SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5372

SUSTAV ZA PREPOZNAVANJE GESTI RUKU TEMELJEN NA PODACIMA S MIKROKONTROLERSKE PLOČICE I NEURONSKOJ MREŽI

Eugen Vušak

Zagreb, Lipanj 2018.

Sadržaj

1. Uvod

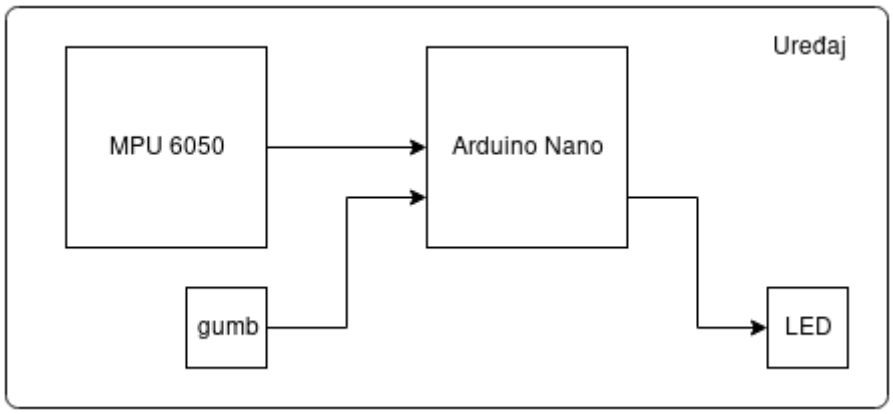
Računala postaju sve veći dio naših života, no ne samo osobna računala, već i mikroračunala koja se nalaze posvuda od kućanskih aparata do sigurnosnih sustava. Jedna procjena kaže da prosječno američko kućanstvo ima oko 60 ugrađenih mikrokontrolera [1]. Kako bismo iskoristili taj potencijal, potrebno je razvijati načine komunikacije između računala i čovjeka. Najpoznatiji takvi uređaji zasigurno su tipkovnica i miš, ali iako dobro utemeljeni, oni nisu dobri za sve tipove komunikacije, a također njihov dizajn minimalno se mijenjao tijekom vremena. Prve tipkovnice potječu još iz doba teleprintera, a nama poznata QWERTY tipkovnica pojavila se davne 1872. godine za tadašnje pisaće strojeve. Prvi miš napravljen je 1964. godine, no ostao je dosta nepopularan sve do pojave računala Macintosh 128K 1984. godine, koji je uključivao verziju miša imenom „Lisa“.

Ideja ovog završnog rada upravo je izazvati tradicionalne metode komunikacije između računala i čovjeka s ciljem kontrole pametne kuće. Također se želi pokazati, iako je ovaj problem rješiv na više načina, da jako popularan programski sustav u strojnom učenju, neuronske mreže, mogu biti korištene i za ovu domenu problema. U tu svrhu potrebno je izraditi uređaj koji će se sastojati od senzora pomaka i mikrokontrolera. Nakon toga, potrebno je prikupiti skup podataka koji će se koristiti za učenje neuronske mreže. Ti podaci zatim se koriste kako bi se dobio model koji će predviđati geste. Posljednji korak je izraditi aplikaciju koja u stvarnom vremenu čita podatke sa spomenutog uređaja, koristi model te predviđa trenutnu gestu.

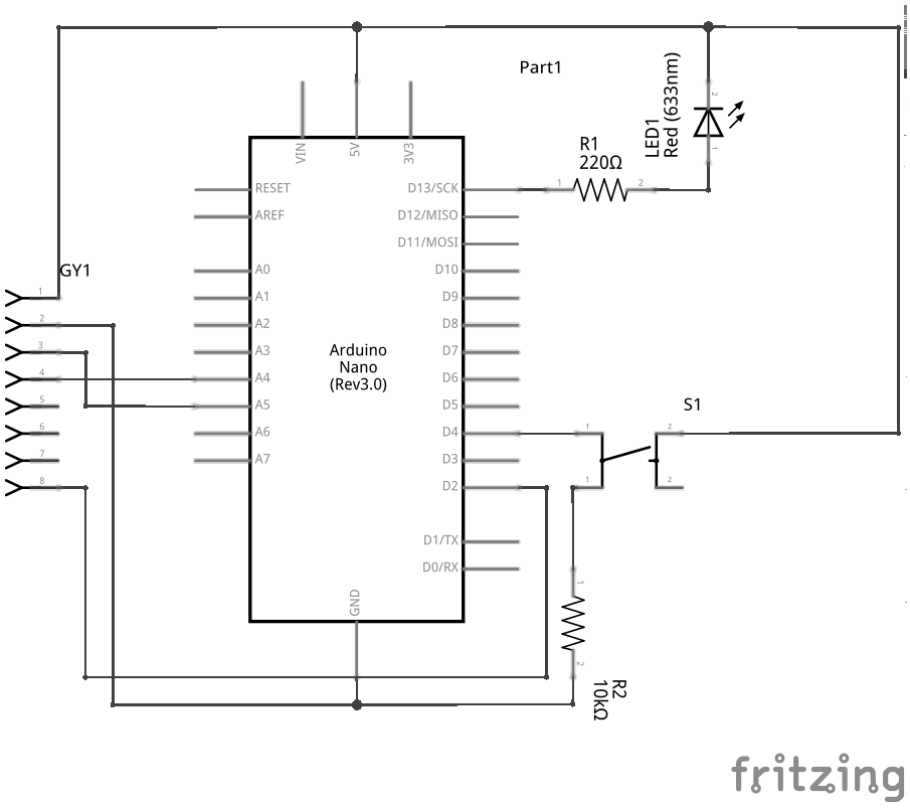
Sličan rad objavili su Blanca MiriamLee-Cosio, Carlos Delgado-Mata i JesusIbanez pod nazivom „ANN for Gesture Recognition using Accelerometer Data“ [2]. U tom radu korišten je daljinski uređaj konzole Wii, a podaci su filtrirani i normalizirani korištenjem brze Fourierove transformacije. Za prepoznavanje gesti u euklidskom prostoru također ja korištena umjetna neuronska mreža.

2. Uređaj

Uređaj se sastoji od nekoliko komponenata. To su u njegovom centru mikrokontrolerska pločica Arduino Nano koji se koristi kao središnja jedinica za obradu podataka, te senzor pokreta MPU 6050 koji je zaslužan za dobivanje podataka o akceleraciji i kutnoj akceleraciji iz stvarnog svijeta. Uređaj također ima indikatorsku LED diodu te gumb za kontrolu.



SLIKA 1: IDEJNA BLOK SHEMA UREĐAJA



SLIKA 2: ELEKTRIČNA SHEMA UREĐAJA

2.1. Komponente

2.1.1. Arduino Nano

U ovom radu, za centralnu komponenta uređaja odabrao sam mikrokontrolersku pločicu Arduino Nano, upravo zbog malih dimenzija. Veličina Arduino Nanoa je 18 x 45 mm. Pločica se napaja korištenjem konektora Mini B USB, a također se može napajati i pomoću nereguliranog napona između 6 i 20 V na pinu 30 ili reguliranog 5 voltnog napona na pinu 27. Na pločici se nalazi ATmega328P mikrokontroler, koji se također nalazi na Arduinovoj popularnijoj ali fizički većoj verziji Uno. Njegova radna frekvencija je 16 MHz što je i više nego dovoljno za potrebe ovog rada. ATmega328P ima 32 KB Flash memorije na koju se spremaju programi (no 2 KB su iskorištena za punitelja (engl. Bootloader)). Uz to također ima 2 KB statičke radne memorije (skrač. SRAM) koja je dostupna programu u izvršavanju, te 1KB EEPROM memorije čiji sadržaj se ne gubi nakon nestanka napajanja. Nano ima 14 digitalnih ulazno izlaznih pinova, 6 od kojih se mogu koristiti za pulsno širinsku modulaciju (skrač. PWM), tj. mogu na izlazu dati signal koji koristi pluseve određene širine i tako dobiti efikasnu vrijednost napona od 0 – 5 V. Pinovi D0 (RX) i D1 (TX) koriste se za serijsku komunikaciju. Uz digitalne pinove Nano ima i 8 analognih ulaznih pinova 10 bitne rezolucije. Za ovaj rad bitni su samo pinovi A4 (SDA) i A5 (SDL) koji podržavaju I^2C komunikaciju. Za nju postoji standardna Arduino knjižnica funkcija Wire.h. [3]

Programiranje Arduino Nano-a je vrlo jednostavno, te sve što je potrebno je jedan Mini B USB na USB A kabel, računalo i Arduino IDE dostupno za preuzimanje na adresi https://www.arduino.cc/en/Main/Software, također Arduino nudi *Web Editor* s kojim je moguće programirati koristeći pretraživač weba. Jezik korišten za programiranje je malo prilagođeni C/C++. Prije prevođenja koda, Arduino IDE napravi male pred-procesorske promjene kako bi se kod pretvorio u C++ kod, a onda se šalje na avr-gcc prevodioca. Taj kod se zatim povezuje sa standardnim Arduino knjižnicama, što sveukupno rezultira jednim intel hex dokumentom koji se zatim zapisuje na pločicu. [4]

Svaki program sastoji se od minimalno dvije funkcije, a to su *loop*() i *setup*(). Funkcija *setup*() izvodi se samo jednom, prilikom svakog pokretanja programa (pritiskom tipke *reset*, pojavom napajanja ili programerskim ponovnim pokretanjem). Nakon završetka funkcije *setup*(), sve do kraja programa izvodi se petlja *loop*(), tako da svaki put kada ona završi pozove se ponovno.

```
1. void setup() {
2.   // put your setup code here, to run once:
3. }
4. void loop() {
5.   // put your main code here, to run repeatedly:
6. }
```

KOD 1: MINIMALNI ARDUINO PROGRAM

2.1.2. MPU-6050

MPU-6050 druga je bitna komponenta uređaja. To je senzor firme InvenSense, a dizajniran je da bude male potrošnje, male cijene ali visoki performansi koje zahtijevaju pametni telefoni, tableti i nosivi uređaji. U sebi ima ugrađen troosni žiroskop i troosni akcelerometar na istoj silicijskoj boji, što znači da ima 6 osi slobode. Omogućava I^2C i SPI komunikaciju, a veličina samog senzora je 4 x 4 x 0.9 mm. Također, moguće je pratiti brze i spore kretnje tako što je omogućeno programiranje opsega mjerenja i za žiroskop i za akcelerometar. Za žiroskop, opsezi su ±250, ±500, ±1000, i ±2000 °/s, a za akcelerometar oni iznose ±2g, ±4g, ±8g, i ±16g. Dodatno ,MPU-6050 ima ugrađen senzor temperature i unutarnji oscilator [5]. U ovom radu koristi se pločica GY-521 koja na sebi ima MPU-6050, ali i neke dodatne komponente poput regulatora napona.

Kako bismo olakšali programiranje, potrebno je izraditi vlastitu knjižnicu za rad sa senzorom. U sklopu toga napravljena su dva razreda. Prvi od tih razreda je razred „Data“ koji će biti korišten za reprezentiranje podataka koje u jednom mjerenju izmjeri senzor.

TABLICA 1: OPISI BITNIJIH FUNKCIJA RAZREDA DATA

Data()	Stvara novu instancu s vrijednostima 0, 0, 0, 0, 0, 0
Data (int16_t ax, int16_t ay, int16_t az, int16_t gx, int16_t gy, int16_t gz);	Stvara novu instancu s vrijednostima ax, ay, az, gx, gy, gz
toString()	Pretvara objekt u String objekt u formatu „ax,ay,az,gx,gy,gz“
add(Data other);	Zbraja dvije instance razreda Data
sub(Data other);	Oduzima dvije instance razreda Data
div(int16_t c);	Dijeli instancu razreda Data konstantom c

U detalje ovih funkcija neće se ulaziti, no sav kod dostupan je na https://github.com/eugen-vusak/zavrsni-rad/tree/master/Device

Drugi razred je sam MPU_6050 razred koji omogućuje osnovne funkcionalnosti potrebne za ovaj rad.

