Senzor pokreta kao digitalna brava

Pavičić, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Elektrotehnika i informacijska tehnologija

SENZOR POKRETA KAO DIGITALNA BRAVA

Diplomski rad

Marko Pavičić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 25.08.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Marko Pavičić
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. Pristupnika, godina upisa:	D-1378, 07.10.2021.
OIB studenta:	23689261882
Mentor:	prof. dr. sc. Davor Vinko
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Krešimir Grgić
Član Povjerenstva 1:	prof. dr. sc. Davor Vinko
Član Povjerenstva 2:	Luka Filipović, mag. ing. el.
Naslov diplomskog rada:	Senzor pokreta kao digitalna brava
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zadatak diplomskog rada je realizirati digitalnu bravu korištenjem senzora pokreta (engl. Gesture sensor) na Arduino Nano 33 BLE Sense razvojnoj pločici. Za više informacija javiti se mentoru: davor.vinko@ferit.hr
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	25.08.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 15.09.2023.

Ime i prezime studenta:	Marko Pavičić
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1378, 07.10.2021.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Senzor pokreta kao digitalna brava**

izrađen pod vodstvom mentora prof. dr. sc. Davor Vinko

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U današnjem digitalnom dobu, tehnološki napredak neprestano mijenja različite aspekte svakodnevnih života, tradicionalne metode i prakse zastarijevaju dok se nove, inovativne tehnologije i ideje integriraju u svakodnevne procese. Jedno od područja koje doživljava ove promjene je sigurnost i kontrola pristupa. Tradicionalne fizičke brave i ključevi polako se zamjenjuju digitalnim rješenjima koji nude veću sigurnost i jednostavnost korištenja. Digitalne brave predstavljaju osnovnu komponentu suvremenih sigurnosnih sustava koja integrira visoku tehničku naprednost kako bi osigurala zaštitu imovine i osoba.

Cilj ovog diplomskog rada je realizirati digitalnu bravu koristeći senzor pokreta (engl. Gesture sensor) koji se nalazi na Arduino Nano 33 BLE Sense razvojnoj pločici. Senzor pokreta koristi se za prepoznavanje gesti, odnosno pokreta koje korisnik izvodi, te na temelju prepoznatih gesti sustav kontrolira otvaranje i zatvaranje vrata. Rješenje uključuje prikupljanje podataka s akcelerometra i žiroskopa, treniranje modela za prepoznavanje gesti te povezivanje web sučelja s Arduino pločicom putem Bluetooth-a kako bi se otvorila ili zatvorila vrata na temelju prepoznatih gesti.

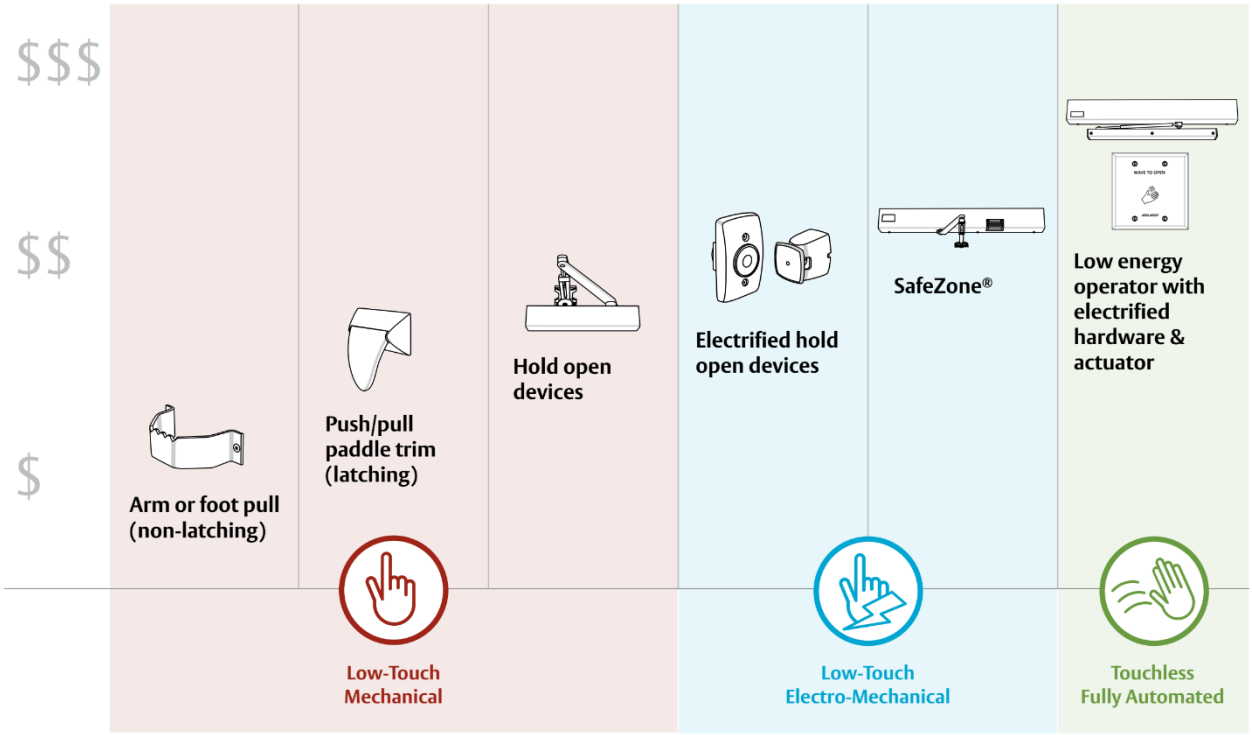
Kroz integraciju ovih komponenata - senzora pokreta na Arduino Nano 33 BLE Sense razvojnoj pločici, modela za prepoznavanje gesti te povezanog web sučelja - ovaj diplomski rad teži stvoriti funkcionalno rješenje koje demonstrira kako se tehnologija može primijeniti na polju sigurnosti i kontrole pristupa.

2. PREGLED POSTOJEĆIH RJEŠENJA

U ovom poglavlju napravljen je pregled sličnih postojećih rješenja za otvaranje vrata bez dodirivanja kvake rukom. Analizirane su različite metode i tehnologije koje se koriste kako bi se postigla sigurna, higijenska i praktična kontrola pristupa.

2.1. Safer2Open™ rješenja tvrtke ASSA ABLOY

ASSA ABLOY nudi čitav spektar rješenja (Slika 2.1.) koja zahtijevaju manje dodira od tradicionalnih brava ili rješenja bez dodira. Safer2Open™ za svrhu ima povećavanja razine čistoće u svim okruženjima a posebno je usmjerena na okruženja izložena velikom broju ljudi [1]. Njihova rješenja smanjuju broj dodirnih točaka te time doprinose smanjenju prenošenja klica.



Slika 2.1.1. *Spektar Safer2Open™ rješenja tvrtke ASSA ABLOY* [1]

Jedno od rješenja koje je slično zadatku ovog diplomskog rada jest „ASSA ABLOY – Norton Door Controls 704 Wave to Open Switch“ (Slika 2.1.2.) koje koristi infracrveni senzor za prepoznavanje pokreta ruke. Nakon što je pokret ruke prepoznat vrata se automatski otvaraju bez potrebe za dodirivanjem kvake. Ovo rješenje nalazi primjenu u javnim ustanovama, toaletima, operacijskim salama i sl. Nedostatak ovog rješenja jest sigurnost, nema identifikacije osobe koja koristi bravu tako da svatko može otvoriti vrata koja koriste ovu bravu.

Slika 2.1.2. *„ASSA ABLOY – Norton Door Controls 704 Wave to Open Switch“* [2]

2.2. Touchless Hand Gesture Door Opener

Ovo rješenje nije komercijalan proizvod već „uradi-sam“ projekt koji predstavlja nadogradnju prethodno opisanog rješenja. Koristi Arduino Nano razvojnu pločicu i dva infracrvena senzora blizine. Za razliku od prethodnog rješenja ovo rješenje osim što prepoznaje ruku implementira i sigurnosnu zaštitu uz pomoć lozinke za ulaz. Lozinka se unosi na način da mahanjem ruke preko cijelog uređaja (preko oba infracrvena senzora blizine) unosimo binarnu jedinicu dok zadržavanjem ruke ispred senzora unosimo binarnu nulu. Ukoliko je unesena točna lozinka vrata se otvaraju. Implementirana je i promjena lozinke putem Bluetootha te je zaštićena zasebnom lozinkom kako lozinku ne bi mijenjala neautorizirana osoba.

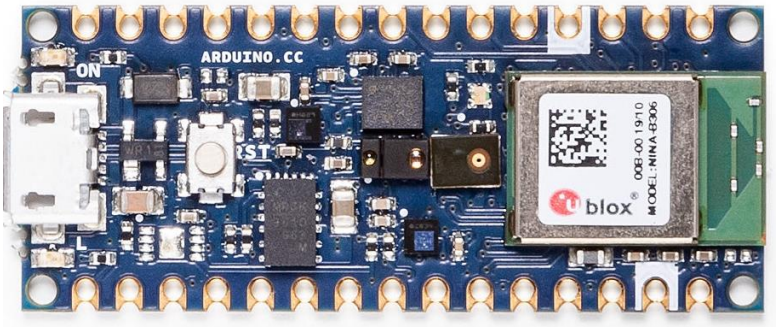
Slika 2.2.1. *„Touchless Hand Gesture Door Opener“* [3]

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju opisane su korištene tehnologije, razvojna okruženja i hardver upotrijebljeni za implementaciju rješenja.

3.1. Arduino Nano 33 BLE Sense

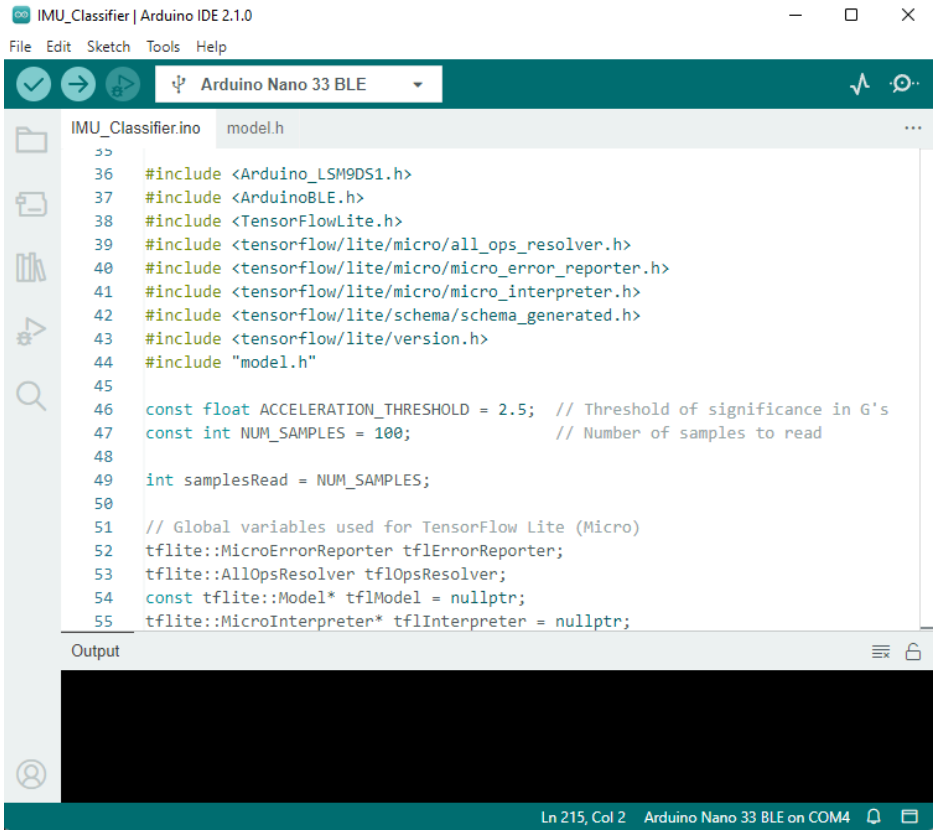
Arduino Nano 33 BLE je razvojna pločica temeljena na Nordic Semiconductors nRF52840 sistemu na čipu (engl. System on a chip) [4]. Pločica je malih dimenzija što omogućava lako rukovanje, ima ugrađen Bluetooth Low Energy (BLE) modul koji omogućuje bežičnu komunikaciju s drugim uređajima poput računala i pametnih telefona koji će biti korišten za spajajanje s web sučeljem. Također, Arduino Nano 33 BLE ima integrirane senzore kao što su inercijski senzor s 9 osi, senzor vlažnosti i temperature, mikrofon, barometarski senzor, senzor geste, blizine te boje i intenziteta svjetla. Senzor najvažniji za implementaciju riješenja je inercijski senzor s devet osi koji osim kompasa s tri osi uključuje žiroskop sa tri osi kao i akcelerometar sa tri osi koji će biti korišteni za prepoznavanje gesti.



Slika 3.1.1. *Arduino Nano 33 BLE Sense pločica [4]*

3.2. Arduino IDE

Arduino IDE (Integrated Development Environment) je razvojno okruženje koje se koristi za programiranje Arduino pločica. IDE pruža jednostavno sučelje za pisanje, uređivanje i prenošenje programa na Arduino pločicu [5]. Također, IDE ima ugrađene biblioteke koje olakšavaju korištenje različitih senzora i komponenti s Arduino pločicom, ukoliko biblioteka nije ugrađena postoji jednostavan izbornik biblioteka koji omogućava naknadno preuzimanje biblioteka iz Arduino repozitorija ili uvoz putem zip datoteke. Za razvoj programskog rješenja korištena je nova verzija Arduino IDE 2 koja predstavlja veliko unaprjeđenje prethodnika koje dolazi s obnovljenim korisničkim sučeljem, programom za ispravljanje pogrešaka, značajkom automatskog dovršavanja i ostalim poboljšanjima.



