SafetyNet AI

Sistema indossabile di rilevamento delle cadute basato sulle Reti Neurali Cellulari (CNN) con filtraggio dati di Kalman

Domenico Formosi - Gabriel Schifone - Antonio Roma - Francesco Suma - Giovanni Pepe

ITT E.FERMI

Abstract

Current wearable fall detection systems mainly use methods that apply a motion threshold with long-distance communications such as 3G/4G applied to visual systems [1], or machine learning algorithms with possible short-distance communications such as Bluetooth or Wi-Fi [2], [6]-[9]. The first method has the problem of low algorithm precision, while the second has the problem of short transmission distance. To solve these problems, an Arduino Nano 33 BLE development board with an integrated accelerometer sensor, commercially available [10], is used, capable of learning critical events or accidental situations in a neural way. This document describes a wearable system on a motorcycle helmet using the customizable Arduino Nano 33 BLE sense board, capable of learning critical events and recognizing them offline with Machine Learning. The algorithm exploits the knowledge of Cellular Neural Networks [6],[7], using the TensorFlow Lite development library combined with data filtering with the Kalman method [3]-[5], [10]. The Kalman filtering is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement. The Kalman filter has numerous applications in technology [3], including IoT [5]. Specifically, Kalman filters are used in reading data from sensors, to avoid false positives or false negatives [5]. The prototype created, after a learning phase and after managing the parameters of the Kalman filter, allows accurate detection of strong accelerations on a motorcycle helmet, in order to predict and/or anticipate an accidental fall or extreme event with high precision. The accuracy of the fall training model has reached over 98.85% with peaks

close to 99.84%; while the goodness of the Kalman filter has shown that the system is extremely portable and provides high success percentages in fall detection in terms of precision, loss, and reduction of reading noise.

Introduzione

Attualmente sono state proposte diverse soluzioni per il rilevamento delle cadute accidentali[1],[2]. Tali soluzioni sono classificate in due tipi principali in base alla tecnologia dei sensori utilizzata: sistemi non indossabili (NWS: Not Wearable System)e sistemi indossabili (WS: Wearable System). In particolare, i sistemi NWS utilizzano sensori basati sulla visione, e che sono strategicamente distribuiti nel luogo di azione. Tali sistemi si sono dimostrati potenti e robusti nel rilevare le cadute; tuttavia, questi sistemi hanno costi elevati, possono essere ovviamente efficaci solo in ambienti chiusi e potrebbero generare problemi di privacy, senza contare che nella maggior parte dei casi non sono sistemi portatili. Per superare queste limitazioni, sono stati proposti sistemi WS. Questi ultimi sistemi, in genere utilizzano sensori inerziali, come un accelerometro o un giroscopio, solitamente solidali al corpo e/o oggetto di cui si vuole rilevare un movimento preconfigurato. Gli accelerometri sono sempre più utilizzati nei sistemi WS perché offrono svariati vantaggi: basso consumo energetico; convenienza; leggerezza; facilità d'uso; taglia piccola; e soprattutto, estrema portabilità. Uno dei vantaggi essenziali dell'utilizzo del metodo basato sulla soglia è che è meno complesso e meno computazionalmente intensivo rispetto agli altri metodi. Recentemente, sono stati proposti sistemi WS basati su approcci di apprendimento automatico CNN (ML: Machine Learning) per affrontare queste limitazioni e migliorare l'accuratezza del

rilevamento delle cadute [6] -[9]. Al fine di migliorare la robustezza della lettura dai sensori del giroscopio, al sistema Arduino Nano 33 ble sense sia in fase di apprendimento che di rilevamento in modalità Machine Learning è stato abbinato un filtro di Kalman [3]-[5] con l'intenzione di migliorare l'apprendimento e la precisione. Inizialmente, l'idea del filtro di Kalman fu proposta all'inizio del 1960 dall'ingegnere e matematico Rudolf Emil Kalman nel risolvere il problema della stima delle traiettorie per il programma Apollo 11 [3]. Col tempo tale filtro è stato impiegato in miriadi di applicazioni, destando attenzione ed interesse via via crescente. Oggi esistono diverse varianti del filtro Kalman originale. Questi filtri vengono utilizzati diffusamente dalle applicazioni che si basano su una stima, tra cui la visione artificiale, sistemi di guida e navigazione, econometria ed elaborazione di segnali. Tecnicamente parlando, il filtro di Kalman trova la proiezione dei dati attraverso una stima dell'errore di lettura del sensore, usandoli per una correzione continua dei risultati.

