine them since they have to be aggregated as a stream rather than blocks of images against the traditional blocks of letters system. Hashing each keypoint, on the other hand, is an expensive operation as we may require (21 * 2) 42 hashes.

4.2 Approach 2: Training model with mediapipe model maker

In this approach, we would train a model with mediapipe model maker. We can use the default 'gesture_recognizer' package from mediapipe to train the model.

To capture the data, we can use cv2's videocaputure and capturing a frame in every 0.6 seconds. This number is a balance between speed of capture and ability to move hands between each frame. We store each frame in it's own categorical directory.

Export this model and import it in the application

4.3 Approach 3: Training model with Tensorflow

In this, we train a custom model with tensorflow. I found a source online which uses mediapipe's landmark detection and trains a model on these keypoints to get the gesture data.

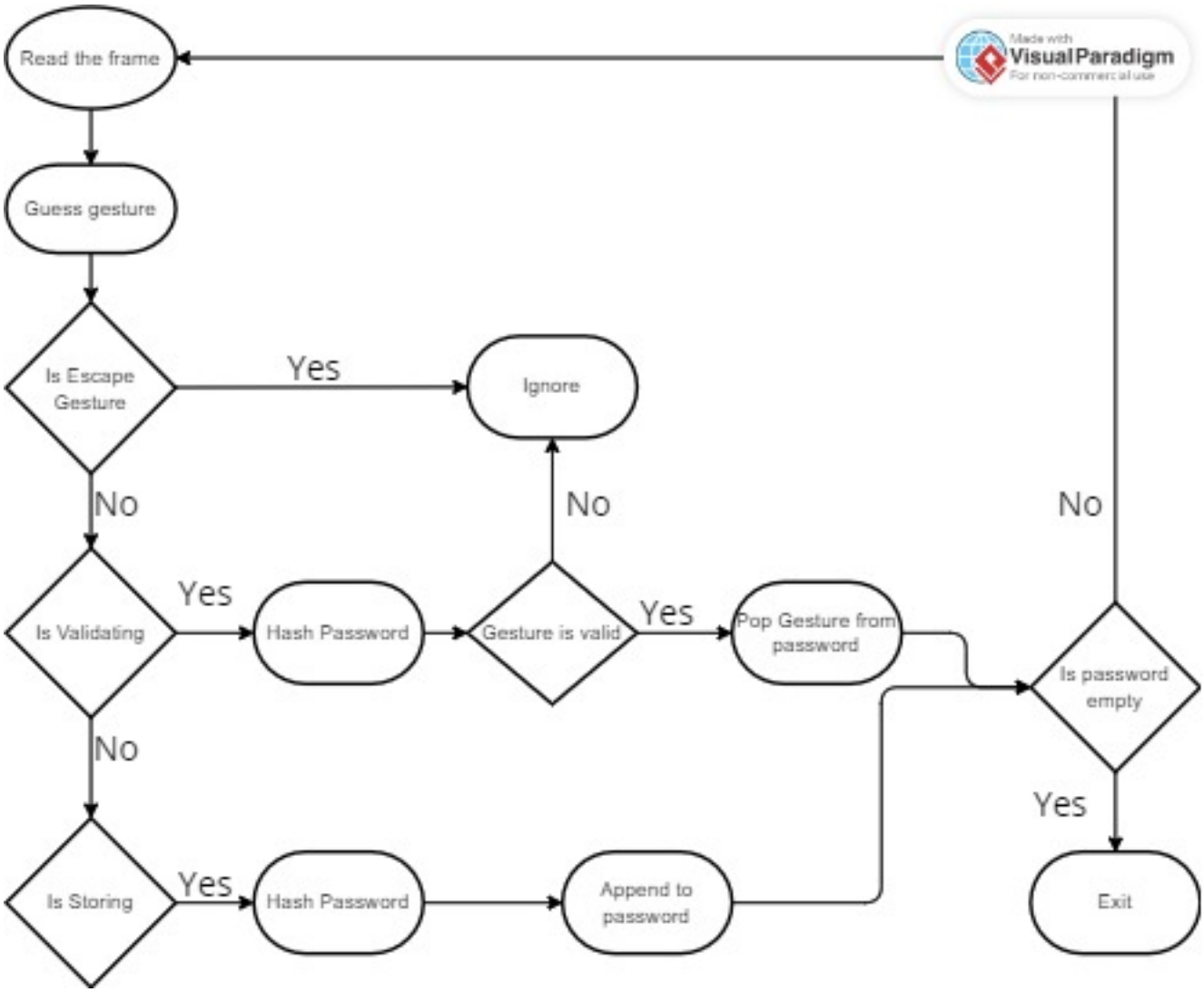
I was able to increase the model's accuracy by a small percentage (1 percent) by adding an extra dense layer to it. This model can then further be used to get the gesture.

