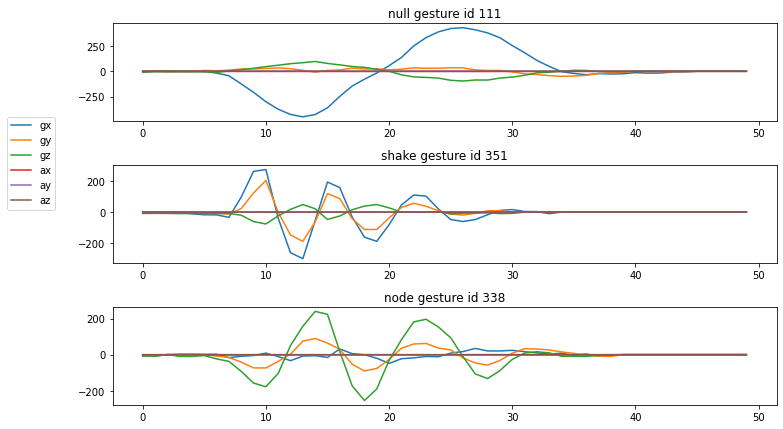
**Design Document**

1. Gesture Classifier

Design process of the gesture classifier is the following.

1. Perform exploratory data analysis by sampling each gesture and plotting it using matplotlib. From the pictures below, I decide to ignore AX, AY, AZ. Moreover, for each GX, GY, and GZ, I make the length of each segment into 50. I choose 50 because most of the segment length is between 40-50. If I make it shorter, I may lose many information and if I pad it to the longer, I may end up using much memory.

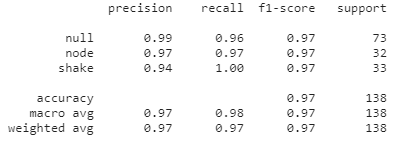
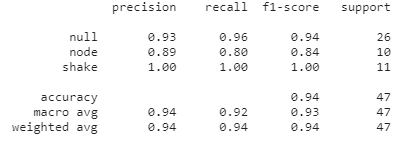


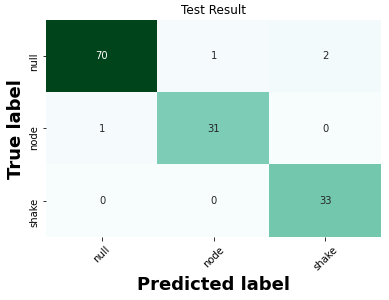
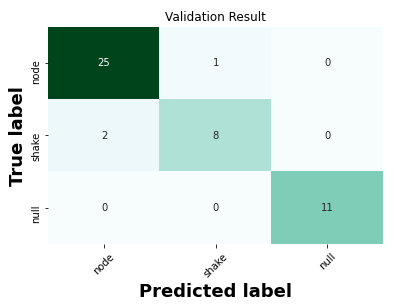
1. I utilize library named “tsfel”[[1]](#footnote-1) to extract features from time series signal. I extract various features that are supported by the library. Afterwards, I use the extracted features to make prediction. After making prediction, I measure the prediction power using Gini importance coefficient. Finally, I choose the best 10 features.

The features can be divided into 3 categories: statistical, temporal, and spectral. The chosen features are the following:

* Spectral: fundamental frequency, median frequency, spectral entropy, spectral kurtosis, spectral slope, spectral spread, wavelet entropy
* Statistical: interquartile range
* Temporal: autocorrelation, peak to peak distance

1. For classification training, I split the data into training + test set and validation set with ratio of 9:1. I train different models: SVM, Decision Tree (DT), and RandomForest (RF). Using 3-fold stratified validation, I get average accuracy of 94%, 85%, and 95% respectively for each classifier. The reason I use k-fold stratified validation was because I want to make the distribution of each category to be the same in each fold. Moreover, I choose k=3 because if I choose larger k, the datapoint will be too few for each fold. Therefore, I choose RF as the classifier. The confusion matrix and the training result for RF classifier are shown as the following (left is test set; right is validation set).

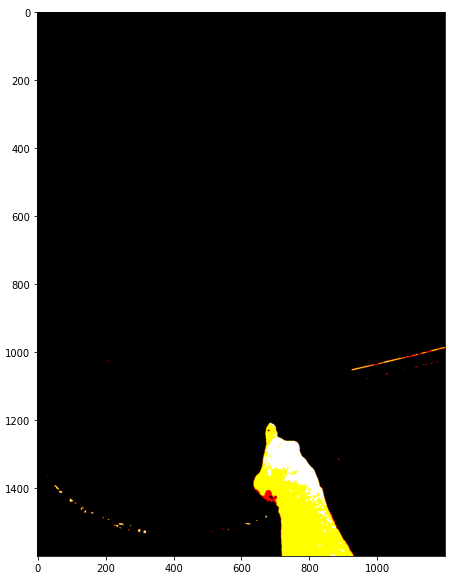
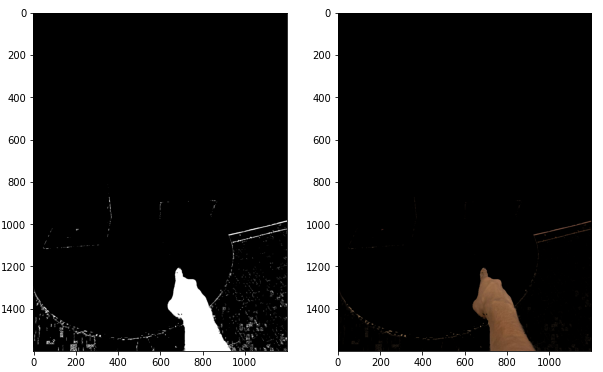
 