Presentazione della proposta progettuale

La nostra proposta progettuale prevede lo sviluppo di un sistema indossabile per il rilevamento dello spostamento repentino della testa indotto da un evento accidentale, quale: caduta accidentale nel vuoto, da scale, decelerazione su motoveicoli dovuto a brusche frenate, praticamente in qualsivoglia evento ove la struttura del collo subisce forti traumi o sollecitazioni. Tale sistema indossabile sfrutta le capacità dei dispositivi intelligenti a basso consumo e una rete neurale per il riconoscimento del rilevamento del movimento, seguendo un approccio Machine Learning, utilizzando una rete neurale per il rilevamento del movimento dell'articolazione testa/collo.

Mentre altri lavori correlati sfruttano più sensori che raccolgono movimenti e inviano dati a un dispositivo che li analizza [1],[2], in questo lavoro abbiamo utilizzato un singolo dispositivo per il monitoraggio e il riconoscimento delle attività attraverso una rete neurale distribuita su una scheda Arduino nano 33 BLE Sense. La scheda ha una dimensione ridotta di 45×18 mm, che la rende adatta a prototipi di indossabili, ed è dotata di diversi sensori integrati per misurare le variabili ambientali [10]. Nella Figura 1 , si possono vedere i componenti principali della scheda e le interfacce di input/output. Questa scelta porta vantaggi di versatilità e portabilità, poiché le soluzioni delle altre opere correlate sono vincolate ad ambienti interni che si basano su infrastrutture non portatili o richiedono più sensori da indossare sul corpo.

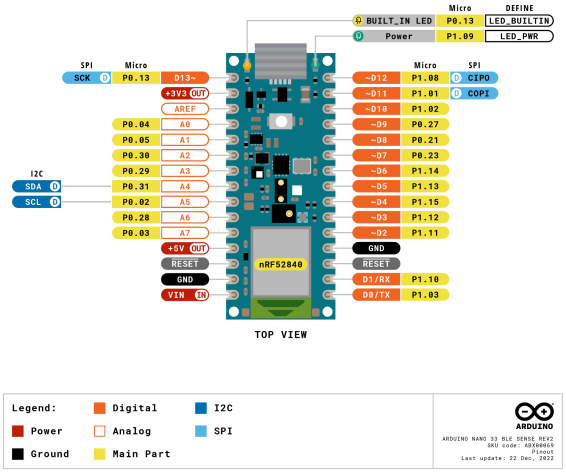


Fig. 1 PinOut Arduino NANO 33 BLE SENSE REV2

I microcontrollori, come quelli utilizzati sulle schede Arduino, sono sistemi informatici a chip singolo, economici e indipendenti. Sono i computer invisibili incorporati all'interno di miliardi di gadget che usiamo tutti i giorni, come sistemi indossabili, droni, stampanti 3D, giocattoli, cuociriso, prese intelligenti, scooter elettrici, lavatrici. La tendenza a collegare questi dispositivi fa parte di ciò che viene chiamato Internet of Things.

Arduino è una piattaforma open source e una community il cui obiettivo è

rendere accessibile a tutti lo sviluppo di applicazioni sui microcontrollori. La scheda che usiamo qui ha un microcontrollore Arm Cortex-M4 a 64 MHz con 1 MB di memoria Flash e 256 KB di RAM. È piccola rispetto a cloud, PC o dispositivo mobile, ma ragionevole per gli standard dei microcontrollori; come una gomma da masticare.

Per quanto riguarda il Machine Learning [6]-[9], ci sono tecniche che possono essere usati per adattare i modelli di rete neurale a dispositivi con memoria limitata come i microcontrollori. La libreria impiegata con l'Arduino Nano 33 BLE Sense è stata la TinyML. Tale libreria affianca il Machine Learning con i Tensor Flow per l'apprendimento ed il riconoscimento dei movimenti. Per evitare falsi allarmi, i segnali di lettura dei sensori di movimento e giroscopici sia in fase di apprendimento che di valutazione sono filtrati attraverso dei filtri di Kalman [3]-[5].

La fig.2 riporta sinteticamente la struttura della nostra proposta progettuale.

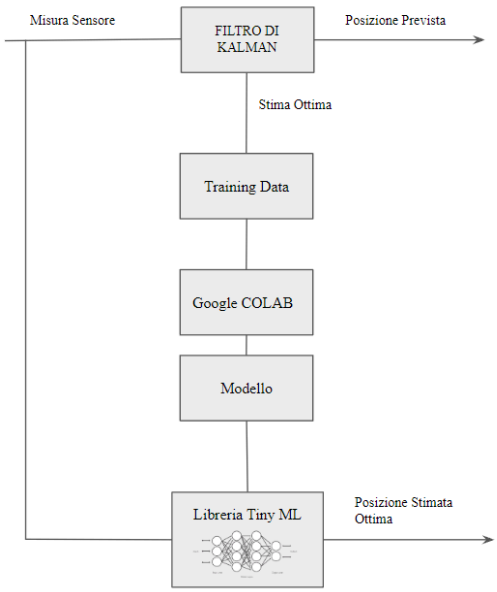


Fig. 2 Schema a blocchi della nostra proposta progettuale

ll filtro di Kalman è un algoritmo utilizzato per il filtraggio dei dati costruito sulla base di una media tra il prossimo valore predetto e il prossimo valore stimato [3]. Questo filtro è spesso utilizzato per ottenere una migliore valutazione di un dato ottenuto dalla lettura di più sensori, ognuno caratterizzato da un rumore di misura avente caratteristiche differenti nel tempo. Il filtro di Kalman, nella letteratura scientifica viene anche definito come stimatore dei dati

togliendo le fonti di rumore che causano disturbo[5]. L'algoritmo si compone in due fasi (fig.3):

