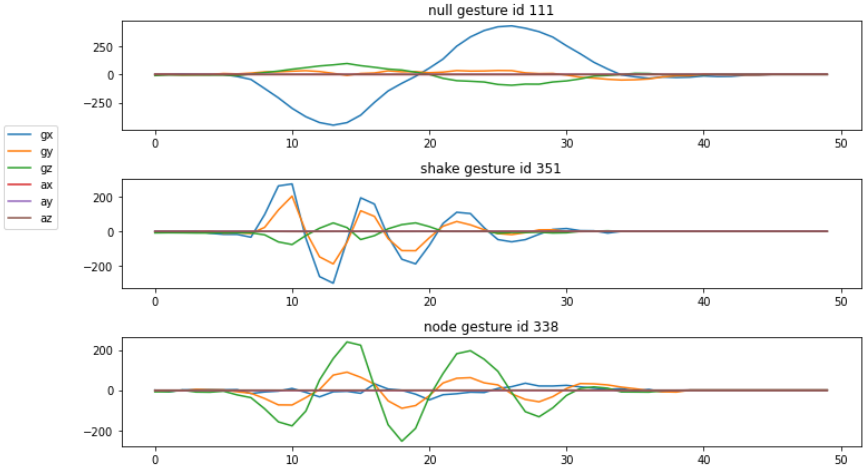
Design Document

A. 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.



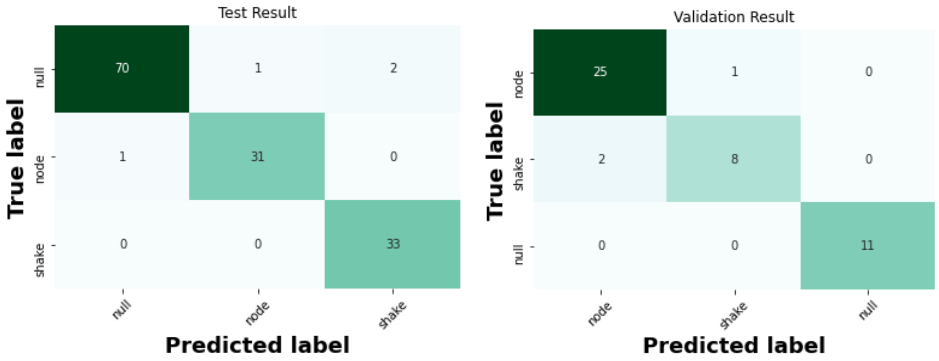
2. I extract various features, then measure the prediction power of each feature using Gini importance coefficient. In the end, I choose top 10 most important features. The features can be divided into 3 categories: statistical, temporal, and spectral. The chosen features are the following:
   - Spectral: fundamental frequency, median frequency, spectral entropy, spectral kurtosis, spectral slope, spectral spread, wavelet entropy
   - Statistical: interquartile range
   - Temporal: autocorrelation, peak to peak distance

3. 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).

	precision	recall	f1-score	support
null	0.99	0.96	0.97	73
node	0.97	0.97	0.97	32
shake	0.94	1.00	0.97	33
accuracy			0.97	138
macro avg	0.97	0.98	0.97	138
weighted avg	0.97	0.97	0.97	138

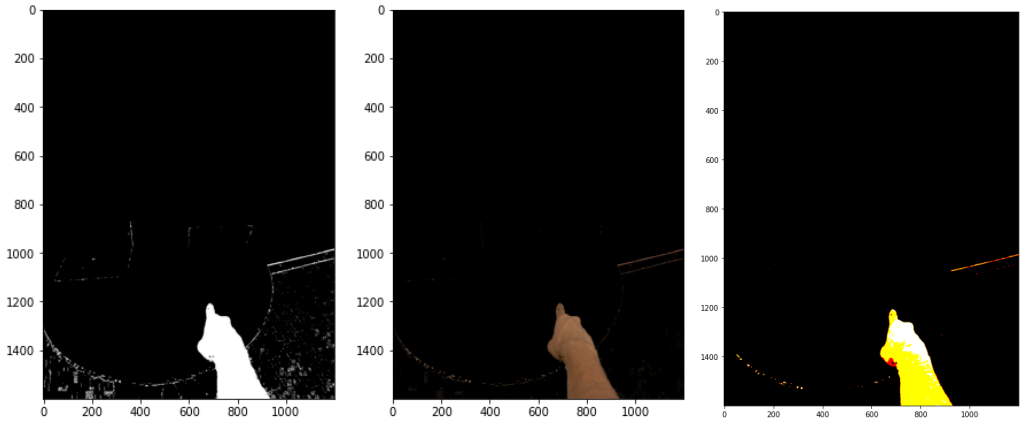
	precision	recall	f1-score	support
null	0.93	0.96	0.94	26
node	0.89	0.80	0.84	10
shake	1.00	1.00	1.00	11
accuracy			0.94	47
macro avg	0.94	0.92	0.93	47
weighted avg	0.94	0.94	0.94	47



