Guideline on data collection and labelling

First, write code to implement Step 1) to 4).

1) Configure the sampling frequency of each sensor. The minimal recommended sampling frequency is 10Hz (10 samples per second), but a frequency of 50Hz is often preferred for better accuracy.
2) Collect sensor data continuously and assign a timestamp to each data point as it is collected. This timestamp will help synchronize and align data from different sensors. You can use the system clock or a real-time clock module to generate timestamps.
3) Store the raw sensor readings in text files or .csv files. You can find example code in Appendix I. In this setup, your computer acts as the server side, and Arduino is the client. Ensure both devices are connected to the same Wi-Fi (e.g., your phone hotspot), and are listening on the same port.
4) Data synchronization.
    - If you are using a microcontroller like Arduino and reading data from multiple sensors within the same loop iteration, the timing difference between readings of different sensors might be minimal. This can be considered synchronized sampling, and you do not need to explicitly synchronize the sensor readings afterwards.
    - If the data acquisition process does not guarantee synchronized sampling, you may need to synchronize the readings post-acquisition. This is common when sensors have different sampling rates or when their data acquisition is controlled by independent processes. For example, polling different sensors in a round-robin fashion can lead to slight timing differences between readings. You can synchronize the sensor readings based on their timestamps to ensure alignment of data from different sensors. You can find example code from the appendix II.
    - Store the synchronized sensor readings in new text or .csv files.

    Here is an example of the file format you can use for storing the sensor readings. Assume that the sampling frequency is 50Hz. Note that if you are using IMUs, they provide multi-dimensional data. You will need to ensure that each row in your data corresponds to a single dimension. In case of synchronized sampling, you can directly store data in this format during data collection. In case of asynchronized sampling, the file should contain only one 'timestamp' column after synchronizing the sensor readings.

Timestamp	Sensor 1	Sensor 2	Sensor 3	…	Acce_x	Acce_y	Acce_z
0							
0.02							
0.04							
…							

Second, to perform a quick test of your prototype, follow steps 5) – 6) to collect labelled data for training and testing machine learning models. You can use the training data you collect in the upcoming tutorials on various machine learning methods. You can start working on Step 5 – 6 already after the mid-term feedback session.

5) Invite a small number of subjects (e.g., 3 – 5 people) to try your gloves/insoles/socks and collect the sensor data.

   For each subject, ask them to repeat each gesture at least 10 times. For activities like walking, you can simply record data for 30s – 60s per activity per subject. It is a good practice to have 1 data file per subject per session.

6) Manually annotate the data with labels by recording the starting and ending timestamps of each gesture/activity.
   If you have 8 gestures, label these gestures with 0, 1, 2, …, 7, respectively. For all the other gestures, you can label them as 8.
   Add a column named 'label' to the data table for these annotations.

Timestamp	Sensor 1	Sensor 2	Sensor 3	…	Acce_x	Acce_y	Acce_z	Label
0								
0.02								
0.04								
…								

Appendix I

Here is the example code of **collecting sensor data and storing them in .csv file**.
For the **Arduino** part:

```
#include <SPI.h>
#include <WiFiNINA.h>

// WiFi credentials
const char* ssid = "zyyyyy";    // Replace with your WiFi SSID
const char* password = "zy561526"; // Replace with your WiFi password

// Server details (your computer's IP and port)
const char* serverIP = "172.20.10.3";  // Replace with your computer's local IP/ the IP after your computer connect to the hotspot
const int serverPort = 5001;  // Must match the server Python script

int sensorPin1 = A2;
int sensorValue1 = 0;

WiFiClient client;

void setup() {
  Serial.begin(115200);
  while (!Serial);
```

```
  // Connect to WiFi
  Serial.print("Connecting to WiFi...");
  while (WiFi.begin(ssid, password) != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("Connected!");

  // Connect to the server
  Serial.print("Connecting to server...");
  while (!client.connect(serverIP, serverPort)) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("Connected to server!");
}

void loop() {
  if (client.connected()) {
    sensorValue1 = analogRead(sensorPin1);  // Read from a sensor (adjust as needed)
    String data = String(sensorValue1 + ",");  // Convert to string
    client.print(data);  // Send data
    Serial.print("Sent: ");
    Serial.println(data);
    delay(100);  // Adjust sending rate/frequency
  } else {
    Serial.println("Disconnected, reconnecting...");
    client.stop();
    delay(2000);
    client.connect(serverIP, serverPort);
  }
}
```

For the **server** part:

```
import socket
import csv
import datetime
import joblib

# Server settings
HOST = "0.0.0.0"  # Listens on all available interfaces
PORT = 5001  # Must match Arduino's serverPort

# Create a socket (IPv4, TCP)
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen(1)
```

```
print(f"Listening for connections on port {PORT}...")

# Accept connection
client_socket, client_address = server_socket.accept()
print(f"Connected to {client_address}")

# Open CSV file for writing data
csv_filename = "./saving_data/training_sensor_data_1.csv"
with open(csv_filename, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Sensor Value", "label"])  # Write header

    try:
        while True:
            data = client_socket.recv(1024).decode("utf-8").strip()
            if data:
                timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                print(f"Received: {data}")

                # Save to CSV
                writer.writerow([timestamp, data, "pressed"]) # adjust the label here
                file.flush()  # Ensure data is written immediately
    except KeyboardInterrupt:
        print("Server stopped.")
    finally:
        client_socket.close()
        server_socket.close()
```

Appendix II

Here is one example of **timestamp-based synchronization** using *Pandas* library. This example synchronizes the sensor readings from two files, sensor1.csv and sensor2.csv.

```
import pandas as pd

# Load sensor data into DataFrames
sensor1_data = pd.read_csv('sensor1.csv', parse_dates=['timestamp'])
sensor2_data = pd.read_csv('sensor2.csv', parse_dates=['timestamp'])

# Convert timestamp columns to datetime and set them as the index
sensor1_data.set_index('timestamp', inplace=True)
sensor2_data.set_index('timestamp', inplace=True)

# Resample both dataframes to a common frequency (e.g., 20 milliseconds) and interpolate
missing values
sensor1_resampled = sensor1_data.resample('20ms').mean().interpolate()
sensor2_resampled = sensor2_data.resample('20ms').mean().interpolate()

# Synchronize by joining on the index (timestamps)
synchronized_data = sensor1_resampled.join(sensor2_resampled, how='outer').interpolate()
```

```
# Reset the index to bring the timestamp back as a column
synchronized_data.reset_index(inplace=True)

# Save the synchronized data to a new CSV file
synchronized_data.to_csv('synchronized_data.csv', index=False)

print(synchronized_data)
```