- Una previsione puntuale dei valori forniti dal sensore in istanti di tempo successivi, legati anche alla dinamica di Machine Learning;
- Una lettura dei valori dei sensori di movimento e spostamento
- Il confronto/correzione con la previsione iniziale per ottenere la stima ottima finale con una misura del rumore

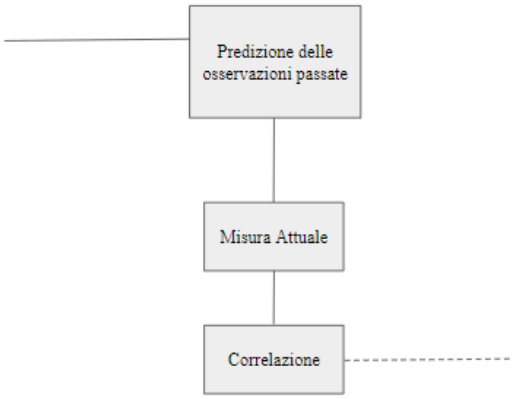


Fig. 3 Algoritmo del filtro di Kalman

La parte di osservazione consiste nella lettura dei sensori anch'essi affetti da errore, l'aggiornamento consiste nella minimizzazione della differenza fra la previsione e l'osservazione, quindi l'errore viene corretto da un fattore di correzione detto K il quale viene usato per aggiornare le posizioni puntuali dei sensori giroscopici. L'algoritmo è stato implementato, sfruttando la piattaforma di sviluppo IDE di Arduino Nano 33 BLE Sense, applicando cinque fasi applicate in maniera iterativa per ogni sensore accelerometro (fig. 4) [10]:

1. Predizione: si prevede lo stato corrente dei sensori di movimento affetti da rumore;
2. Osservazione dello stato: si effettua la misura dei sensori di accelerazione affetti anch'essi da rumore;
3. Minimizzazione dell'errore tra la previsione dei movimenti ottenuta dai sensori di accelerazione e i corrispondenti valori di spostamento effettivamente letti e rilevati.
4. Aggiornamento e calcolo del termine di correzione K per la stima ottima dei movimenti
5. Ritorno al punto 1

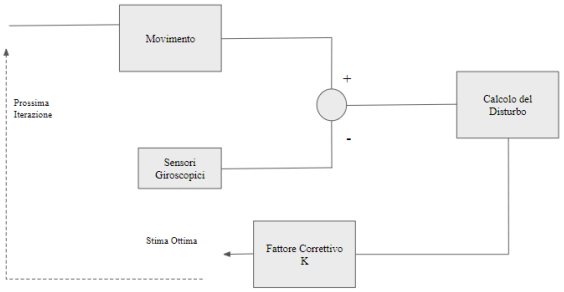


Fig.4 Le fasi del Filtro di Kalman implementate su Arduino Nano 33 BLE Sense.

A seguire, nella fig. 5 è schematizzato il procedimento funzionale del Machine Learning abbinato all'uso delle Reti Neurali Cellulari [6]-[9]. Il passaggio del Training Data dei dati di movimento è realizzato attraverso l'uso delle librerie Tensor Flow e TinyML installate sul microcontrollore Arduino Nano 33 BLE SENSE che abbinate all'analisi fornita dal tool di sviluppo di Google Colab, fornirà un modello facilmente utilizzabile nell'ambiente di sviluppo IDE Arduino. Tale modello consiste, solitamente un file header con estensione .h, e servirà come strumento per realizzare l'applicazione di riconoscimento delle cadute accidentali [10].

