

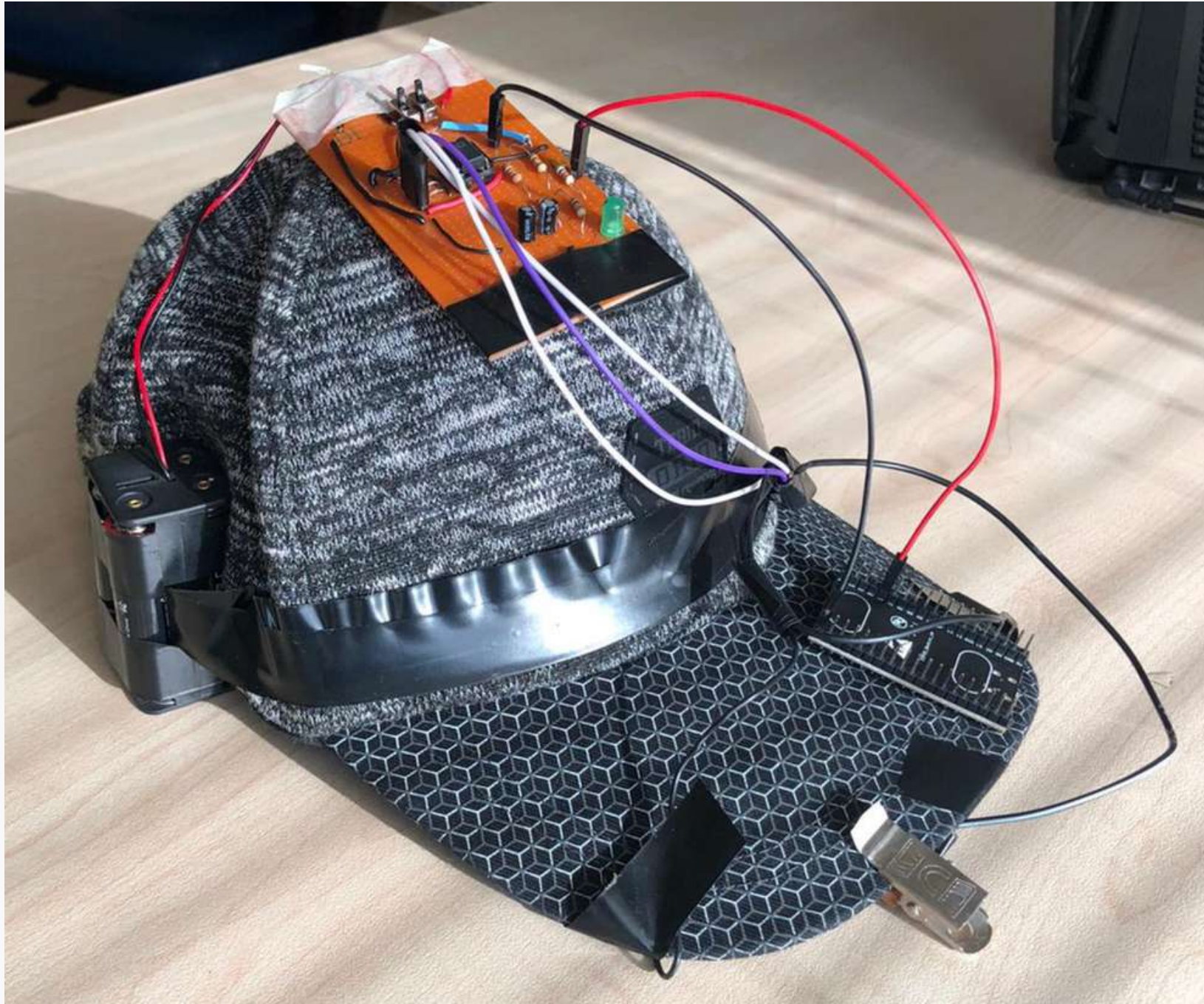


CSE396

Final Presentation

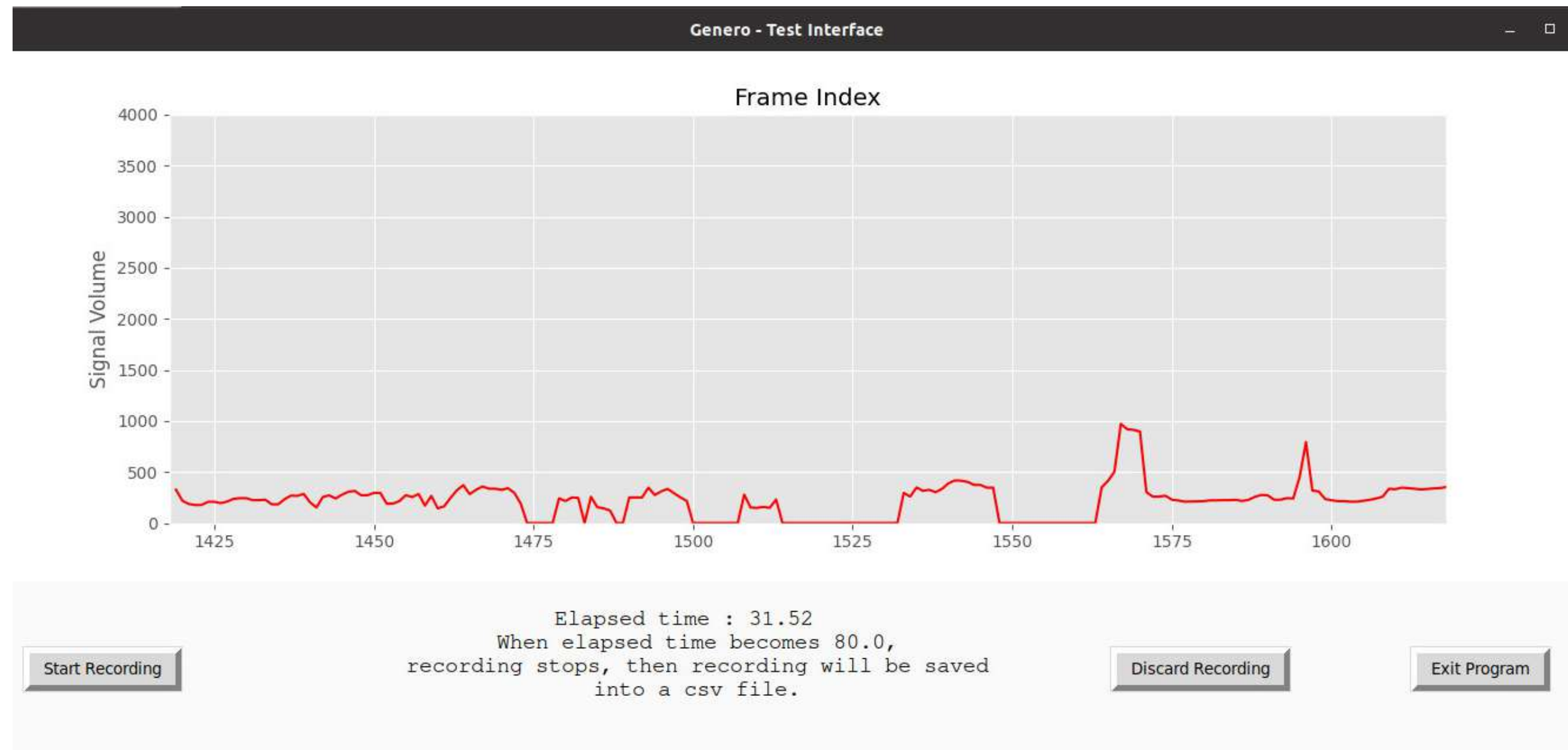
GENERO

HARDWARE



TEST INTERFACE

- We used Python to make test interface
- For interactive objects (buttons, frame, etc.) we used Tkinter library.
- For the graph we used matplotlib library.
- Frame index shows which frame is received at the moment
- Signal value represents the size of the signal