TABLICA 2: OPISI SVIH FUNKCIJA RAZREDA MPU_6050

MPU_6050(int MPU_adr);	Stvara novu instancu razreda MPU_6050 te postavlja adresu I^2C adresu senzora
wakeUp();	„Budi“ senzor iz stanja niske potrošnje
setAccelRange(int16_t range);	Postavlja mjerni opseg akcelerometra ±2g, ±4g, ±8g, ili ±16g
setGyroRange(int16_t range);	Postavlja mjerni opseg žiroskopa ±250, ±500, ±1000, i ±2000 °/s
measure();	Zahtjeva trenutne podatke od senzora te ih vraća ih kao instancu razreda Data

Sve ove funkcije osim konstruktora koriste knjižnicu Wire.h za I^2C komunikaciju i zapisuju odgovarajuće bitove u odgovarajuće registre.

TABLICA 3: REGISTRI KORIŠTENI U RADU

Ime registra	Adresa registra	Namjena registra
GYRO_CONFIG	0x1B	Konfiguracija žiroskopa
ACCEL_CONFIG	0x1C	Konfiguracija akcelerometra
ACCEL_XOUT_H	0x3B	Viših 8 bitova akceleracije u smjeru osi x
ACCEL_XOUT_L	0x3C	Nižih 8 bitova akceleracije u smjeru osi x
ACCEL_YOUT_H	0x3D	Viših 8 bitova akceleracije u smjeru osi y
ACCEL_YOUT_L	0x3E	Nižih 8 bitova akceleracije u smjeru osi y
ACCEL_ZOUT_H	0x3F	Viših 8 bitova akceleracije u smjeru osi z
ACCEL_ZOUT_L	0x40	Nižih 8 bitova akceleracije u smjeru osi z
TEMP_OUT_H	0x41	Viših 8 bitova temperature
TEMP_OUT_L	0x42	Nižih 8 bitova temperature
GYRO_XOUT_H	0x43	Viših 8 bitova kutne akceleracije u smjeru osi x
GYRO_XOUT_L	0x44	Nižih 8 bitova kutne akceleracije u smjeru osi x
GYRO_YOUT_H	0x45	Viših 8 bitova kutne akceleracije u smjeru osi y
GYRO_YOUT_L	0x46	Nižih 8 bitova kutne akceleracije u smjeru osi y
GYRO_ZOUT_H	0x47	Viših 8 bitova kutne akceleracije u smjeru osi z
GYRO_ZOUT_L	0x48	Nižih 8 bitova kutne akceleracije u smjeru osi z
PWR_MGMT_1	0x6B	Kontrola rada uređaja i izvora oscilatora

[6]

wakeUp() – u registar PWR_MGMT_1 (0x6B) zapisuje 0 kako bi probudila uređaj

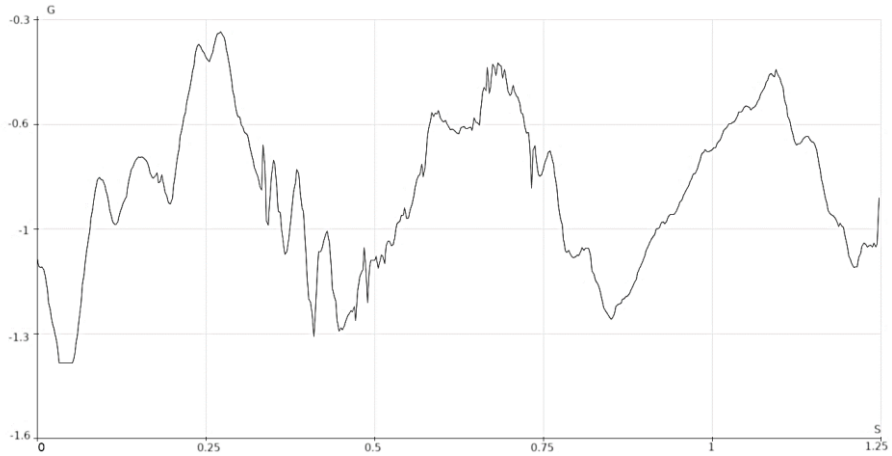
setAccelRange() – u registar ACCEL_CONFIG (0x1C) zapisuje ovisno o željenom opsegu

setGyroRange() – u registar GYRO_CONFIG (0x1B) zapisuje ovisno o željenom opsegu

measure() – traži sveukupno 14 registara, počevši od registra ACCEL_XOUT_H (0x3B). Budući da su podaci 16 bitni a registri 8bitni, svaki podatak nalazi se u dva registra, prvi se pomiče 8 bitova u desno te se radi logička operacija ILI s drugim.

2.2. Filtriranje šuma

MPU-6050 je vrlo osjetljiv senzor te će registrirati puno šuma. Taj šum nije poželjan jer će otežati generalizaciju neuronskoj mreži, tj. taj šum potrebno je filtrirati.

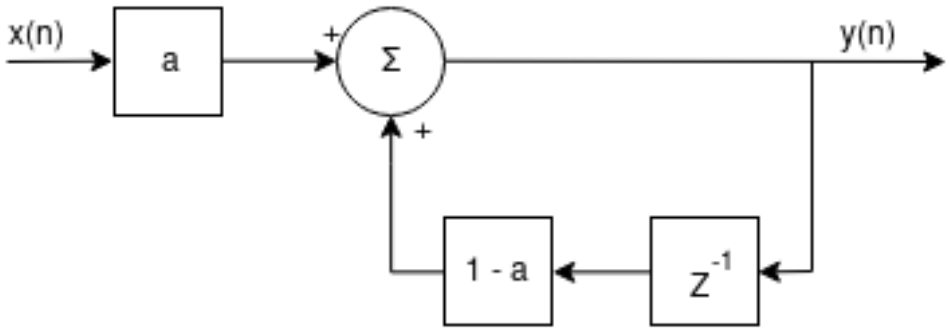


SLIKA 3: PODACI SA SENZORA BEZ FILTRIRANJA

Na Slici 3 moguće je vidjeti kako podaci imaju puno velikih skokova i visokih frekvencija. Ono što se želi ostvariti je nisko propusni digitalni filter.

Filter koji je korišten u ovom radu je *Filter s beskonačnim impulsnim odzivom* (skrač. IIR filter) prvog reda. Da je filter prvog reda znači da uzima podatke jednu vremensku konstantu u prošlost. Filter je dizajniran po generalnom modelu IIR filtera prvog reda s predavanja na Fakultetu Elektrotehnike i Računarske Znanosti, Sveučilišta u Michiganu [7]. Takav filter navodi dvije konstante: a_0 i b_0. No kada želimo da niska frekvencija prigušenja bude 0 dB, tada zbroj konstanti a_0 i b_0 treba biti točno jedan [8], zato je uzeto $a_0 = a$ i $b_0 = 1 - a$. Diferencijalna jednadžba koja opisuje ovaj filter je

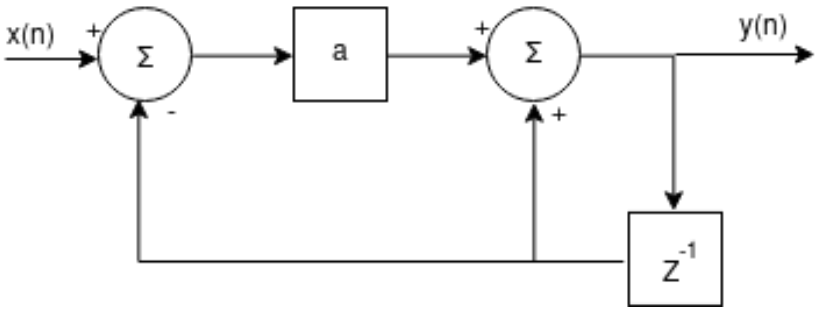
$$y(n) = a \times x(n) + (1 - a) \times y(n - 1).$$



SLIKA 4: BLOK SHEMA IIR FILTERA PRVOG REDA

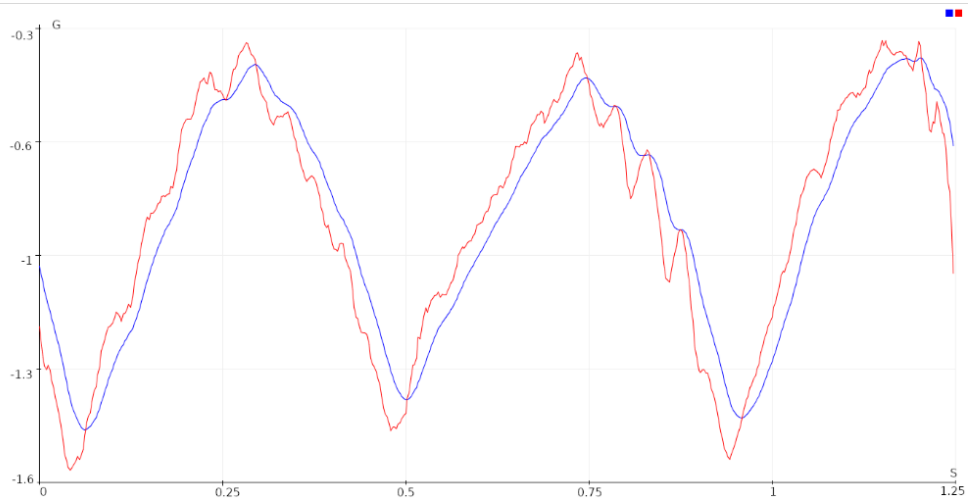
Primjećujemo da postoje dva mjesta množenja, prvo množenje s a, a zatim množenje s 1-a, kako bismo optimirali filter jednostavnom algebrom možemo svesti dva množenja na jedno dijeljenje čime dobivamo našu konačnu diferencijalnu jednadžbu

$$y(n) = y(n-1) + \left[\frac{(x(n) - y(n))}{a}\right].$$



SLIKA 5: OPTIMIZIRANA BLOK SHEMA IIR FILTERA PRVOG REDA

Sada kada na naše podatke rekurzivno primjenjujemo ovaj filter možemo vidjeti očitu razliku.

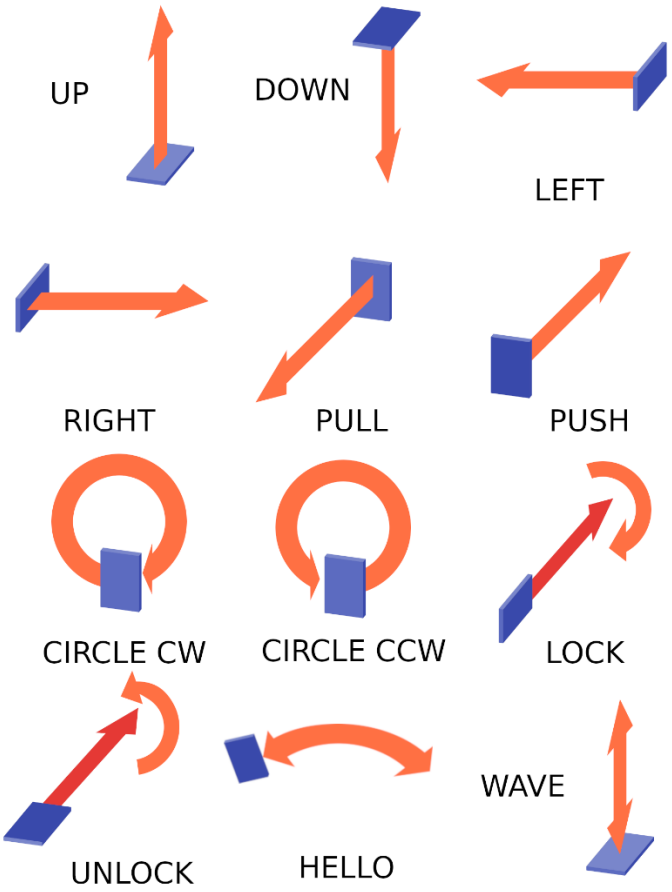


SLIKA 6: PODACI PRIJE (CRVENO) I POSLIJE (PLAVO) FILTRIRANJA

Primjećujemo da su podaci puno glađi bez naglih skokova koje smo primjećivali prije. No jedna od posljedica je da sada naš signal ima kašnjenje od razmaka između uzorkovanja podataka pomnoženog s redom filtera. Kako je naš filter prvog reda, a frekvencija uzorkovanja 50ms, naše kašnjenje će biti 20ms, što je prihvatljivo u sklopu ovog rada.

