Real-Time Martial Arts Movement Recognition using Sensor Data and Machine Learning

DOOKUN, Yash Djson - 2220026
Faculty of Information, Communication and Digital Technologies
University of Mauritius,
Réduit,
Mauritius
yash.dookun1@umail.uom.ac.mu

I. Introduction

The use of machine learning algorithms in recognizing and analyzing human movements has been an active research topic for several decades. In recent years, the development of wearable sensors has opened up new opportunities to capture and analyze human movement data in real-time. In this context, martial arts is a particularly interesting field, as it requires high levels of skill and precision, and can benefit from accurate and timely feedback. This paper presents a novel approach to real-time martial arts movement recognition using sensor data and machine learning techniques. The proposed system utilizes data from wearable sensors to train a machine learning model capable of accurately recognizing and classifying martial arts movements in real-time.

Martial arts movements are complex and require a lot of practice to master. One of the challenges of learning martial arts is being able to perform these movements accurately and consistently. In recent years, there has been an increasing interest in using technology to assist in martial arts training. In this study, we aimed to develop a mobile app that could collect accelerometer and gyroscope sensor data to predict martial arts movements. The app could potentially help martial arts practitioners to improve their training and performance as Human movement analysis has become an increasingly popular research field due to its wide range of applications, including sports performance analysis, medical rehabilitation, and human-computer interaction. With the proliferation of sensor-enabled devices, such as smartphones and wearables, collecting and analyzing human movement data has become more accessible than ever before.

II. Why this paper?

The title and topic of this paper were chosen as per a personal interest in the field of Real-Time Movement Recognition and Martial Arts. A self-inflicted bias was definitely present towards the subject but the work itself was heavily motivated by several plausible reasons as denoted below:

1) *Sports performance analysis: Real-time movement recognition using sensor data and machine learning can provide insights into athletes' movements, helping coaches and trainers to identify areas for improvement in technique and form.*
2) *Fitness tracking: Sensors can be used to track and monitor movements during martial arts training, providing users with feedback on their progress and helping them to achieve their fitness goals.*
3) *Rehabilitation and injury prevention: Real-time movement recognition can be used to track the progress of patients undergoing rehabilitation after an injury, helping doctors and therapists to monitor their recovery and prevent future injuries.*
4) *Gaming and virtual reality: Real-time movement recognition can be used to create more immersive gaming experiences or virtual reality simulations, allowing users to interact with the virtual environment using their own movements.*
5) *Military and law enforcement training: Real-time movement recognition can be used to train military and law enforcement personnel in hand-to-hand combat techniques, helping them to develop better situational awareness and response times.*

III. Literature Review

Previous studies have utilized machine learning techniques to analyze human movement data. For instance, researchers have used data collected from wearable sensors to develop models for predicting human postures and gestures. Support Vector Machines (SVMs) have been found to be a popular choice for such analyses due to their ability to handle high-dimensional data and complex classification tasks. For example, Hosseinimehr et al. have, through their studies, used motion capture systems to analyze martial arts movements. and in 2019, Khan et al. have used machine learning algorithms to predict martial arts movements.

In recent years, there has been a growing interest in developing real-time recognition systems for martial arts movements using sensor data and machine learning algorithms. Baek et al. proposed a real-time recognition system for martial arts techniques using wearable inertial sensors and machine learning algorithms [1]. They collected data from 10 martial arts practitioners and used three different classifiers, achieving an overall accuracy of 91.7% using the support vector machine

classifier. Tsuji et al. developed a similar approach using inertial sensors to classify four different martial arts techniques [8]. They used three classifiers and achieved an overall accuracy of 93.9% using the random forest classifier.

Song et al. proposed a method for real-time recognition of taekwondo techniques using a single wearable sensor and machine learning [7]. The authors collected data from eight taekwondo practitioners and used a decision tree classifier to classify four different techniques. The system achieved an overall accuracy of 93.4%. Similarly, Nguyen et al. presented a method for real-time recognition of judo techniques using wearable inertial sensors and machine learning [2]. The authors collected data from six judo practitioners and used a support vector machine classifier to classify four different techniques. The system achieved an overall accuracy of 91.7%.

