RELAZIONE PROGETTO DI RETI DI SENSORI E DISPOSITIVI INDOSSABILI

Filippo Silvestri, VR486238

INTRODUZIONE

Durante la parte pratica del corso, abbiamo imparato ad utilizzare il dispositivo Nordic Thingy:52, nel contesto di acquisizione di dati inerziali tramite Python. In questo progetto, mi occupo della raccolta di dati inerziali, e di come utilizzarli per allenare un algoritmo di deep learning chiamato rete neurale convoluzionale (CNN).

Ho deciso di allenare l'algoritmo per riconoscere una probabilità binaria: se l'user sta sciando o meno.

METODO

Per raccogliere i dati, inizialmente ho provato a trasferire l'esecuzione del codice su un cellulare Android (meno ingombrante rispetto ad un Portatile, e quindi più semplice da portare sugli impianti sciistici), senza però successo, in quanto la libreria Bleak non supporta Android, e quindi era impossibile accedere allo stack bluetooth di Android. Non essendo riuscito a sviare questo problema, ho deciso in un approccio diverso:

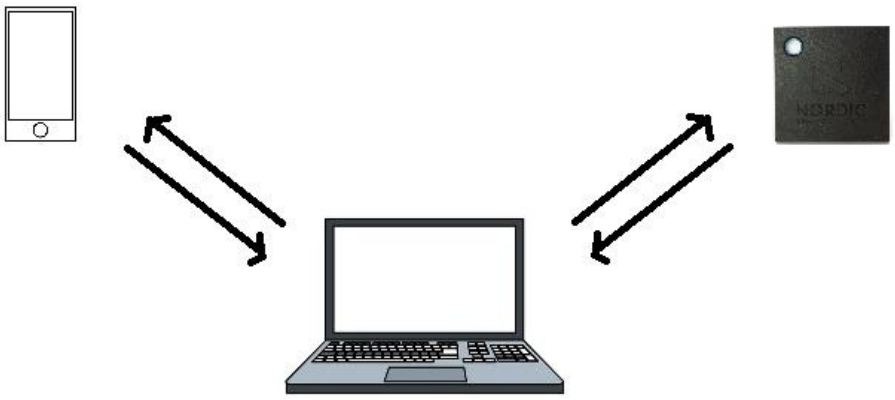


Immagine 1: schema della Body Area Network

In questo schema, osserviamo una BAN con i seguenti nodi:

- Portatile: il nodo centrale, dove avviene la computazione. È il "server" nel nostro schema.
- Thingy:52: è il sensore che rileva i dati inerziali. Tramite protocolli specifici, li invia continuamente al portatile.
- Cellulare: è il nostro "client". È un'interfaccia grafica che ci permette di osservare i dati acquisiti, e di "mandare comandi" al sensore passando per il computer.

Questo significa che una volta fatto partire il programma da computer, tutte le altre azioni necessarie possono essere fatte da cellulare.

Una volta raccolti i dati, utilizziamo un apposito script per allenare e creare il modello. Successivamente, abbiamo l'algoritmo di intelligenza artificiale pronto, e basta far ripartire il programma principale per osservare la predizione sul cellulare.

DESCRIZIONE DEL CODICE

- **main.py**

```
async def main():  1 usage
    # indirizzo MAC del Thingy52
    my_thingy_addresses = ["DC:82:24:3D:29:80"]
    # scannerizzazione dei dispositivi disponibili
    discovered_devices = await scan()
    # lista che filtra tra i dispositivi disponibili e quelli conosciuti
    my_devices = find(discovered_devices, my_thingy_addresses)

    # creazione classe
    thingy52 = Thingy52Client(my_devices[0])
    # connessione WiFi al telefono
    thingy52.connect_to_phone()
    # connessione BLE al Thingy52
    await thingy52.connect()
    thingy52.save_to("stop_recording")
    # call per ricevere i dati IMU
    await thingy52.receive_inertial_data()

if __name__ == '__main__':
    asyncio.run(main())
```

Immagine 2: main.py

Tramite la funzione *scan,* facciamo la ricerca dei dispositivi bluetooth disponibili. Tramite la funzione *find,* inseriamo in *my_devices* i dispositivi salvati disponibili (in questo caso solo il thingy). Tramite la classe *Thingy52Client*, creiamo una classe che gestisce la comunicazione con il sensore e con il cellulare. Utilizzando le funzioni specifiche della classe, ci connettiamo al cellulare e al Thingy, e poi cominciamo a ricevere i dati inerziali.