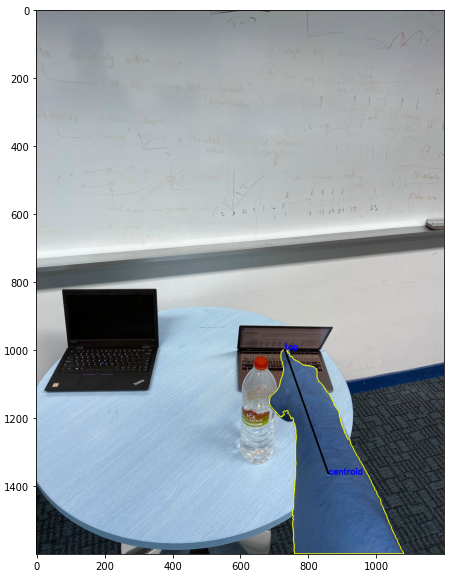
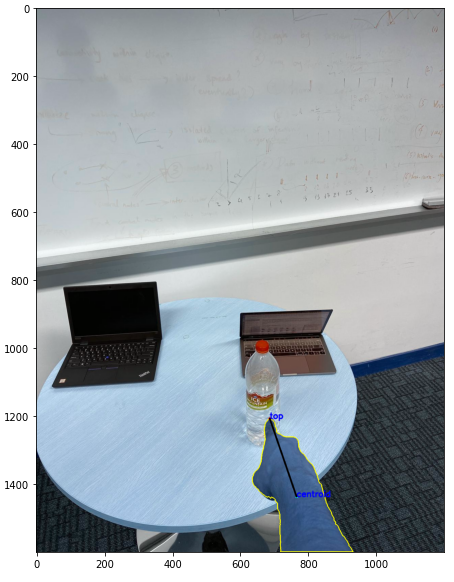
1. Pointing Resolver

Instead of using the provided model to get the pointing direction, I use basic image processing technique. My observation is the extracted point from the hand model is not accurate. Accuracy of line of best fit suffers if there are any outliers (I found that sometimes the extracted points are outside the hand).

The design intuition of my approach is I want to detect hand in the picture, afterwards I extract the fingertip. The algorithm is the following. First, I convert the image from RGB to HSV because it is easier to separate color in the HSV image. Subsequently, I perform masking, blurring, and thresholding in the image. The result of masking, blurring, and thresholding are shown in the image below respectively.

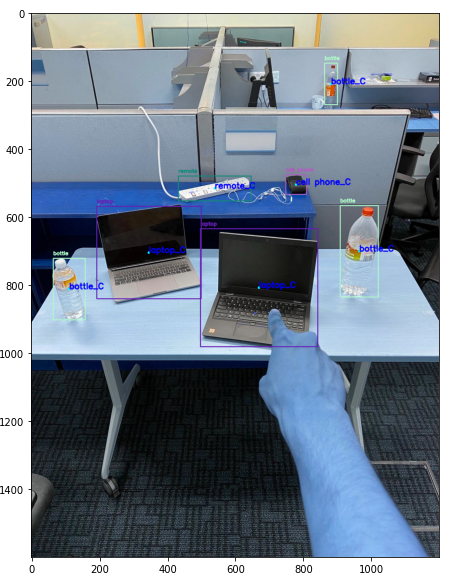


Then I detect the largest contour to get the hand. I get the fingertip by extracting the most top y coordinate. I also extract the centroid of the contour, so I can get a line equation using those 2 points. I show the results of my approach below.

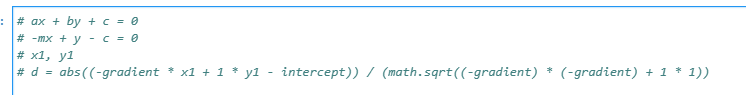


1. Target Resolver

The algorithm to resolve the target is the following. First, I calculate the centroid of all the detected objects as in the Figure below



Subsequently, for each centroid, I calculate the distance between the centroid and the line.