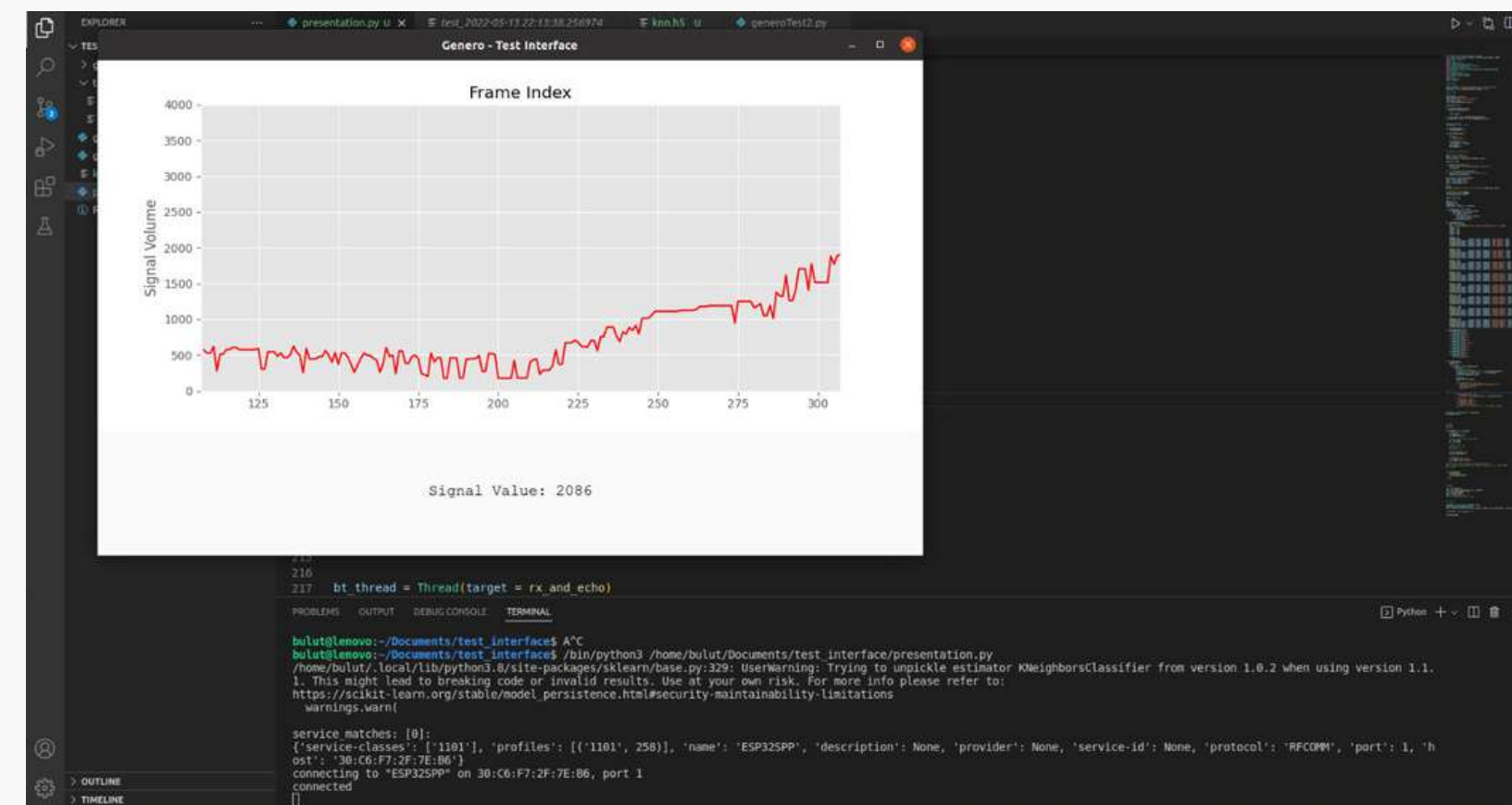
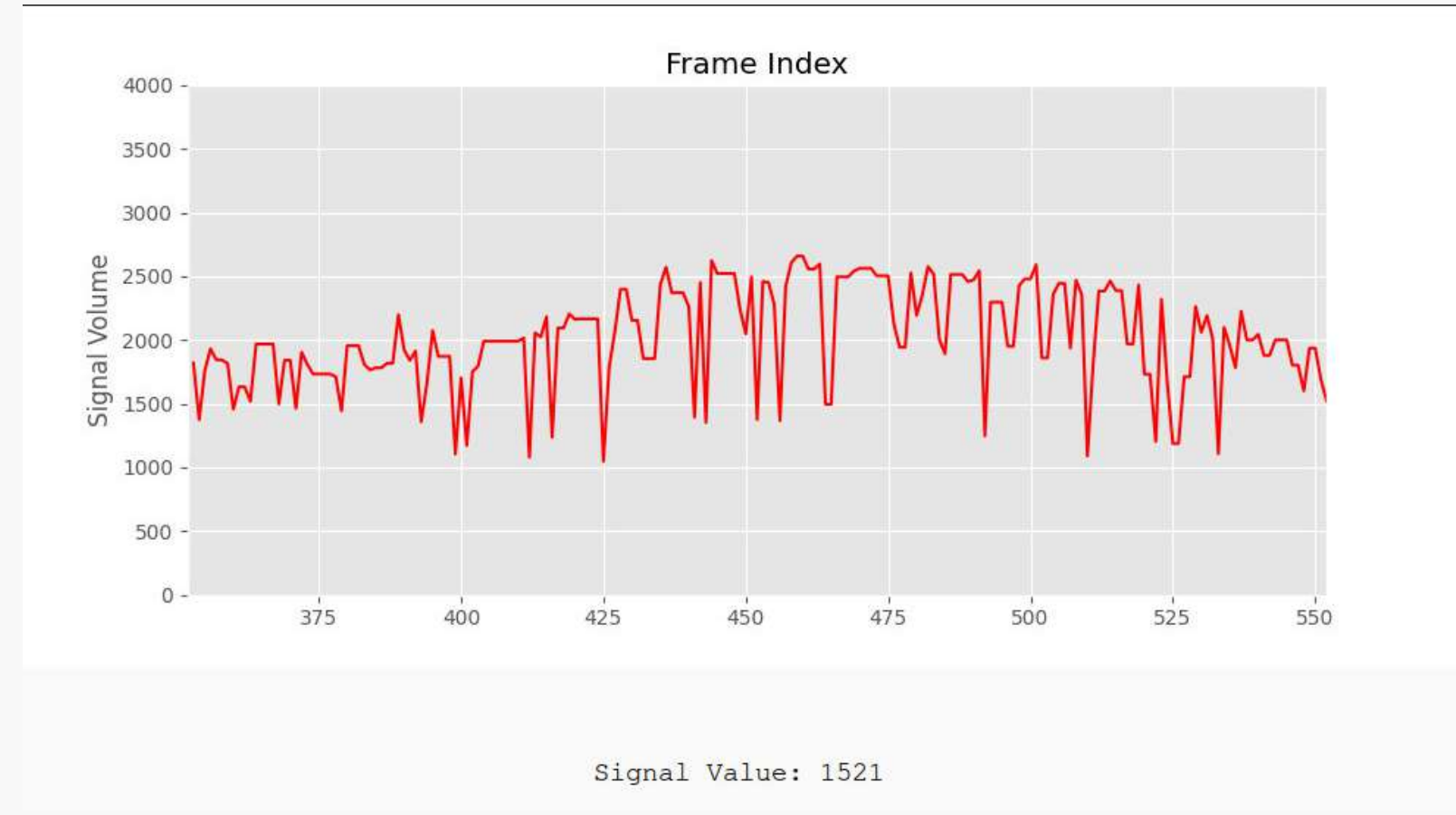
DATA ACQUISITION