- **Thingy52Client.py, funzione __init__**

```python
class Thingy52Client(BleakClient):  2 usages

    def __init__(self, device: BLEDevice):
        super().__init__(device.address)
        self.mac_address = device.address

        # variabili per il modello AI
        self.model = ort.InferenceSession('training/CNN_60.onnx')
        self.classes = ["skiing", "still"]
        self.prediction = "None"

        # Data buffer
        self.buffer_size = 60
        self.data_buffer = []

        # Recording information
        self.recording_name = None

        # variabili per la connessione al telefono
        self.server_socket = None
        self.client_socket = None
        self.client_address = None
        self.host = '0.0.0.0'
        self.port = 12345
```

Immagine 3: Thingy52Client.py -- __init__

La classe Thingy52Class si basa sulla classe BleakClient. Nella funzione __init__ (initialization) settiamo le variabili per il funzionamento della connessione al Thingy:52, della connessione al telefono, e dell'algoritmo di AI.

- **Thingy52Client.py, funzione connect_to_phone**

```
# funzione di connessione al telefono
def connect_to_phone(self):  1 usage
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server_socket.bind((self.host, self.port))
    self.server_socket.listen(1)
    print(f"Listening for connections on {self.host}:{self.port}")
    self.client_socket, self.client_address = self.server_socket.accept()
    print(f"Connected to {self.client_address}")
    self.client_socket.setblocking(False)
```

Immagine 4: Thingy52Client.py – connect_to_phone

Questa funzione utilizza la libreria *socket* per creare una comunicazione tra un *client* (telefono) e un *server* (portatile). Affinché la comunicazione sia funzionante, il router deve essere settato per permettere a pacchetti di passare nella porta specifica. Questo va fatto sia sul Router, che sul Firewall del portatile. Utilizzare l'Hotspot del cellulare risolve questo problema.

- **Thingy52Client.py, funzione connect**

```
async def connect(self, **kwargs) -> bool:  1 usage
    print(f"Connecting to {self.mac_address}")
    await super().connect(**kwargs)

    try:
        print(f"Connected to {self.mac_address}")
        # se si connette con successo, cambia il colore del LED
        await change_status(self,  status: "connected")
        return True
    except Exception as e:
        print(f"Failed to connect to {self.address}: {e}")
        return False
```

Immagine 5: Thingy52Client.py – connect

Questa funzione utilizza la libreria *bleak* per connettersi con un dispositivo bluetooth. In aggiunta, se la connessione avviene con successo, tramite un'altra funzione (*change_status)*, cambia il colore del LED del Thingy52.

- **Thingy52Client.py, funzione receive_inertial_data**

```python
async def receive_inertial_data(self, sampling_frequency: int = 60):  1 usage
    # Set the sampling frequency
    payload = motion_characteristics(motion_processing_unit_freq=sampling_frequency)
    await self.write_gatt_char(TMS_CONF_UUID, payload)
    # call per chiedere la trasmissione dei dati
    await self.start_notify(TMS_RAW_DATA_UUID, self.raw_data_callback)
    # Change the LED color to red, recording status
    await change_status(self,  status: "recording")
    # loop che dura fino allo stop del programma
    try:
        while True:
            await asyncio.sleep(0.1)
    # in caso di errore / cancellazione
    except asyncio.CancelledError:
        await self.stop_notify(TMS_RAW_DATA_UUID)
        await self.disconnect()
        print("Stopped notification")
        self.client_socket.close()
        self.server_socket.close()
        print("stopped server")
```

Immagine 6: Thingy52Client – receive_inertial_data

Questa funzione si occupa di preparare un *payload* contenente l'UUID necessario per iniziare il ricevimento dei dati inerziali, e successivamente di creare un *loop* che si ferma solo in caso di chiusura del programma. Questo *loop* continua a chiamare la funzione *raw_data_callback*.