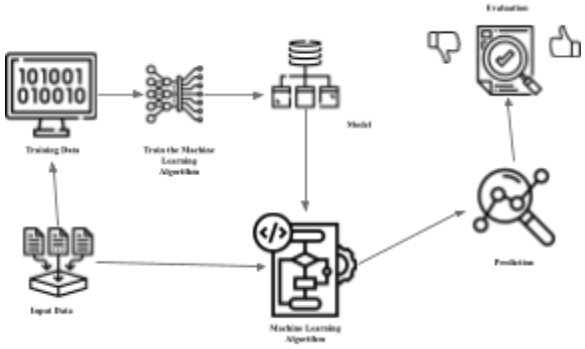


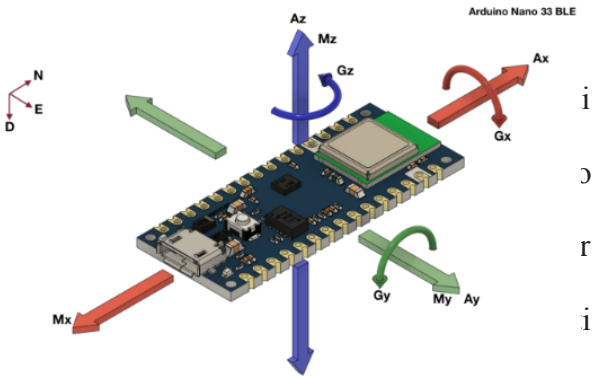
Fig. 5 Lo schema del Machine Learning attraverso le Reti Neurali Cellulari.

Google Colab è uno strumento gratuito presente nella suite Google che consente di scrivere codice python direttamente dal proprio browser, molto utile per chi si occupa di Data Science e/o Big Data con supporto Python. Le macchine virtuali messe a disposizione in Google Colab ospitano un ambiente configurato che consente di concentrarsi sin da subito sui progetti di Data Science: sono presenti numerose librerie Python, tra cui moltissime

di Data Science come Keras e Tensor Flow, si può usufruire di componenti aggiuntive di calcolo, per esempio nell'implementazione di reti neurali con Tensor Flow. Importare dati in Google Colab è molto semplice ed inoltre è uno strumento molto flessibile e integrato con altri servizi cloud come ad esempio Bigquery.

La proposta progettuale: fase di apprendimento

Nella fase di apprendimento abbiamo utilizzato il Machine Learning per consentire alla scheda Arduino di riconoscere i gesti. Abbiamo acquisito i dati di movimento dalla scheda Arduino Nano 33 BLE Sense, per poi importarli in Tiny ML per addestrare un modello neurale e poi definire ed avviare il classificatore risultante sulla scheda Arduino. Questo è stato reso più semplice nel nostro caso poiché la scheda Arduino Nano 33 BLE Sense che abbiamo utilizzato ha un processore Arm



i

o

r

i

direttamente dai log dei dati dei sensori della scheda Arduino tramite lo stesso cavo USB utilizzato per programmare la scheda al nostro laptop o PC. Il set di dati contenenti la lettura dei movimenti dell'articolazione collo e testa sono stati raccolti tramite la finestra di log di Arduino Nano in formato .csv nella misura di circa 1800 campioni ordinati in 6 letture di sensori ciascuna. Nello specifico (fig. 6), ogni lettura del dataset contiene:

- accelerazione_x;
- accelerazione_y;
- accelerazione_z;
- giroscopio_x;
- giroscopio_y;
- giroscopio_z.

Fig. 6 Sensori dell'Arduino Nano 33 BLE sense

A tal fine, furono raccolti 50 campioni in modo da poter simulare una rotazione/movimento testa/collo troppo repentina e 50 eventi correlati a movimenti regolari e non traumatici. Ogni record del dataset sia in fase di addestramento dei movimenti che di classificazione furono filtrati secondo lo stimatore di Kalman [4], [10]. La fig. 7 riporta un dettaglio della lettura del sensore di 'accelerazione x' operata dal log del sistema di sviluppo IDE Arduino nell'intervallo di 0.5 secondi, dove la linea in blu indica il valore disturbato letto direttamente dal sensore di accelerazione; mentre in rosso il valore esautorato dal rumore di lettura ottenuto dal filtraggio di Kalman. Dalla figura si osserva come il valore disturbato, indicato con la linea blu, sia esautorato dal rumore e migliorato fino ad ottenere i valori indicati dalla linea rossa.

Fig.7 Particolare di lettura del sensore 'accelerazione x' della scheda Arduino Nano 33 BLE Sense nella raccolta di dati per l'apprendimento.

La proposta progettuale: pre-elaborazione dati e apprendimento Machine Learning.

In questa fase abbiamo usato il supporto Google Colab [10] per addestrare il nostro modello di Machine Learning utilizzando i dati raccolti dalla scheda Arduino nella sezione precedente. Colab fornisce un sistema a riga di comando che ci consente di eseguire l'addestramento Tensor Flow agendo direttamente sul browser di navigazione. Il supporto Google Colab ci ha aiutato a sviluppare i passaggi della nostra proposta progettuale, ed in particolare sono

riportati i passaggi salienti passati alla piattaforma Google Colab:

- Configurazione dell'ambiente Python

```
# Setup environment
!apt-get -qq install xxd
!pip install pandas numpy
matplotlib
!pip install
tensorflow==2.0.0-rc1
```

