DESIGN FILE FOR IS708 PROJECT

a)i) Gesture Classifier

Time Series Feature Extraction Library (TSFEL) is used for feature extraction from the input data. The features extracted are simple statistical values which are :

1. Interquartile range.
2. Max value
3. Mean value
4. Median value
5. Min value
6. Root mean square
7. Standard deviation
8. Variance

These features were selected as they are believed to be an accurate presentation of the data and are used as an input to the classifier. The model selected to perform classification is a random forest. The reason random forest is selected as the classifier is due to the random property of the model which mitigates the factor of overfitting. Random forests is a model that consists of decision trees which split on a subset of features every split. When growing the trees, additional randomness is added to the model. Random forest searches for the best feature among a random subset of features instead of searching for the most important feature when splitting the node.

```
*** Feature extraction finished ***
     0_Interquartile range     0_Max  ...  5_Standard deviation   5_Variance
0                 0.959229  0.000000  ...             25.625386   656.660405
1                 0.140442  0.000000  ...             59.763956  3571.730451
2                 0.131958  0.000000  ...             31.189312   972.773214
3                 0.324677  0.000000  ...             50.676149  2568.072111
4                 0.302948  0.000000  ...             20.802056   432.725531
..                     ...       ...  ...                   ...          ...
457               0.366821  0.000000  ...             24.214236   586.329225
458               0.361359  0.000000  ...             15.003519   225.105592
459               0.190643  0.000000  ...             46.934772  2202.872806
460               0.185852  0.000000  ...             21.639548   468.270051
461               0.034943 -0.897095  ...             96.357272  9284.723822
```

Figure 1: The features extracted from input data.

```
Gesture code: 1
*** Feature extraction started ***
Progress: |████████████████████████████████████████████████| 100% Complete

*** Feature extraction finished ***
Nodding
```

Figure 2: The gesture classifier returning the correct gesture type when it is fed input data of the Nodding class

```
Gesture code: 2
*** Feature extraction started ***
Progress: |████████████████████████████████████████████████| 100% Complete

*** Feature extraction finished ***
127.0.0.1 - - [28/Apr/2021 14:01:43] "Shaking
```

Figure 3: The gesture classifier returning the correct gesture type when it is fed input data of the Shaking class

ii) Data preprocessing

Due to the data being different in size for every file, preprocessing must be done on the data to be fed into the classifier. First, the timestamp and gesture types columns are removed as they are not needed in feature extraction. Next, a fixed length of 50 is determined for the size of each input file. If the input file is less than 50 rows, the empty rows are padded with zeros and if they are larger than 50 rows, they are trimmed down to 50 rows. 50 rows has been determined to be the best length for the input data as sufficient info will still be retained despite the trimming, and to minimize the amount of dummy data required to ensure the information is not affected.

```
# Drops the first and last columns of the data as we do not need them for the feature extraction - Timestamp and gesture type
# Ensure the data being fed in has these columns, else features will not be extracted properly
raw_segment_dataframe.drop(raw_segment_dataframe.columns[len(raw_segment_dataframe.columns) - 1], axis=1,
                           inplace=True)

raw_segment_dataframe.drop(raw_segment_dataframe.columns[0], axis=1, inplace=True)

# If the length of the csv files are less than 50 rows, pad the empty blocks with zeros
if len(raw_segment_dataframe) < 50:
    for i in range(len(raw_segment_dataframe), 50):
        raw_segment_dataframe.loc[i] = 0

# If the length is greater than 50 rows, trim it to 50 rows
else:
    raw_segment_dataframe = raw_segment_dataframe.head(50)
```

Figure 4: Snippet of code showing how data preprocessing is performed on the input data

iii) Training methodology

A simple 60:40 split of the input data into training and test sets are used to partition the data. This splitting is chosen to prevent the random forest from overfitting on the data and to leave sufficient data over for testing.

```
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.4, random_state=10)
```

Figure 5: Snippet of code showing how the input data is split into training and test sets

iv) Classification report

As shown in Figure 6, the random forest provides very high accuracy, precision and recall on all 3 gestures, showing that random forest was a good choice for the classifier. **(Note : NO = Null gesture)**

```
*** Feature extraction finished ***
Accuracy: 95.675675675675%
              precision    recall  f1-score   support

          NO       0.95      0.97      0.96        95
    NODDING        0.96      0.98      0.97        48
    SHAKING        0.97      0.90      0.94        42

    accuracy                           0.96       185
   macro avg       0.96      0.95      0.96       185
weighted avg       0.96      0.96      0.96       185
```

Figure 6: Classification report

b) Pointing Direction Resolver

The pointing direction resolver takes a set of keypoints and uses the points to create a line of best fit and obtains the gradient and intercept. A formula for line of best fit was used to obtain the gradient and intercept. We first calculate the means of the x and y values and then use those values to calculate the gradient and intercept.