- **Thingy52Client.py, funzione raw_data_callback**

```
def raw_data_callback(self, sender, data):  1 usage
    # Handle the incoming accelerometer data here
    receive_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")

    # Accelerometer
    acc_x = (struct.unpack( format: 'h', data[0:2])[0] * 1.0) / 2 ** 10
    acc_y = (struct.unpack( format: 'h', data[2:4])[0] * 1.0) / 2 ** 10
    acc_z = (struct.unpack( format: 'h', data[4:6])[0] * 1.0) / 2 ** 10

    # Gyroscope
    gyro_x = (struct.unpack( format: 'h', data[6:8])[0] * 1.0) / 2 ** 5
    gyro_y = (struct.unpack( format: 'h', data[8:10])[0] * 1.0) / 2 ** 5
    gyro_z = (struct.unpack( format: 'h', data[10:12])[0] * 1.0) / 2 ** 5

    # Compass
    comp_x = (struct.unpack( format: 'h', data[12:14])[0] * 1.0) / 2 ** 4
    comp_y = (struct.unpack( format: 'h', data[14:16])[0] * 1.0) / 2 ** 4
    comp_z = (struct.unpack( format: 'h', data[16:18])[0] * 1.0) / 2 ** 4
```

Immagine 7: Thingy52Client.py – raw_data_callback – parte 1

Questa prima parte della funzione si occupa di dividere la struttura dati ricevuta (*data*) nelle sue 9 componenti (acc xyz, gyro xyz, comp xyz). La forma dei dati della struttura è definita sul sito della Nordic.

```
try:
    new_name = self.client_socket.recv(1024).decode('utf-8')
    new_name_bool = True
    print(f"\nnew command: {new_name}")
except BlockingIOError:
    new_name_bool = False
if new_name_bool:
    # cambia il file di salvataggio in quel nome
    self.save_to(new_name)
# se il nome è valido (non è stop_recording)
if self.recording_name != f"{self.mac_address.replace( _old: ':',  _new: '-')}_stop_recording.csv":
    # append al file l'ultima riga
    with open(f"training/data/{self.recording_name}", "a+") as file:
        file.write(f"{receive_time},{acc_x},{acc_y},{acc_z},{gyro_x},{gyro_y},{gyro_z}\n")
```

Immagine 8: Thingy52Client.py – raw_data_callback – parte 2

Questa seconda parte si occupa di controllare se il cellulare ha inviato un messaggio al portatile (contenente il nome dell'azione da registrare). In caso positivo (*new_name_bool* == True), allora il nome del *file* su cui salvare i dati inerziali cambia.

Successivamente, salviamo i dati inerziali sul *file* corretto.

```
if len(self.data_buffer) == self.buffer_size:
    input_data = np.array(self.data_buffer, dtype=np.float32).reshape(1, self.buffer_size, 6)
    input_ = self.model.get_inputs()[0].name
    cls_index = np.argmax(self.model.run( output_names: None,  input_feed: {input_: input_data})[0], axis=1)[0]
    self.prediction = self.classes[cls_index]
    self.data_buffer.clear()
self.data_buffer.append([acc_x, acc_y, acc_z, gyro_x, gyro_y, gyro_z])

# invia la previsione al telefono
self.client_socket.sendall(self.prediction.encode('utf-8'))
# stampa i valori
print(f"\r{self.mac_address} | {receive_time} - Accelerometer: X={acc_x: 2.3f},"
      f" Y={acc_y: 2.3f}, ...Z={acc_z: 2.3f}, prediction: {self.prediction}", end="", flush=True)
```

Immagine 9: Thingy52Client.py – raw_data_callback – parte 3

Questa terza e ultima parte si occupa di aggiornare il *buffer* con l'ultimo secondo di dati, e di utilizzarlo per aggiornare la *prediction*. Successivamente inviamo la predizione al cellulare, e stampiamo i valori inerziali.

