Lab Report: **Motion Recognition using IMU Sensor Fusion**

Student Name: Charanjit Bangalore Kumar
Student ID: 12504134
Course Name: Embedded System
Instructor: Prof. Tobias Schaffer

Date: 11/06/2025

1 Introduction

This lab focuses on recognizing human motion using Inertial Measurement Unit (IMU) data from a Raspberry Pi Sense HAT. The goal is to capture sensor data, train a neural network to classify motion patterns, and deploy the trained model to perform real-time gesture recognition and respond by lighting LEDs accordingly.

2 Objectives

The main objectives of this lab are:

- Collect motion data using the Raspberry Pi Sense HAT (accelerometer + gyroscope).
- Label four motion types: `move_none`, `move_circle`, `move_shake`, `move_twist`.
- Train a neural network model using TensorFlow in Google Colab.
- Convert the trained model to TensorFlow Lite format.
- Deploy and run the model on Raspberry Pi to recognize gestures in real time.
- Display the LED matrix based on predicted gestures.

3 Methodology

This project follows a pipeline that includes data collection, pre-processing, model training, and deployment. IMU data are captured from the Raspberry Pi's Sense HAT for different gestures. This labeled sensor data is then used to train a neural network using TensorFlow. Once trained, the model is converted to TensorFlow Lite and deployed back onto the Raspberry Pi for real-time inference and LED-based gesture feedback.

3.1 Data Collection

Motion data was collected using the Raspberry Pi Sense HAT's IMU sensors (accelerometer + gyroscope). For each gesture, 50 samples were collected at a sampling rate of **50 Hz**, giving **1.0 second** of data per gesture instance. Each sample contains six features: three-axis accelerometer values and three-axis gyroscope values.

- Sensors: 3-axis accelerometer and 3-axis gyroscope
- Sampling rate: 50 Hz
- Samples per gesture: 50
- Total features per gesture: $50 \times 6 = 300$
- Data format: `[acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z]`

3.2 Data Preprocessing

No explicit preprocessing was applied to the IMU sensor data. The raw accelerometer and gyroscope readings were directly flattened into a one-dimensional array of 300 features per gesture sample and used as input for the neural network. Due to the consistency of the data collected in a controlled environment, this direct usage yielded sufficiently accurate model performance. However, for more general use cases, preprocessing steps such as normalization or filtering would typically be necessary.

3.3 Model Training

Model training is conducted using TensorFlow in Google Colab. The architecture consists of a feedforward neural network with two hidden layers and dropout for regularization.

- Input layer: 1800 neurons
- Hidden layers:
  - Dense(128, ReLU)
  - Dropout(0.2)
  - Dense(64, ReLU)
- Output layer: Dense(4, Softmax) for classifying the four gestures
- Optimizer: Adam
- Loss function: Categorical crossentropy
- Epochs: 15
- Batch size: 32

Training and validation sets are split using an 80:20 ratio. Model performance is tracked using accuracy and validation loss.

3.4 Model Conversion to TensorFlow Lite

The trained model is converted to TensorFlow Lite format using the `TFLiteConverter`. This produces a compact model file (`.tflite`) optimized for running on embedded systems.

- Conversion tool: `tf.lite.TFLiteConverter`
- Output: `gesture_model.tflite`
- Benefit: Smaller size, optimized for low-latency inference on resource-constrained hardware

3.5 Real-Time Inference and Deployment

The `gesture_model.tflite` file is deployed to the Raspberry Pi, where it is used to classify incoming real-time sensor data. The Raspberry Pi uses the Sense HAT LED matrix to display the predicted gesture using a specific color code.

- Red: Circular motion
- Green: Shake
- Blue: Twist
- Off/Colorless: No movement

The TFLite interpreter loads the model, collects live sensor data, reshapes it to a 1D array of 1800 float32 values, and runs inference. Based on the prediction, the corresponding color is displayed on the LED matrix.

4 Software and Hardware Used

- Programming language: Python
- Libraries: NumPy, TensorFlow, scikit-learn, Sense HAT API
- Hardware: Raspberry Pi 4 with Sense HAT for deployment; model training performed on a PC with AMD Ryzen 7 5700U 1.80 GHz with Radeon Graphics

5 Code Repository

The full source code for this project is available on GitHub at:

`https://github.com/CharanjitBK/Motion-Recognition-using-IMU-Sensor-Fusion`

This repository includes:

- Source code files
- Installation instructions
- Motion datasets
- Documentation and usage guidelines

6 Code Implementation

The project implementation was divided into several stages: data collection, preprocessing, model training, model conversion to TensorFlow Lite, and real-time inference using the Raspberry Pi Sense HAT. Python was used for the entire pipeline, using libraries such as NumPy, TensorFlow, and the Sense HAT API.