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad m = \frac{\sum_{i=1}^{n} (x_i - \overline{X})(y_i - \overline{Y})}{\sum_{i=1}^{n} (x_i - \overline{X})^2}$$

$$\overline{Y} = \frac{\sum_{i=1}^{n} y_i}{n} \qquad b = \overline{Y} - m\overline{X}$$

Figure 7,8,9: Calculating the mean of the x and y values, then using those values to calculate the gradient and the intercept

```
# Finding the total x values and y values and obtaining the mean of them.
for i in range(rows):
    x = points2[i][0]
    total_x = total_x + x

for i in range(rows):
    y = points2[i][1]
    total_y = total_y + y

mean_x = total_x / rows
mean_y = total_y / rows

total = 0
totalX2 = 0

# Calculating the gradient and y-intercept of the line of best fit
for i in range(rows):
    x = points2[i][0]
    y = points2[i][1]
    total = total + ((x - mean_x) * (y - mean_y))
    totalX2 = totalX2 + ((x - mean_x) ** 2)

gradient = total / totalX2
intercept = mean_y - (gradient * mean_x)
```

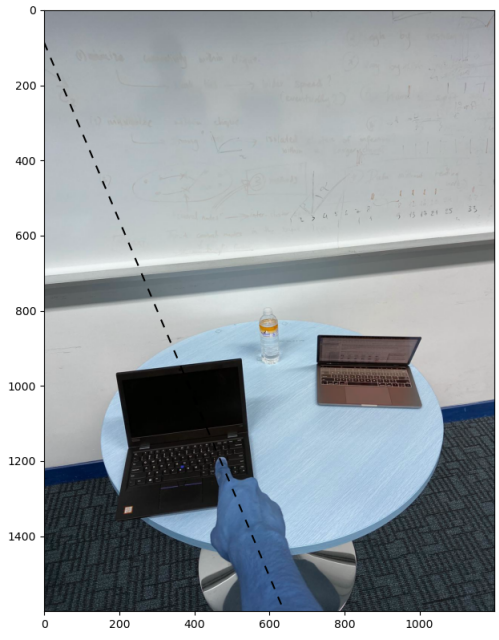

Figure 10,11: Code showing how the gradient and intercept is calculated and plotting the line of best fit

c) Target resolver

In the target resolver, the inputs from the object detector and pointing resolver are used to determine the object the hand is pointing to. The midpoint of the bounding box for each object detected is obtained and the perpendicular distance from the midpoint of the bounding box and the line of best fit is obtained. The object with the shortest distance between the bounding box and the line of best fit will be selected as the target.

d) Virtual Object Renderer and displaying the final scene image

When the android application receives the data from the target resolver, it will draw a bounding box around the object specified. This is done using Canvas. Afterwards, when the Android application receives the gesture type from the server API, it will render an object (Cube if “Nodding”, Sphere if “Shaking”) on the side where the bounding box is (Left/Right side of the screen)

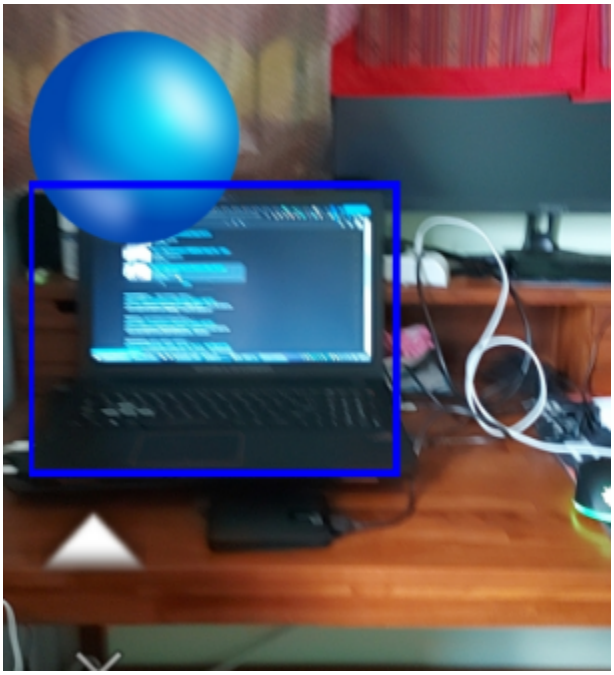

Figure 12: Bounding box and object rendered on the laptop which is the targeted object

```
/*   Bounding box rendering    */
//Preparing the canvas for drawing bounding box
int w = (int)width;
int h = (int)height;

//Bitmap setup and assignment to canvas
Bitmap.Config conf = Bitmap.Config.ARGB_8888;
Bitmap bmp = Bitmap.createBitmap(w, h, conf); // this creates a MUTABLE bitmap
canvas = new Canvas(bmp);

// Draws the bounding box using the coordinates from the server API
Paint myPaint = new Paint();
myPaint.setStyle(Paint.Style.STROKE);
myPaint.setColor(Color.rgb( red: 0,  green: 0,  blue: 255));
myPaint.setStrokeWidth(10);
canvas.drawRect(xLeft, yTop, xRight, yBottom, myPaint);
boundingBoxDraw.setImageBitmap(bmp);
```

```
public void respondToGesture(String gesture) {
    String nodding = "\"Nodding\"";
    String shaking = "\"Shaking\"";

    //If bounding box is on the left side of the screen, we render
    if (targetScreenArea.equals("LEFT"))
    {

        //If the gesture is nodding(gesture 1), we draw a cube and
        if (gesture.equals(nodding))
        {
            cubeLeft.setVisibility(View.VISIBLE);
            sphereLeft.setVisibility(View.GONE);
            cubeRight.setVisibility(View.GONE);
            sphereRight.setVisibility(View.GONE);
        }
```

Figure 13: Snippet of code performing the bounding box and object rendering