3. Geste

U radu je programirano 12 gesti, to su UP, DOWN, LEFT, RIGHT, PULL, PUSH, CIRCLE CW, CIRCLE CCW, LOCK, UNLOCK, HELLO i WAVE. Sve geste opisane su za lijevu ruku. Trajanje svake geste je 1,2 sekunde, a uzorkovanje je izvedeno frekvencijom 50Hz. Razmak između svakog vremenskog trenutka je 20ms.



SLIKA 7: VIZUALIZACIJA GESTI

Gesta UP opisuje pomak ruke prema gore, držeći dlan okrenut prema dolje.

Gesta DOWN opisuje pomak ruke prema dolje, držeći dlan okrenut također prema dolje.

Gesta LEFT opisuje pomak ruke s lijeva na desno, držeći dlan okrenut prema desno.

Gesta RIGHT opisuje pomak ruke s desna na lijevo, držeći dlan okrenut prema desno.

Gesta PULL opisuje pomak ruke od naprijed prema nazad, držeći dlan okrenut prema naprijed.

Gesta PUSH opisuje pomak ruke od nazad prema naprijed, držeći dlan okrenut prema naprijed.

Gesta CIRCLE CW opisuje kretanje ruke po kružnici radijusa približno 15cm u smjeru kazaljke na satu, držeći dlan okrenut prema naprijed.

Gesta CIRCLE CCW opisuje kretanje ruke po kružnici radijusa približno 15cm u smjeru suprotnom od kazaljke na satu, držeći dlan okrenut prema naprijed.

Gesta LOCK opisuje pomak ruke od nazad prema naprijed, držeći dlan okrenut prema dolje, a zatim rotaciju ruke oko osi x u smjeru suprotnom od kazaljke na satu.

Gesta LOCK opisuje pomak ruke od nazad prema naprijed, držeći dlan okrenut prema desno, a zatim rotaciju ruke oko osi x u smjeru kazaljke na satu.

Gesta HELLO opisuje naizmjenično kretanje lijevo-desno, držeći dlan ruke okrenut prema naprijed.

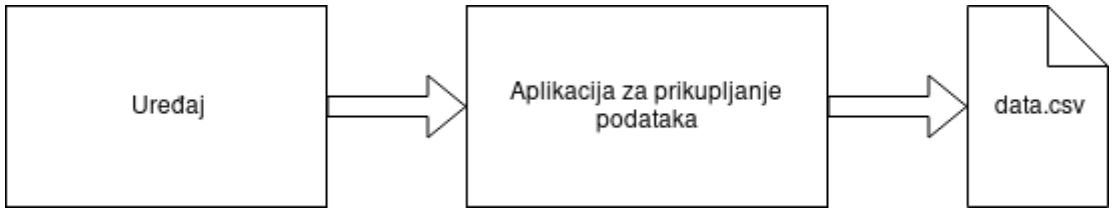
Gesta WAVE opisuje naizmjenično kretanje gore-dolje, držeći dlan ruke okrenut prema dolje.

4. Aplikacija za prikupljanje podataka

Kako bi uspješno naučili neuronsku mrežu potreban nam je set podataka. Za većinu primjena moguće je pronaći dostupan set podataka na stranicama poput www.kaggle.com ili https://datasource.kapsarc.org/pages/home/. Za primjenu prikazanu u ovom radu takav set ne postoji te ga je bilo potrebno stvoriti. Prikupljanje podataka je spor i zahtjevan proces, te nema unaprijed definiranog načina kako mu pristupiti. Potrebno je ponavljati pokuse onoliko puta koliko smatramo da nam je podataka dovoljno. Kako bi se olakšalo prikupljanje podataka, izrađena je aplikacija koja to omogućuje.

Aplikacija je pisana u programskom jeziku *Processing* [9] radi jednostavnosti povezivanja s Arduinom. Processing se nadograđuje na programski jezik Java, no koristi jednostavniju sintaksu i grafičko sučelje. Funkcionira slično kao i Arduino, tj. ima dvije bitne funkcije. Prva je *setup*(), koja kao i kod Arduina se odvija jednom pri pozivu programa. Druga je *draw*(), koja je ekvivalent Arduinove *loop*() funkcije te se vrti u petlji od kraja izvođenja *setup*() funkcije do kraja izvođenja programa. Processing također omogućuje rukovanje događajima u kodu poput pritiska tipke na tipkovnici. Nama najvažnija funkcija je *serialEvent*(*Serial port*) koja se izvodi svaki put kada se na nekom serijskom portu dovedu podaci. Pošto naš uređaj koristi serijsku komunikaciju za slanje podataka u ovoj funkciji odvija se obrada podataka. Kôd cijele aplikacije dostupan je na Githubu na sljedećoj adresi: https://github.com/eugen-vusak/zavrsni-rad/blob/master/DataCollector/DataCollector.pde

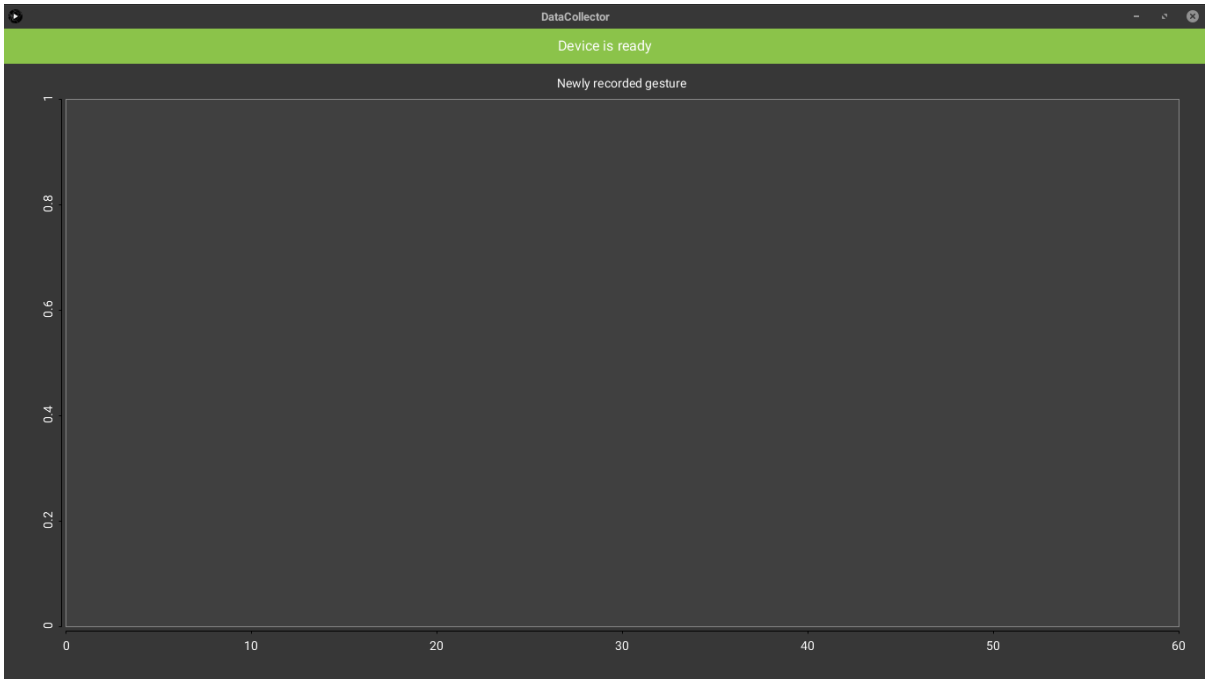
Aplikacija ima jednostavan zadatak. Od uređaja zatražiti podatke, ovisno o pritisku tipke dodijeliti tim podacima oznaku pripadajuće geste, te tako označene podatke nadodati u .csv datoteku.



SLIKA 8: BLOK SHEMA PROLAZA PODATAKA PRILIKOM PRIKUPLJANJA

4.1. Korištenje aplikacije

Prilikom pokretanja, aplikacija otvara odgovarajući serijski port te, ako je to izvedeno uspješno, dojavljuje da je povezivanje s uređajem bilo uspješno, nakon čega čeka da uređaj dojavi da je spreman za korištenje.



SLIKA 9: APLIKACIJA NAKON ŠTO JE UREĐAJ DOJAVIO SPREMNOST

Sada je moguće odašiljati zahtjeve za dobivanjem podataka. Svaka gesta odgovara jednoj tipki na tipkovnici. Ovdje je to izvedeno :

1 → UP

2 → DOWN

…

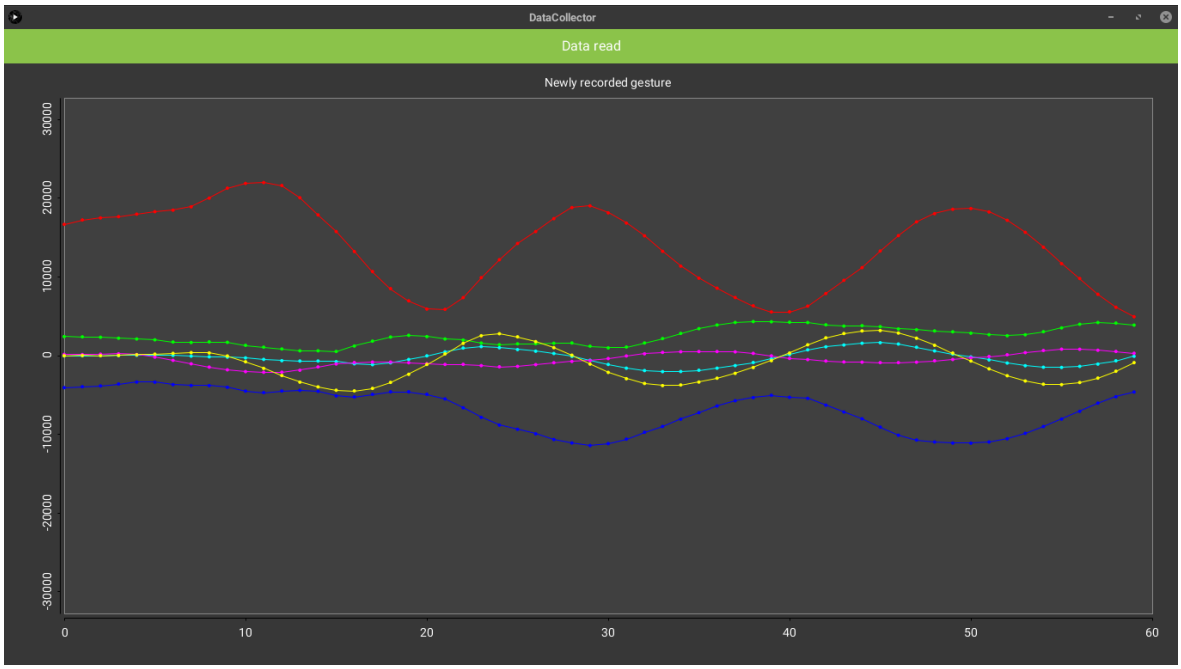
0 → UNLOCK

' → WAVE

+ → HELLO

Preslikavanje tipke u odgovarajuću gestu moguće je mijenjati promjenom mape *keyToGestureName*.

Nakon što je odgovarajuća tipka pritisnuta, započinje očitavanje geste.



SLIKA 10: IZGLED APLIKACIJE NAKON ŠTO SU PROČITANI PODACI

Kada su podaci pročitani moguće ih je odbaciti pritiskom BACKSPACE ili ih nadodati u .csv datoteku pritiskom tipke ENTER, što je također popraćeno odgovarajućom porukom.