- **client_phone.py, analisi del codice**

```
#server_ip = "192.168.56.241"
server_ip = "192.168.56.185"
server_port = 12345
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port))
print(f"Connected to server: {server_ip}:{server_port}")
```

Immagine 10: client_phone.py – server creation

Questa prima parte di codice si occupa della connessione del *client* (cellulare) al *server* (portatile)

```
def send(text):  1 usage
    message = text
    client_socket.send(message.encode())
def stop_recording():  1 usage
    message = "stop_recording"
    client_socket.send(message.encode())
def close():  3 usages (2 dynamic)
    try:
        client_socket.close()
    except Exception as e:
        root.destroy()
def update_prediction(message):  1 usage
    if message =="skiing":
        color = "green"
    else:
        color = "red"
    canvas.itemconfig(rect2, fill=color)
    canvas.itemconfig(rect2_text, text=message)
```

questa seconda parte definisce le funzioni che verranno utilizzate dalla GUI, quando verrà pressato uno dei bottoni.

Immagine 11: client_phone.py – functions

```
def receive_message():  1 usage
    while True:
        try:
            message = client_socket.recv(1024).decode()
            if message:
                root.after( ms: 0, update_prediction,  *args: message)
        except Exception as e:
            break

Thread(target=receive_message, daemon=True).start()
```

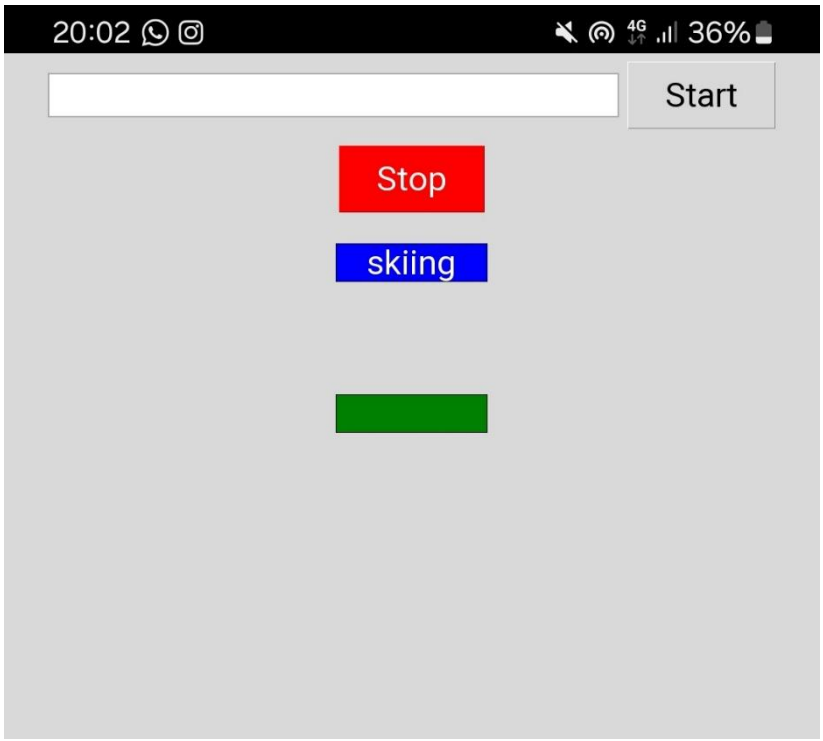
Immagine 12: client_phone.py – loop function