Slika 3.2.1. *Arduino IDE 2 korisničko sučelje*

3.3. TensorFlow

TensorFlow je popularna biblioteka otvorenog koda za strojno učenje i umjetnu inteligenciju [6]. Biblioteka pruža bogat skup alata i funkcionalnosti za izgradnju, treniranje i evaluaciju različitih modela strojnog učenja te ju koristimo za treniranje modela za prepoznavanje gesti. Kako bi se model za prepoznavanje gesti mogao koristiti na Arduino pločici korištena je TensorFlow Lite biblioteka za mikrokontrolere koja je dizajnirana za pokretanje modela strojnog učenja na mikrokontrolerima sa malom količinom memorije.

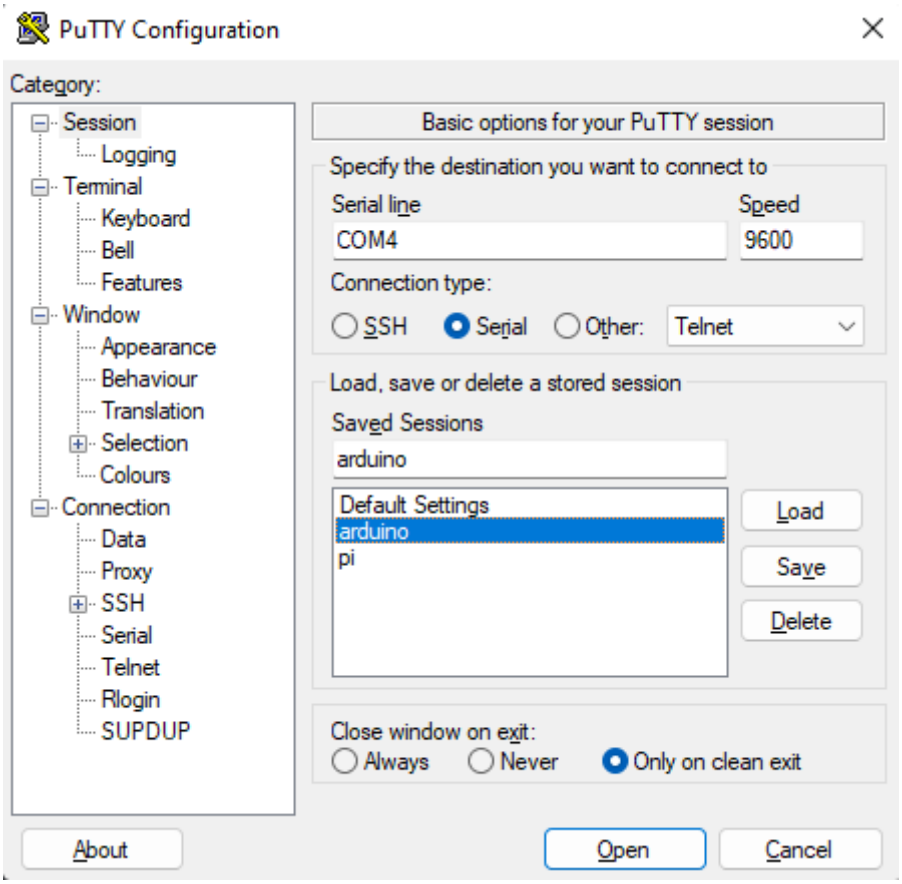
3.4. Google Colab

Google Colab je besplatno razvojno okruženje temeljeno na oblaku koje omogućuje izvođenje Jupyter bilježnica [7]. Pomoću Google Colab-a izvedena je Jupyter bilježnica kojom izvodimo Python kod za treniranje modela u oblaku. Colab pruža moćne resurse u oblaku, uključujući GPU i TPU, što ubrzava proces treniranja modela. Također, Colab nudi mogućnost suradnje i jednostavno dijeljenje bilježnica s drugima.

3.5. PuTTY

PuTTY je besplatni klijentski program otvorenog koda koji podržava SCP, SSH, Telnet, Rlogin i SUPDUP mrežne protokole. Osim pokretanja udaljenih sesija preko mreže podržava i spajanje na lokalni serijski port [8].

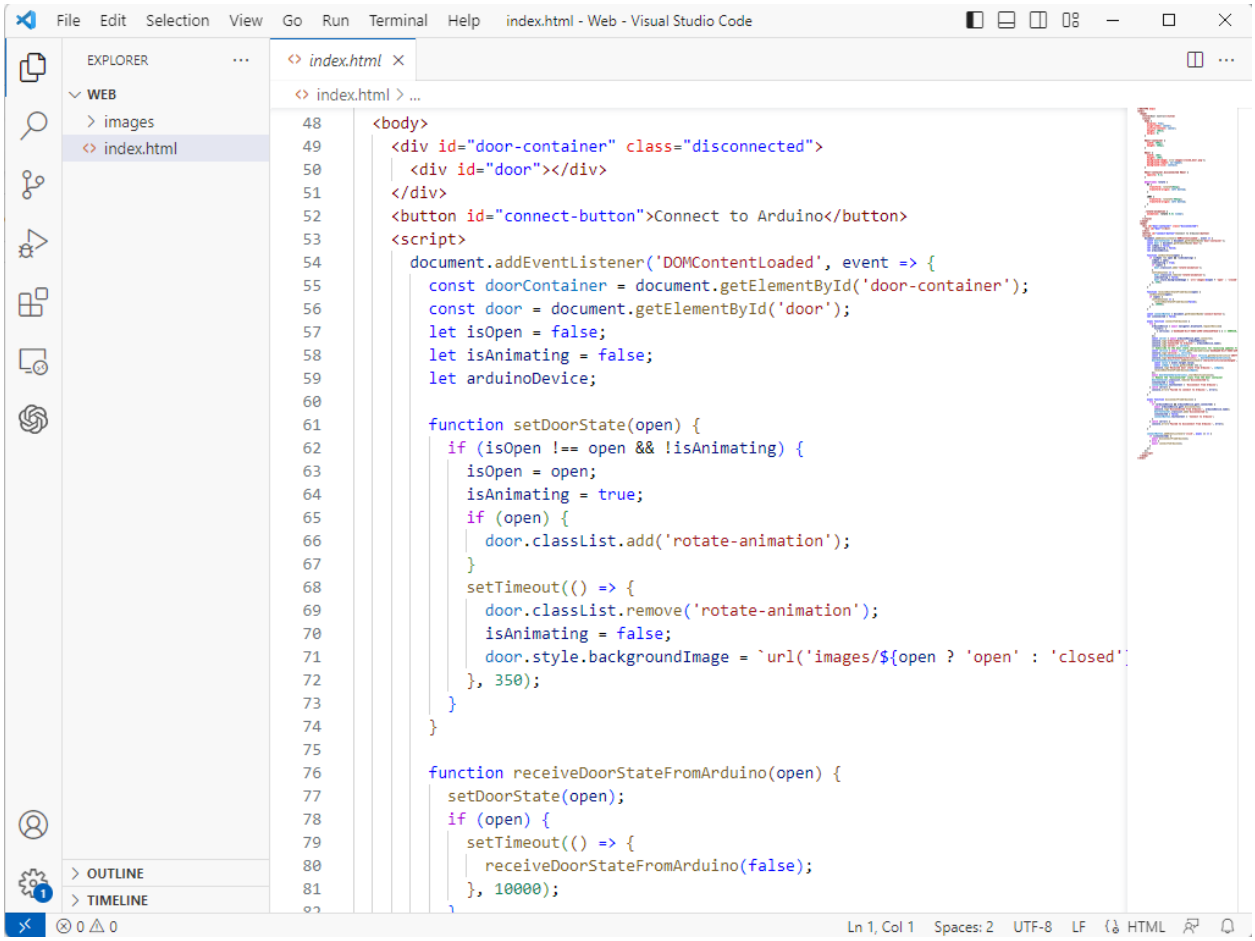
Za serijsku komunikaciju s Arduinom koristit će se PuTTY jer pruža jednostavan način za spremanje čitave sesije u datoteku. Ova funkcionalnost se koristi za čitanje podataka o gestama i spremanje u CSV datoteku.



Slika 3.5.1. *PuTTY korisničko sučelje*

3.6. Visual Studio Code

Visual Studio Code je lagan, ali moćan uređivač izvornog koda koji radi na Windows, macOS i Linux operacijskim sustavima. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js i ima bogat ekosustav proširenja za druge jezike i okruženja (kao što su C++, C#, Java, Python, PHP, Go, .NET) [9].




```
48  <body>
49    <div id="door-container" class="disconnected">
50      <div id="door"></div>
51    </div>
52    <button id="connect-button">Connect to Arduino</button>
53    <script>
54      document.addEventListener('DOMContentLoaded', event => {
55        const doorContainer = document.getElementById('door-container');
56        const door = document.getElementById('door');
57        let isOpen = false;
58        let isAnimating = false;
59        let arduinoDevice;
60
61        function setDoorState(open) {
62          if (isOpen !== open && !isAnimating) {
63            isOpen = open;
64            isAnimating = true;
65            if (open) {
66              door.classList.add('rotate-animation');
67            }
68            setTimeout(() => {
69              door.classList.remove('rotate-animation');
70              isAnimating = false;
71              door.style.backgroundImage = `url('images/${open ? 'open' : 'closed'
72            }, 350);
73          }
74        }
75
76        function receiveDoorStateFromArduino(open) {
77          setDoorState(open);
78          if (open) {
79            setTimeout(() => {
80              receiveDoorStateFromArduino(false);
81            }, 10000);
```

Slika 3.6.1. *Visual Studio Code korisničko sučelje*

4. IZRADA I KORIŠTENJE DIGITALNE BRAVE

Ovo poglavlje sadržava pregled izrađenog rješenja digitalne brave temeljene na senzoru pokreta. Implementacija se sastoji od tri ključna dijela: prikupljanje podataka o gestama uz pomoću inercijskog senzora, treniranje modela za prepoznavanje gesti te prepoznavanje gesti i otvaranje vrata.

4.1. Prikupljanje podataka o gestama

Prvi korak u izradi digitalne brave je prikupljanje podataka o gestama uz pomoć inercijskog senzora s devet osi koji dolazi s ugrađenim kompasom, akcelerometrom i žiroskopom. Za prepoznavanje gesti prikupljat će se podaci žiroskopa i akcelerometra kako bi se precizno prepoznali pokreti ruke korisnika.

4.1.1. Prikupljanje podataka o gestama – programsko rješenje

Za izradu programskog rješenja korišten je ugrađeni LSM9DS1 senzor te pripadajuća Arduino biblioteka koja omogućava čitanje podataka sa senzora.

Nakon uključivanja potrebne datoteke zaglavlja *Arduino_LSM9DS1.h* [10] na početku koda definirane su dvije konstante:

- *ACCELERATION_THRESHOLD* – uz pomoć ove konstante postavljamo prag detekcije kojime reguliramo koliko brz pokret rukom mora biti da bi snimanje geste započelo
  Prag je postavljen na ubrzanje 2,5 puta sile gravitacije tako da se izbjegne detekcija slučajnih, manjh pokreta.
- *NUM_SAMPLES* – ovom konstantom definiramo broj uzoraka koje ćemo isčitati sa senzora, tom konstantom reguliramo duljinu izvođenja geste
  Broj uzoraka je postavljen na 100 što uz stopu uzorkovanja LSM9DS1 senzora koja od 119 Hz daje trajanje geste od otprilike 0,84 sekunde.

Također, definirana je varijabla *samplesRead* koja predstavlja broj dosad isčitanih uzoraka te je inicijalizirana na konstantu *NUM_SAMPLES* koja iznosi 100.

