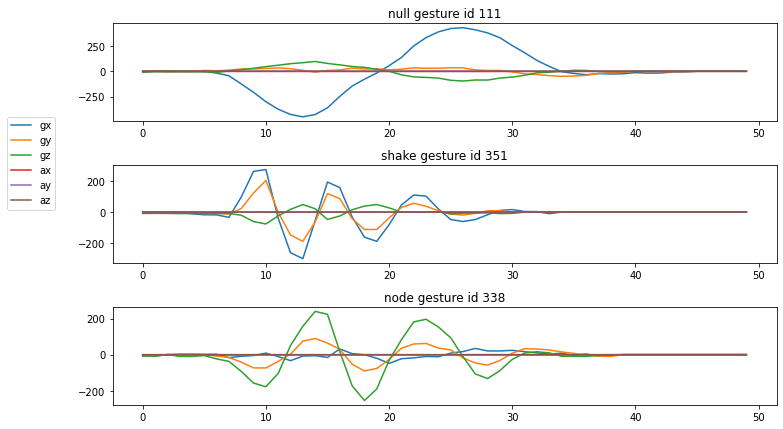
**Design Document**

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. I decide to ignore AX, AY, AZ. Moreover, for each GX, GY, and GZ, I make the length of each segment into 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 longer, I may end up using much memory.

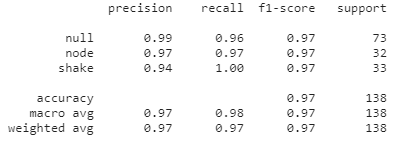
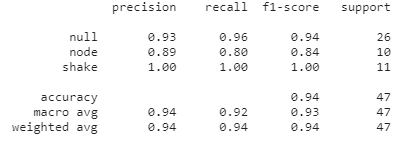


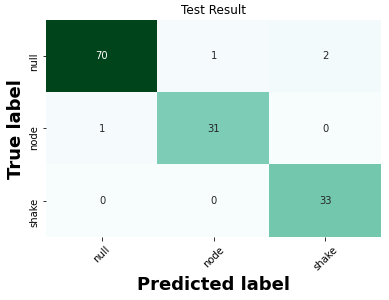
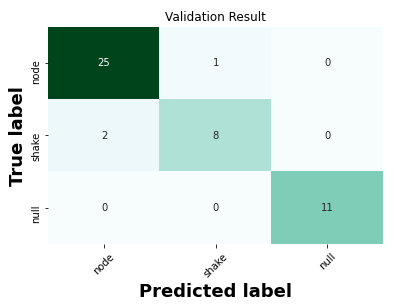
1. I extract features by leveraging library “tsfel”[[1]](#footnote-1). The feature selection method is inspired from this paper[[2]](#footnote-2). I extract various supported features, then I use it to make predictions. Hence, I measure the prediction power of each feature using Gini importance coefficient and I choose the 10 top 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. I train three different classifiers: SVM, Decision Tree (DT), and Random Forest (RF). I split the data into (training + test dataset) and (validation dataset) with ratio 9:1. Further, I use k-fold stratified cross validation in the training phase. I choose k=3 because the number of training data is small. Moreover, I choose stratified to make sure the distribution of each gesture category is the same in each fold.

The training accuracy is 94%, 85%, and 95% for SVM, DT, and RF, respectively. Therefore, I choose RF as the classifier. The result and confusion matrix for RF are shown below (left is test set; right is validation set).

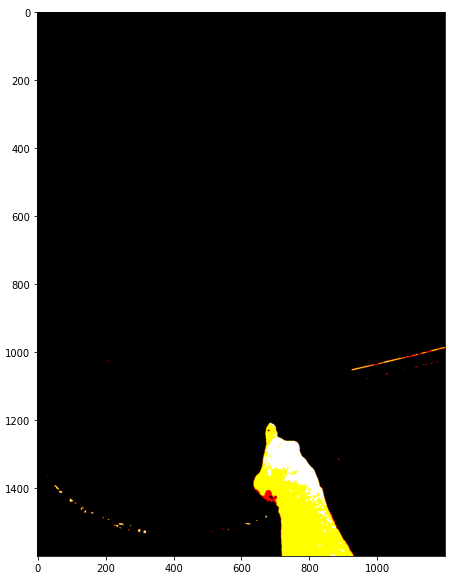
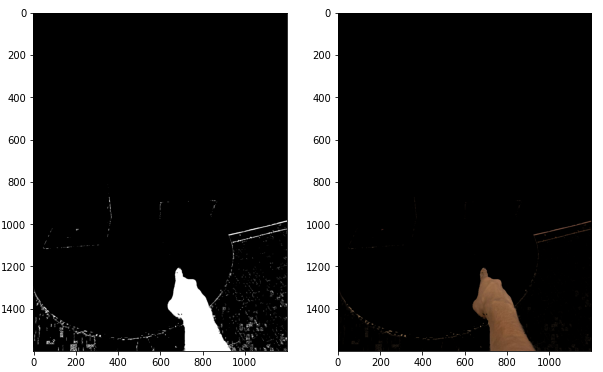
 