- Caricamento dei dati in formato .csv:

```
import matplotlib.pyplot as
plt
import numpy as np
import pandas as pd

filename = "movimento.csv"

df =
pd.read_csv("/content/sample
_data/" + filename)

index = range(1,
len(df['aX']) + 1)

plt.rcParams["figure.figsize
"] = (20,10)

plt.plot(index, df['aX'],
'g.', label='x',
linestyle='solid',
marker=',')
plt.plot(index, df['aY'],
'b.', label='y',
linestyle='solid',
marker=',')
plt.plot(index, df['aZ'],
'r.', label='z',
linestyle='solid',
marker=',')
plt.title("Acceleration")
plt.xlabel("Sample #")
plt.ylabel("Acceleration
(G)")
plt.legend()
plt.show()

plt.plot(index, df['gX'],
'g.', label='x',
linestyle='solid',
marker=',')
plt.plot(index, df['gY'],
'b.', label='y',
linestyle='solid',
marker=',')
plt.plot(index, df['gZ'],
'r.', label='z',
linestyle='solid',
marker=',')
plt.title("Gyroscope")
plt.xlabel("Sample #")
plt.ylabel("Gyroscope
(deg/sec)")
plt.legend()
plt.show()
```

A seguire, nelle figg. 8 e 9 sono riportati, rispettivamente, alcuni dettagli delle misure delle tre accelerazioni fornite dai rispettivi sensori di Arduino Nano 33 BLE Sense assieme a quelle delle rotazioni registrate dai sensori inerziali; entrambi filtrati con lo stimatore Kalman e pronti per essere appresi dal modello di rete neurale dati della piattaforma Google Colab. I colori blu, rosso e verde indicano rispettivamente gli assi di rotazione z, x e y, secondo le schema di fig. 6. Il filtro di Kalman è stato implementato direttamente nel tool IDE Arduino, tenendo conto delle capacità computazionali della scheda Arduino Nano, cercando un buon compromesso tra qualità della stima e velocità di esecuzione del calcolo delle misure.

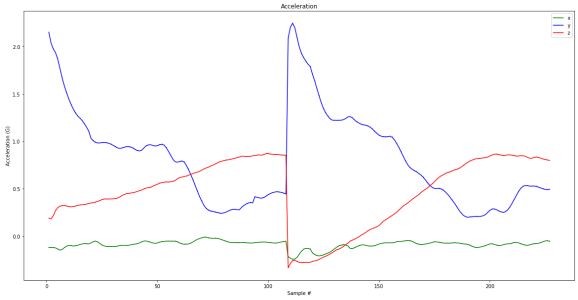


Fig.8 Dettaglio delle tre accelerazioni misurate dei rispettivi sensori di Arduino Nano 33 Ble Sense

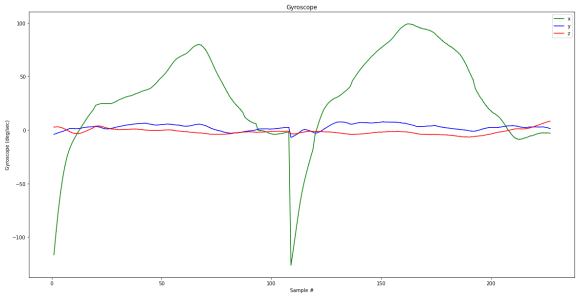


Fig.9 Dettaglio delle tre rotazioni inerziali misurate dei rispettivi sensori di Arduino Nano 33 Ble Sense

L'insieme dei valori forniti dai sei sensori descrivono il movimento dell'articolazione collo/testa che permette di individuare accelerazioni/decelerazioni improvvise per il riconoscimento delle cadute accidentali.

- Analisi e preparazione dei dati:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
print(f"TensorFlow version =
{tf.__version__}\n")

# Set a fixed random seed value,
for reproducibility, this will
allow us to get
# the same random numbers each
time the notebook is run
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)

# the list of gestures that data
is available for
GESTURES = [
    "movimento",
]

SAMPLES_PER_GESTURE = 119

NUM_GESTURES = len(GESTURES)

# create a one-hot encoded matrix
that is used in the output
ONE_HOT_ENCODED_GESTURES =
np.eye(NUM_GESTURES)

inputs = []
outputs = []
# read each csv file and push an
input and output
for gesture_index in
range(NUM_GESTURES):
  gesture =
GESTURES[gesture_index]
  print(f"Processing index
{gesture_index} for gesture
'{gesture}'.")

  output =
ONE_HOT_ENCODED_GESTURES[gesture_i
ndex]

  df =
pd.read_csv("/content/sample_data/
" + gesture + ".csv")

  # calculate the number of
gesture recordings in the file
  num_recordings = int(df.shape[0]
/ SAMPLES_PER_GESTURE)

  print(f"\tThere are
{num_recordings} recordings of the
{gesture} gesture.")