5. Podaci

Kao što je već navedeno, podaci su spremljeni u datoteku formata .csv, gdje je svaki stupac odvojen zarezom, a stupci su:

- Oznaka geste
- Akceleracija u osi x u trenutku T1
- Akceleracija u osi y u trenutku T1
- Akceleracija u osi z u trenutku T1
- Kutna akceleracija u osi x u trenutku T1
- Kutna akceleracija u osi y u trenutku T1
- Kutna akceleracija u osi z u trenutku T1
- Akceleracija u osi x u trenutku T2
- Akceleracija u osi y u trenutku T2
- …

Za svaku gestu prikupljeno je približno 150 različitih uzoraka.

5.1. Vizualizacija

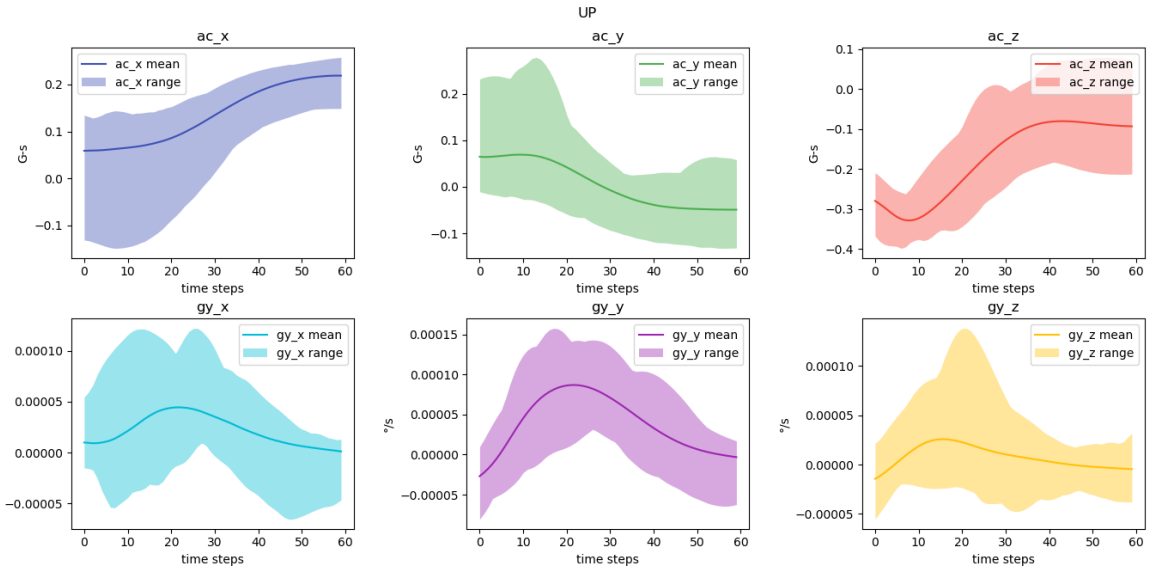
Kada bismo pogledali samo jedan red od svih podataka on bi izgledao ovako:

UP,-4358,6966,-24103,610,-2003,-752,-4876,6748,-24940,961,-1732,-513,-5234,6180,-25434,1162,-1410,-312,-5523,5670,-25666,1211,-1055,-150,-5725,5285,-25731,1183,-665,-1,-5779,5066,-25602,1114,-264,139,-5756,5040,-25353,…

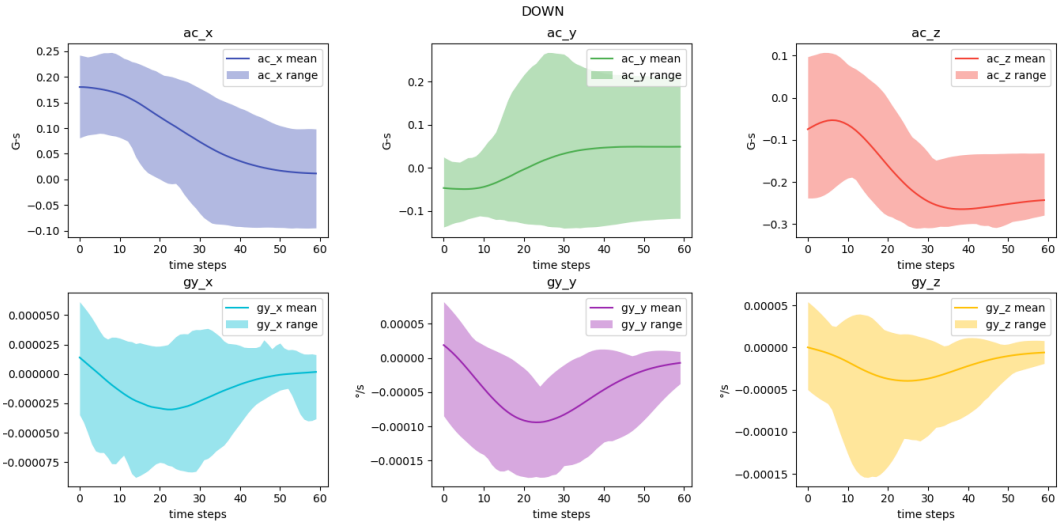
Takvi podaci namijenjeni su računalu, dok čovjeku oni vrlo malo znače. Kako bismo lakše razumjeli podatke, potrebno ih je vizualizirati na neki način. U ovom radu korištena su dva tipa vizualizacije. Prvi je sam opseg vrijednosti za pojedinu gestu, a drugo je pomoću algoritma za redukciju dimenzionalnosti t-SNE.

5.1.1. Opseg podataka

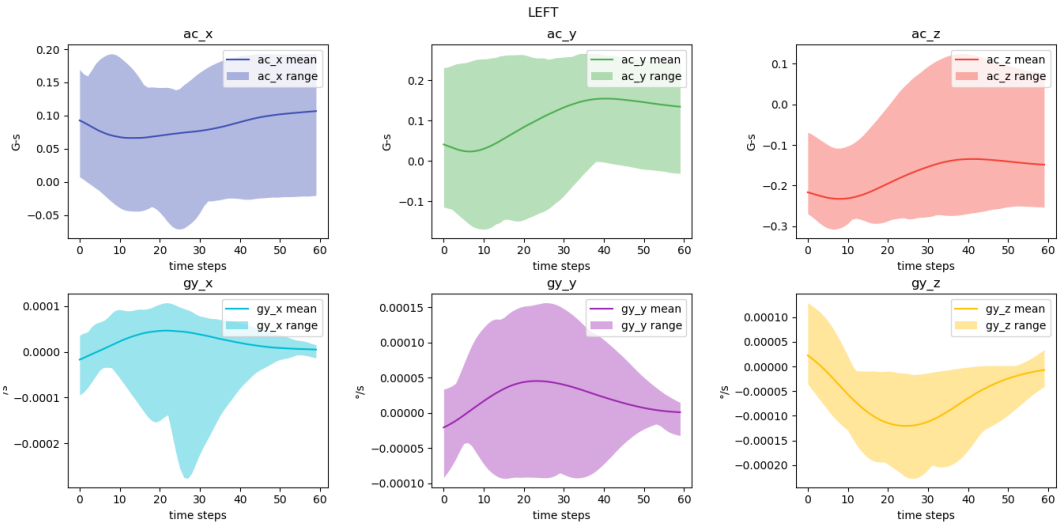
Prva vizualizacija prikazuje koliko svaki od 6 stupnjeva slobode variraju za svaku gestu i srednju vrijednost za nju, točnije koje su maksimalne i minimalne granice geste. Na podacima prikazani su svi podaci, koji će kasnije biti razdijeljeni na dva skupa, jedan za učenje i drugi za treniranje. Program za stvaranje ove vizualizacije dostupan je na: https://github.com/eugen-vusak/zavrsni-rad/blob/master/RecognitionApp/Data/Visualization/Programs/min-max.py