U *setup()* funkciji otvara se serijska komunikacija te se inicira inercijski senzor. Ukoliko dođe do pogreške prilikom inicijalizacije senzora ispisuje se pogreška i izvođenje programa se zaustavlja. Također, ispisuje se tekst zaglavlja koji će biti korišten za treniranje modela za prepoznavanje gesti.

```
#include <Arduino_LSM9DS1.h>

const float ACCELERATION_THRESHOLD = 2.5;  // Threshold of significance in G's
const int NUM_SAMPLES = 100;               // Number of samples to read

int samplesRead = NUM_SAMPLES;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  // Initialize the IMU
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  Serial.println("accelerometerX,accelerometerY,accelerometerZ,gyroscopeX,gyroscopeY,gyroscopeZ");
}
```

Slika 4.1.1. *Inicijalizacija programa za prikupljanje podataka o gestama*

Unutar *loop()* funkcije definirano je šest varijabli, po tri varijable za svaku od osi za akcelerometar i žiroskop u koje će se spremati vrijednosti isčitane sa senzora. Funkcija *loop()* sadrži dvije *while* petlje. Prva *while* petlja se izvodi sve dok je varijabla *samplesRead* jednaka konstanti *NUM_SAMPLES* i na početku provjerava jesu li dostupni podaci akcelerometra te ukoliko jesu, čita ih i sprema u pripadajuće varijable, zatim se ubrzanje u svim smjerovima zbraja te ukoliko prelazi ranije definirani prag detekcije varijabla *samplesRead* se postavlja na nulu.

```
void loop() {
  float accelerometerX, accelerometerY, accelerometerZ;
  float gyroscopeX, gyroscopeY, gyroscopeZ;

  // Wait for significant motion
  while (samplesRead == NUM_SAMPLES) {
    if (IMU.accelerationAvailable()) {
      // Read the acceleration data
      IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);

      // Sum up the absolute values
      float accelerationSum = fabs(accelerometerX) + fabs(accelerometerY) + fabs(accelerometerZ);

      // Check if it's above the threshold
      if (accelerationSum >= ACCELERATION_THRESHOLD) {
        // Reset the sample read count
        samplesRead = 0;
        break;
      }
    }
  }
```

Slika 4.1.2. *Varijable i prva while petlja loop() funkcije*

Druga *while* petlja počinje sa izvođenjem kada dođe do ubrzanja većeg od praga detekcije i izvodi se sve dok je varijabla *samplesRead* manja od konstante *NUM_SAMPLES* što uz vrijednost varijable NUM_SAMPLES 100 i frekvenciju uzorkovanja senzora od 119 Hz daje vrijeme izvršavanja od otprilike 0,84 sekunde. Vrši se provjera jesu li podaci akcelerometra i žiroskopa dostupni te ukoliko jesu spremaju se u odgovarajuće varijable. Varijabla *samplesRead* uvećava se za jedan i ispisuju se podaci s akcelerometra i žiroskopa u CSV formatu za kasnije treniranje modela. Kada varijabla *samplesRead* dosegne vrijednost *NUM_SAMPLES* ispisuje se prazan red koji označava kraj čitanja geste, te izvođenje programa vraća u prvu petlju gdje se čeka ponovno ubrzanje veće od praga detekcije.

```
// Check if all the required samples have been read since
// the last time the significant motion was detected
while (samplesRead < NUM_SAMPLES) {
  // Check if new acceleration and gyroscope data is available
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    // Read the acceleration and gyroscope data
    IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);
    IMU.readGyroscope(gyroscopeX, gyroscopeY, gyroscopeZ);

    samplesRead++;

    // Print the sensor data
    Serial.print(accelerometerX, 3);
    Serial.print(',');
    Serial.print(accelerometerY, 3);
    Serial.print(',');
    Serial.print(accelerometerZ, 3);
    Serial.print(',');
    Serial.print(gyroscopeX, 3);
    Serial.print(',');
    Serial.print(gyroscopeY, 3);
    Serial.print(',');
    Serial.print(gyroscopeZ, 3);
    Serial.println();

    if (samplesRead == NUM_SAMPLES) {
      Serial.println();
    }
  }
}
```

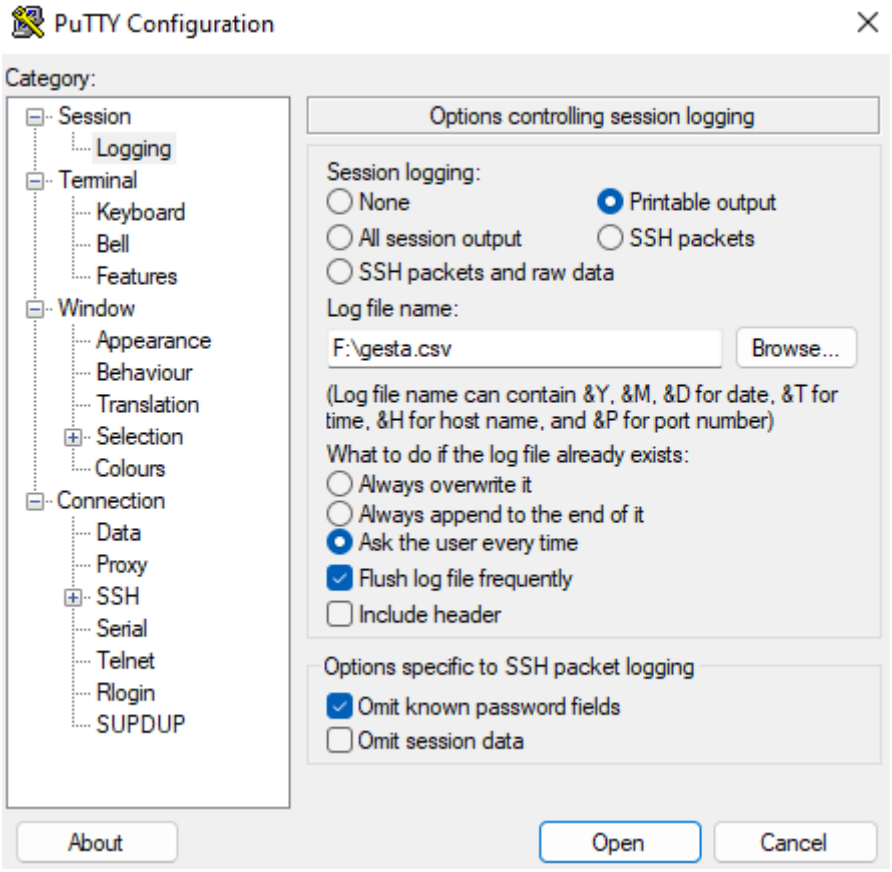
Slika 4.1.3. *Druga while petlja loop() funkcije*

4.1.2. Prikupljanje podataka o gestama – primjena

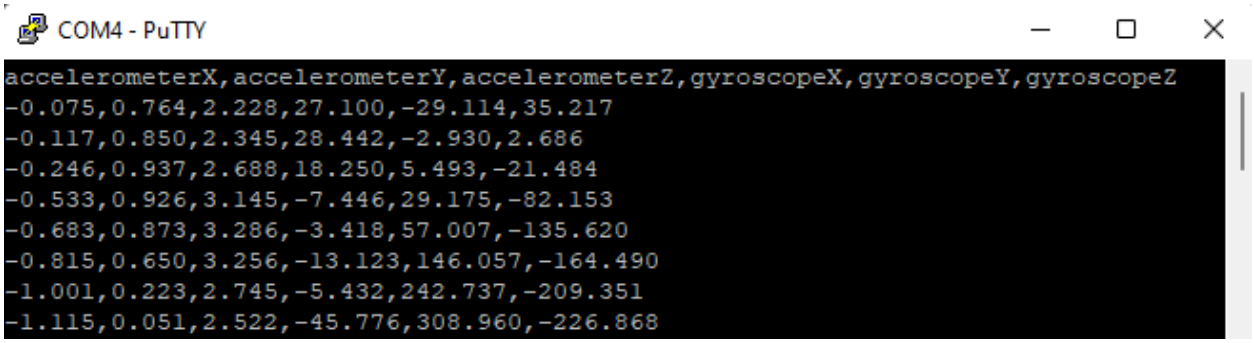
Prikupljanje podataka o gestama vrši se izvođenjem iznad opisanog programskog na Arduino Nano 33 BLE Sense razvojnoj pločici. Serijski ispis sprema se u CSV datoteku korištenjem Putty programa.

Kako bi se podaci sa senzora spremili u CSV datoteku korišten je ranije opisan besplatni program PuTTY. Unutar PuTTY programa, nakon odabira odgovarajućeg serijskog porta potrebno je uključiti spremanje sesije u datoteku. Za omogućavanje spremanja sesije u datoteku potrebno je odabrati Session->Logging kategoriju iz lijevog izbornika kategorija, pod Session logging odabrati Printable output, definirati ime i put spremljene datoteke te isključiti opciju Include header.

Konačne opcije prikazane su na slici 4.1.4. te nakon njihovog postavljanja otvaramo sesiju klikom na gumb Open. Primjer ispisa uspješno uspostavljenje serijske komunikacije prikazan je na slici 4.1.5. Preporučeno je snimanje deset do dvanaest uzoraka svake od gesti, odnosno prilikom prikupljanja podataka o pojedinoj gesti, gestu je potrebno ponoviti barem deset do dvanaest puta.



Slika 4.1.4. *Postavke unutar PuTTY programa za spremanje podataka sesije u CSV datoteku*



Slika 4.1.5. *Ispis serijske komunikacije unutar PuTTY programa*

4.2. Treniranje modela za prepoznavanje gesti

Nakon što su podaci o gestama snimljeni, sljedeći korak je treniranje modela za prepoznavanje gesti. Za treniranje modela koristit će se CSV datoteke dobivene u prethodnom koraku, Python razvojno okruženje u oblaku Google Colab te TensorFlow biblioteka.

U svrhu treniranja modela za prepoznavanja gesti napravljena je Jupyter bilježnica sa detaljno raspisanim koracima u Google Colab-u.

4.2.1. Treniranje modela za prepoznavanje gesti – programsko rješenje

Na početku je potrebno postaviti Python razvojno okruženje, očistiti radni direktorij i instalirati potrebne pakete. Paketi potrebni za izvođenje su:

- xxd – pretvaranje modela u datoteku zaglavlja
- pandas – učitavanje podataka iz CSV datoteka
- numpy – rad s višedimenzionalnim nizovima i matematičke operacije nad njima
- matplotlib – vizualizacija podataka
- tensorflow – biblioteka za treniranje modela

```
# Remove all files and directories in the current working directory
!rm -r *

# Install the 'xxd' package
!apt-get -qq install xxd

# Install required Python packages for data manipulation, numerical computations, and data visualization
!pip install pandas numpy matplotlib

# Install TensorFlow
!pip install tensorflow==2.12.0
```

Slika 4.2.1. *Postavljanje razvojnog okruženja u Google Colab-u*

Nakon postavljanja razvojnog okruženja slijedi prenošenje CSV datoteka na Google Colab. Klikom na *Files* karticu u izborniku s lijeve strane otvara se radni direktorij trenutne sesije koji postoji samo za vrijeme trajanja sesije. Datoteke se mogu prenijeti povlačenjem u otvoreni direktorij ili odabirom datoteka klikom na *Upload to session storage.*

Sljedeći korak je čitanje podataka iz CSV datoteka i pripremanje istih za daljnju obradu. Naredni kod potrebno je izmjeniti ovisno o odabranom broju uzoraka po gesti i o nazivima CSV datoteka. Lista *GESTURES* predstavlja imena gesti odnosno nazive prenesenih CSV datoteka. Varijabla *SAMPLES_PER_GESUTRE* označava broj uzoraka podataka za svaku gestu.

Nadalje, stvara se one-hot enkodirana reprezentacija gesti, *ONE_HOT_ENCODED_GESTURES* što je matrica dimenzija NUM_GESTURES x NUM_GESTURES sa jedinicama na dijagonali i nulama na ostalim mjestima.

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

# Set a random seed for reproducibility
SEED = 1111
np.random.seed(SEED)
tf.random.set_seed(SEED)

# Define the list of gestures
GESTURES = [
    "c_up_push",
    "random_1",
    "random_2",
    "random_3",
    "random_4"
]

# Define the number of samples per gesture
SAMPLES_PER_GESTURE = 100

# Get the number of gestures
NUM_GESTURES = len(GESTURES)

# Create one-hot encoded representations of the gestures
ONE_HOT_ENCODED_GESTURES = np.eye(NUM_GESTURES)
```

Slika 4.2.2. *Uključivanje potrebnih paketa i deklaracija varijabli*

Inicijalizira se lista ulaza i izlaza, *inputs* je lista koja će sadržavati sve ulaze za treniranje modela, *outputs* je lista koja će sadržavati izlaze. Iterira se kroz svaku gestu odnosno svaku CSV datoteku i podaci iz datoteke se učitavaju uz pomoć *read_csv* funkcije pandas biblioteke. Prilikom treniranja korištena je *c_up_push.csv* datoteka koja sadrži uzorke ispravne geste koja će otvarati vrata i četiri CSV datoteke koje sadržavaju uzorke nasumičnih gesti kako bi model znao raspoznati ispravnu gestu od nasumičnih pokreta rukom. Računa se broj uzoraka trenutne geste i iterira se kroz svaki uzorak. Za trenutni uzorak stvaraju se tenzori ulaska sa normaliziranim vrijednostima iz akcelerometra i žiroskopa. Očitane vrijednosti sa žiroskopa i akcelerometra normaliziraju se na vrijednosti između nula i jedan te se dodaju u varijablu *tensor*. Nakon što su obrađeni svi uzorci unutar zapisa geste varijable *tensor* dodaje se u listu *inputs* a varijabla *output* dodaje se u listu *outputs*.

```
# Initialize lists to store inputs and outputs
inputs = []
outputs = []

# Iterate over each gesture
for gesture_index in range(NUM_GESTURES):
    gesture = GESTURES[gesture_index]
    print(f"Processing index {gesture_index} for gesture '{gesture}'.")

    # Get the one-hot encoded output for the current gesture
    output = ONE_HOT_ENCODED_GESTURES[gesture_index]

    # Read the data from the CSV file for the current gesture
    df = pd.read_csv("/content/" + gesture + ".csv")

    # Calculate the number of recordings for the current gesture
    num_recordings = int(df.shape[0] / SAMPLES_PER_GESTURE)

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

    # Iterate over each recording
    for i in range(num_recordings):
        tensor = []

        # Iterate over each sample in the recording
        for j in range(SAMPLES_PER_GESTURE):
            index = i * SAMPLES_PER_GESTURE + j

            # Normalize and scale the accelerometer and gyroscope data
            tensor += [
                (df['accelerometerX'][index] + 4) / 8,
                (df['accelerometerY'][index] + 4) / 8,
                (df['accelerometerZ'][index] + 4) / 8,
                (df['gyroscopeX'][index] + 2000) / 4000,
                (df['gyroscopeY'][index] + 2000) / 4000,
                (df['gyroscopeZ'][index] + 2000) / 4000
            ]

        # Append the tensor as input and the output to the respective lists
        inputs.append(tensor)
        outputs.append(output)
```

Slika 4.2.3. *Iteracije kroz uzorke gesti i kreiranje tenzora*

Konačno, nakon što su obrađene sve geste, liste *inputs* i *outputs* pretvaraju se u numpy nizove uz pomoć funkcije *array* numpy biblioteke.