Listing 1: Sample code for IMU data collection

```
# 1. Data Collection using Sense HAT
from sense_hat import SenseHat
import time

sense = SenseHat()
samples = []

# Collect 300 samples per gesture (approx. 3 seconds at 100 Hz)
for _ in range(300):
    acc = sense.get_accelerometer_raw()
    gyro = sense.get_gyroscope_raw()
    sample = [acc['x'], acc['y'], acc['z'], gyro['x'], gyro['y'], gyro['
        z']]
    samples.append(sample)
    time.sleep(0.01)  # 100 Hz sampling rate
```

Listing 2: Model Training in Google Colab using Dense Neural Network

```
# 2. Model Training in Google Colab
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split

# Preprocessed data (X: features, y: one-hot encoded labels)
# X.shape = (num_samples, 300), y.shape = (num_samples, 4)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
   random_state=42)

# Define the model architecture
model = Sequential([
    tf.keras.layers.Input(shape=(300,)),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(4, activation='softmax')  # 4 classes
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
```

```
    epochs=15,
    batch_size=32
)
```

Listing 3: TensorFlow Lite Model Conversion

```
# 3. Model Conversion to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TFLite model
with open("gesture_model.tflite", "wb") as f:
    f.write(tflite_model)
```

Listing 4: Real-Time Inference and LED Feedback on Raspberry Pi

```
# 4. Real-Time Inference with LED Feedback
import tflite_runtime.interpreter as tflite
import numpy as np

# Load the TFLite model
interpreter = tflite.Interpreter(model_path="gesture_model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Collect one sample and reshape it
sample = np.array(samples).flatten().reshape(1, 300).astype(np.float32)

# Run inference
interpreter.set_tensor(input_details[0]['index'], sample)
interpreter.invoke()
predictions = interpreter.get_tensor(output_details[0]['index'])
predicted_class = np.argmax(predictions)

# LED display feedback
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0)]  #
    Colors for each gesture
sense.clear(colors[predicted_class])
```

7 Results

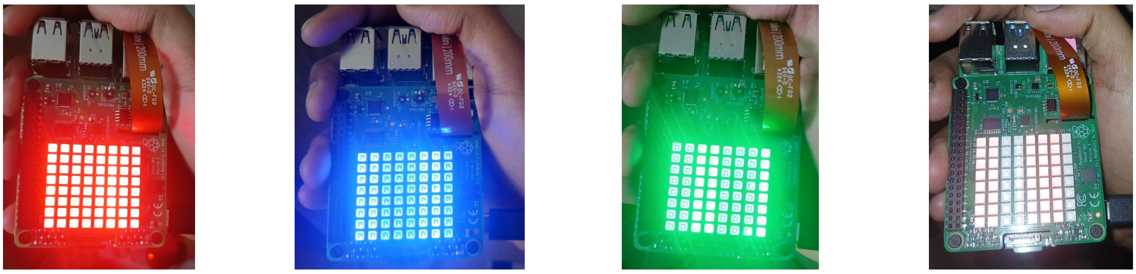
The model was trained for 15 epochs using a dataset collected from four motion types. The following is a summary of the training and validation performance.

- Final training accuracy: **100%**
- Final validation accuracy: **97.92%**
- Validation loss stabilized around: **0.0795**

The training accuracy quickly converged to 100% after the second epoch, indicating that the model fit the training data well. The validity accuracy remained high and stable after the first epochs, suggesting effective generalization without overfitting.

For a full breakdown, here is a summary excerpt from the training log.

```
Epoch 1/15 - accuracy: 0.5788 - val_accuracy: 0.9583
Epoch 2/15 - accuracy: 1.0000 - val_accuracy: 0.9792
...
Epoch 15/15 - accuracy: 1.0000 - val_accuracy: 0.9792
```



(a) (Red) (b) (Blue) (c) Green (d) Colourless

Figure 1: LED matrix output showing gesture recognition for four motion types, red for circular motion, green for shaking, blue for twist and colour does not change when motion does not changes.

8 Challenges, Limitations, and Error Analysis

8.1 Challenges Faced

- Synchronizing real-time data capture and classification on Raspberry Pi.
- Differentiating similar motion patterns like shake and twist.

8.2 Error Analysis

- The model occasionally misclassified the shake as twist.
- Errors caused by drifting IMU sensor and inconsistent user gesture speeds.

8.3 Limitations of the Implementation

- Limited to four predefined gestures.
- The model does not generalize well to new users without retraining.
- Real-time inference performance is limited by the Raspberry Pi CPU capabilities.

9 Discussion

The results align with expectations, showing that simple feedforward neural networks can effectively classify motion data when provided with consistent samples. The use of TensorFlow Lite enables real-time inference on embedded hardware with limited resources. Future improvements could involve leveraging recurrent neural networks (e.g., LSTM) to better capture temporal dependencies in the IMU time series data.

10 Conclusion

This lab successfully demonstrated the end-to-end pipeline of collecting IMU sensor data, training a neural network model, converting it to TensorFlow Lite, and deploying it on a Raspberry Pi to perform real-time gesture recognition. The project highlights the practical application of machine learning techniques embedded in resource-constrained environments.

11 References

- TensorFlow Documentation: `https://www.tensorflow.org`
- Sense HAT API: `https://pythonhosted.org/sense-hat/`
- Prof. Tobias Schaffer, Embedded Systems Lab05 Notes