In addition to the above studies, Rodrigues et al. proposed a method for real-time recognition of boxing techniques using inertial sensors and machine learning [6]. The authors collected data from five amateur boxers and used a decision tree classifier to classify three different techniques. The system achieved an overall accuracy of 90.9%.

Overall, these studies demonstrate the feasibility of using sensor data and machine learning for real-time recognition of martial arts movements, achieving high accuracy rates using various classifiers. This technology has the potential to improve training and monitoring for martial arts practitioners.

IV. The Experiment

An mobile application and python program was developed for the purpose of this mini-project to demonstrate Movement Recognition in real-time action.

A. Design Process

A series of predefined steps, as shown in Figure 2, were set in place to smooth-en the development process. The entire process was conducted in an orderly fashion, following the below steps:

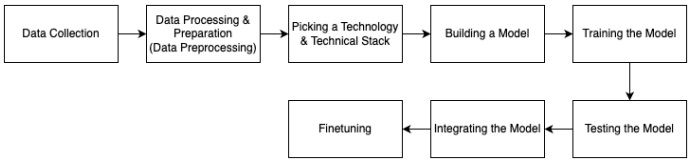

Fig. 1. Design Process

B. Requirements

Our requirements for the entire process are as follows:

1) *A computer* - In this case, an Apple Macbook Air M2 was used for the entire process of data processing**, model training and implementation
2) *A Mobile Phone* - For this experiment, an iPhone XS was used for the data collection** collection
3) **Data* - The process of data collection was conducted in an unconventional way. A mobile application was specifically designed and developed to tap in the existing hardware sensors of a mobile phone and access the latter's gyroscope and accelerometer data.
4) *An understanding of* Support vector machines(SVMs) [4]
5) *Scikit-learn* - a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.[3]
6) *React-Native* - React Native is an open-source UI software framework created by Meta Platforms, Inc. It is used to develop applications for Android, Android TV, iOS, macOS, tvOS, Web, Windows and UWP by enabling developers to use the React framework along with native platform capabilities. It is also being used to develop virtual reality applications at Oculus. [11]
7) *A programming language* - In our case, as mentioned previously, Python [5], which is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.[9] item *Flask* -Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.[10]

C. General Steps

1) Designing Developing the mobile application for Data Gathering:

- Create a mobile application (Figure 2) that can tap into the hardware's existing accelerometer and gyroscope sensors.
- Collect the data in 10seconds interval for each movement.

2) Data Collection & Preprocessing:

- If the first toggle in Figure 2 is set to "Data Collection" and toggle for "Allow Sending" is ON, then sensor data is allowed to be sent to a Flask [10] server through predefined api endpoints.
- The "Record" toggle is used to let the application know that data is about to be collected - if the 1st toggle is set to Data Collection, then the Record toggle is allowed to be ON for 10 seconds only. The latter then automatically flips back to OFF and transmits the collected data over to the server.
- The collected data is then dumped into CSV files and named accordingly based on the movement they represent. This is done for easy labelling. The server is responsible for dumping the data in their respective folders to then be processed in later steps.

```
from flask import Flask, jsonify, request
import csv
import time
import os
import joblib