```
# Convert the input and output lists to numpy arrays
inputs = np.array(inputs)
outputs = np.array(outputs)
```

Slika 4.2.4. *Pretvorba listi u numpy nizove*

Kako bi se model učinkovito trenirao, ulazne i izlazne parove treba nasumično rasporediti i podijeliti u različite skupove za treniranje, provjeru valjanosti i testiranje. Podaci se dijele na način:

- Skup za treniranje – sastoji se od 60% ukupnih ulaznih i izlaznih parova i koristi se za treniranje modela
- Skup za provjeru valjanosti – sastoji se od 20% ukupnih ulaznih i izlaznih parova i koristi se za mjerenje performansi modela tijekom treninga
- Skup za testiranje – sastoji se od 20% ukupnih ulaznih i izlaznih parova i koristi se za testiranje izvedbe modela nakon završetka faze treniranja

Varijabla *num_inputs* sadrži ukupan broj tenzora u listi *inputs*, uz pomoć te informacije kreiramo listu od nula do vrijednosti varijable *num_inputs* umanjenje za jedan koristeći funkciju *arrange* numpy biblioteke. Ova lista predstavlja indekse ulaznih tenzora, te ćemo ih nasumično rasporediti koristeći *random.shuffle* funkciju kako bismo izbjegli obrasce u podacima tijekom treniranja. Liste *inputs* i *outputs* se dijele u skupove funkcijom *split* i spremaju se u nizove *inputs_train*, *input_test*, *inputs_validate*, *outputs_train*, *outputs_test* i *outputs_validate*.

```
# Get the total number of inputs
num_inputs = len(inputs)

# Create an array of indices from 0 to num_inputs
randomize = np.arange(num_inputs)

# Shuffle the indices randomly
np.random.shuffle(randomize)

# Randomize the order of inputs and outputs using the randomized indices
inputs = inputs[randomize]
outputs = outputs[randomize]

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

# Split the inputs based on the defined splits
inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT, TEST_SPLIT])

# Split the outputs based on the defined splits
outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT, TEST_SPLIT])
```

Slika 4.2.5. *Nasumično raspoređivanje i podjela podataka na skupove*

Nakon raspoređivanja i raspodjele podataka u skupove slijedi treniranje modela. Uz pomoć Keras API-ja visoke razine koji pruža visokoproduktivno sučelje za rješavanje problema strojnog učenja kreira se prazan sekvencijalni model funkcijom *tf.keras.Sequential()*. Dodani su slojevi potrebni za treniranje modela te se model kompajlira sa optimizatorom, gubitkom i metrikama. Model se zatim trenira, koriste se ranije kreirani nizovi *inputs_train* i *outputs_train*, parametar *epochs* je postavljen na 600 što znači da će se treniranje sastojati od 600 prolaza kroz set za treniranje, parametar *batch_size* je postavljen na 1 što znači da će se unutranji parametri modela ažurirati prolaskom kroz svaki uzorak u setu za treniranje [11].

```
# Create a sequential model
model = tf.keras.Sequential()

# Add a dense layer with 50 units and ReLU activation function
model.add(tf.keras.layers.Dense(50, activation='relu'))

# Add a dense layer with 15 units and ReLU activation function
model.add(tf.keras.layers.Dense(15, activation='relu'))

# Add a dense layer with NUM_GESTURES units and softmax activation function
# Softmax is used because we expect only one gesture to occur per input
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax'))

# Compile the model with optimizer, loss, and metrics
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

# Train the model
# Use inputs_train and outputs_train as training data
# Train for 600 epochs with a batch size of 1
# Use inputs_validate and outputs_validate as validation data
history = model.fit(inputs_train, outputs_train, epochs=600, batch_size=1, validation_data=(inputs_validate, outputs_validate))
```

Slika 4.2.6. *Treniranje modela*

Trenirani model se zatim pretvara u TensorFlow Lite format koristeći konverter dan u TensorFlow biblioteci. Kreira se konverter koristeći *tf.lite.TFLiteConverter.from_keras_model()* i pozivanjem funkcije *convert*, pretvoreni model sprema se u varijablu *tflite_model* i sprema se na lokalnu pohranu u datoteku *gesture_model.tflite* te se ispisuje veličina pohranjene datoteke.

```
# Convert the model to TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the converted model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

# Get the size of the saved model file
import os
basic_model_size = os.path.getsize("gesture_model.tflite")

# Print the size of the saved model
print("Model is %d bytes" % basic_model_size)
```

Slika 4.2.7. *Pretvorba modela u TensorFlow Lite format*

Trenirani model potrebno je enkodirati u datoteku zaglavlja kako bismo model mogli prenijeti na Arduino. Kreira se *model.h* datoteka koja se dodaje početna linija koda koja započinje deklaraciju polja koje će sadržavati trenirani model. Datoteka *gesture_model.tflite* pretvara se u heksadecimalni oblik i dodaje se u *model.h* datoteku kojoj se dodaje i zagrada za kraj deklaracije polja. Ovim postpukom kreirana je *model.h* datoteka koja sadržava model za prepoznavanje gesti te će u sljedećem koraku biti prenesena na Arduino.

```
# Create the model.h file and write the initial line
!echo "const unsigned char model[] = {" > /content/model.h

# Convert the content of gesture_model.tflite to hexadecimal representation and append it to model.h
!cat gesture_model.tflite | xxd -i >> /content/model.h

# Write the closing line to model.h
!echo "};" >> /content/model.h
```

Slika 4.2.8. *Enkodiranje modela u datoteku zaglavlja*

4.2.2. Treniranje modela za prepoznavanje gesti – primjena

Nakon što su sve geste spremljene u odgovarajuće CSV datoteke pokreće se iznad opisana Jupyter bilježnica unutar Google Colab razvojnog okruženja kojom treniramo model i generiramo datoteku zaglavlja koja sadržava trenirani model.

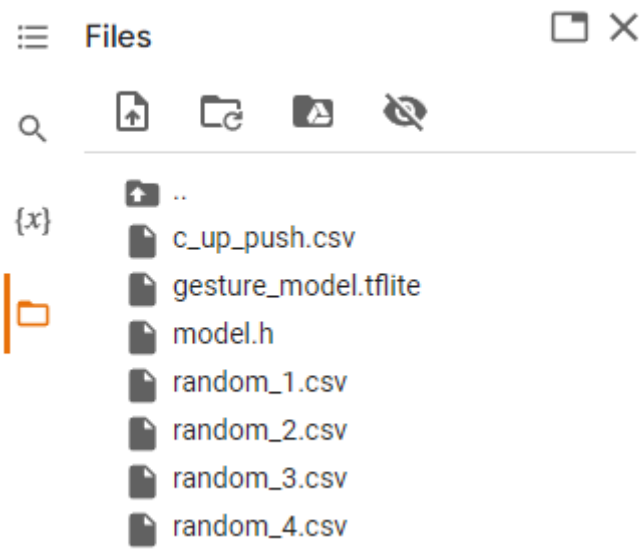
Na kraju procesa treniranja modela generira se *model.h* datoteka zaglavlja koja sadrži trenirani model u obliku Arduino datoteke zaglavlja koja se uključuje u koraku prepoznavanja gesti i otvaranja vrata.

Encode the Model in an Arduino Header File

The next cell creates a constant byte array that contains the TensorFlow Lite model. The provided code converts the model to an Arduino header file format, which can be easily included and used in your Arduino sketch.

```
[6]  # Create the model.h file and write the initial line
     !echo "const unsigned char model[] = {" > /content/model.h

     # Convert the content of gesture_model.tflite to hexadecimal representation and append it to model.h
     !cat gesture_model.tflite | xxd -i >> /content/model.h

     # Write the closing line to model.h
     !echo "};" >> /content/model.h

     # Get the size of the model.h file
     import os
     model_h_size = os.path.getsize("model.h")

     # Print the size of the model.h file
     print(f"Header file, model.h, is {model_h_size:,} bytes.")
     print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")

     Header file, model.h, is 773,828 bytes.

     Open the side panel (refresh if needed). Double click model.h to download the file.
```

Slika 4.2.9. *Ispis posljednje ćelije Jupyter bilježnice za treniranje modela*



Slika 4.2.10. *Konačno stanje izbornika datoteka unutar Google Colab okruženja*

Prilikom treniranja modela uočeno je da je poželjno imati što kompleksniju gestu (koja sadrži kretnje na svim osima) i geste bi se trebale razlikovati jedna od druge kako bi detekcija bila što točnija. U ovome radu korištena je jedna kompleksna gesta koja se koristila kao gesta za otvaranje vrata, dok je druga CSV datoteka predstavljala „ostale“ geste odnosno geste koje se mogu pojaviti no ne trebaju otvoriti vrata.

4.3. Prepoznavanje gesti i otvaranje vrata

Zadnji korak u izradi digitalne brave, nakon prikupljanja podataka i treniranja modela, je prepoznavanje gesti uz pomoć treniranog modela te otvaranje vrata ukoliko je točna gesta prepoznata. Kako bi se demonstrirala funkcionalnost otvaranja vrata uz pomoć prepoznavanja gesti korišten je ugrađeni Bluetooth® 5 Low Energy modul i izrađena je web stranica koja se putem Bluetooth-a spaja na Arduino.

4.3.1. Prepoznavanje gesti i otvaranje vrata – programsko rješenje

Dio koda koji se odnosi na čitanje podataka iz senzora identičan je onome opisanom u poglavlju 4.1., čeka se ubrzanje veće od postavljenog praga detekcije i isčitava se 100 uzoraka.

Dodane su datoteke zaglavlja potrebne za rad sa BLE modulom [12], datoteke zaglavlja TensorFlow Lite biblioteke, datoteke zaglavlja iz TensorFlow Lite Micro okruženja i datoteka zaglavlja dobivena u prošlom koraku koja sadržava trenirani model. Definirane su globalne varijable za TensorFlow Lite Micro:

- *tflErrorReporter* – izvještavanje o pogreškama
- *tflOpsResolver* – pruža operacije koje interpreter koristi za pokretanje modela
- *tflModel* – pokazivač na TensorFlow Lite model
- *tflInterpreter* – interpreter za pokretanje TensorFlow Lite modela
- *tflInputTensor* – pokazivač na ulazne tenzore modela
- *tflOutputTensor* – pokazivač na izlazne tenzore modela

```
#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
#include "model.h"
```

Slika 4.3.1. *Datoteke zagljavlja*

```
// Global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;
tflite::AllOpsResolver tflOpsResolver;
const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;
```

Slika 4.3.2. *Globalne varijable za TensorFlow Lite Micro*

Stvara se statički memorijski spremnik za tenzore, definirane su dvije varijable:

- *TENSOR_ARENA_SIZE* – veličina spremnika za tenzore
- *tensorArena* – spremnik za tenzore

Ostale varijable potrebne za izvođenje programa:

- *gestureNames* – niz koji mapira indeks geste na ime geste
- *CORRECT_GESTURE_INDEX* – indeks ispravne geste
- *NUM*_GESTURES – broj gesti
- *name* – varijabla koja će sadržavati ime uređaja za Bluetooth konekciju
- *openDoorService* – pokazivač na BLE servis
- *openDoorCharacteristic* – pokazivač na BLE karakteristiku

```
// Create a static memory buffer for TFLM
constexpr int TENSOR_ARENA_SIZE = 8 * 1024;
byte tensorArena[TENSOR_ARENA_SIZE] __attribute__((aligned(16)));

// Array to map gesture index to a name
const char* gestureNames[] = {
  "c_up_push",
  "random_1",
  "random_2",
  "random_3",
  "random_4"
};

// Index of the correct gesture
constexpr int CORRECT_GESTURE_INDEX = 0;

#define NUM_GESTURES (sizeof(gestureNames) / sizeof(gestureNames[0]))

// String to calculate the local and device name
String name;

BLEService* openDoorService = nullptr;
BLEUnsignedCharCharacteristic* openDoorCharacteristic = nullptr;
```

Slika 4.3.3. *Varijable potrebne za izvođenje programa*

Unutar *setup()* funkcije inicira se inercijski senzor i BLE modul, ukoliko dođe do pogreške zaustavlja se izvođenje programa i ispisuje se greška. Zatim se stvaraju BLE servis i BLE karakteristika sa svojim karakterističnim UUID-evima, karakteristici se postavljaju dozvole za čitanje i obavještavanje. Dohvaća se BLE adresa uređaja te se uz pomoć nje tvori lokalno ime uređaja koje se zatim i postavlja. Stvoreni BLE servis postavlja se kao oglašavani servis i dodaje dodaje mu se prethodno stvorena karakteristika. Servis se dodaje u BLE i postavlja se početna vrijednost koja predstavlja stanje otvorenosti vrata na 0. Pozivanjem funkcije *BLE.advertise()* započinje Bluetooth oglavašavanje.