Application Algorithm

```
password := empty
mode := none
escape_gesture

on_video_capture:
    if mode is recording and gesture is not escape_gesture:
        update_password(append(gesture))
    else if mode is entering and gesture is not escape_gesture:
        if front(password) is gesture:
            pop(password)
    if password is empty:
        exit
    process_key_press_for_mode(key_press)
```

Flowchart



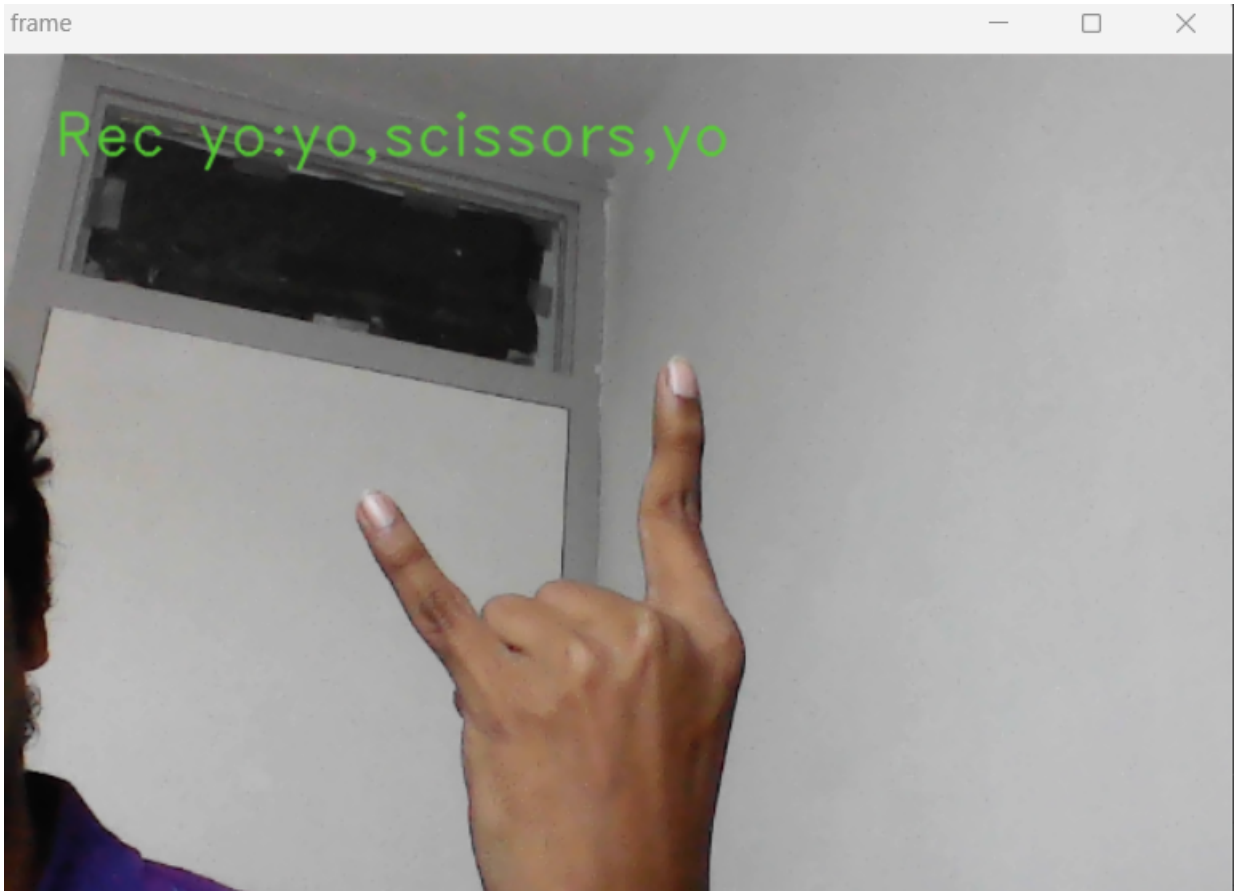
5. Experimental Discussion

For Approach 1, there was a sharp issue with error calculation observed. It was also observed that the errors vary from person to person as well which may add to the security of the application. For the experiment, I rounded the difference between the stored and input hand landmark position to 3 decimal and supported an error of 0.1. This may not be optimum and need to be trained. However, hashing the hand position data is an expensive operation. I tried hashing each position before submitting them and it affected the main video capture thread.