To collect data we used test interface and a video clip from our unity app simultaneously together.

SUBJECT GROUP QUALITIES

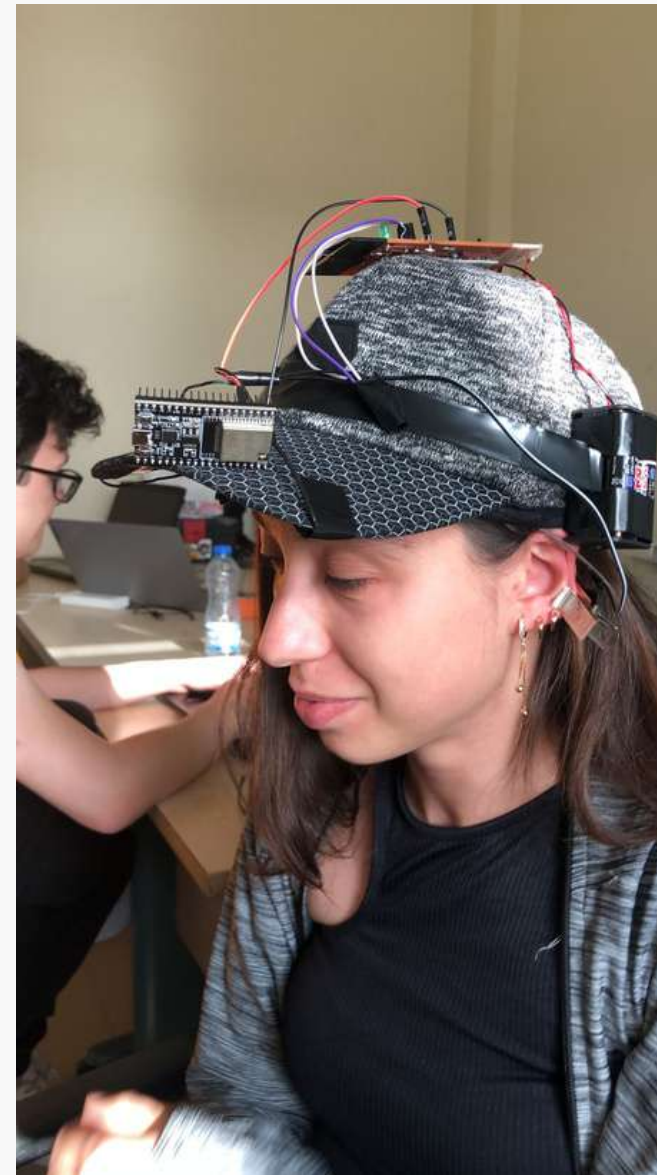
AGE : 19 - 24

NUMBER : 115 PEOPLE



HOW DO WE COLLECT DATA

- Put the hat on the head (copper bands must be touching to forehead)
- Reference band must touch to the ear
- Ask people to focus and do some hand movements
- Receive and record the signals formed in the brain with the eeg sensor



A CLIP FROM DATA COLLECTION



DATA PRE PROCESSING

- We used machine learning methods to make the data meaningful
- We used keras python library
- All data from every person is in a file separately and results are labeled as parmak1_repeat1, parmak1_repeat2 etc.
- We did a pre-processing to make all the data ready for the learning model

DATA PROCESSING MODELS

- We used machine learning models to make the data meaningful
- Z Score Normalization
- test_size : 0.25
- **THESE MODELS:**
 - Decision tree
 - K-Nearest Neighbors
 - Best random forest model
 - Deep neural network model

RESULTS OF MODELS

● DECISION TREE (14.3%)