After that, I choose the top 3 objects that has the least distance and put it in the list. (I will refer this list as top-3-least-distance-list later). The intuition is the ambiguous object should be closer to each other, then the distance should be close to each other too. Also, there will be no more than 3 objects that can be close to each other because the distance of the 4th object centroid will be too far from the pointing line. Next, I solve the ambiguity using speech command input. Leveraging the pretrained spacy model, I tokenize the speech command, then extract the token that has dependency “direct object” using dependency parsing technique (there will be no more than 1 direct object in 1 sentence!). If no “direct object” found, I extract token with noun PoS (Part of Speech) tag instead. The later technique is used in case the former technique fails to get the direct object. It is worth to note that the PoS tagging has higher probability to success than dependency parsing because dependency parsing depends on the sentence structure.

Afterwards, I match the extracted object (from the command text) with the top-3-least-distance list. If there is any match, then keep it in the result list. There are 3 cases:

1. Normal condition.

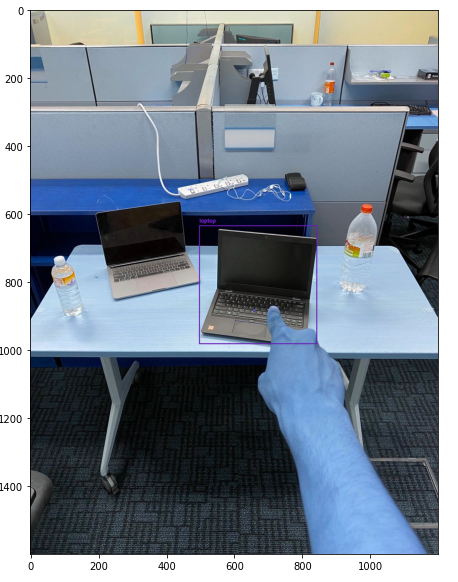
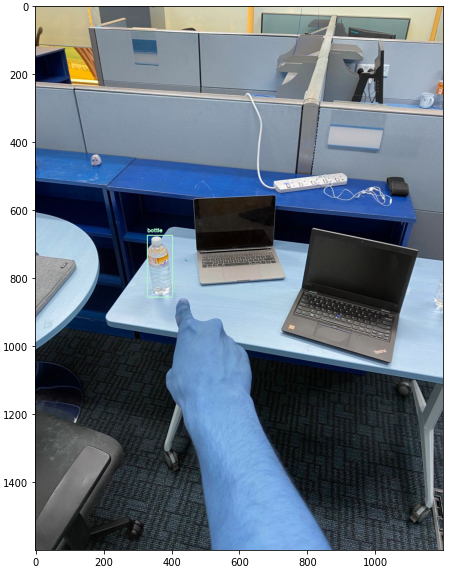
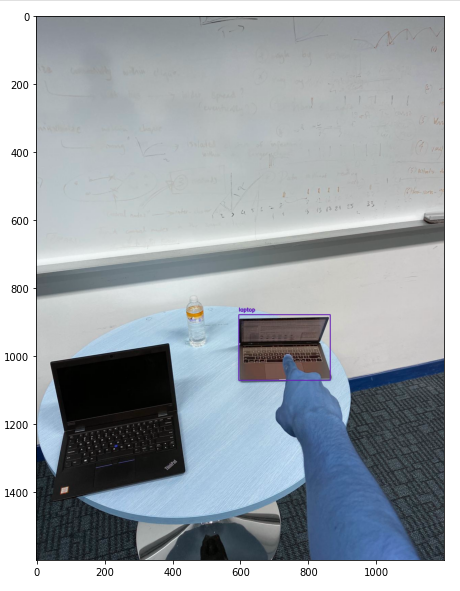
For example, there are 5 objects: chair, laptop, remote, cup, bottle. The laptop, cup and bottle are close to each other. The command input is “select that laptop” and user points ambiguously to the laptop direction. In this case, top-3-least-distance-list will contain [laptop, cup, bottle], the command input object is “laptop”, then my algorithm will return laptop.

1. There are multiple objects that match with the command input object OR the detected object is less than or equal 3 (in this case, the top-3-least-distance-list will be the same as the detected objects list).

For example, there are 3 objects: 1 laptop on the right, 1 laptop on the left, and a bottle. The top-3-least-distance-list will also contain [laptop, laptop, bottle]. Suppose that the user point to the laptop on the right. After we matched top-3-least-distance-list with command input object which is “laptop”, the result list will contain [laptop, laptop]. In this case, the algorithm will return the closest object, which is laptop on the right side.

1. If the voice command is invalid due to reasons such as unclear human tone, my algorithm will return the object with the closest distance instead. For example, we say “laptop” and the speech recognizer recognizes it as “fliptop”.

Some output examples are shown below. The command input object is laptop, bottle, and laptop, respectively.

1. M. Barandas *et al.*, “TSFEL: Time Series Feature Extraction Library,” *SoftwareX*, vol. 11, p. 100456, Jan. 2020, doi: [10.1016/j.softx.2020.100456](https://doi.org/10.1016/j.softx.2020.100456). [↑](#footnote-ref-1)