  for i in range(num_recordings):
    tensor = []
    for j in
range(SAMPLES_PER_GESTURE):
      index = i *
SAMPLES_PER_GESTURE + j
      # normalize the input data,
between 0 to 1:
      # - acceleration is between:
-4 to +4
      # - gyroscope is between:
-2000 to +2000
      tensor += [
          (df['aX'][index] + 4) /
8,
          (df['aY'][index] + 4) /
8,
          (df['aZ'][index] + 4) /
8,
          (df['gX'][index] + 2000)
/ 4000,
          (df['gY'][index] + 2000)
/ 4000,
          (df['gZ'][index] + 2000)
/ 4000
```

```
      ]

    inputs.append(tensor)
    outputs.append(output)

# convert the list to numpy array
inputs = np.array(inputs)
outputs = np.array(outputs)
print("Data set parsing and
preparation complete.")
```

- Creazione e addestramento del modello:

```
# Randomize the order of the
inputs, so they can be evenly
distributed for training, testing,
and validation
#
https://stackoverflow.com/a/377104
86/2020087
num_inputs = len(inputs)
randomize = np.arange(num_inputs)
np.random.shuffle(randomize)

# Swap the consecutive indexes (0,
1, 2, etc) with the randomized
indexes
inputs = inputs[randomize]
outputs = outputs[randomize]

# Split the recordings (group of
samples) into three sets:
training, testing and validation
TRAIN_SPLIT = int(0.6 *
num_inputs)
TEST_SPLIT = int(0.2 * num_inputs
+ TRAIN_SPLIT)

inputs_train, inputs_test,
inputs_validate = np.split(inputs,
[TRAIN_SPLIT, TEST_SPLIT])
outputs_train, outputs_test,
outputs_validate =
np.split(outputs, [TRAIN_SPLIT,
TEST_SPLIT])

print("Data set randomization and
splitting complete.")