```
openDoorService = new BLEService("be28eab6-8cc7-4d63-a396-134e1e26fb1d");
openDoorCharacteristic = new BLEUnsignedCharCharacteristic("e04ff209-6ae4-45e3-8a38-0baefdba9b53", BLERead | BLENotify);

String address = BLE.address();

Serial.print("address = ");
Serial.println(address);

address.toUpperCase();

name = "BLESense-";
name += address[address.length() - 5];
name += address[address.length() - 4];
name += address[address.length() - 2];
name += address[address.length() - 1];

Serial.print("name = ");
Serial.println(name);

BLE.setLocalName(name.c_str());
BLE.setAdvertisedService(*openDoorService);
openDoorService->addCharacteristic(*openDoorCharacteristic);
BLE.addService(*openDoorService);
openDoorCharacteristic->writeValue(0);

BLE.advertise();
```

Slika 4.3.4. *Postavljanje Bluetooth konekcije unutar setup() funkcije*

Prethodno deklarirane varijable za TensorFlow Lite Micro se inicijaliziraju, prvo se dohvaća TensorFlow Lite reprezentacija modela iz uvezene datoteke zaglavlja koja sadržava trenirani model, i provjerava se kompatibilnost verzije modela i verzije TensorFlow Lite sheme, ukoliko nisu kompatibilne ispisuje se odgovarajuća poruka i prekida se izvođenje programa.

Nakon toga stvara se interpreter koji će izvršavati model, dodjeljuje mu se model (*tflModel*), objekt koji pruža operacije za pokretanje modela (*tflInterpreter*), memorija za pohranu tensora i njena veličina (*tensorArena*, *TENSOR_ARENA_SIZE*) te objekt za izvještavanje o pogreškama (*tflErrorReporter*). Pozivanjem funkcije *tflInterpreter->AllocateTensors()* alocira se memorija za ulazne i izlazne tenzore modela.

Dalje se dohvaćaju pokazivači na ulazni i izlazni tenzor modela i spremaju se u varijable *tflInputTensor* i *tflOutputTensor*.

```
// Get the TFL representation of the model byte array
tflModel = tflite::GetModel(model);
if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
  Serial.println("Model schema mismatch!");
  while (1);
}

// Create an interpreter to run the model
tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, TENSOR_ARENA_SIZE, &tflErrorReporter);

// Allocate memory for the model's input and output tensors
tflInterpreter->AllocateTensors();

// Get pointers for the model's input and output tensors
tflInputTensor = tflInterpreter->input(0);
tflOutputTensor = tflInterpreter->output(0);
```

Slika 4.3.5. *Postavljanje TensorFlow Lite okruženja unutar setup() funkcije*

Na početku *loop()* funkcije vrši se provjera povezanosti sa centralnim Bluetooth uređajem, ukoliko poveznost postoji ispisuje se adresa centralnog uređaja. Ostatak koda izvodi se samo ukoliko postoji povezanost sa centralnim uređajem.

```
BLEDevice central = BLE.central();
if (central) {
  Serial.print("Connected to central: ");
  // Print the central's BT address
  Serial.println(central.address());

  while (central.connected()) {
    float accelerometerX, accelerometerY, accelerometerZ;
    float gyroscopeX, gyroscopeY, gyroscopeZ;

    // Wait for significant motion
    while (samplesRead == NUM_SAMPLES) {
      if (IMU.accelerationAvailable()) {
        // Read the acceleration data
        IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);

        // Sum up the absolute values
        float accelerationSum = fabs(accelerometerX) + fabs(accelerometerY) + fabs(accelerometerZ);

        // Check if it's above the threshold
        if (accelerationSum >= ACCELERATION_THRESHOLD) {
          // Reset the sample read count
          samplesRead = 0;
          break;
        } else {
          break;
        }
      }
    }
```

Slika 4.3.6. *Provjera povezanosti sa centralnim uređajem i praga detekcije*

Vrši se čitanje podataka gesti sa inercijskog senzora na način opisan u poglavlju 4.1. te se podaci spremaju u ulazne tenzore modela.

```
// Normalize the IMU data between 0 and 1, and store it in the model's input tensor
tflInputTensor->data.f[samplesRead * 6 + 0] = (accelerometerX + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 6 + 1] = (accelerometerY + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 6 + 2] = (accelerometerZ + 4.0) / 8.0;
tflInputTensor->data.f[samplesRead * 6 + 3] = (gyroscopeX + 2000.0) / 4000.0;
tflInputTensor->data.f[samplesRead * 6 + 4] = (gyroscopeY + 2000.0) / 4000.0;
tflInputTensor->data.f[samplesRead * 6 + 5] = (gyroscopeZ + 2000.0) / 4000.0;
```

Slika 4.3.7. *Normalizacija podataka inercijskog senzora i spremanje u ulazne tenzore modela*

Nakon učitavanja geste (kada varijabla *samplesRead* bude jednaka varijabli *NUM_SAMPLES*) pozivanjem funkcije tflInterpreter->Invoke() izvršava se inferencija, interpreter obrađuje ulazne podatke i generira izlazne rezultate temeljene na modelu. Pojam inferencije odnosi se na proces izvršavanja TensorFlow Lite modela na uređaju kako bi se napravila predviđanja na temelju ulaznih podataka. Da bi se izvršila inferencija TensorFlow Lite modela, ona mora biti pokrenuta kroz interpreter. Interpreter TensorFlow Lite biblioteke dizajniran je da bude jednostavan i brz te koristi statički poredak grafova i prilagođeni alokator memorije kako bi osigurao minimalno kašnjenje učitavanja, inicijalizacije i izvršenja [13]. Provjerava se uspješnost izvođenja inferencije, statusni kod *kTfLiteOk* označava uspješno izvršavanje, ukoliko inferencija nije bila uspješna ispisuje se odgovarajuća poruka i prekida se izvođenje programa. For petljom prolazi se geste, ispisuje se ime geste i pripadajuća vrijednost izlaznog tenzora (*tflOutputTensor->data.f[i]*) sa preciznošću od šest decimalnih mjesta. Provjerava se je li indeks trenutne geste jednak prethodno postavljenom indeksu ispravne geste i ukoliko je vrijednost izlaznog tenzora za tu gestu veća od 0.8 (sigurnost da je gesta prepoznata veća od 80%), ukoliko su oba uvjeta ispunjena ispisuje se odgovarajuća poruka i karakteristika (*openDoorCharacteristic*) se postavlja na 1, što obavještava centralni uređaj (web-stranicu) da je došlo do promjene karakteristike čime se pokreće otvaranje vrata.

```
if (samplesRead == NUM_SAMPLES) {
  // Run inference
  TfLiteStatus invokeStatus = tflInterpreter->Invoke();
  if (invokeStatus != kTfLiteOk) {
    Serial.println("Invoke failed!");
    while (1);
    return;
  }

  // Loop through the output tensor values from the model
  for (int i = 0; i < NUM_GESTURES; i++) {
    Serial.print(gestureNames[i]);
    Serial.print(": ");
    Serial.println(tflOutputTensor->data.f[i], 6);
    // Check if the read gesture is the correct one
    if (i == CORRECT_GESTURE_INDEX && tflOutputTensor->data.f[i] > 0.8) {
      Serial.println("Correct gesture detected.");
      openDoorCharacteristic->writeValue(1);
    }
  }
  Serial.println();
}
```

Slika 4.3.8. *Inferencija modela i detekcija ispravne geste*

U svrhu demonstracije funkcionalnosti otvaranja vrata izrađena je jednostavna web-stranica. Prilikom izrade web-stranice korišteni su HTML, CSS i JavaScript te Visual Studio Code uređivač koda.

U HTML dijelu stranice unutar <*head*> oznaka postavljen je naslov stranice i unutar <*style*> oznaka definirani su CSS stilovi elemenata.

CSS dio stranice sastoji se od sljedećih dijelova:

- *body* – postavlja visinu tijela dokumenta na 100% visine preglednika i centrira sadržaj stranice
- *#door-container* – definira stil za kontejner koji sadržava sliku vrata, postavlja visinu i širinu kontejnera na 400 piksela
- *#door* – definira stil za element vrata, visinu i širinu postavlja na 100% kontejnera i početnu sliku postavlja na sliku zatvorenih vrata

- *#door-container.disconnected* – povećava prozirnost slike vrata dok nema konekcije s Arduino uređajem
- *@keyframes rotate* – definira CSS animaciju *rotate* koja rotira vrata za 90 stupnjeva po Y osi
- *.rotate-animation* – koristi se za primjenjivanje animacije *rotate* na element vrata trajanja pola sekunde

```
<head>
  <title>Door Control</title>
  <style>
    body {
      display: flex;
      align-items: center;
      justify-content: center;
      height: 100vh;
      margin: 0;
    }

    #door-container {
      width: 400px;
      height: 400px;
    }

    #door {
      width: 100%;
      height: 100%;
      background-image: url('images/closed_door.png');
      background-repeat: no-repeat;
      background-size: contain;
    }

    #door-container.disconnected #door {
      opacity: 0.2;
    }

    @keyframes rotate {
      0% {
        transform: rotateY(0deg);
        transform-origin: left bottom;
      }

      100% {
        transform: rotateY(-90deg);
        transform-origin: left bottom;
      }
    }

    .rotate-animation {
      animation: rotate 0.5s linear;
    }
  </style>
</head>
```

Slika 4.3.9. *Zaglavlje stranice*

Unutar <*body*> oznaka opisano je tijelo stranice koje se sastoji od kontejnera vrata, elementa unutar kontejnera koji sadrži sliku vrata (slike preuzete s izvora [14]) te gumba za uspostavljanje ili prekidanje Bluetooth veze s Arduinom.

```
<div id="door-container" class="disconnected">
  <div id="door"></div>
</div>
<button id="connect-button">Connect to Arduino</button>
```

Slika 4.3.10. *Elementi unutar tijela stranice*

JavaScript dio koda koristi se za spajanje na Arduino putem Bluetootha i primjenu odgovarajućih CSS stilova kada je Arduino uređaj spojen ili odspojen, te kada su vrata otvorena ili zatvorena.

Na početku su definirane varijable koje sadržavaju reference na HTML elemente (kontejner vrata i slika vrata) te varijable koje prate stanje otvorenosti vrata, stanje animiranja slike vrata i varijabla koja sadrži informacije o povezanom Arduino uređaju koja će se koristiti za provjeru povezanosti.

```
const doorContainer = document.getElementById('door-container');
const door = document.getElementById('door');
let isOpen = false;
let isAnimating = false;
let arduinoDevice;
```

Slika 4.3.11. *Definicija varijabli doorContainer, door, isOpen, isAnimating i arduinoDevice*

Funkcija *setDoorState(open)* postavlja stanje vrata (otvoreno ili zatvoreno ovisno o paramteru *open*) i animira rotaciju vrata. Ako su vrata otvorena, dodaje se klasa *.rotate-animation* elementu vrata kako bi se pokrenula rotacija. Nakon završetka animacije, elementu *door* se postavlja odgovarajuća slika ovisna o stanju otvorenosti vrata.

```
function setDoorState(open) {
  if (isOpen !== open && !isAnimating) {
    isOpen = open;
    isAnimating = true;
    if (open) {
      door.classList.add('rotate-animation');
    }
    setTimeout(() => {
      door.classList.remove('rotate-animation');
      isAnimating = false;
      door.style.backgroundImage = `url('images/${open ? 'open' : 'closed'}_door.png')`;
    }, 350);
  }
}
```

Slika 4.3.12. *Funkcija setDoorState(open)*

Funkcija *receiveDoorStateFromArduino(open)* prima informaciju o stanju vrata od Arduino uređaja i poziva funkciju *setDoorState(open)* kako bi se postavilo stanje vrata na temelju primljenih podataka. Ako su vrata otvorena, postavlja se vremensko kašnjenje od 10 sekundi, a zatim se ponovno poziva funkcija *receiveDoorStateFromArduino* s parametrom *false* kako bi se simuliralo zatvaranje vrata nakon 10 sekundi.

```
function receiveDoorStateFromArduino(open) {
  setDoorState(open);
  if (open) {
    setTimeout(() => {
      receiveDoorStateFromArduino(false);
    }, 10000);
  }
}
```

Slika 4.3.13. *Funkcija receiveDoorStateFromArduino(open)*

Dalje su definirane varijable *connectButton* koja sadržava referencu na gumb za povezivanje ili prekidanje veze s Arduinom i varijabla *isConnected* koja prati stanje povezanosti s Arduinom.

```
const connectButton = document.getElementById('connect-button');
let isConnected = false;
```

Slika 4.3.14. *Definicija varijabli connectButton i isConnected*

Funkcija *connectToArduino()* služi za uspostavljanje Bluetooth veze s Arduino uređajem. Filtrira dostupne uređaje prema određenoj UUID vrijednosti servisa (vrijednost postavljena na slici 4.3.4.). Ukoliko je veza uspješno uspostavljena, pretplaćuje se na obavijesti o karakteristici stanja vrata, uklanja klasu *.disconnected* sa kontejnera vrata, postavlja vrijednost varijable *isConnected* na *true* te mijenja tekst gumba za spajanje u gumb za prekid veze.