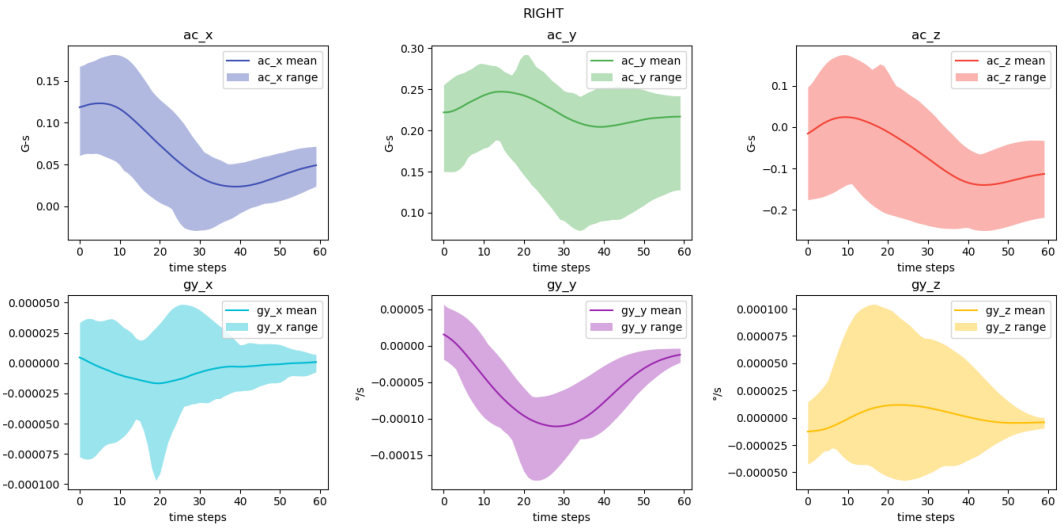
SLIKA 11: OPSEG PODATAKA ZA GESTU UP



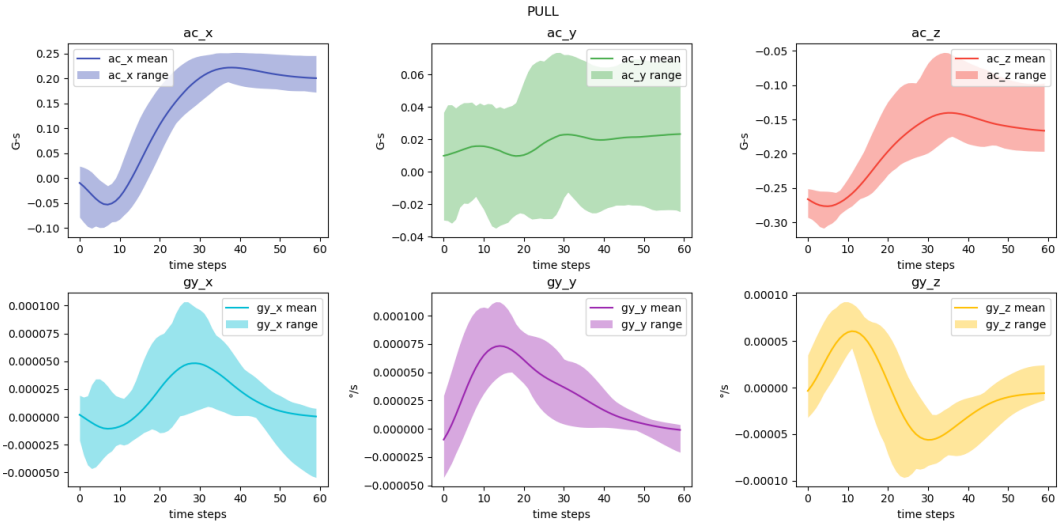
SLIKA 12: OPSEG PODATAKA ZA GESTU DOWN



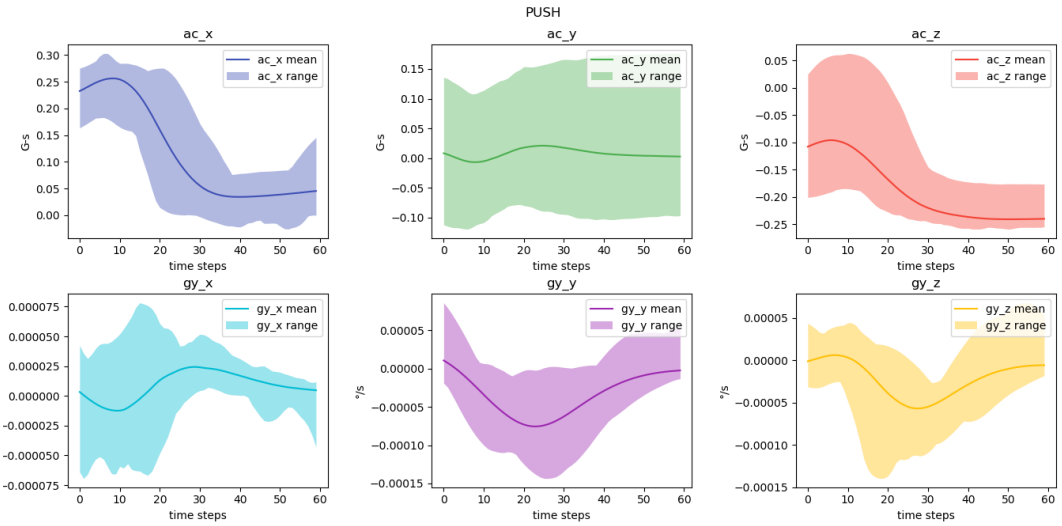
SLIKA 13: OPSEG PODATAKA ZA GESTU LEFT



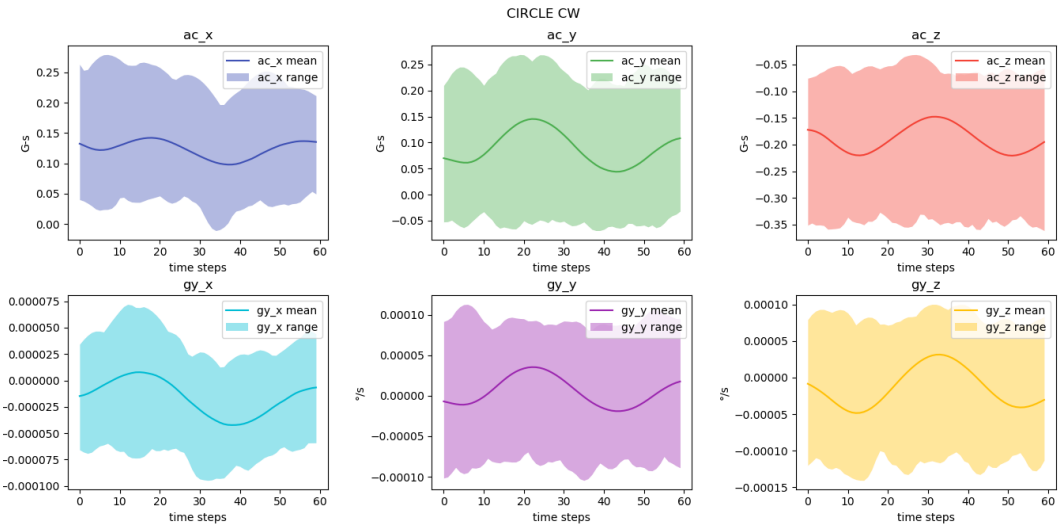
SLIKA 14: OPSEG PODATAKA ZA GESTU RIGHT



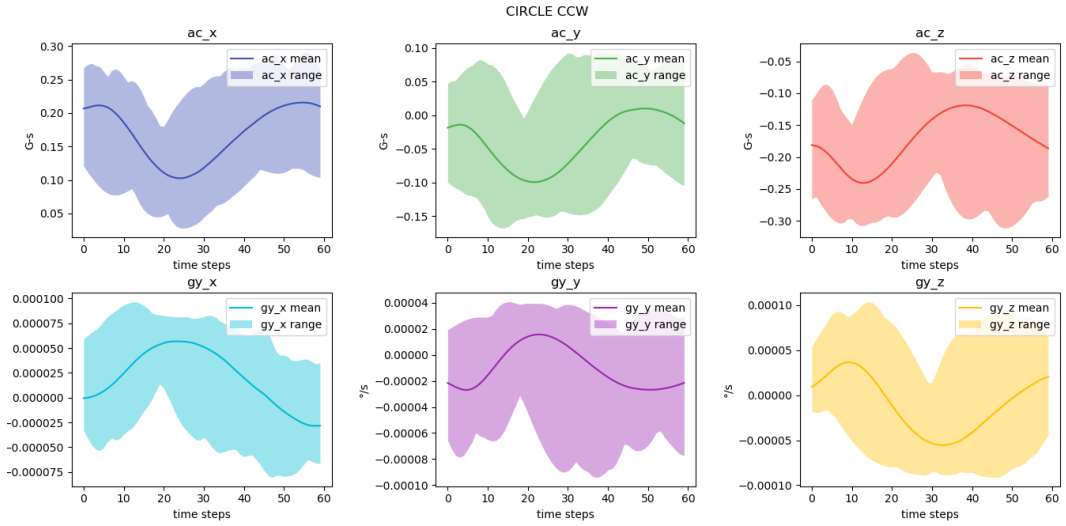
SLIKA 15: OPSEG PODATAKA ZA GESTU PULL



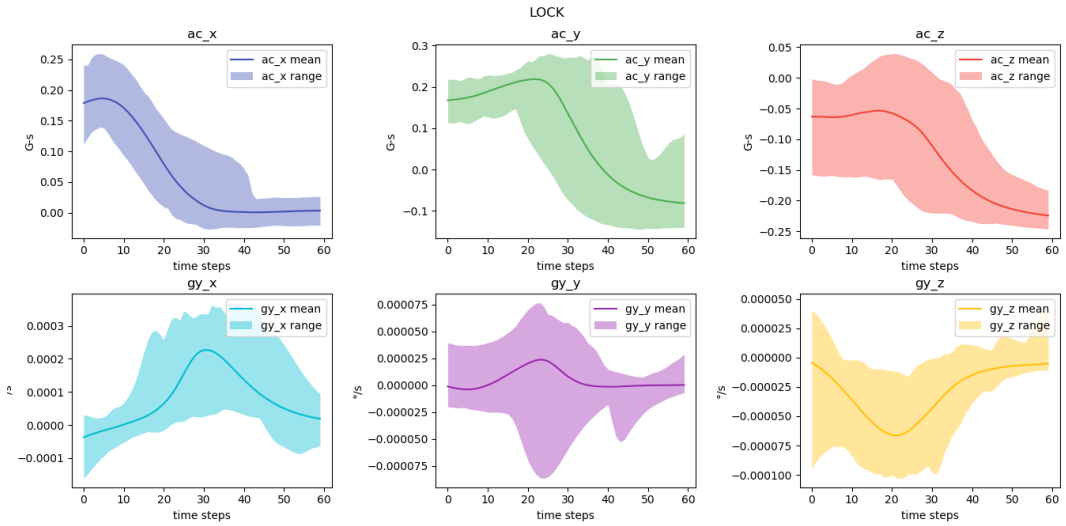
SLIKA 16: OPSEG PODATAKA ZA GESTU PUSH



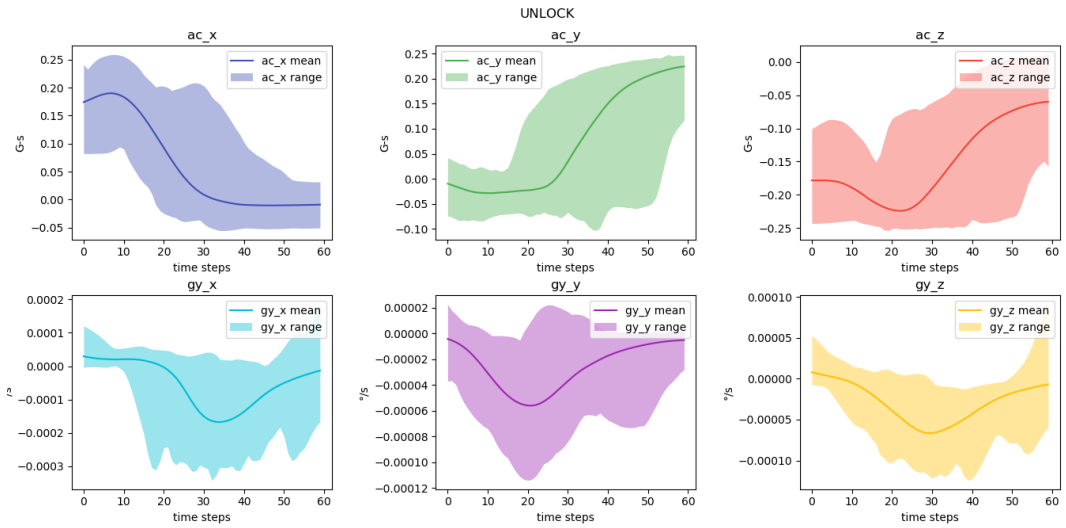
SLIKA 17: OPSEG PODATAKA ZA GESTU CIRCLE CW



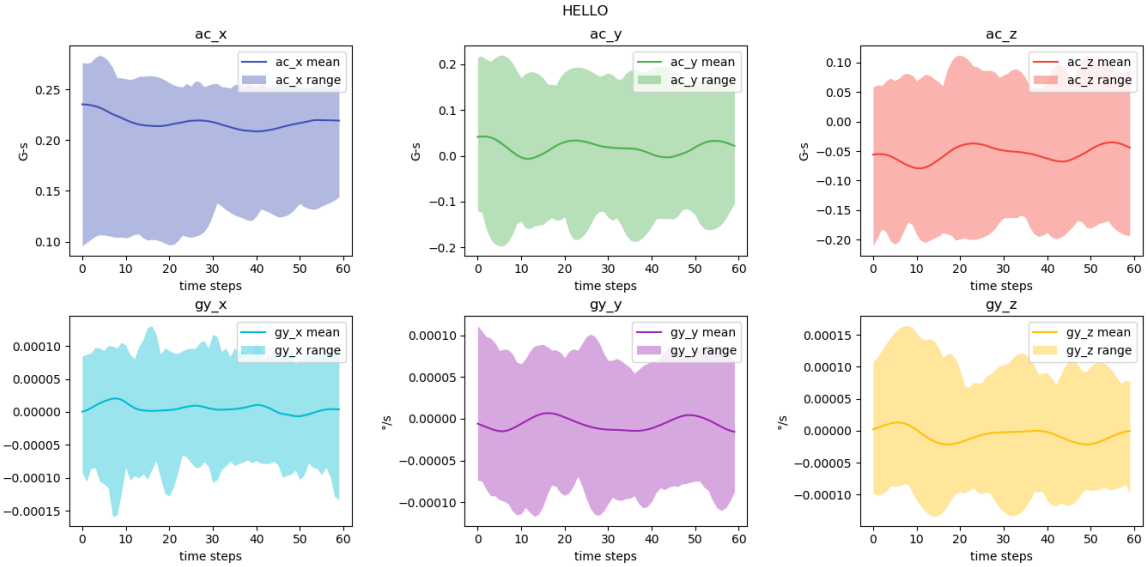
SLIKA 18: OPSEG PODATAKA ZA GESTU CIRCLE CCW