# build the model and train it
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(50
, activation='relu')) # relu is
used for performance
model.add(tf.keras.layers.Dense(15
, activation='relu'))
model.add(tf.keras.layers.Dense(NU
M_GESTURES, activation='softmax'))
# softmax is used, because we only
expect one gesture to occur per
input
model.compile(optimizer='rmsprop',
loss='mse', metrics=['mae'])
history = model.fit(inputs_train,
outputs_train, epochs=600,
batch_size=1,
validation_data=(inputs_validate,
outputs_validate))
```

Per sfruttare i vantaggi delle reti neurali, per il Machine Learning, mantenendo un modello semplice, in grado di essere implementato sulla scheda Arduino, si è adottato un modello di rete neurale non molto complesso, composto da

tre livelli: uno strato iniziale con 50 neuroni avente funzione di attivazione sigmoidea (funzione con andamento ad S); un livello intermedio con 15 neuroni con funzione di attivazione sigmoidea; ed infine un livello finale con quattro neuroni e una funzione di attivazione softmax (una specie di funzione con andamento esponenziale) [6]-[9].

Nella fig. 10 sono riportati l'evoluzione della fase di apprendimento (training test) con quella di validazione (validation test). Dopo poco tempo, espresso nel caso delle Reti Neurali Cellulare col termine di Epoch (epoche) [6], il modello cellulare converge, riconoscendo correttamente il movimento in fase di apprendimento. Tale convergenza è testimoniata dal fatto che la linea blu raggiunge il valore 1.0; mentre la bontà dell'apprendimento è dedotta dal fatto che l'errore di validazione, indicato con la linea rossa, scende in tempi molto stretti verso il valore 0. In entrambi i casi l'accuratezza della caduta del modello di addestramento ha raggiunto oltre il 98,85% con picchi prossimi al 99,84%.

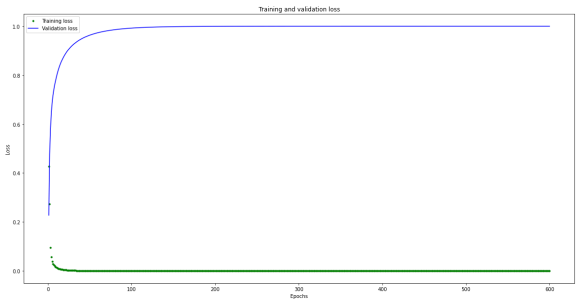


Fig.10 Dettaglio del risultato dell'addestramento Machine Learning condotto con Google Colab.

Il modello è stato creato e addestrato utilizzando le librerie Tensor Flow e Keras, con supporto Colab. Il modello ottenuto è stato convertito in una versione Tensor Flow Lite, ed adattato per essere caricato nell'IDE di Arduino, sotto forma di file header (model.h), e quindi caricata nella RAM della scheda Nano 33 BLE Sense.

Conclusioni: Risultati, Considerazioni e Sviluppi Futuri.

Per simulare il movimento di caduta nel vuoto o un evento accidentale, si è usato

un casco motociclistico, posizionando tramite nastro isolante la scheda Arduino Nano sulla superficie di copertura, vicino alla visiera del casco. Il cavo USB è stato collegato ad un PC, dal quale era in esecuzione lo scketch implementato sull'IDE Arduino col modello neurale ottenuto. Attraverso il serial monitor, fornito dall'IDE Arduino, è possibile saggiare l'affidabilità del sistema di riconoscimento a Machine Learning leggendo i valori rilevati assieme alla stringa che avvisa il superamento della soglia di riconoscimento del movimento. La fig. 11 riporta una possibile schermata del serial monitor dell'IDE Arduino dopo un riconoscimento di possibile caduta accidentale. La probabilità che l'evento sia riconosciuto aumenta man mano che il valore raggiunge un valore molto prossimo allo 0.97. La possibilità di ridurre i falsi allarmi di riconoscimento è merito anche all'uso del filtro di Kalman anche in fase di riconoscimento, oltre che in fase di addestramento.

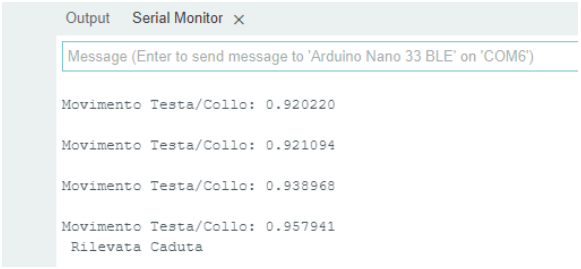


Fig.11 Possibile risultato del riconoscimento Machine Learning del movimento testa.

Attualmente tutti i dati sono visualizzati mediante il terminale offerto dall'IDE di Arduino e il rilevamento stesso della caduta può essere visualizzato mediante questa interfaccia, com'è possibile vedere nella fig. 11.

In questo lavoro abbiamo presentato un sistema per il rilevamento delle cadute, sfruttando il movimento articolazione testa/collo. Il sistema sfrutta una smart sensor board su cui utilizziamo una rete neurale addestrata a riconoscere e monitorare i movimenti catturati da un casco motociclistico. Il sistema è a livello

prototipale con auspicabili migliorie, specialmente nei tempi di dati di apprendimento e validazione. Al fine di raggiungere un buon compromesso tra velocità di esecuzione e precisione nei risultati, il sistema ha impiegato un numero non molto alto di dati in fase di apprendimento; tutto ciò ha comportato la scelta di un semplice modello di rete neurale per meglio adattarsi alle limitate capacità computazionali di tali dispositivi.

Come sviluppi futuri, al fine di rendere l'esperienza del prodotto più "user-friendly" si era pensato ad un prototipo di applicazione mobile, in particolare IOS, mediante il linguaggio di programmazione Swift.

Come riportato nella fig. 12, l'applicazione prevede un numero minimo di funzionalità, collegate ai valori di lettura dei sei sensori della scheda Arduino Nano 33 BLE Sense, con alert incorporato.

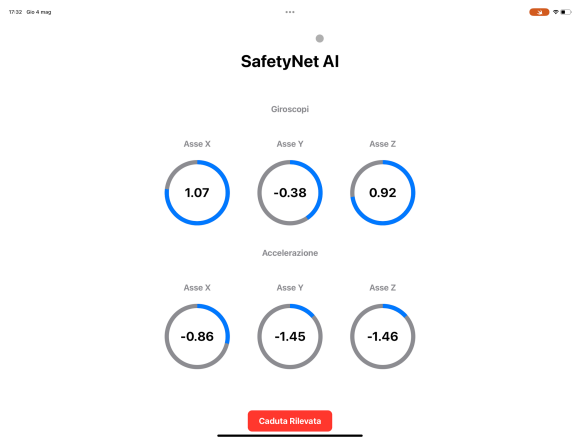


Fig.12 Prototipo di Applicazione Mobile con lettura dei sensori da Arduino Nano 33 BLE Sense

Nelle nostre intenzioni il prototipo dell'applicazione mobile dovrebbe permettere la comunicazione tra il microcontroller con un altro modulo software, installato in un altro Kit Arduino di tipo Uno. A tal scopo sono state analizzate due tecnologie in grado di assolvere a tale funzione. Come indicato nelle figg. 13 e 14, la prima tecnologia a nostra disposizione è quella di sfruttare direttamente il protocollo HTTP, andando a configurare un web server sul kit Arduino Uno, ed attraverso richieste GET al server,

ricevere i dati dai sensori del microcontrollore Nano 33 BLE Sense con quelli riconosciuti dal sistema neurale di apprendimento.

Fig. 13. Schema di funzionamento del Web Server tramite Arduino Uno.

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

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC address of your Ethernet shield
IPAddress ip(192, 168, 1, 177); // IP address of your Arduino
EthernetServer server(80); // create a server at port 80

void setup() {
  Ethernet.begin(mac, ip);
  server.begin();
}

void loop() {
  EthernetClient client = server.available(); // listen for incoming clients
  if (client) {
    if (client.connected()) {
      String request = client.readStringUntil('\r'); // read the client's request
      if (request.indexOf("GET /sensors") != -1) { // if the request is for the sensors
        client.println("HTTP/1.1 200 OK"); // send the HTTP response header
        client.println("Content-Type: text/html");
        client.println("Connection: close");
        client.println();
        client.print("Sensor 1: ");
        client.println(analogRead(A0)); // send the value of sensor 1
        client.print("Sensor 2: ");
        client.println(analogRead(A1)); // send the value of sensor 2
      }
      client.stop(); // close the connection
    }
  }
}
```