Z Score Normalization

Bütün özellikler dahil olacak şekilde sınıflandırma

Decision Tree

```
In 11 1 from sklearn.tree import DecisionTreeClassifier
      2 from sklearn.metrics import (precision_score, recall_score, f1_score, accuracy_score, mean_squared_error, mean_absolute_error, roc_curve,
      3 classification_report, auc)
      4
      5 model = DecisionTreeClassifier()
      6 model.fit(x_train, y_train)
      7
      8
      9 y_pred = model.predict(x_test)
     10 accuracy = accuracy_score(y_test, y_pred)
     11 recall = recall_score(y_test, y_pred, average="weighted")
     12 precision = precision_score(y_test, y_pred, average="weighted")
     13 f1 = f1_score(y_test, y_pred, average="weighted")
     14
```

```
15 print("accuracy")
16 print("%.3f" %accuracy)
17 print("precision")
18 print("%.3f" %precision)
19 print("recall")
20 print("%.3f" %recall)
21 print("f1score")
22 print("%.3f" %f1)
```

```
▼ accuracy
0.143
precision
0.145
recall
0.143
f1score
0.143
```

RESULTS OF MODELS

KNN (30%)

```
main.ipynb X
home > bulut > Desktop > proje_yedek > test > main.ipynb > # setup
+ Code + Markdown | ▶ Run All | Clear Outputs of All Cells | Outline ...

KNN

n_neighbors = list(range(1,31))

optimal_n_neighbors = 0
best_acc = 0
for i in n_neighbors:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    if best_acc < knn.score(x_test,y_test):
        best_acc = knn.score(x_test,y_test)
        optimal_n_neighbors = i

print("Optimal N-Neighbors: {}, Accuracy: {}".format(optimal_n_neighbors,best_acc))

... Optimal N-Neighbors: 3, Accuracy: 0.3

optimal_weights = 'uniform'
best_acc = 0
weights = ['uniform', 'distance']
for i in weights:
    knn = KNeighborsClassifier(n_neighbors = 3, p = 2, weights = i)
    knn.fit(x_train, y_train)
    if best_acc < knn.score(x_test,y_test):
        best_acc = knn.score(x_test,y_test)
        optimal_weights = i

print("Optimal Weights: {}, Accuracy: {}".format(optimal_weights,best_acc))

... Optimal Weights: uniform, Accuracy: 0.3

Best K-NN Model

knn = KNeighborsClassifier(n_neighbors = 3, p = 2)
knn.fit(x_train, y_train)
print("Accuracy: {}".format(knn.score(x_test,y_test)))
```

BEST RANDOM FOREST (20%)

```
File Edit Selection View Go Run Terminal Help
main.ipynb X
home > bulut > Desktop > proje_yedek > test > main.ipynb > # setup
+ Code + Markdown | ▶ Run All | Clear Outputs of All Cells | Outline ...

max_depth = [5, 8, 15, 25, 30]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth)

gridF = GridSearchCV(RandomForestClassifier(), hyperF, cv = 3, verbose = 1,
                    n_jobs = -1)
bestF = gridF.fit(x_train, y_train)

... Fitting 3 folds for each of 25 candidates, totalling 75 fits

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:676: UserWarning: The least populated class in y
warnings.warn(

# print best parameter after tuning
print(bestF.best_params_)

print('\n')

# print how our model looks after hyper-parameter tuning
print(bestF.best_estimator_)

... {'max_depth': 5, 'n_estimators': 300}

RandomForestClassifier(max_depth=5, n_estimators=300)

Best Random Forest Model

import numpy as np
clf = RandomForestClassifier(max_depth=5, n_estimators=300)

clf.fit(x_train, y_train)

clf.score(x_test, y_test)

... 0.2
```


RESULTS OF MODELS

● DNN (10.78%)