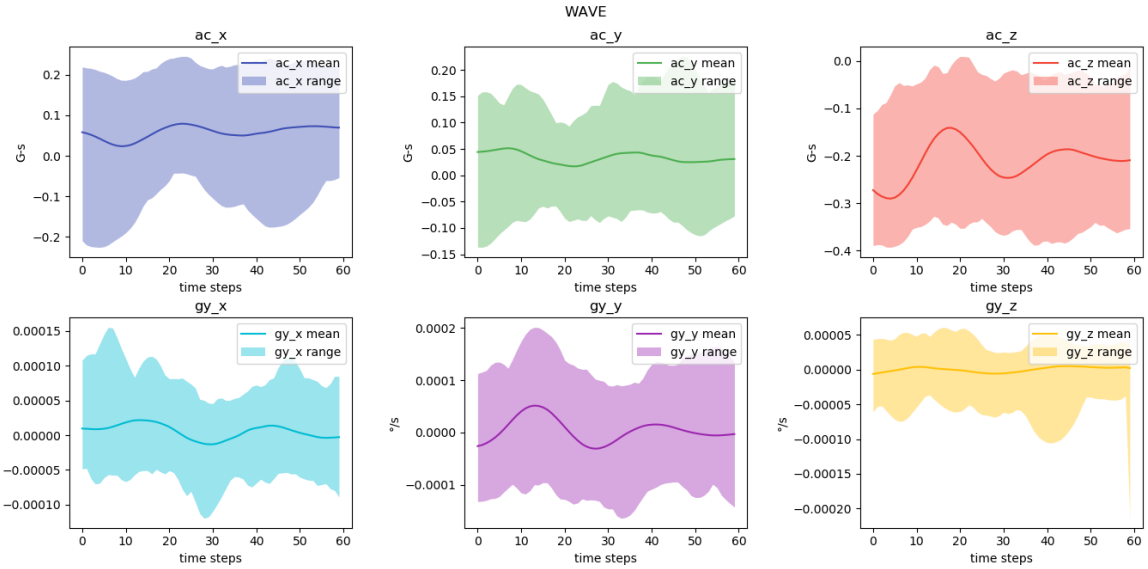
SLIKA 19: OPSEG PODATAKA ZA GESTU LOCK



SLIKA 20: OPSEG PODATAKA ZA GESTU UNLOCK



SLIKA 21: OPSEG PODATAKA ZA GESTU HELLO



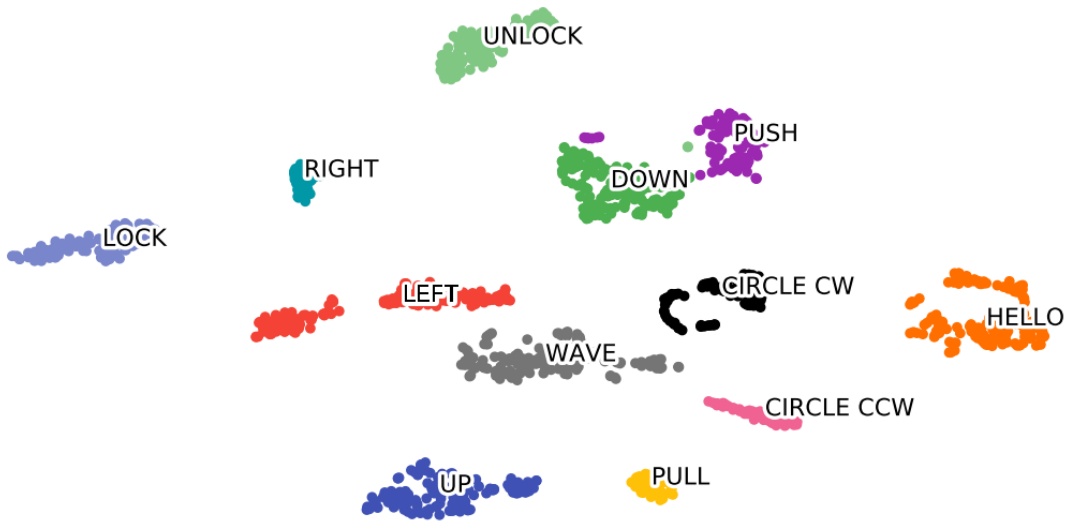
SLIKA 22: OPSEG PODATAK ZA GESTU WAVE

5.1.2. t-SNE

t-SNE (engl. t-distributed stochastic neighbor embedding) je algoritam iz grane strojnog učenja za redukciju dimenzionalnosti podataka, razvijen od Laurens van der Maatena i Geoffreyja Hintona [10]. To je nelinearna tehnika podatna za vizualizaciju podataka visokih dimenzija u nisko dimenzionalnom prostoru tipa dvodimenzionalnom ili trodimenzionalnom. Preciznije, t-SNE modelira svaki višedimenzionalni objekt u manje dimenzionalni objekt tako da točke koje su u originalnom prostoru bile blizu ostanu blizu i u novom prostoru te obratno za udaljene objekte.

t-SNE algoritam sastoji se od dva glavna koraka. Prvo se stvara distribucija vjerojatnosti parova višedimenzionalnih objekata tako da objekti koji su bliže imaju veću vjerojatnost da budu izabrani, dok različiti objekti imaju ekstremno malu vjerojatnost kako bi bili izabrani. Drugo, t-SNE radi sličnu distribuciju vjerojatnosti za konačan prostor u koji podaci trebaju biti reducirani, a zatim minimizira Kullback–Leibler divergenciju, koja u matematici predstavlja koliko jedna distribucija divergira od druge, u odnosu na lokacije točaka u prostoru. t-SNE se koristi za vizualizaciju podataka širokih primjena poput istraživanja u računalnoj sigurnosti, analizi glazbe, istraživanju raka, bioinformatici i obradi signala. Često je korišteno kako bi se prikazali podaci visokih dimenzija koji se koriste za učenje umjetnih neuronskih mreža, što je upravo primjena koju mi koristimo. Program za t-SNE vizualizaciju dostupan je na:

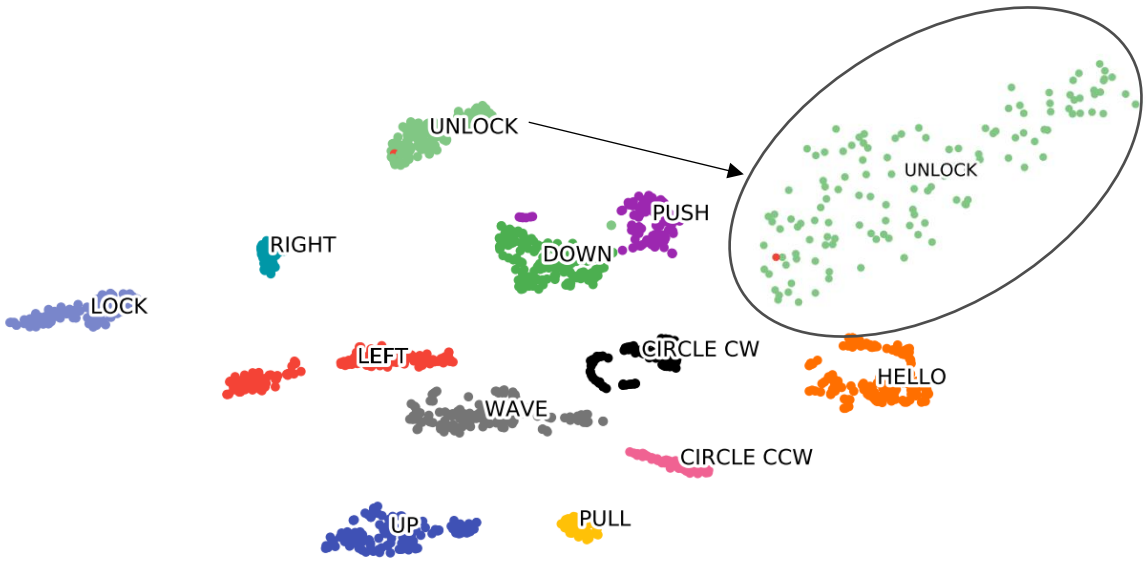
https://github.com/eugen-vusak/zavrsni-rad/blob/master/RecognitionApp/Data/Visualization/Programs/tsne.py



SLIKA 23: VIZUALIZACIJA PODATAKA POMOĆU T-SNE ALGORITMA

Iz ove slike moguće je uočiti grupiranje podataka po gestama, no također možemo vidjeti da su geste DOWN i PUSH bliske. Ipak, ovaj prikaz pokazuje da je moguće podatke klasificirati te daje bitan uvid u nastavak rada.

Još jedna primjena t-SNE algoritma jesu pogreške u označavanju. Ljudi nisu savršeni te se greške mogu dogoditi, a na ovaj način možemo lako ukloniti podatke koji očito ne odgovaraju svojoj oznaci. Možemo pogledati primjer kada jednoj UNLOCK gesti zamijenimo oznaku u npr. LEFT



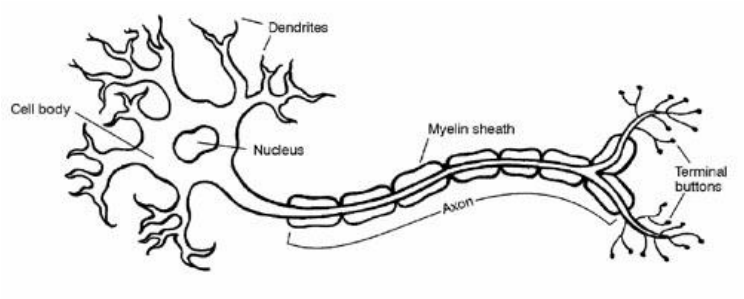
SLIKA 24: PRIMJER VIZUALIZACIJE S KRIVOM OZNAKOM

6. Neuronska mreža

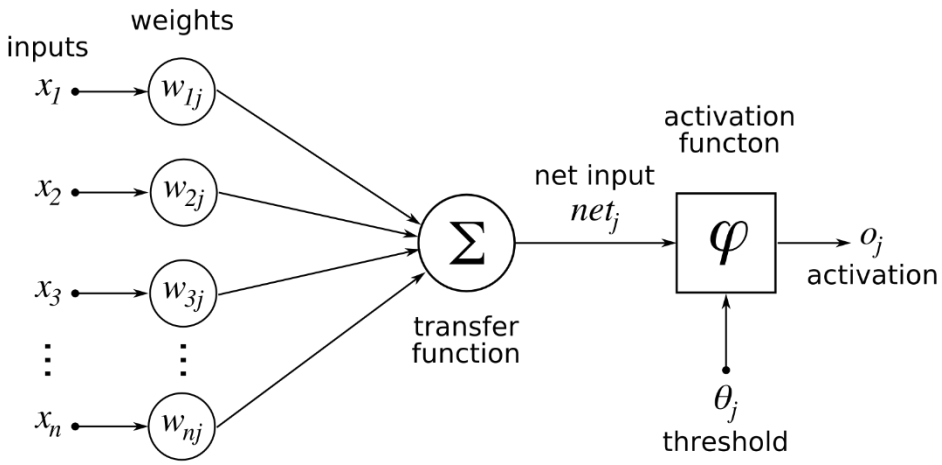
6.1. Teorijski uvod

Najbitniji dio ovog sustava je zasigurno neuronska mreža. Neuronska mreža je računalni sustav za učenje koji je modeliran ugrubo po ljudskom mozgu. Sastoji se od težinski povezanih neurona gdje težina modelira debljinu sinapse u mozgu.

Umjetni neuron sadrži određen broj ulaza koji predstavljaju dendrite stvarnog neurona. Tamo gdje je tijelo stanice neurona, umjetni neuron ima težinsko sumiranje ulaza, a na svome izlazu emulira aksone neurona.



SLIKA 25: STRUKTURA STVARNOG NEURONA



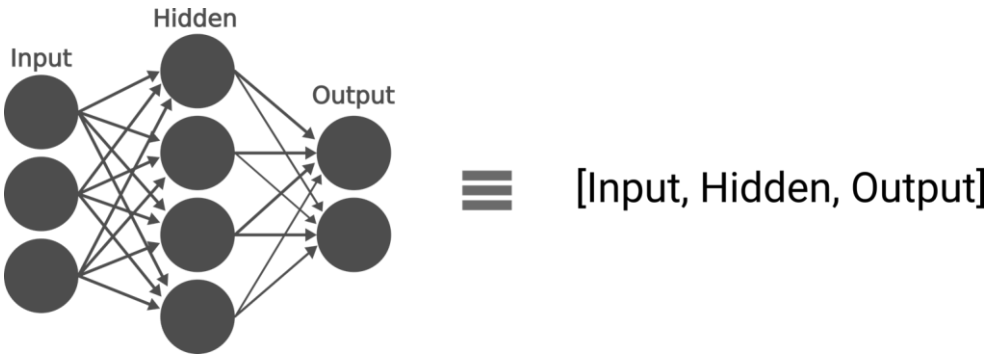
SLIKA 26: MODEL UMJETNOG NEURONA

Umjetni neuron je zapravo samo matematička funkcija.

$$o_j = \varphi\left(\sum_{i=1}^{n} x_i w_{ij}\right)$$

Neuronska mreža zapravo skup međusobno povezanih neurona. U ovom radu pričat će se samo o unaprijednoj neuronskoj mreži, no postoji veliki broj različitih konfiguracija neurona i time tipova neuronskih mreža.