Fig. 14. Parte del codice del Web Server per il Kit Arduino Uno.

Un altro modello efficiente di programmazione, sarebbe quello di implementare un WebSocket su una scheda Arduino Uno, con la possibilità di inviare dati dal server al client senza che quest'ultimo debba effettuare continuamente richieste per ottenere gli aggiornamenti; riducendo così la quantità di traffico di rete e migliorando le prestazioni dell'applicazione. Inoltre, essendo i WebSocket full-duplex, consentirebbero di gestire interazioni in tempo reale tra client e server. Le figg. 15 e 16 mostrano lo schema concettuale del WebSocket ed una possibile implementazione.

Fig. 14 Schema di funzionamento del WebSocket tramite Arduino Uno.

```
void loop() {
  // Gestisci le richieste dei client
  webSocket.loop();

  // Controlla se c'è un nuovo client connesso
  if (webSocket.newClient()) {
    // Invia i valori dei sensori al nuovo client
    webSocket.sendTXT(webSocket.lastIndex(), getSensorValues());
  }
}

String getSensorValues() {
  // Leggi i valori dei sensori
  int sensor1Value = analogRead(sensor1Pin);
  int sensor2Value = analogRead(sensor2Pin);

  // Costruisci una stringa con i valori dei sensori
  String sensorValues = "{\"sensor1\": " + String(sensor1Value) + ", \"sensor2\": " + String(sensor2Value) + "}";

  // Ritorna la stringa con i valori dei sensori
  return sensorValues;
}
```

Fig. 15 Parte del codice del WebSocket per il Kit Arduino Uno.

BIBLIOGRAFIA

[1] Design of a Wearable Healthcare Emergency Detection Device for Elder Persons / Amato, Flora; Balzano, Walter; Cozzolino, Giovanni. - In: APPLIED SCIENCES. - ISSN 2076-3417. - 12:5(2022). [10.3390/app12052345]

[2] Dias, P.V.G.; Costa, E.D.M.; Tcheou, M.P.; Lovisolo, L. Fall detection monitoring system with position detection for elderly at indoor environments under supervision. In Proceedings of the 2016 8th IEEE Latin-American Conference on Communications (LATINCOM), Medellin, Colombia, 16–18 November 2016; pp. 1–6.

[3] Shih Yu Chang, Hsiao-Chun Wu, "Tensor Kalman Filter and Its Applications", IEEE Transactions on Knowledge and Data Engineering, vol.35, no.6, pp.6435-6448, 2023.

[4] Zhehao Jin, Andong Liu, Wen-An Zhang, Li Yu, Chun-Yi Su, "A Learning Based Hierarchical Control Framework for Human–Robot Collaboration", IEEE Transactions on Automation Science and Engineering, vol.20, no.1, pp.506-517, 2023.

[5] Sameer Qazi, Bilal A. Khawaja, Qazi Umar Farooq, "IoT-Equipped and AI-Enabled Next Generation Smart Agriculture: A Critical Review, Current Challenges and Future Trends", IEEE Access, vol.10, pp.21219-21235, 2022.

[6] Giuseppe Grassi, Vecchio Pietro, "Wind energy prediction using a two-hidden layer neural network", Communication in NonLinear Science and Numerical Simulation, Volume 15, Issue 9, September 2010, Pages 2262-2266

[7] Kariniotakis GN, Stavrakakis GS, Nogaret EF. Wind power forecasting using advanced neural networks models. IEEE Trans Energy Convers 1996;11:762–7

[8] Grassi G, Grieco LA. Object-oriented image analysis using the CNN universal machine: new analogic CNN algorithms for motion compensation image synthesis and consistency observation. IEEE Trans CAS-I 2003;50:488–99.

[9] Flores P, Tapia A, Tapia G. Application of a control algorithm for wind speed prediction and active power generation. Renew Energy 2005;30:523–36.

[10] Tutorial Nano 33BLE Sense: https://developers-it.googleblog.com/2019/11/come-muovere-i-primi-passi-con-il-machine-learning-su-arduino.html