- activation : 'softmax'
- optimizer : 'adam'
- loss : 'categorical_crossentropy'
- epochs : 10
- batch_size : 1024

```
3450/3450 [=====] - 0s 10us/step - loss: 2.1727 - acc: 0.1229 - val_loss: 2.1226 - val_acc: 0.1200
...
3450/3450 [=====] - 0s 6us/step - loss: 2.0822 - acc: 0.1386 - val_loss: 2.0858 - val_acc: 0.1191
Epoch 51/100
3450/3450 [=====] - 0s 6us/step - loss: 2.0805 - acc: 0.1342 - val_loss: 2.0853 - val_acc: 0.1287
Epoch 52/100
3450/3450 [=====] - 0s 6us/step - loss: 2.0837 - acc: 0.1371 - val_loss: 2.0841 - val_acc: 0.1296 Epoch 53/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0826 - acc:
0.1359 - val_loss: 2.0836 - val_acc: 0.1139 Epoch 54/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0861 - acc: 0.1165 - val_loss: 2.0871 - val_acc: 0.1061 Epoch 55/100 3450/3450
[=====] - 0s 7us/step - loss: 2.0830 - acc: 0.1258 - val_loss: 2.0850 - val_acc: 0.1217 Epoch 56/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0820 - acc: 0.1348 -
val_loss: 2.0835 - val_acc: 0.1226 Epoch 57/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0793 - acc: 0.1400 - val_loss: 2.0849 - val_acc: 0.1026 Epoch 58/100 3450/3450
[=====] - 0s 6us/step - loss: 2.0824 - acc: 0.1241 - val_loss: 2.0864 - val_acc: 0.1043 Epoch 59/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0820 - acc: 0.1313 -
val_loss: 2.0843 - val_acc: 0.1217 Epoch 60/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0787 - acc: 0.1333 - val_loss: 2.0841 - val_acc: 0.1183 Epoch 61/100 3450/3450
[=====] - 0s 6us/step - loss: 2.0804 - acc: 0.1243 - val_loss: 2.0867 - val_acc: 0.1096 Epoch 62/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0817 - acc: 0.1342 -
val_loss: 2.0836 - val_acc: 0.1096 Epoch 63/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0823 - acc: 0.1278 - val_loss: 2.0848 - val_acc: 0.1052 Epoch 64/100 3450/3450
[=====] - 0s 6us/step - loss: 2.0816 - acc: 0.1267 - val_loss: 2.0882 - val_acc: 0.1052
...
Epoch 99/100 3450/3450 [=====] - 0s 6us/step - loss: 2.0782 - acc: 0.1345 - val_loss: 2.0852 - val_acc: 0.1183 Epoch 100/100 3450/3450 [=====] - 0s 6us/step - loss:
2.0776 - acc: 0.1313 - val_loss: 2.0853 - val_acc: 0.1078
```

<keras.callbacks.History at 0x256cb1e4940>

```
y_pred = dnn.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
acc = accuracy_score(np.argmax(y_test, axis=1), y_pred)
print('DNN Test Accuracy: %.4f' % acc)
```

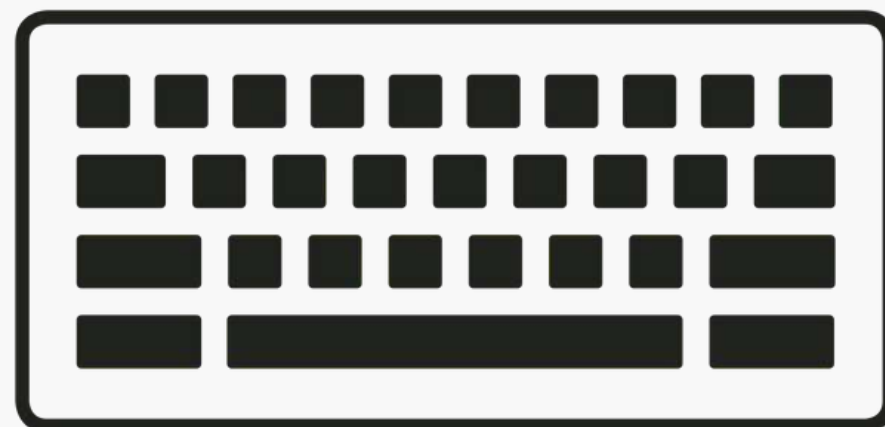
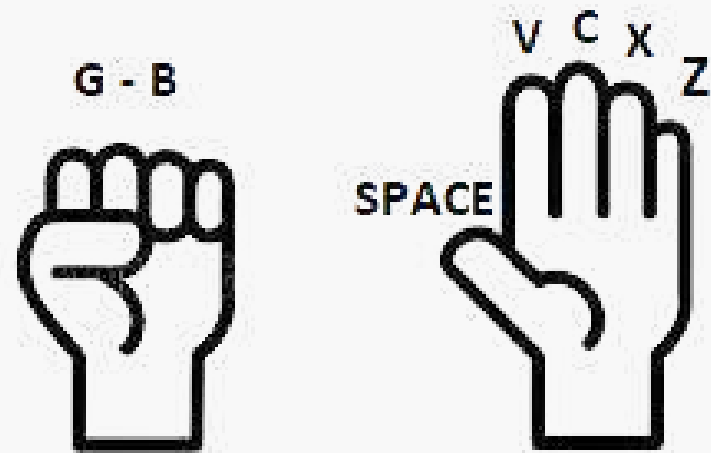
DNN Test Accuracy: 0.1078

Python

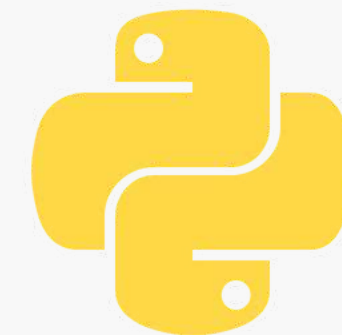
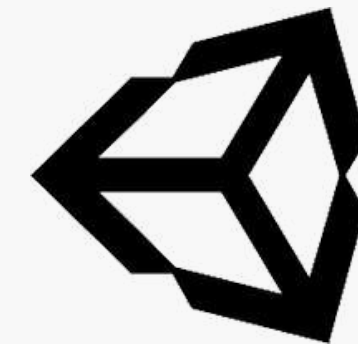
RESULT OF DATA PROCESSING

- After compiling the model we made a real-time working script that communicates with esp-32 to get signals.
- Puts all 50 inputs to the model and redirects the output to the unity simulation and simulation shows the finger movements.