Questa funzione si occupa di ricevere il messaggio contenente la *prediction* dal portatile, e di aggiornare la GUI nell'eventualità. La funzione *Thread* fa si che questa funzione venga ripetuta continuamente in background.

```
root = tk.Tk()
root.title("Reti project App")
frame = tk.Frame(root)
frame.pack(pady=10)

entry = tk.Entry(frame, width=30)
entry.pack(side=tk.LEFT, padx=5)

name_button = tk.Button(frame, text="Start", command=lambda: send(entry.get()))
name_button.pack(side=tk.LEFT, padx=5)

stop_button = tk.Button(root, text="Stop", command=stop_recording, fg="white", bg="red")
stop_button.pack(pady=10)

canvas = tk.Canvas(root, width=300, height=300)
canvas.pack(pady=10)

rect1 = canvas.create_rectangle(50, 20, 250, 70, fill="blue", outline="black")
rect2 = canvas.create_rectangle(50, 220, 250, 270, fill="grey", outline="black")

rect2_text = canvas.create_text(150, 45, text="prediction", fill="white")

root.protocol( name: "WM_DELETE_WINDOW", close)
root.mainloop()
```

Immagine 13: client_phone.py – GUI creation

Questa sezione finale di codice si occupa di creare la GUI che contiene 2 bottoni (start e stop recording), una barra per inserire il nome della registrazione, e un quadrato dove possiamo osservare la *prediction*. La GUI viene creata tramite la libreria TKinter.

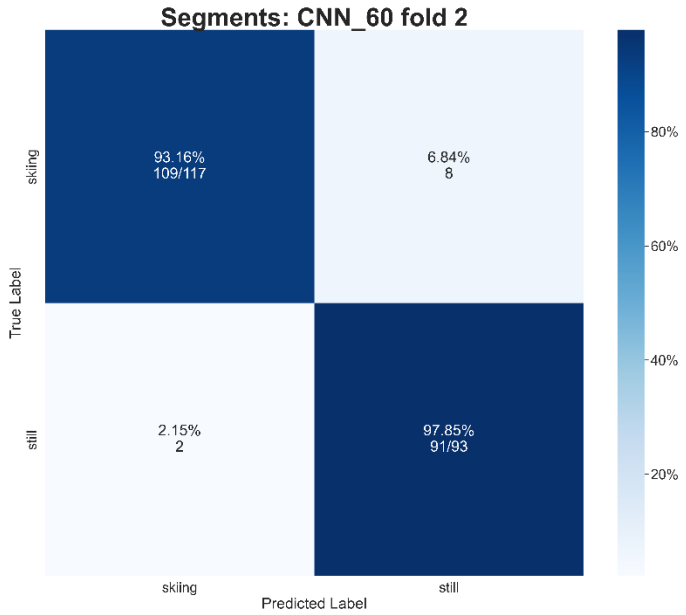


In questa immagine, possiamo vedere la GUI creata dal codice appena descritto.

Immagine 14: client_phone.py – GUI screen

RISULTATI

Dopo aver preparato la *Body Area Network* (portatile nello zaino, sensore in tasca e cellulare in mano), ho registrato 2 azioni: *skiing* e *still*. Successivamente, ho fatto partire lo script *train.py*, creando 3 *confusion matrix* diverse:



tra le 3 matrici, quella con risultati migliori è la seconda.

Immagine 15: confusion matrix – fold 2

Successivamente, facendo partire lo script *convert_model.py*, ho creato il modello CNN_60.onnx dal fold 2. Questo modello viene usato nella classe Thingy52Client per computare la predizione.

Successivamente, ho testato la predizione del modello durante una discesa, e i risultati erano comparabili a quelli della confusion matrix.

CONCLUSIONI

L’azione binaria scelta (sciare o meno) è facilmente prevedibile da un modello CNN. Questo risultato era aspettato, in quanto le differenze del segnale inerziale in movimento e non, sono facilmente visibili anche nel segnale grezzo. Nonostante ciò, il progetto consisteva in una *proof-of-concept* sul funzionamento delle BAN, quindi la scelta dell’algoritmo di intelligenza artificiale non era troppo importante (anche perché la CNN nasce come metodo di analisi di immagini).

Nella parte di acquisizioni dati, non ho riscontrato troppi problemi, se non durante i futili tentativi di far girare il codice su un dispositivo Android.

il codice funziona bene, ed è possibile espanderlo per allenare algoritmi che predicono azioni anche più complesse, con la facilitazione dell’interfaccia per cellulare.