```
async function connectToArduino() {
  try {
    arduinoDevice = await navigator.bluetooth.requestDevice({
      filters: [
        { services: ['be28eab6-8cc7-4d63-a396-134e1e26fb1d'] } // SERVICE_UUID
      ]
    });
    const server = await arduinoDevice.gatt.connect();
    console.log('arduinoDevice:', arduinoDevice);
    console.log('Connected to Arduino:', arduinoDevice.name);
    console.log('server:', server);
    // Subscribe to the door state characteristic for receiving updates from Arduino
    const service = await server.getPrimaryService('be28eab6-8cc7-4d63-a396-134e1e26fb1d');
    console.log('service:', service);
    const doorStateCharacteristic = await service.getCharacteristic('e04ff209-6ae4-45e3-8a38-0baefdba9b53');
    console.log('doorStateCharacteristic:', doorStateCharacteristic);
    doorStateCharacteristic.addEventListener('characteristicvaluechanged', event => {
      const value = event.target.value;
      const isOpen = value.getUint8(0) === 1;
      console.log('Received door state from Arduino:', isOpen);
      receiveDoorStateFromArduino(isOpen);
    });
    await doorStateCharacteristic.startNotifications();
    // Remove the "disconnected" class from the door container
    doorContainer.classList.remove('disconnected');
    isConnected = true;
    connectButton.textContent = 'Disconnect from Arduino';
  } catch (error) {
    console.error('Failed to connect to Arduino:', error);
  }
}
```

Slika 4.3.15. *Funkcija connectToArduino()*

Funkcija *disconnectFromArduino()* prekida vezu s Arduino uređajem ukoliko je uređaj povezan, dodaje klasu *.disconnected* na kontejner vrata, postavlja vrijednost *isConnected* varijable na *false* i mijenja tekst gumba za prekid veze u gumb za spajanje.

```
async function disconnectFromArduino() {
  try {
    if (arduinoDevice && arduinoDevice.gatt.connected) {
      await arduinoDevice.gatt.disconnect();
      console.log('Disconnected from Arduino:', arduinoDevice.name);
      doorContainer.classList.add('disconnected');
      isConnected = false;
      connectButton.textContent = 'Connect to Arduino';
    }
  } catch (error) {
    console.error('Failed to disconnect from Arduino:', error);
  }
}
```

Slika 4.3.16. *Funkcija disconnectFromArduino()*

Na gumb *connectButton* postavljen je slušatelj događaja koji se poziva na klik gumba, ovisno o stanju *isConnected* pozivaju se funkcije za spajanje ili prekid veze s Arduinom.

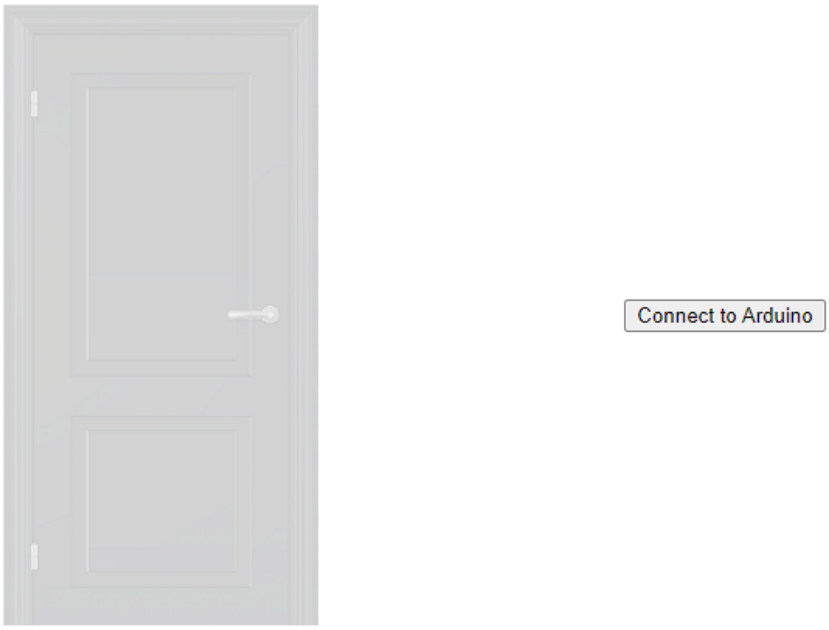
```
connectButton.addEventListener('click', async () => {
  if (isConnected) {
    await disconnectFromArduino();
  } else {
    await connectToArduino();
  }
});
```

Slika 4.3.17. *Slušatelj klik događaja na gumb connectButton*

4.3.2. Prepoznavanje gesti i otvaranje vrata – primjena

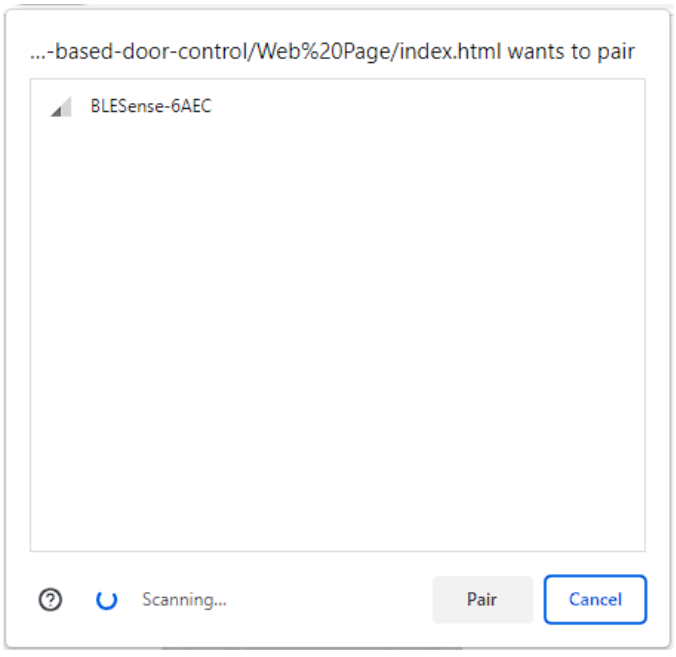
Nakon preuzimanja *model.h* datoteke generirane u poglavlju 4.2, ista se prenosi u direktorij koji sadrži kod za prepoznavanje gesti. Programski kod se zatim prenosi na Arduino Nano 33 BLE Sense razvojnu pločicu.

Kada je programski kod prenesen i pokrenut na razvojnoj pločici pokreće se web-stranica otvaranjem datoteke index.html. Slika vrata je prozirna prije spajanja s Arduino pločicom putem Bluetootha kako bi se indiciralo da još nema konekcije između centrale i pločice.



Slika 4.3.18. *Web-stranica prije spajanja s razvojnom pločicom*

Gumb „Connect to Arduino“ služi za spajanje s pločicom putem Bluetootha, klikom na gumb (u web-pregledniku Google Chrome, na uređaju s Bluetooth podrškom) otvara se izbornik koji prikazuje samo razvojnu pločicu (zbog postavljenih filtera). Klikom na ime uređaja i klikom na gumb „Pair“ ostvaruje se konekcija između centrale (web-preglednika) i razvojne pločice.



Slika 4.3.19. *Izbornik za spajanje s Bluetooth uređajima*

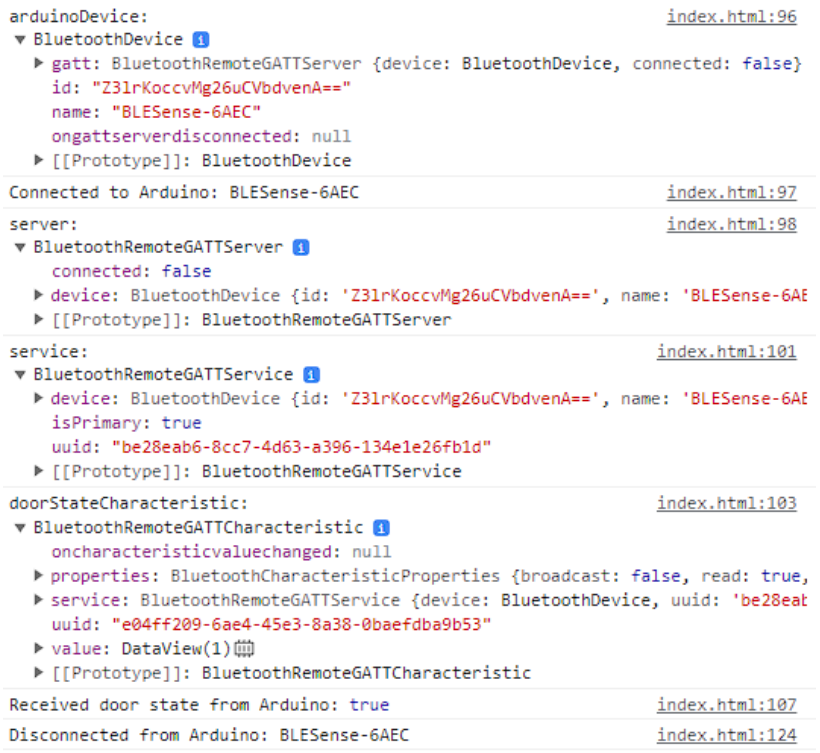
Nakon uspješnog spajanja, slika vrata prestaje biti prozirna i web-stranica čeka podatke od razvojne pločice.

Slika 4.3.20. *Web-stranica nakon spajanja s razvojnom pločicom*

Nakon što je putem Bluetooth konekcije poslana poruka da se vrata otvore, vrata se otvaraju uz animaciju i ostaju otvorena 10 sekundi nakon čega se sama zatvaraju.

Slika 4.3.21. *Web-stranica nakon primitka poruke o otvaranju vrata*

Klikom na gumb „Disconnect from Arduino“ prekida se Bluetooth konekcija između web-stranice i razvojne pločice te se stanje web-stranice vraća u ono sa slike 4.3.18.



Slika 4.3.22. *Ispis konzole za čitav proces od spajanja, otvaranja vrata do prekida veze*

5. EVALUACIJA RADA SUSTAVA

Kako bi se postigla zadovoljavajuća preciznost detekcije gesti korisnika, nakon što je kompletno programsko rješenje implementirano bilo je potrebno mijenjati parametre sustava.

Isprva, korištena je jedna CSV datoteka koja je predstavljala ispravnu gestu, ona koja otvara vrata, i jedna CSV datoteka koja je predstavljala nasumične pokrete rukom, koji ne otvaraju vrata te je korišteno osam uzoraka za ispravnu i osam uzoraka za nasumične geste. Rezultati detekcije su u ovom slučaju bili loši i nezadovoljavajući, detekcija gesti nije radila ispravno te se u nekim slučajevima ispravna gesta detektirala kao nasumična i obrnuto.

Kako bi se poboljšala detekcija povećao se broj uzoraka pojedine geste na dvanaest što je dalo bolje rezultate, tada se ispravna gesta detektirala no rezultatu modela se moglo vjerovati tek kada je bio preko 99% siguran da je prepoznata pojedina gesta.

Zadnje potrebno poboljšanje bilo je dodati dodatne nasumične geste kako bi se pokrilo što više mogućih pokreta rukom koja nisu ispravna gesta. Izrađene su četiri CSV datoteke nasumičnih pokreta i jedna CSV datoteka ispravne geste. Nakon uvedenih promjena uočeno je kako je dovoljna sigurnost modela od 80% da bi bili sigurni da je detektirana pojedina gesta.

Tijekom evaluacije rada sustava uočeno je kako je za ispravnu gestu dobro koristiti kompleksniji pokret rukom kako bi šansa slučajnog rekreiranja točne geste za otvaranje vrata bila što manje i kako bi sustav bio sigurniji.

U konačnici, kombinacija povećanja broja uzoraka, postavljanje visokih pragova sigurnosti za detekciju ispravne geste, koristeći kompleksnije pokrete i raznovrsne nasumične geste, doprinijela je značajnom poboljšanju preciznosti detekcije gesti korisnika. Ovaj pristup omogućio je sustavu da pouzdano i precizno prepoznaje ispravne geste te smanjuje mogućnost slučajnih i nepoželjnih detekcija.

6. ZAKLJUČAK

Zadatak ovog diplomskog rada bio je realizirati digitalnu bravu korištenjem senzora pokreta na Arduino Nano 33 BLE Sense razvojnoj pločici. Napravljen je pregled sličnih postojećih rješenja, pregled korištenih tehnologija te opis programskog rješenja i testiranje realiziranog sustava. Tijekom izrade rješenja korištena su moderna razvoja okruženja (Arduino IDE 2, Google Colab) te popularna biblioteka za strojno učenje (TensorFlow).

Izvedeno rješenje sastoji se od tri dijela:

- Prikupljanje podataka o gestama – spremanje podataka žiroskopa i akcelerometra gesti u CSV datoteke
- Treniranje modela za prepoznavanje gesti – treniranje modela za prepoznavanje gesti uz pomoć CSV datoteka iz prethodnog dijela, generiranje datoteke zaglavlja koja sadržava trenirani model
- Prepoznavanje gesti i otvaranje vrata – primjena treniranog modela za prepoznavanje gesti, izrada web-stranice s kojom se komunicira putem Bluetooth konekcije i ovisno o prepoznatoj gesti otvaraju vrata

Rješenje je uspješno prikupljalo podatke o gestama, treniralo model za prepoznavanje gesti i omogućavalo otvaranje vrata na temelju prepoznate geste. Kvaliteta prepoznavanja gesti je evaluirana kroz testiranje sustava sa različitim kombinacijama gesti i pragovima detekcije sve dok nije postignuta zadovoljavajuća točnost.

Daljnji razvoj rješenja bi mogao sadržavati podršku za više gesti koje otvaraju vrata, podrška za geste različitih vremena trajanja, pojednostavljenje cijelog procesa od prikupljanja podataka o gestama do otvaranja vrata te izrada kućišta za razvojnu pločicu.

LITERATURA

[1] »Hands-free opening solutions from ASSA ABLOY,« [Mrežno]. Available: https://www.assaabloydss.com/en/solutions/solutions-by-challenge/reducing-the-spread-of-germs/hands-free-solutions. [Pokušaj pristupa 4 svibanj 2023.].

[2] »ASSA ABLOY – Norton Door Controls 704 Wave to Open Switch,« [Mrežno]. Available: https://www.allmar.com/products/assa-abloy-norton-door-controls-704-wave-to-open-switch/. [Pokušaj pristupa 4 svibanj 2023.].

[3] C. C. C. N. Ryan Chan, »Touchless Hand Gesture Door Opener,« [Mrežno]. Available: https://www.hackster.io/316847/touchless-hand-gesture-door-opener-474c2a. [Pokušaj pristupa 5 svibanj 2023.].

[4] »Nano 33 BLE Sense | Arduino Docs,« [Mrežno]. Available: https://docs.arduino.cc/hardware/nano-33-ble-sense. [Pokušaj pristupa 4 svibanj 2023.].

[5] »Getting Started with Arduino IDE 2,« [Mrežno]. Available: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2. [Pokušaj pristupa 6 svibanj 2023.].

[6] »TensorFlow,« [Mrežno]. Available: https://www.tensorflow.org/. [Pokušaj pristupa 6 svibanj 2023.].

[7] »Google Colab FAQ,« [Mrežno]. Available: https://research.google.com/colaboratory/faq.html. [Pokušaj pristupa 7 svibanj 2023.].

[8] S. Tatham, »PuTTY User Manual,« [Mrežno]. Available: https://the.earth.li/~sgtatham/putty/0.78/htmldoc/. [Pokušaj pristupa 7 svibanj 2023.].

[9] »Documentation for Visual Studio Code,« [Mrežno]. Available: https://code.visualstudio.com/docs. [Pokušaj pristupa 8 svibanj 2023.].

[10] »Arduino_LSM9DS1,« [Mrežno]. Available: https://www.arduino.cc/reference/en/libraries/arduino_lsm9ds1/. [Pokušaj pristupa 8 svibanj 2023.].

[11] »arduino_tinyml_workshop.ipynb,« [Mrežno]. Available: https://colab.research.google.com/github/arduino/ArduinoTensorFlowLiteTutorials/blob/master/GestureToEmoji/arduino_tinyml_workshop.ipynb#scrollTo=kGNFa-lX24Qo. [Pokušaj pristupa 8 Svibanj 2023].

[12] »ArduinoBLE,« [Mrežno]. Available: https://www.arduino.cc/reference/en/libraries/arduinoble/. [Pokušaj pristupa 8 svibanj 2023.].

[13] »TensorFlow Lite inference,« [Mrežno]. Available: https://www.tensorflow.org/lite/guide/inference. [Pokušaj pristupa 18 Kolovoz 2023].

[14] »Door Png - Open Door Closed Door,« [Mrežno]. Available: https://www.pngkit.com/view/u2a9o0i1e6t4r5t4_door-png-open-door-closed-door/. [Pokušaj pristupa 15 svibanj 2023.].

SAŽETAK

U ovom diplomskom radu je provedeno istraživanje i implementacija digitalne brave korištenjem senzora pokreta na Arduino Nano 33 BLE Sense razvojnoj pločici. Rješenje se sastoji od tri ključna dijela. Prvo, podaci o gestama su prikupljani putem žiroskopa i akcelerometra te su spremljeni u CSV datoteke. Zatim je korištena TensorFlow biblioteka za strojno učenje kako bi se trenirao model za prepoznavanje gesti koristeći prikupljene podatke. Nakon treniranja, generirana je datoteka zaglavlja koja sadrži trenirani model. Konačno, trenirani model se koristi za prepoznavanje gesti te se uspostavlja komunikacija s web-stranicom putem Bluetooth-a kako bi se otvorila vrata ovisno o prepoznatoj gesti.

Rješenje je uspješno prikupljalo podatke o gestama, treniralo model za prepoznavanje gesti te omogućavalo otvaranje vrata na temelju prepoznate geste. Kvaliteta prepoznavanja gesti je evaluirana kroz testiranje sustava s različitim kombinacijama gesti i pragovima detekcije sve dok nije postignuta zadovoljavajuća točnost.

Ključne riječi: Arduino, Digitalna brava, Kontrola pristupa, Prepoznavanje gesti, TensorFlow

ABSTRACT

In this master's thesis, the research and implementation of a digital lock using a motion sensor on the Arduino Nano 33 BLE Sense development board was carried out. The solution consists of three key parts. First, gesture data was collected via gyroscope and accelerometer and saved in CSV files. The TensorFlow machine learning library was then used to train a gesture recognition model using the collected data. After training, a header file containing the trained model is generated. Finally, the trained model is used for gesture recognition and communicates with a website via Bluetooth to open the door depending on the recognized gesture.

The solution successfully collected data on gestures, trained a model for gesture recognition and enabled door opening based on the recognized gesture. The quality of gesture recognition was evaluated by testing the system with different combinations of gestures and detection thresholds until satisfactory accuracy was achieved.

Key words: Arduino, Digital lock, Access control, Gesture recognition, TensorFlow

ŽIVOTOPIS

Marko Pavičić rođen je u Osijeku 26. lipnja 1999. godine. U Osijeku završava osnovnu školu Jagode Truhelke. Nakon završene osnovne škole upisuje Elektrotehničku i prometnu školu Osijek gdje 2018. stječe zvanje Elektrotehničar. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku preddiplomski studij Elektrotehnike, smjera „Komunikacije i informatika“.

Preddiplomski studij elektrotehnike završava 2021. godine sa završnim radom „Mobilna aplikacija za anketiranje u stvarnom vremenu“ i stječe akademski naziv sveučilišni prvostupnik (baccalaureus) inženjer Elektrotehnike. Iste godine upisuje diplomski studij Elektrotehnike, smjera „Komunikacije i informatika“ i odjela „DKB – Mrežne tehnologije“

PRILOZI

Prilog 1: Programski kod za prikupljanje podataka o gestama