Unaprijedna neuronska mreža zove se tako, jer podaci u svakom trenutku putuju prema naprijed. Sastoji se od liste slojeva, gdje svaki sloj ima određen broj neurona. Prvi sloj zove se ulazni sloj te ima onoliko neurona koliko je velik vektor ulaznih podataka. U našem slučaju to je 360 neurona. Izlazni sloj može biti proizvoljan te također ovisi o primjeni. U našoj primjeni, koja je klasifikacija podataka, uobičajeno je da svaka klasa odgovara jednom neuronu. Točnije, izlazni sloj naše neuronske mreže ima onoliko neurona koliko ima gesti koje prepoznajemo, a to je 12. Slojevi koji se nalaze između ulaznog i izlaznog sloja nazivaju se skriveni slojevi, a njih može biti proizvoljan broj. Što je više slojeva, to je veća sposobnost mreže da interpretira podatke, no preveliki broj može dovesti do sporog učenja, loše generalizacije, pa čak i do toga da mreža uopće ne može naučiti. Za sve jednostavnije sustave sasvim su dovoljna jedan do dva skrivena sloja. Veličina tih slojeva također je proizvoljna, te je nešto što se određuje eksperimentiranjem.



SLIKA 27: UNAPRIJEDNA NEURONSKA MREŽA KAO LISTA SLOJEVA

No neuronska mreža koja nije naučena nema koristi. Učenje neuronske mreže zapravo se svodi na minimizaciju njene pogreške. Želimo da neuronska mreža radi jako male pogreške i to u jako malo slučajeva. No to nije nužno lagan posao. Ako zamislimo neuronsku mrežu kao funkciju koja za određeni ulaz daje određeni izlaz sa svojim parametrima koji su težine (i pomaci) kojih može biti i više milijuna (a i više), onda imamo funkciju čija dimenzija je u milijunima. Traženje minimuma jednodimenzionalne funkcije svodi se na jednostavnu derivaciju. Međutim, sa svakom dimenzijom taj zadatak postaje sve teži i teži. Zato se u ovom radu koristimo vrlo popularnim algoritmom*, Gradijentnim spustom*, u kojemu se prvo traži gradijent u točki te se pomičemo po gradijentu prema manjoj vrijednosti za maleni korak određen hiperparametrom. Nakon pomaka ponovno računamo gradijent u novo dobivenoj točki.

Još jedino što nam je preostalo reći kako se računa gradijent u točki. Za to postoji efikasan algoritam, *Propagacija pogreške unatrag* (engl. Backpropagation). Taj algoritam naziva se tako jer uzima pogrešku koju je dala neuronska mreža te, korištenjem pravila ulančavanja, računa parcijalne derivacije od zadnjeg sloja prema prvom.

Sada kada su pokriveni neki osnovni koncepti, neuronsku mrežu se dalje modelira matematički.

Neuronska mreža koja se koristi u ovom kodu modelirana je po primjeru iz online knjige Michaela Nielsena [11], uz nekoliko izmjena. Za to je potrebno definirati neke pojmove:

X - ulazni vektor u neuronsku mrežu

$Z^{(l)}$ - vektor ulaza u sloj l

$A^{(l)}$ - vektor izlaza iz sloja l

$W^{(l)}$ - matrica težina (engl. weights) sloja l

$B^{(l)}$ - vektor pomaka (engl. biases) sloja l

$E^{(l)}$ - vektor pogrešaka (engl. errors) sloja l

$\Delta^{(l)}$ - vektor delti sloja l gdje je delta funkcija od E i Z

C - trošak (engl. cost) je suma svih članova vektora pogreške zadnjeg sloja

Y - očekivani izlazni vektor iz neuronske mreže

Sada se mogu opisati tri algoritma koji se koriste. To su *feedforward* algoritam za računanje izlaza, algoritam propagacije unatrag (engl. *backpropagation*) za računanje gradijenta i gradijentni spust, kojim pomičemo težine tako da ukupna pogreška pada.

6.1.1. Feedforward

Prvi sloj je jedinstven te za njega vrijedi

$$Z^{(1)} = X \tag{1}$$

dok za ostale čvorove vrijede jednakosti :

$$Z^{(l)} = W^{(l)} \cdot A^{(l-1)} + B^{(l)} \tag{2}$$

$$A^{(l)} = f(Z^{(l)}) \tag{3}$$

6.1.2. Propagacija pogreške unatrag (Backpropagation)

U ovom slučaju posljednji sloj je jedinstven a za njega vrijedi

$$E^{(L)} = A^{(L)} - Y \tag{4}$$

no za posljednji sloj se delta računa kao i za ostale slojeve

Za ostale slojeve zatim vrijedi:

$$E^{(l)} = {W^{(l+1)}}^T \cdot \Delta^{(l+1)} \tag{5}$$

$$\Delta^l = E^l \odot f'(Z^l) \tag{6}$$

Sada kada su izračunate delte za svaki sloj, uz pomoć njih mogu se izračunati parcijalne derivacije

$$\frac{\partial C}{\partial W^{(l)}} = \Delta^{(l)} \cdot A^{(l-1)} \tag{7}$$

$$\frac{\partial C}{\partial B^{(l)}} = \Delta^{(l)} \tag{8}$$

Sada se pomoću tih vrijednosti mogu ažurirati težine.

6.1.3. Gradijentni spust

$$W^{(l)} = W^{(l)} - \mu \cdot \frac{\partial C}{\partial W^{(l)}} \tag{9}$$

$$W^{(l)} = W^{(l)} - \mu \cdot \frac{\partial C}{\partial W^{(l)}} \tag{10}$$

Malo grčko slovo μ („mi“) je hiperparametar koji predstavlja brzinu učenja (engl. learning rate), a odabire se prije početka učenja. On određuje brzinu pomaka po krivulji u smjeru gradijenta, točnije koliko su veliki takvi skokovi.

6.2. Implementacija

Sada se može opisati kod koji modelira neuronsku mrežu definiranu u prošlom poglavlju. Navedeno je ranije da je neuronska mreža opisana listom slojeva, te da je za opisivanje mreže potrebno prvo modelirati sloj. Postoje dva tipa sloja. To su standardni sloj i ulazni sloj. Svaki od tih slojeva modeliran je svojom klasom, a nalaze se u paketu *layers.py.*
Izdvoji se najbitniji dio sloja, a to je funkcija output(), koja za neki ulazni vektor daje na izlazu novi izlazni vektor po formulama (1), (2) i (3).

```
1.    def output(self, input):
2.        self.Z = np.dot(self.weights, input) + self.biases
3.        self.A = self.function(self)
4.        return self.A
```

KOD 2: FUNKCIJA OUTPUT RAZREDA LAYER

ovdje se primjećuju formule (2) u liniji 2 i (3) u liniji 3.

U razredu InputLayer način računanja je dosta sličan, no nešto jednostavniji. Razlika je da se u liniji 2 koristi formula (1) a ne formula (2), no za računanje A također koristimo (3).

```
1.    def output(self, input):
2.      self.Z = input
3.      self.A = self.function(self)
4.      return self.A
```

KOD 3: FUNKCIJA OUTPUT RAZREDA INPUTLAYER

Potrebno je primijetiti da se poziva funkcija *function*() u oba slučaja. Ta funkcija prima se kroz konstruktor, a ima oblik:

```
1. def ime_funkcije(layer, deriv = False){
2.     if deriv:
3.         #vrati rezultat derivacije funkcije za layer.Z
4.     #vrati rezultat funkcije za layer.Z
```

KOD 4: KOSTUR AKTIVACIJSKE FUNKCIJE

Može se primijetiti da funkcija ne prima samo layer.Z, već i cijeli sloj. To je radi bržeg računanja derivacije koja nekad ovisi samo o već prije izračunatom layer.A, tako da nije potrebno ponovno provoditi kompleksne kalkulacije. Postoji datoteka s već unaprijed oblikovanim funkcijama, ActivationFunctions.py

Sada kada je poznato što radi jedan sloj, može se preći na neuronske mreže. Potpuni kod implementirane neuronske mreže nalazi se u modulu Neural_Networks u __init__.py datoteci. Započinje se ponovno s *feedforward* algoritmom. U neuronskoj mreži, izvedba tog algoritma svodi se na iteriranje kroz sve slojeve u listi slojeva te spajanje izlaza prošlog sloja na izlaz sljedećeg sloja. Ulaz prvog sloja je ulaz u feedforward() funkciju.

```
1. def feedforward(self, input):
2.     for layer in self.layers:
3.         input = layer.output(input)
4.     return input
```

KOD 5: FUNKCIJA FEEDFORWARD RAZREDA FEEDFORWARDNEURALNETWORK

Algoritam propagacije pogreške unatrag malo je složeniji, zato je najbolje raščlaniti ga po dijelovima.

```
1.    def backpropagate(self,output, desired_output):
2.        #calculate for the last layer first
3.        output_layer = self.layers[-1]
4.        output_layer.error = output - desired_output
5.
6.
7.        self.E += (output_layer.error**2 / 2).sum()
8.
9.        deriv = np.multiply(output_layer.A, 1-output_layer.A)
10.
11.       output_layer.delta = np.multiply(output_layer.error, deriv)


12.
13.       #from penultimate layer to second layer
14.       for l in reversed(range(1,len(self.layers)-1)):
15.
16.               layer = self.layers[l]
17.
18.               layer_plus_one = self.layers[l+1]
19.               layer.error = np.dot(
20.                    layer_plus_one.weights.T,
21.                    layer_plus_one.delta
22.               )
23.
24.
25.               deriv = layer.function(layer, deriv = True)
26.
27.               layer.delta = np.multiply(layer.error, deriv)
```

KOD 6: FUNKCIJA BACKPROPAGATE RAZREDA FEEDFORWARDNEURALNETWORK

U liniji 4 izračuna se pogreška za posljednji sloj po (4), zatim po (6) izračuna se delta za isti taj sloj u liniji 9. Linija 7 računa kvadratnu pogrešku po članovima vektora *error* za iscrtavanje grafa. Sada za sve slojeve od predzadnjeg do drugog, u liniji 19, računamo pogrešku po (5), a u liniji 24 deltu po (6).

Još jedini dio koji je ostao implementirati je gradijentni spust. U ovoj izvedbi on malo odstupa od gore navedenih formula, jer je omogućen takozvani stohastički gradijentni spust koji ne ažurira veze nakon svakog ulaznog podatka, već akumulira vrijednosti gradijenta kroz nekoliko ulaza (engl. mini batch gradient descent) ili kroz sve (engl. batch gradient descent). Vrijednosti se zatim ažuriraju prosječnom vrijednošću tih akumuliranih vrijednosti.

```
1.    def gradientDescent(self):
2.        for l in range(1,len(self.layers)):
3.
4.            layer = self.layers[l]
5.            layer_minus_one = self.layers[l-1]
6.
7.            layer.pC_pW += np.dot(layer.delta, layer_minus_one.A.T)

8.            layer.pC_pB += layer.delta
```

KOD 7: FUNKCIJA GRADIENTDESCENT RAZREDA FEEDFORWARDNEURALNETWORK

Sada za svaki sloj osim prvoga računamo vrijednosti gradijenta i akumuliramo ih po formulama (7) i (8).

U ovom trenutku neuronska mreža spremna je za učenje. Učenje se obavlja po epohama. Jedna epoha je jedna iteracija kroz cijeli skup podataka. U svakoj epohi želi se prolaziti kroz određene dijelove ispitnih primjera (engl. mini batch), a za svaki testni primjer u njoj se želi izvoditi algoritme onim redom kojim su i opisani.