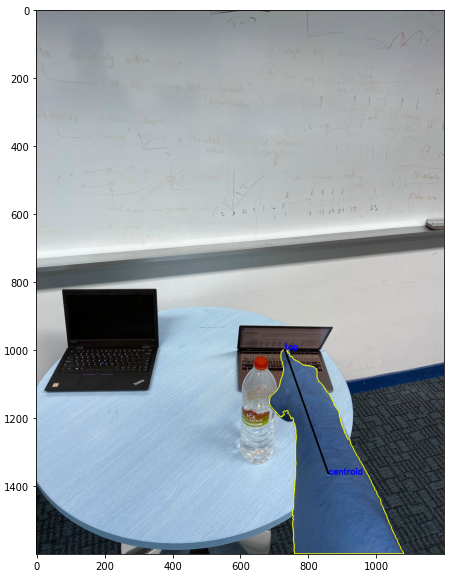
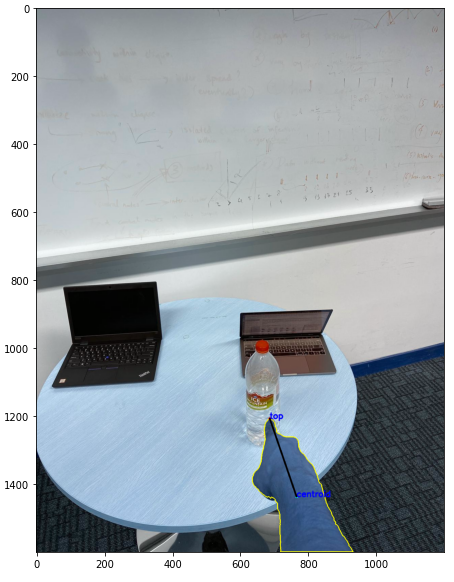
1. Pointing Resolver

I use basic image processing techniques to extract gradient of intercept of the pointing direction. My observation is accuracy of “line of best fit” method suffers if there are any outliers (I found that some points are outside the hand).

The high-level approach is: 1) detect hand 2) extract fingertip 3) draw a line which parallel to the hand. First, I convert the image from RGB to HSV to make color separation easier. Afterwards, I perform masking, blurring, and thresholding in the HSV image. The results are shown below respectively.



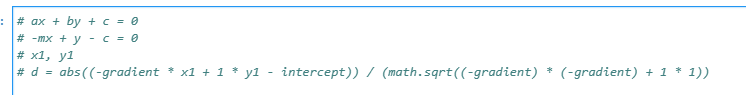
I detect hand by extracting the largest contour. I get the fingertip coordinate by extracting the most top coordinate in the contour. I also extract the centroid of the contour, so I can create a line equation using those 2 coordinates. The results are shown below.



1. Target Resolver

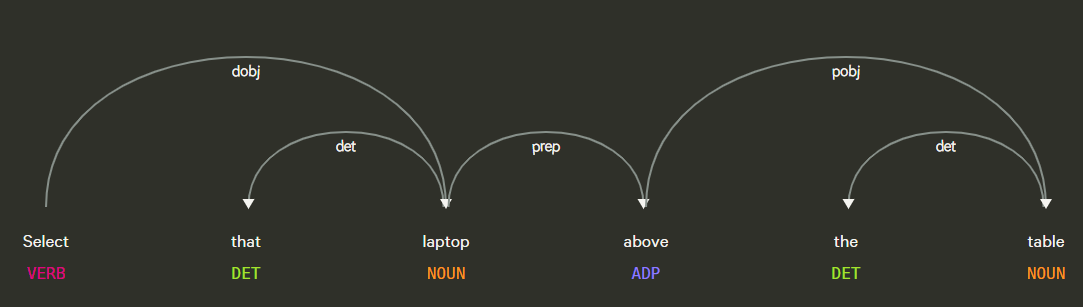
Input: list of detected objects, gradient + intercept, command text

The algorithm is the following. First, I calculate the centroid of all the detected objects. Then, for each centroid, I calculate the distance between the centroid and the line equation which is obtained from the gradient and intercept.



Afterwards, I put maximum 3 objects which have the shortest distance in the list of distance. I only choose 3 because there will be no more than 3 objects that can be ambiguously close to each other due to limited physical space (if the 4th object exists, the distance will be too far away). If there no objects that have ambiguous position, then it does not matter because the aim of this step is to narrow down the possible output.