```
/*
  Motion Detection and Sensor Data Logging

  This Arduino sketch demonstrates motion detection using the LSM9DS1 IMU (Inertial Measurement Unit).
  It waits for a significant motion event, defined by a threshold acceleration, and then starts logging
  sensor data (acceleration and gyroscope) for a fixed number of samples.

  The logged sensor data is printed over the serial connection in CSV format, with each line representing
  a sample and containing the following values:
    accelerometerX, accelerometerY, accelerometerZ, gyroscopeX, gyroscopeY, gyroscopeZ

  This code assumes the Arduino LSM9DS1 library is installed for interfacing with the LSM9DS1 IMU sensor.

  Library: Arduino_LSM9DS1
  Author: Arduino
  Version: 1.1.1
  More information: https://github.com/arduino-libraries/Arduino_LSM9DS1

  To store the logged sensor data into a .csv file you can use Putty -> Session -> Logging -> Printable output
  More information: https://putty.org.ru/articles/capture-putty-session-log.html

*/

#include <Arduino_LSM9DS1.h>

const float ACCELERATION_THRESHOLD = 2.5;  // Threshold of significance in G's
const int NUM_SAMPLES = 100;               // Number of samples to read

int samplesRead = NUM_SAMPLES;

void setup() {
  Serial.begin(9600);
  while (!Serial);
```

```
  // Initialize the IMU
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  Serial.println("accelerometerX,accelerometerY,accelerometerZ,gyroscopeX,gyroscopeY,gyroscopeZ");
}

void loop() {
  float accelerometerX, accelerometerY, accelerometerZ;
  float gyroscopeX, gyroscopeY, gyroscopeZ;

  // Wait for significant motion
  while (samplesRead == NUM_SAMPLES) {
    if (IMU.accelerationAvailable()) {
      // Read the acceleration data
      IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);

      // Sum up the absolute values
      float accelerationSum = fabs(accelerometerX) + fabs(accelerometerY) + fabs(accelerometerZ);

      // Check if it's above the threshold
      if (accelerationSum >= ACCELERATION_THRESHOLD) {
        // Reset the sample read count
        samplesRead = 0;
        break;
      }
    }
  }
```

```
  // Check if all the required samples have been read since
  // the last time the significant motion was detected
  while (samplesRead < NUM_SAMPLES) {
    // Check if new acceleration and gyroscope data is available
    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
      // Read the acceleration and gyroscope data
      IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);
      IMU.readGyroscope(gyroscopeX, gyroscopeY, gyroscopeZ);

      samplesRead++;

      // Print the sensor data
      Serial.print(accelerometerX, 3);
      Serial.print(',');
      Serial.print(accelerometerY, 3);
      Serial.print(',');
      Serial.print(accelerometerZ, 3);
      Serial.print(',');
      Serial.print(gyroscopeX, 3);
      Serial.print(',');
      Serial.print(gyroscopeY, 3);
      Serial.print(',');
      Serial.print(gyroscopeZ, 3);
      Serial.println();

      if (samplesRead == NUM_SAMPLES) {
        Serial.println();
      }
    }
  }
}
```

Prilog 2: Programski kod za treniranje modela za prepoznavanje gesti

Setup Python Environment

To set up the Python environment and install the necessary dependencies for the notebook, please run the following cell

```
# Remove all files and directories in the current working directory
!rm -r *

# Install the 'xxd' package
!apt-get -qq install xxd

# Install required Python packages for data manipulation, numerical computations, and data visualization
!pip install pandas numpy matplotlib

# Install TensorFlow
!pip install tensorflow==2.12.0
```

Upload Data

1. Click on the "Files" tab located in the left-side menu of Colab.
2. Drag and drop your desired `.csv` files from your computer onto the "Files" tab.

Alternatively, you can click on the "Upload" button within the "Files" tab and browse for the files you want to upload. By following these steps, you will successfully upload your CSV files into Colab for further processing.

Parse and prepare the data

To parse and prepare the data in the next cell, you need to update the `GESTURES` list with the gesture data you've collected in `.csv` format.

1. Replace the values in the `GESTURES` list with the names of the gestures you have collected. For example:

```
GESTURES = [ "gesture_1", "gesture_2", "gesture_3" ]
```

2. Make sure the names in the `GESTURES` list match the actual filenames of your CSV files.

By updating the `GESTURES` list, the code will correctly process and transform the CSV files of the corresponding gestures for training.

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

# Set a random seed for reproducibility
SEED = 1111
np.random.seed(SEED)
tf.random.set_seed(SEED)

# Define the list of gestures
GESTURES = [
    "c_up_push",
    "random_1",
    "random_2",
    "random_3",
    "random_4"
]

# Define the number of samples per gesture
SAMPLES_PER_GESTURE = 100

# Get the number of gestures
NUM_GESTURES = len(GESTURES)

# Create one-hot encoded representations of the gestures
ONE_HOT_ENCODED_GESTURES = np.eye(NUM_GESTURES)

# Initialize lists to store inputs and outputs
inputs = []
outputs = []

# Iterate over each gesture
for gesture_index in range(NUM_GESTURES):
    gesture = GESTURES[gesture_index]
    print(f"Processing index {gesture_index} for gesture '{gesture}'.")

    # Get the one-hot encoded output for the current gesture
    output = ONE_HOT_ENCODED_GESTURES[gesture_index]
```

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

    # Calculate the number of recordings for the current gesture
    num_recordings = int(df.shape[0] / SAMPLES_PER_GESTURE)

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

    # Iterate over each recording
    for i in range(num_recordings):
        tensor = []

        # Iterate over each sample in the recording
        for j in range(SAMPLES_PER_GESTURE):
            index = i * SAMPLES_PER_GESTURE + j

            # Normalize and scale the accelerometer and gyroscope data
            tensor += [
                (df['accelerometerX'][index] + 4) / 8,
                (df['accelerometerY'][index] + 4) / 8,
                (df['accelerometerZ'][index] + 4) / 8,
                (df['gyroscopeX'][index] + 2000) / 4000,
                (df['gyroscopeY'][index] + 2000) / 4000,
                (df['gyroscopeZ'][index] + 2000) / 4000
            ]

        # Append the tensor as input and the output to the respective lists
        inputs.append(tensor)
        outputs.append(output)

# Convert the input and output lists to numpy arrays
inputs = np.array(inputs)
outputs = np.array(outputs)

print("Data set parsing and preparation complete.")
```

Randomize and split the input and output pairs for training

In order to train the model effectively, the input and output pairs need to be randomized and split into different sets for training, validation, and testing. Here's how the data is divided:

- Training Set: This set comprises 60% of the total input and output pairs. It is used to train the model by optimizing its parameters and updating the weights based on the provided input-output pairs.
- Validation Set: This set consists of 20% of the total input and output pairs. It is used to measure the model's performance during training. The validation set helps assess how well the model generalizes to unseen data and allows for monitoring the model's progress.
- Testing Set: This set also represents 20% of the total input and output pairs. It is used to test the model's performance after the training phase is completed. The testing set evaluates the model's ability to make accurate predictions on new, unseen data.

By splitting the data into these sets, you can train, validate, and evaluate the model's performance throughout the training process, ensuring reliable and accurate results.