HARDWARE - UNITY COMMUNICATION



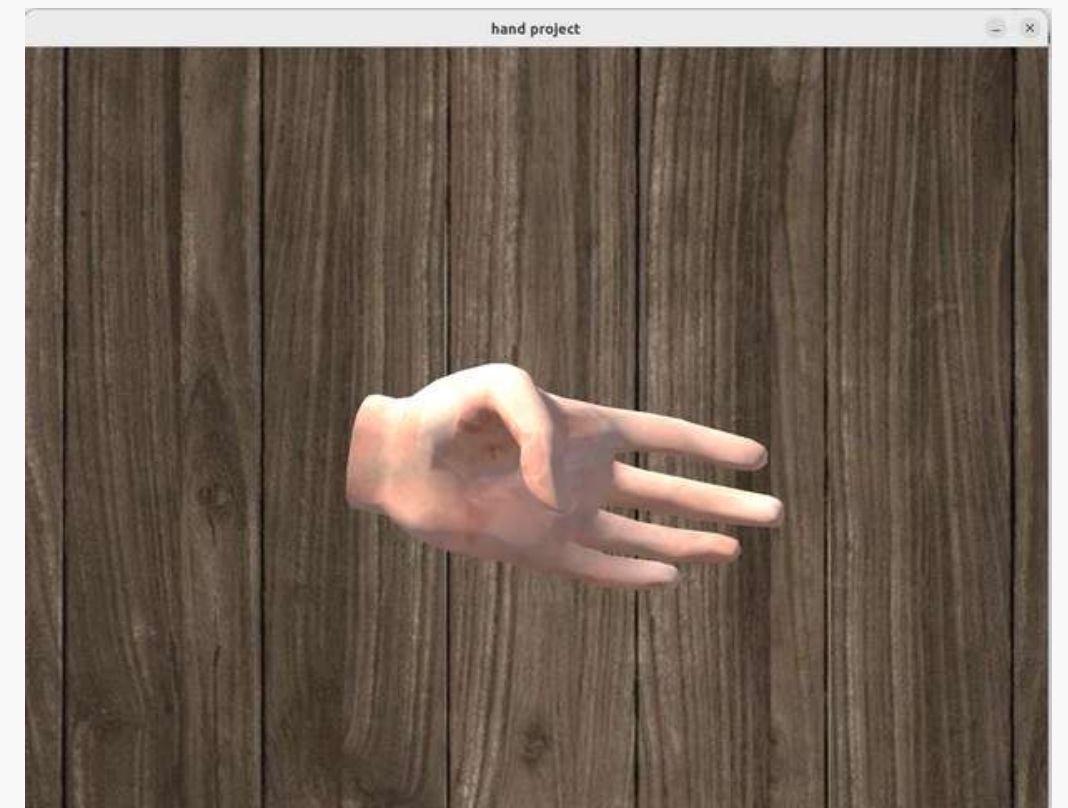
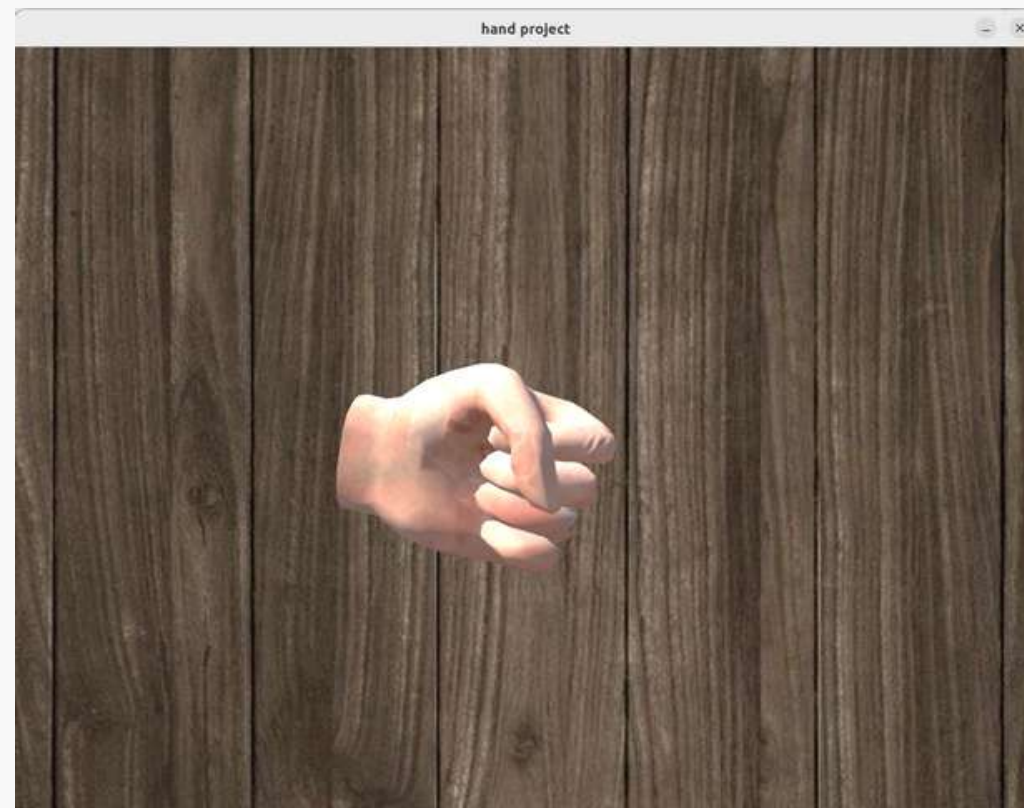
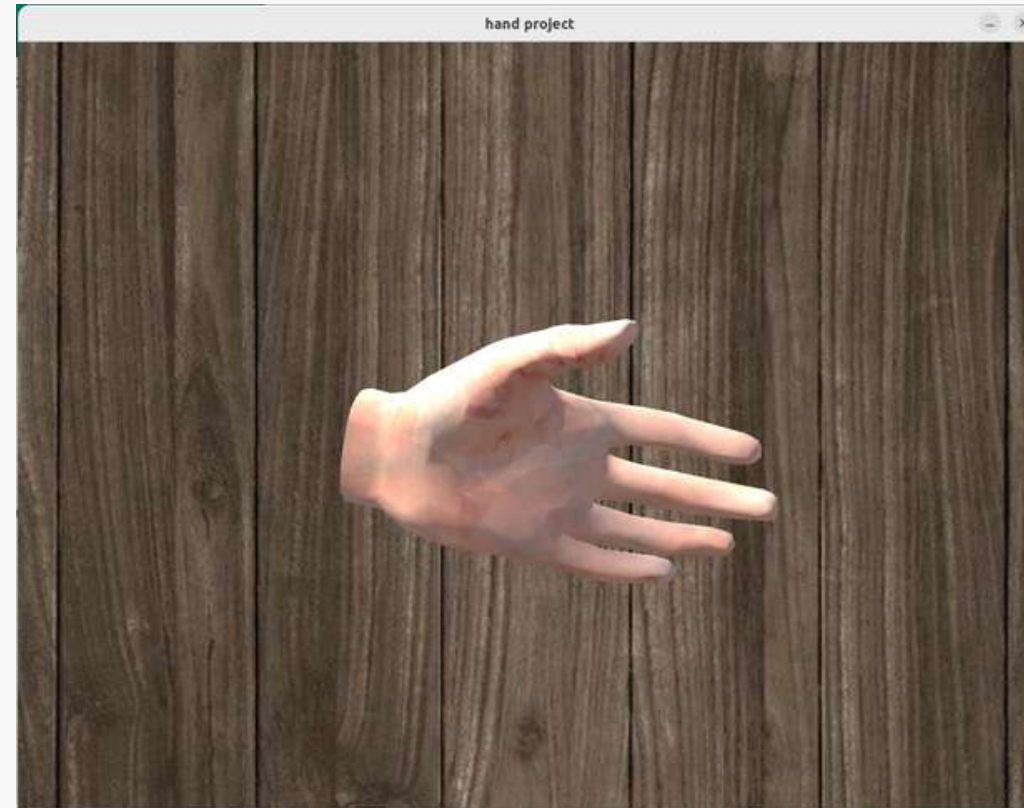
- **CLASSIFICATION OF OUTPUTS AFTER DATA PROCESSING**
- **ACCORDING TO THE DATA WE HAVE CLASSIFIED, DIFFERENT KEYS ARE PRESSED.**
- **ASSIGNING THE KEYS FOR THE NECESSARY HAND MOVEMENTS.**



UNITY

- **DIFFERENT HAND COMBINATIONS ARE SHOWN IN UNITY**

- **DIFFERENT KEYS ARE USED FOR EACH HAND MOVEMENT**



OUR TEAM

- Abdurrahman Bulut
- Ahmet Tuğkan Ayhan
- Buse Elbirgiç
- Doğukan Güler
- Mehmet Hüseyin Yıldız
- Muhammet Fikret Atar
- Salih Tangel
- Ömer Faruk Erol
- Özlem Sevri
- Yusuf Fatih Şişman
- Yusuf Talha Altun
- Yunus Emre Yumşak



THANK YOU