Next, I solve the ambiguity by applying NLP techniques to the command input. Leveraging the pretrained spacy[[3]](#footnote-3) model, I tokenize the command text, extract the token with dependency “direct object” using dependency parsing technique. The example of dependency parsing output is shown below. The direct object is indicated by “dobj” which corresponds to “laptop”.



If “direct object” is found, then the output (I will refer as command object later) will be “laptop”. Otherwise, I will handle it in the case 3 below.

Afterwards, I match the command object with the object name in the distance list. If match is found, then append it to 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, the distance list will contain [laptop, cup, bottle], the command object is “laptop”, then the result list will contain [laptop] and the algorithm will return laptop.

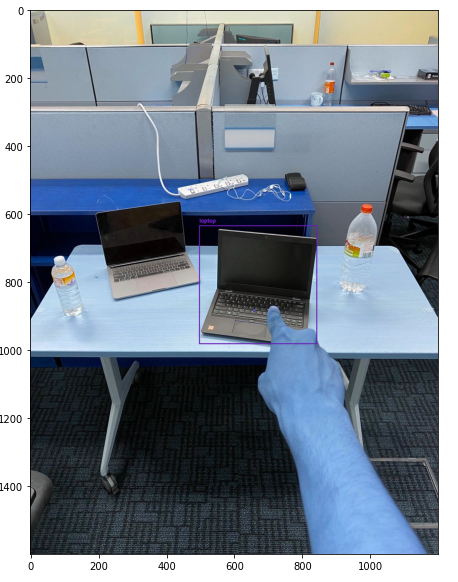
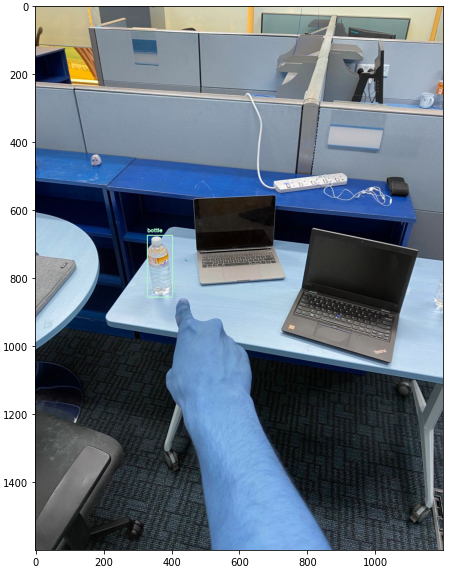
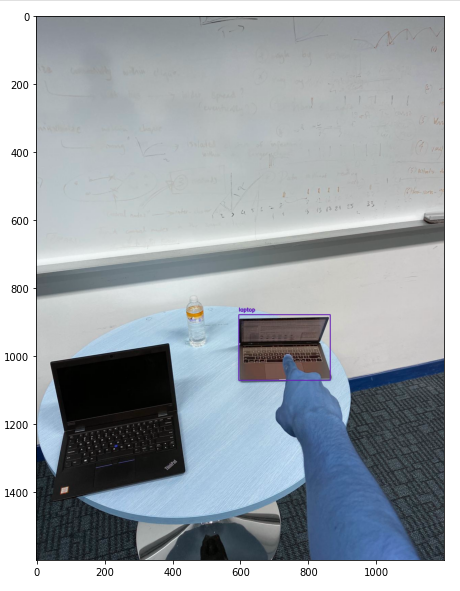
1. Command object matches multiple objects in the distance list OR the detected object is less than or equal 3 (in this case, the distance list will be the same as the detected objects list).

For example, there are 2 objects: 1 laptop on the right and 1 laptop on the left. The distance list will contain [laptop, laptop]. Suppose that the user points to the laptop on the right side and the command object is “laptop”. The result list will contain [laptop, laptop]. In this case, the algorithm will return object with the closest distance to the line, which is laptop on the right side.

1. Voice command is invalid due to reasons such as unclear human tone or pronunciation.

For example, there are 2 objects: laptop (right), laptop (left), and bottle. Suppose that the user points to a laptop on the right side and the speech recognizer recognizes it as “fliptop”. In this case, the command object will not have any match in the distance list which contains [laptop, laptop]. Therefore, the algorithm will return the object with the closest distance to the line. Because user points to the laptop on the right side, then it has a closer distance than laptop on the left side. Hence, the algorithm will return laptop.

Some output examples are shown below. The command 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)
2. R. Doshi, N. Apthorpe, and N. Feamster, “Machine Learning DDoS Detection for Consumer Internet of Things Devices,” 2018 IEEE Security and Privacy Workshops (SPW), pp. 29–35, May 2018, doi: 10.1109/SPW.2018.00013. [↑](#footnote-ref-2)
3. https://spacy.io/ [↑](#footnote-ref-3)