```
# Get the total number of inputs
num_inputs = len(inputs)

# Create an array of indices from 0 to num_inputs
randomize = np.arange(num_inputs)

# Shuffle the indices randomly
np.random.shuffle(randomize)

# Randomize the order of inputs and outputs using the randomized indices
inputs = inputs[randomize]
outputs = outputs[randomize]

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

# Split the inputs based on the defined splits
inputs_train, inputs_test, inputs_validate = np.split(inputs, [TRAIN_SPLIT, TEST_SPLIT])

# Split the outputs based on the defined splits
outputs_train, outputs_test, outputs_validate = np.split(outputs, [TRAIN_SPLIT, TEST_SPLIT])

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

Build & Train the Model

Build and train a TensorFlow model using the high-level Keras API.

```
# Create a sequential model
model = tf.keras.Sequential()

# Add a dense layer with 50 units and ReLU activation function
model.add(tf.keras.layers.Dense(50, activation='relu'))

# Add a dense layer with 15 units and ReLU activation function
model.add(tf.keras.layers.Dense(15, activation='relu'))

# Add a dense layer with NUM_GESTURES units and softmax activation function
# Softmax is used because we expect only one gesture to occur per input
model.add(tf.keras.layers.Dense(NUM_GESTURES, activation='softmax'))

# Compile the model with optimizer, loss, and metrics
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

# Train the model
# Use inputs_train and outputs_train as training data
# Train for 600 epochs with a batch size of 1
# Use inputs_validate and outputs_validate as validation data
history = model.fit(inputs_train, outputs_train, epochs=600, batch_size=1, validation_data=(inputs_validate, outputs_valida
```

Convert the Trained Model to Tensor Flow Lite

In the next cell, the model is converted to the TensorFlow Lite format, and the size of the model in bytes is printed out.

```
# Convert the model to TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the converted model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

# Get the size of the saved model file
import os
basic_model_size = os.path.getsize("gesture_model.tflite")

# Print the size of the saved model
print("Model is %d bytes" % basic_model_size)
```

Encode the Model in an Arduino Header File

The next cell creates a constant byte array that contains the TensorFlow Lite model. The provided code converts the model to an Arduino header file format, which can be easily included and used in your Arduino sketch.

```
# Create the model.h file and write the initial line
!echo "const unsigned char model[] = {" > /content/model.h

# Convert the content of gesture_model.tflite to hexadecimal representation and append it to model.h
!cat gesture_model.tflite | xxd -i >> /content/model.h

# Write the closing line to model.h
!echo "};" >> /content/model.h

# Get the size of the model.h file
import os
model_h_size = os.path.getsize("model.h")

# Print the size of the model.h file
print(f"Header file, model.h, is {model_h_size:,} bytes.")
```

```
print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")
```

Classifying IMU Data

Now it's time to switch back to `imu_classification` and run our new model on the Arduino Nano 33 BLE Sense to classify the accelerometer and gyroscope data.

Prilog 3: Programski kod za prepoznavanje gesti i otvaranje vrata

Prilog 3.1: Arduino programski kod

```
/*
  Gesture Recognition and Door Control

  This Arduino sketch demonstrates gesture recognition using an IMU (Inertial Measurement Unit) sensor and
  TensorFlow Lite for microcontrollers. It waits for a significant motion event based on a threshold
  acceleration and then collects sensor data (acceleration and gyroscope) for a fixed number of samples.
  The collected data is then processed by a pre-trained machine learning model to recognize gestures.
  If the recognized gesture matches the expected gesture a "true" message is sent via the Bluetooth
  connection, triggering the door opening on the web page.

  The code assumes the following libraries are installed:
  - Arduino_LSM9DS1: For interfacing with the LSM9DS1 IMU sensor
  - ArduinoBLE: For Bluetooth Low Energy (BLE) communication
  - TensorFlowLite Micro: For running the machine learning model

  Library: Arduino_LSM9DS1
  Author: Arduino
  Version: 1.1.1
  More information: https://github.com/arduino-libraries/Arduino_LSM9DS1

  Library: ArduinoBLE
  Author: Arduino
  Version: 1.3.4
  More information: https://github.com/arduino-libraries/ArduinoBLE

  Library: TensorFlowLite Micro
  Author: TensorFlow
  Version: 2.1.0-ALPHA
  More information: https://github.com/tensorflow/tflite-micro

  Model: model.h
  Description: Pre-trained machine learning model for gesture recognition

*/

#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
```

```
#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/version.h>
#include "model.h"

const float ACCELERATION_THRESHOLD = 2.5;  // Threshold of significance in G's
const int NUM_SAMPLES = 100;               // Number of samples to read

int samplesRead = NUM_SAMPLES;

// Global variables used for TensorFlow Lite (Micro)
tflite::MicroErrorReporter tflErrorReporter;
tflite::AllOpsResolver tflOpsResolver;
const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;

// Create a static memory buffer for TFLM
constexpr int TENSOR_ARENA_SIZE = 8 * 1024;
byte tensorArena[TENSOR_ARENA_SIZE] __attribute__((aligned(16)));

// Array to map gesture index to a name
const char* gestureNames[] = {
  "c_up_push",
  "random_1",
  "random_2",
  "random_3",
  "random_4"
};

// Index of the correct gesture
constexpr int CORRECT_GESTURE_INDEX = 0;

#define NUM_GESTURES (sizeof(gestureNames) / sizeof(gestureNames[0]))
```

```
// String to calculate the local and device name
String name;

BLEService* openDoorService = nullptr;
BLEUnsignedCharCharacteristic* openDoorCharacteristic = nullptr;

void setup() {
  Serial.begin(9600);

  // Initialize the IMU
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  if (!BLE.begin()) {
    Serial.println("Failed to initialize Bluetooth!");
    while (1);
  }

  openDoorService = new BLEService("be28eab6-8cc7-4d63-a396-134e1e26fb1d");
  openDoorCharacteristic = new BLEUnsignedCharCharacteristic("e04ff209-6ae4-45e3-8a38-0baefdba9b53", BLERead | BLENotify);

  String address = BLE.address();

  Serial.print("address = ");
  Serial.println(address);

  address.toUpperCase();

  name = "BLESense-";
  name += address[address.length() - 5];
  name += address[address.length() - 4];
  name += address[address.length() - 2];
  name += address[address.length() - 1];
```

```
  Serial.print("name = ");
  Serial.println(name);

  BLE.setLocalName(name.c_str());
  BLE.setAdvertisedService(*openDoorService);
  openDoorService->addCharacteristic(*openDoorCharacteristic);
  BLE.addService(*openDoorService);
  openDoorCharacteristic->writeValue(0);

  BLE.advertise();

  // Get the TFL representation of the model byte array
  tflModel = tflite::GetModel(model);
  if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model schema mismatch!");
    while (1);
  }

  // Create an interpreter to run the model
  tflInterpreter = new tflite::MicroInterpreter(tflModel, tflOpsResolver, tensorArena, TENSOR_ARENA_SIZE, &tflErrorReporter);

  // Allocate memory for the model's input and output tensors
  tflInterpreter->AllocateTensors();

  // Get pointers for the model's input and output tensors
  tflInputTensor = tflInterpreter->input(0);
  tflOutputTensor = tflInterpreter->output(0);
}

void loop() {
  BLEDevice central = BLE.central();
  if (central) {
    Serial.print("Connected to central: ");
    // Print the central's BT address
    Serial.println(central.address());
```

```
while (central.connected()) {
  float accelerometerX, accelerometerY, accelerometerZ;
  float gyroscopeX, gyroscopeY, gyroscopeZ;

  // Wait for significant motion
  while (samplesRead == NUM_SAMPLES) {
    if (IMU.accelerationAvailable()) {
      // Read the acceleration data
      IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);

      // Sum up the absolute values
      float accelerationSum = fabs(accelerometerX) + fabs(accelerometerY) + fabs(accelerometerZ);

      // Check if it's above the threshold
      if (accelerationSum >= ACCELERATION_THRESHOLD) {
        // Reset the sample read count
        samplesRead = 0;
        break;
      } else {
        break;
      }
    }
  }

  // Check if all the required samples have been read since
  // the last time the significant motion was detected
  while (samplesRead < NUM_SAMPLES) {
    // Check if new acceleration and gyroscope data is available
    if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
      // Read the acceleration and gyroscope data
      IMU.readAcceleration(accelerometerX, accelerometerY, accelerometerZ);
      IMU.readGyroscope(gyroscopeX, gyroscopeY, gyroscopeZ);

      // Normalize the IMU data between 0 and 1, and store it in the model's input tensor
      tflInputTensor->data.f[samplesRead * 6 + 0] = (accelerometerX + 4.0) / 8.0;
      tflInputTensor->data.f[samplesRead * 6 + 1] = (accelerometerY + 4.0) / 8.0;
```

```
            tflInputTensor->data.f[samplesRead * 6 + 2] = (accelerometerZ + 4.0) / 8.0;
            tflInputTensor->data.f[samplesRead * 6 + 3] = (gyroscopeX + 2000.0) / 4000.0;
            tflInputTensor->data.f[samplesRead * 6 + 4] = (gyroscopeY + 2000.0) / 4000.0;
            tflInputTensor->data.f[samplesRead * 6 + 5] = (gyroscopeZ + 2000.0) / 4000.0;

            samplesRead++;

            if (samplesRead == NUM_SAMPLES) {
              // Run inference
              TfLiteStatus invokeStatus = tflInterpreter->Invoke();
              if (invokeStatus != kTfLiteOk) {
                Serial.println("Invoke failed!");
                while (1);
                return;
              }

              // Loop through the output tensor values from the model
              for (int i = 0; i < NUM_GESTURES; i++) {
                Serial.print(gestureNames[i]);
                Serial.print(": ");
                Serial.println(tflOutputTensor->data.f[i], 6);
                // Check if the read gesture is the correct one
                if (i == CORRECT_GESTURE_INDEX && tflOutputTensor->data.f[i] > 0.999999) {
                  Serial.println("Correct gesture detected.");
                  openDoorCharacteristic->writeValue(1);
                }
              }
              Serial.println();
            }
          }
        }
      }
    }
  }
}
```

Prilog 3.2: Programski kod web stranice

```
<!DOCTYPE html>
<html>
  <head>
    <title>Door Control</title>
    <style>
      body {
        display: flex;
        align-items: center;
        justify-content: center;
        height: 100vh;
        margin: 0;
      }

      #door-container {
        width: 400px;
        height: 400px;
      }

      #door {
        width: 100%;
        height: 100%;
        background-image: url('images/closed_door.png');
        background-repeat: no-repeat;
        background-size: contain;
      }

      #door-container.disconnected #door {
        opacity: 0.2;
      }

      @keyframes rotate {
        0% {
          transform: rotateY(0deg);
          transform-origin: left bottom;
        }
```

```
      100% {
        transform: rotateY(-90deg);
        transform-origin: left bottom;
      }
    }

    .rotate-animation {
      animation: rotate 0.5s linear;
    }
  </style>
</head>
<body>
  <div id="door-container" class="disconnected">
    <div id="door"></div>
  </div>
  <button id="connect-button">Connect to Arduino</button>
  <script>
    document.addEventListener('DOMContentLoaded', event => {
      const doorContainer = document.getElementById('door-container');
      const door = document.getElementById('door');
      let isOpen = false;
      let isAnimating = false;
      let arduinoDevice;

      function setDoorState(open) {
        if (isOpen !== open && !isAnimating) {
          isOpen = open;
          isAnimating = true;
          if (open) {
            door.classList.add('rotate-animation');
          }
          setTimeout(() => {
            door.classList.remove('rotate-animation');
            isAnimating = false;
            door.style.backgroundImage = `url('images/${open ? 'open' : 'closed'}_door.png')`;
          }, 350);
```

```
  }
}

function receiveDoorStateFromArduino(open) {
  setDoorState(open);
  if (open) {
    setTimeout(() => {
      receiveDoorStateFromArduino(false);
    }, 10000);
  }
}

const connectButton = document.getElementById('connect-button');
let isConnected = false;

async function connectToArduino() {
  try {
    arduinoDevice = await navigator.bluetooth.requestDevice({
      filters: [
        { services: ['be28eab6-8cc7-4d63-a396-134e1e26fb1d'] } // SERVICE_UUID
      ]
    });
    const server = await arduinoDevice.gatt.connect();
    console.log('arduinoDevice:', arduinoDevice);
    console.log('Connected to Arduino:', arduinoDevice.name);
    console.log('server:', server);
    // Subscribe to the door state characteristic for receiving updates from Arduino
    const service = await server.getPrimaryService('be28eab6-8cc7-4d63-a396-134e1e26fb1d');
    console.log('service:', service);
    const doorStateCharacteristic = await service.getCharacteristic('e04ff209-6ae4-45e3-8a38-0baefdba9b53');
    console.log('doorStateCharacteristic:', doorStateCharacteristic);
    doorStateCharacteristic.addEventListener('characteristicvaluechanged', event => {
      const value = event.target.value;
      const isOpen = value.getUint8(0) === 1;
      console.log('Received door state from Arduino:', isOpen);
      receiveDoorStateFromArduino(isOpen);
```

```
            });
            await doorStateCharacteristic.startNotifications();
            // Remove the "disconnected" class from the door container
            doorContainer.classList.remove('disconnected');
            isConnected = true;
            connectButton.textContent = 'Disconnect from Arduino';
          } catch (error) {
            console.error('Failed to connect to Arduino:', error);
          }
        }

        async function disconnectFromArduino() {
          try {
            if (arduinoDevice && arduinoDevice.gatt.connected) {
              await arduinoDevice.gatt.disconnect();
              console.log('Disconnected from Arduino:', arduinoDevice.name);
              doorContainer.classList.add('disconnected');
              isConnected = false;
              connectButton.textContent = 'Connect to Arduino';
            }
          } catch (error) {
            console.error('Failed to disconnect from Arduino:', error);
          }
        }

        connectButton.addEventListener('click', async () => {
          if (isConnected) {
            await disconnectFromArduino();
          } else {
            await connectToArduino();
          }
        });
      });
    </script>
  </body>
</html>
```