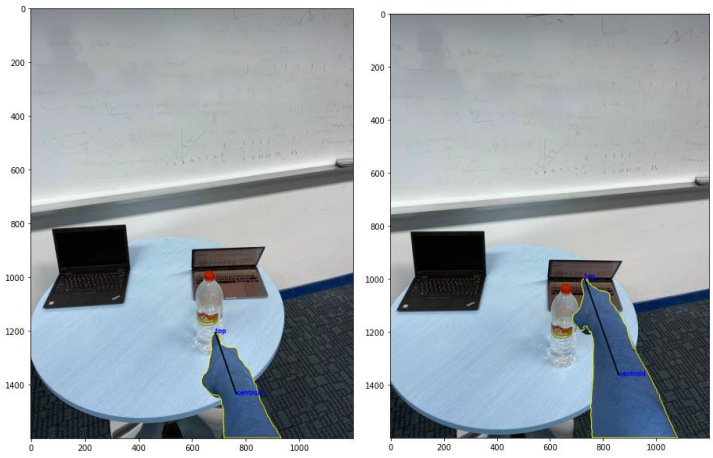
B. Pointing Resolver

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.



C. Target Resolver

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

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. I put the result in the *list of distance* in a shorted order.

```
# ax + by + c = 0
# -mx + y - c = 0
# x1, y1
# d = abs((-gradient * x1 + 1 * y1 - intercept)) / (math.sqrt((-gradient) * (-gradient) + 1 * 1))
```

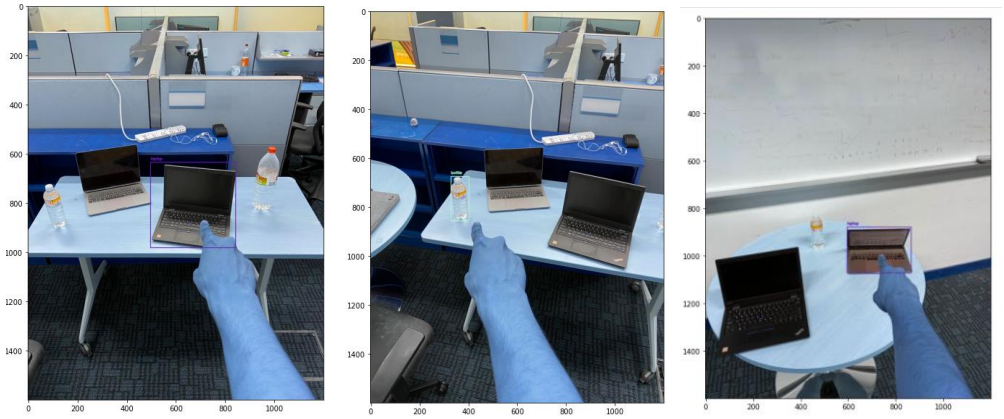
Next, I solve the ambiguity by applying NLP techniques to the command input. There are 2 cases regarding the command input. First, if the command input is composed of one word, I will match it with a predefined word dictionary. Second, in the case where the command input is more than one word, I tokenize the command text, extract the token which has Part of Speech (PoS) value as "NOUN". For example, in phrase "Select that laptop", the word with noun PoS is "laptop". I refer this object as *command object*. It is worthy to note that I can have multiple nouns in one phrase. If such case happens, I will handle it later. The first case is needed because PoS only available in a phrase, not word level. Afterwards, I match the command object with the object name in the *list of distance*. If match is found, then append it to the *list of result*. There are three possible 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, *list of distance* will contain [laptop, cup, bottle, remote, chair], the *command object* is "laptop", then the *list of result* will contain [laptop] and the algorithm will return laptop.

2. Multiple nouns exist in the command text
   For example, there are 2 objects: 1 laptop (right) and 1 laptop (left). *List of distance* will contain [laptop, laptop]. Suppose that the user points to the laptop on the right side and the *command object* is "laptop". The *list of result* 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. This approach can work because I assume that user pointing direction always tend to be close to the targeted object.

3. Voice command is invalid due to reasons such as 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 output is “fliptop”. In this case, the command object will not have any match in the *list of distance* 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, the algorithm will return laptop.

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



D. AR Object Rendering in Android

I decide at which side the object will be rendered. I calculate the centroid of the bounding box and divide the camera fragment in the center. If the centroid is less than the middle point, I will render in the left side. Otherwise, render in the right side.

Drawing the bounding is done by overlaying empty ImageView in front of camera fragment. Subsequently, I create empty bitmap, draw the bounding box in the bitmap using android canvas, then set the bitmap into the ImageView.

I use the provided object to render 3D object in the screen. If the gesture is “Nodding”, it will render thumbs up pose. If the gesture is “Shaking”, it will render fist pose. If the gesture is “Null”, it will not render anything. The screen is cleared when the user presses reset button. The results are shown below.