Approach 2 is the easiest to implement and gave a satisfactory result with an accuracy of around 85 percent. However, it was observed that there was a slight dip in accuracy with addition of new label. This was attributed to the data in the class 'None' from a dataset

that I used. Getting rid of those conflicting dataset helped in improving the results.

Approach 3 was the one that provided the best accuracy, but this was at the cost of adding an extra dense layer and a custom implementation. At the time of the report, the integration to an application for this model was still pending.

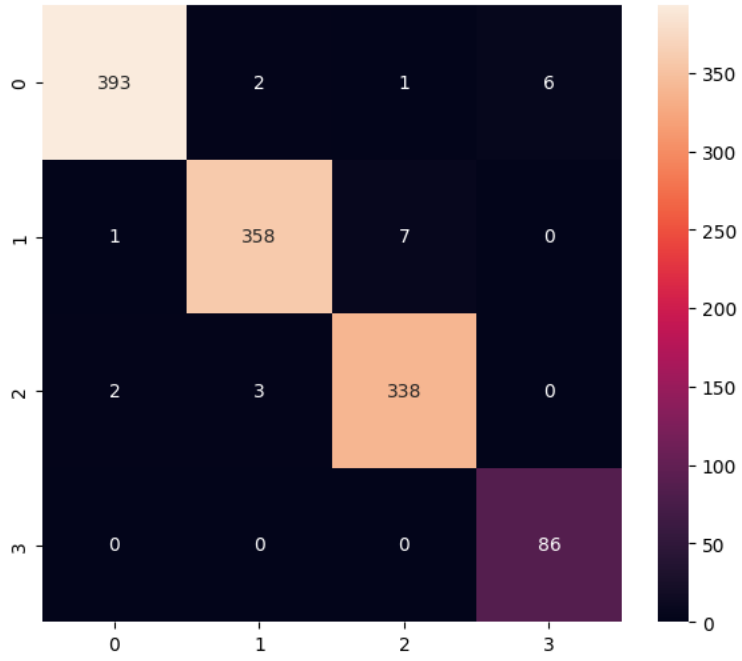


Screenshot: Example output

6. Ablation Study Application of the Proposed Method

- The accuracy of mediapipe model was observed to be 0.845
- The accuracy of Custom model before the change of dense layer was observed to be 0.9632
- The accuracy of custom model after the addition of dense layer was observed to be 0.9816

The Confusion Matrix observed in approach 3 was following:



Change in true positive: +14; Change in f1 score: 0 (upto two decimal points)

7. Application of the Proposed Method

The proposed methods can be used in any type of environment where a hand gesture can be used as a password.

The model and approach using mediapipe can be used in web or mobile application as they are quick and lightweight. Though for complex gestures, it may require seeding the training data carefully as the model trains even with less number of data. The accuracy observed in this case was around 85 percent.

The model and approach using the custom implementation however gave a good response on camera and can be used for instances where we require a better control over the gestures. This was also quick as it was using mediapipe's hand landmark detection in the pipeline before the gesture model.

8. Conclusion

A series of gesture can be used as password for different use cases. The supported gestures can act as a measure of attack vector.

We saw that mediapipe's solution was fairly straight forward and worked out of box for the created system. We also saw that the custom implementation worked well for our use case

too with a slightly better performance.

These systems, however, aren't perfect. An input feedback and correction mechanism while storing the password is important to implement to avoid user from locking out.

8. References

- https://developers.google.com/mediapipe/solutions/vision/hand_landmarker
- https://github.com/topics/mediapipe-hands
- https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/
- https://github.com/MVIG-SJTU/AlphaPose
- https://developers.google.com/mediapipe/solutions/model_maker
- https://docs.python.org/3/library/hashlib.html
- https://en.wikipedia.org/wiki/Transfer_learning
- https://developers.google.com/mediapipe/solutions/vision/hand_landmarker
- https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299
- https://github.com/kinivi/hand-gesture-recognition-mediapipe

Bibliography

[1] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines, 2019.

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