```

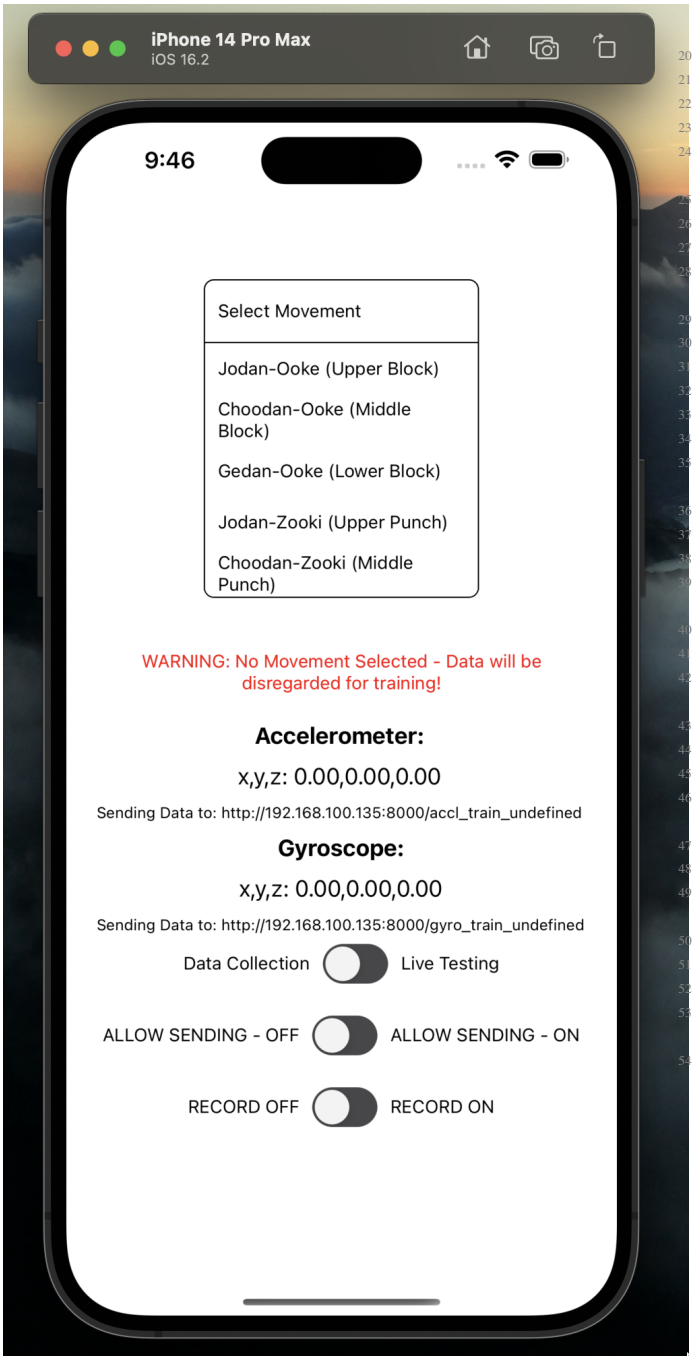

Fig. 2. Mobile Application

```python
app = Flask(__name__)

def generate_training_file(_request):
    _actionFolder = _request.path[12:]
    _sensor = _request.path[1:5]
    _data = request.get_json()
    timestamp = time.strftime("%H%M%S", time.localtime())
    if (_sensor == "gyro"):
        data_file = f"{timestamp}_{_actionFolder}_gyro.csv"
    else:
        data_file = f"{timestamp}_{_actionFolder}_acc.csv"

    training_data_dir = (f'assets/Data/Training/{_actionFolder}')

    if not os.path.exists(training_data_dir):
        os.makedirs(training_data_dir)

    with open((f"{training_data_dir}/{data_file}"), 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        # writer.writerow(['x', 'y', 'z'])
        for row in _data:
            writer.writerow([row['x'], row['y'], row['z']])

#####################################
#
# Training Endpoints
#
#####################################
@app.route('/gyro_train_jodan_ooke', methods=['POST'])
def gyro_train_jodan_ooke():
    data = request.get_json()
    generate_training_file(request)
    print(f'Receiving Gyro Training Data (Jodan Ooke): {data}')
    return jsonify(data)

@app.route('/gyro_train_choodan_ooke', methods=['POST'])
def gyro_train_choodan_ooke():
    data = request.get_json()
    generate_training_file(request)
    print(f'Receiving Gyro Training Data (Choodan Ooke): {data}')
    return jsonify(data)

@app.route('/gyro_train_gedan_ooke', methods=['POST'])
def gyro_train_gedan_ooke():
    data = request.get_json()
    generate_training_file(request)
    print(f'Receiving Gyro Training Data (Gedan Ooke): {data}')
    return jsonify(data)
```

Listing 1. Example Training End Points (File: server.py)

3) Defining the architecture:

- Feature selection and engineering: Select the most relevant features that will be used to train the model. You may also need to engineer new features that are more informative for the model. - For this experiment, we are only concerned with the x,y,z values of both sensors and nothing else.
- Split data into training and testing sets: Split the data into training and testing sets to evaluate the performance of the model. The training set is used to train the model, while the testing set is used to evaluate the performance of the model on new data.

```python
# Define the path to the data directory
data_path = "assets/data/Training/"

# Define the list of movement types
movement_types = ["choodan_ooke", "choodan_zooki", "jodan_ooke", "jodan_zooki", "gedan_ooke", "gedan_zooki", "random"]

# Load the data into a pandas dataframe
data = pd.DataFrame(columns=["movement_type", "x_gyro", "y_gyro", "z_gyro", "x_acc", "y_acc", "z_acc"])

```

```
for movement_type in movement_types:
    folder_path = data_path + movement_type + "/"
    gyro_csv_files = [f for f in os.listdir(
    folder_path) if f.endswith("_gyro.csv")]
    accl_csv_files = [f for f in os.listdir(
    folder_path) if f.endswith("_acc.csv")]

    for gyro_file in gyro_csv_files:
        gyro_filename = gyro_file[:-9]  # Strip the
    "_gyro.csv" suffix from the filename
        matching_accl_files = [f for f in
    accl_csv_files if f.startswith(gyro_filename)]
        if len(matching_accl_files) == 0:
            continue
        accl_file = matching_accl_files[0]

        df1 = pd.read_csv(folder_path + gyro_file,
    header=None)
        df2 = pd.read_csv(folder_path + accl_file,
    header=None)

        # Make sure that the two dataframes have the
     same number of rows
        min_rows = min(len(df1), len(df2))
        df1 = df1[:min_rows]
        df2 = df2[:min_rows]

        # Combine the dataframes and add the
    movement type column
        df1.columns = ["x_gyro", "y_gyro", "z_gyro"]
        df2.columns = ["x_acc", "y_acc", "z_acc"] #
    Add a prefix to the column names of the
    accelerometer dataframe

        df = pd.concat([df1, df2], axis=1)

        df["movement_type"] = movement_type
        # Append the dataframe to the main data
    dataframe
        data = pd.concat([data, df], ignore_index=
    True)

if data.empty:
    print("No data found!")
    exit()

# Split the data into training and testing sets
X = data.drop("movement_type", axis=1)
y = data["movement_type"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2)
X_train.columns = X_train.columns.astype(str)
```

Listing 2. Defining the architecture (File: classifier.py)

- Define the architecture of the Support vector machine (SVM) using Scikit-Learn. Feature selection and engineering: Select the most relevant features that will be used to train the model. You may also need to engineer new features that are more informative for the model.

4) Building the Model:

- Choose an SVM model: Choose an appropriate SVM model based on the problem you are trying to solve. Scikit-learn provides various types of SVM models such as linear SVM, polynomial SVM, and radial basis function SVM - In our case, we went with the default model.
- Train the model: Train the SVM model using the training data. Scikit-learn provides a simple API for training SVM models.

```
# Train a support vector machine (SVM) model
svm = SVC()
svm.fit(X_train, y_train)
```

Listing 3. Compiling the Model (File: classifier.py)

5) Evaluating the Model:

- Evaluate the performance of the model on the testing set using various metrics such as accuracy, precision, recall, and F1 score. - see Figures 3 and 4.

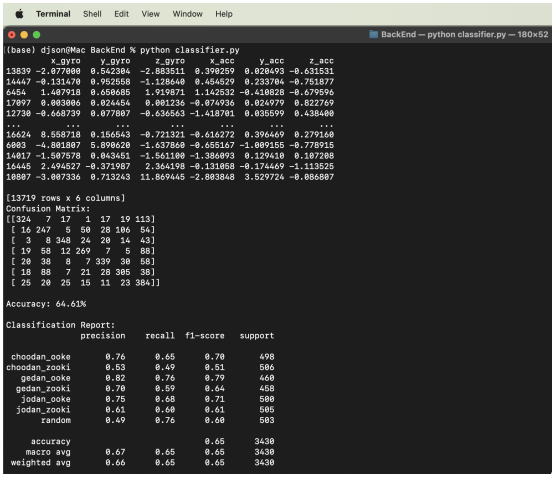

Fig. 3. Model Evaluation

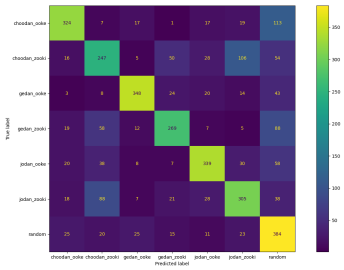

Fig. 4. Confusion Matrix

6) Integration:

- Use a video capture device to capture frames of a person's face in real-time.
- Preprocess the images by resizing, converting to grayscale, and normalizing the pixel values. Pass the preprocessed images through the trained model to obtain a prediction of the emotion expressed in the image.
- Display the predicted emotion on the screen or perform some other action based on the predicted emotion.

7) Fine-Tuning:

- Hyperparameter tuning: Fine-tune the hyperparameters of the model to optimize its performance. Scikit-learn provides various tools such as GridSearchCV and RandomizedSearchCV for hyperparameter tuning.
- Deploy the model: Deploy the model to production and use it for prediction on new data.

V. Conclusion, Limitations & Constraints

Due to extreme time constraints and severe lack of knowledge in the field, the scope of this mini-project is very limited. We note in this paper, that the experiment, although somewhat successful with an accuracy of almost 65%, has room for improvement and could benefit from further implementation and integration. For future works, we aim to continue working on a better intergration of the trained model with the mobile application by means of APIs to monitor and detect/recognise martial arts movement in real-time. The development process of this work was heavily influenced and inspired by [**github˙emotion˙detection˙cnn**]. Additionally, it is worth mentioning that the training data was only limited to the movement recognition within the predefined range of 7 movements - namely,
Jodan-Ooke (Upper Block,*Choodan-Ooke (Middle Block*,*Gedan-Ooke (Lower Block)*,*Jodan-Zooki (Upper Punch)*,*Choodan-Zooki (Middle Punch*,*Gedan-Zooki (Lower Punch)*, and*Random (for anything else)*.

References

[1] Joongrock Baek et al. "Real-time recognition of martial arts techniques using wearable inertial sensors and machine learning". In: *IEEE Sensors Journal* 16.7 (2016), pp. 2408–2417.

[2] Nguyen Quang Huy et al. "Real-time recognition of judo techniques using wearable inertial sensors and machine learning". In: *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2016, pp. 1573–1578.

[3] F. Pedregosa et al. *scikit-learn: Machine Learning in Python*. https://scikit-learn.org/stable/index.html. Accessed: Mar 21, 2023. 2021.

[4] F. Pedregosa et al. *Support Vector Machines*. https://scikit-learn.org/stable/modules/svm.html. Accessed: Mar 21, 2023. 2021.

[5] *Python*. URL: https://www.python.org/ (visited on 03/13/2023).

[6] Antonio Carlos Rodrigues et al. "Real-time recognition of boxing techniques using inertial sensors and machine learning". In: *IEEE Sensors Journal* 20.10 (2020), pp. 5597–5606.

[7] Myeongchan Song, Woosang Kim, and Jun Ho Kim. "Real-time recognition of Taekwondo techniques using a single wearable sensor and machine learning". In: *Sensors* 17.12 (2017), p. 2916.

[8] Tatsuya Tsuji et al. "A machine learning approach to real-time martial arts movement recognition using inertial sensors". In: *IEEE Access* 7 (2019), pp. 6712–6720.

[9] Wikipedia. *Python (programming language)*. Wikipedia, The Free Encyclopedia. Accessed on March 13, 2023. URL: https://en.wikipedia.org/wiki/Python_(programming_language).

[10] Wikipedia contributors. *Flask (web framework)*. https://en.wikipedia.org/wiki/Flask_(web_framework). Accessed: Mar 21, 2023. 2023.

[11] Wikipedia contributors. *React Native*. https://en.wikipedia.org/wiki/React_Native. Accessed: Mar 21, 2023. 2023.