Pseudo kod za takvu mrežu glasio bi:

Za svaku epohu:

Za podatak u setu:

feedforward()

backpropagation()

gradient_descent()

ažuriraj podatke

To je u kodu riješeno ovako:

```
def train(self, training_data, epochs=200, learning_rate=0.5):
        for i in range(epochs):
            print("epoch", i)
            for data in training_data.data_set:
                input, desired_output = data

                output = self.feedforward(input)

                self.backpropagate(output, desired_output)

                self.gradientDescent()

                for layer in self.layers:
                    layer.updateParametars(1, learning_rate)

            self.ErrorAxis.append(self.E / 745)
            self.E = 0
```

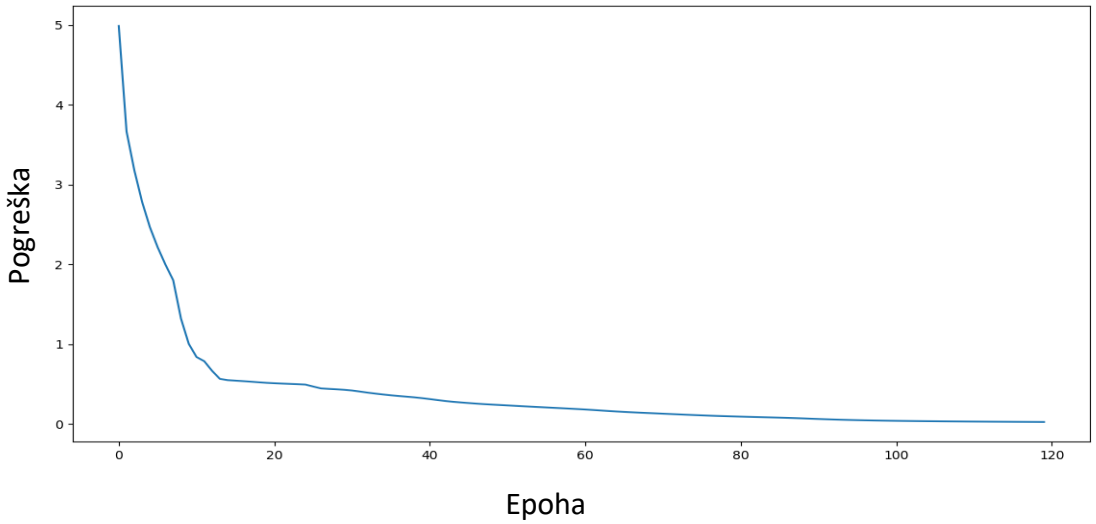
KOD 8: FUNKCIJA TRAIN RAZREDA FEEDFORWARDNEURALNETWORK

Sada se može učiti i ispitati našu neuronsku mrežu. Prvo je potrebno podatke podijeliti na podatke za učenje te podatke za ispitivanje, u omjeru 80 : 20. Omjer je također proizvoljan no omjeri 80 : 20 i 70 : 30 su najčešće korišteni. Neuronska mreža koja će se izraditi imat će ulazni sloj veličine 360 neurona što je ujedno i duljina vektora dobivena uzorkovanjem geste duljine 1.2 sekunde s frekvencijom 50Hz. Testiranje je pokazalo da je

dovoljan samo jedan skriveni sloj, a iskustva su pokazala da broj neurona koji je jednak polovici razlike ulaza i izlaza često daje dobre rezultate.

Nakon što smo izgradili neuronsku mrežu, mogu se učitati podatke za učenje te ih staviti kao ulazni parametar funkciji *train*(). Broj epoha na kojima će se neuronska mreža učiti je 120, a hiperparametar učenja μ je 0.1. Treniranje je izvedeno u datoteci train.py

Na kraju se može nacrtati kako pada pogreška kroz vrijeme. Na kraju potrebno je neuronsku mrežu spremiti kako bi se mogla koristiti kasnije. Kako bi se to moglo izvesti potrebno je serijalizirati objekt razreda *NeuralNetwork*, tj. pretvoriti ga u niz byteova koji se zatim spremaju u datoteku. .



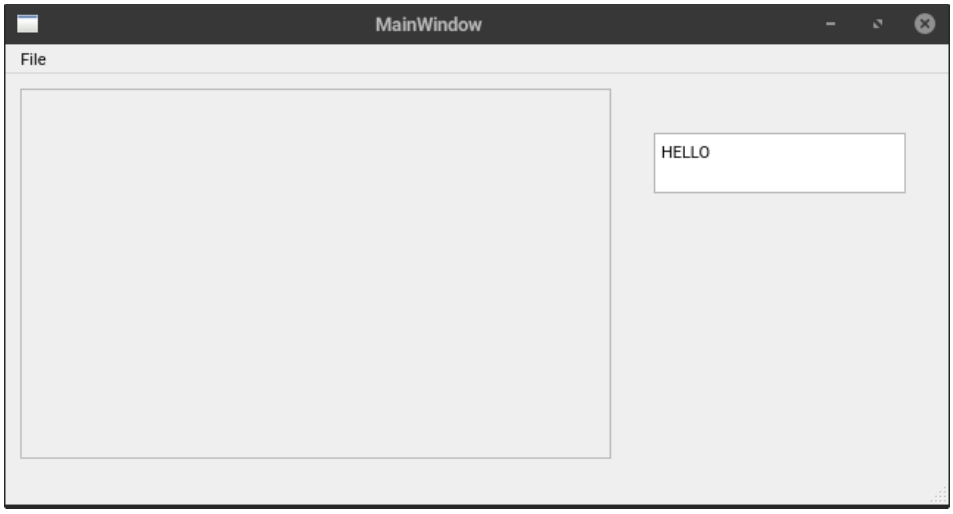
SLIKA 28: PAD POGREŠKE KROZ EPOHE

Tako serijaliziranu neuronsku mrežu može se zatim učitati iz datoteke, učitati podatke za testiranje te iterativno za svaki ispitni primjer pozivati funkciju feedforward() neuronske mreže te provjeriti razlikuje li se dobiveni rezultat od očekivanog. Broje se točno klasificirani primjeri te se zatim na kraju prikazuje postotak točnosti. Testiranje neuronske mreže izvedeno je u datoteci test.py

U slučaju ovog rada, rezultat je **91.429% točnih primjera**, što je dobro, ali može i bolje. Uz malo eksperimentiranja s arhitekturom mreže, količinom podataka i hiperparametrima točnost na ispitnom skupu se može povećati, međutim u tom slučaju potrebno je eksperimentirati na zasebnom, validacijskom skupu podataka koji će služiti za optimizaciju vrijednosti parametara.

7. Aplikacija za prepoznavanje u stvarnom vremenu

Posljednji dio sustava je aplikacija koja prepoznaje geste u stvarnom vremenu. Aplikacija je napisana u jeziku Python 3.6 i koristi programski okvir PyQt5. Za čitanje podataka sa serijskog porta napravljen je razred *Serial* koji je tipa *QThread*, točnije to je dodatna dretva koja provjerava postoje li podaci na serijskom ulazu te, ako oni postoje, odašilje signal koji sadržava te podatke. Glavni program definira grafičko sučelje te pokazuje prozor s njime.



SLIKA 29: GRAFIČKO SUČELJE APLIKACIJE ZA PREPOZNAVANJE U STVARNOM VREMENU

Također, program učitava model neuronske mreže i stvara novi objekt tipa *Serial* te pokreće dretvu. Na dolazak signala sa serijskim podacima provjerava je li došlo do pomaka te, ako je do pomaka došlo, pokreće učitavanje podataka u red iz kojeg, kada se napuni, prepoznaje se gesta tako da se podaci prebace na ulaz modela neuronske mreže. Kada je gesta prepoznata, program ispisuje njenu oznaku na grafičko sučelje te pauzira 2s kako bi se izbjeglo dodatno pogrešno prepoznavanje.

Za pokretanje aplikacije potrebno je imati instaliranu verziju 3.6. Pythona, te instalirane biblioteke *PyQt5* i *pyserial*. Njih je moguće instalirati korištenjem upravitelja paketima „pip“. Aplikacija se zatim pokreće izvođenjem naredbe python3 main.py.

7.1. Vrednovanje

Zbog teške kontrole i velikih sličnosti nekih gesti, te nemogućnosti raspoznavanja lažno pozitivnih primjera, greške aplikacije su dosta česte. Aplikacija prepoznaje dobro oko 50% vremena. Često se dogodi da prepozna neku gestu koju je lako zamijeniti sa statičnim stanjem, poput geste WAVE. Također, geste DOWN i PUSH su gotovo jednake primjenom ovih podataka te ih je vrlo lako zamijeniti. Da bi se to izbjeglo, potrebno je dodati podatke o ne samo kutnim akceleracijama, već o nagibima u određenim osima. Takve podatke nudi skoro svaki pametni sat, koji i je uređaj za koji je ovaj rad namijenjen. Također, uz više podataka, rezultati bi bili bolji no prikupljanje podataka zahtijeva vrijeme, te se očekuje da će se količina podataka povećavati kako vrijeme ide. S obzirom na trenutačne mogućnosti, rezultati su prihvatljivi te pokazuju kako ova metoda može biti uspješno korištena za ovaj problem.

8. Zaključak

Različiti načini komunikacije s računalima postaju sve češći i ljudi svakodnevno traže nove i lakše načine kako olakšati prenošenje informacija strojevima, ne bi li olakšali svakodnevnicu. Neuronske mreže su vrlo široko područje i primjenjive su na puno toga. Ovaj rad prikazuje jednu od mogućih primjena jedne od mogućih varijacija neuronske mreže. Naravno neuronske mreže nisu samo algoritam u koji ubacimo podatke a on izbaci odgovor. Potrebno je puno eksperimentiranja te određivanja različitih hiperparametara i arhitektura kako bi se postigli rezultati. Kako bi ovaj rad bio ne samo istraživanje već i konačni proizvod, potrebno je povećati preciznost prepoznavanja u stvarnom vremenu te uvelike smanjiti prepoznavanje lažno pozitivnih primjera. To je moguće riješiti dodavanjem više znanja o svijetu prilikom prikupljanja podataka, te učenjem podataka koje ne treba prepoznati kao gestu. Neki od mogućih koraka jesu zamjena senzora s kamerama te prikupljanje podataka o poziciji ruke na taj način. Naravno, prvi sljedeći korak je zamijeniti trenutačni uređaj s kupovnim pametnim satom te napraviti aplikaciju za njega koja će odašiljati podatke.

9. Literatura

[1] http://web.sonoma.edu/users/f/farahman/sonoma/courses/es310/resources/aboutmicro.htm, Sonoma State University, 2018

[2] https://www.sciencedirect.com/science/article/pii/S2212017312002411 , Blanca MiriamLee-Cosio, Carlos Delgado-Mata, Jesus Ibanez, Pricedua Technology Volume 3, 2012, 109-120

[3] https://store.arduino.cc/usa/arduino-nano, Arduino, 2018

[4] https://github.com/arduino/Arduino/wiki/Build-Process, Arduino, 2018

[5] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf, InvenSense, 2018

[6] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf, InvenSense, 2018

[7] http://www.eecs.umich.edu/courses/eecs206/archive/spring02/notes.dir/iir4.pdf , The Electrical Engineering and Computer Science, University of Michigan, 2002

[8] http://e-rokodelnica.si/A003/A003_EN.html e-rokodelnica, 2018

[9] https://processing.org/ , Processing, 2018

[10] https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

[11] http://neuralnetworksanddeeplearning.com/, Michael Nielsen, 2017

[12] https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/, Matt Mazur, 2015

[13] https://www.coursera.org/learn/neural-networks , University of Toronto, Geoffrey Hinton, 2018

[14] Make Your Own Neural Network, Tariq Rashid, 2016

[15] http://colah.github.io/posts/2015-08-Backprop/, Christopher Olah, 2015

10. Popis slika

11. Popis tablica

12. Popis kodova

Sustav za prepoznavanje gesti ruku temeljen na podacima s mikrokontrolerske pločice i neuronskoj mreži

Sažetak

U ovom radu pokriven je čitav proces izrade sustava koji koristi neuronske mreže. Mikrokontrolerska pločica, Arduino Nano, sa senzorom MPU-6050, korištena je kao izvor podatka. Pomoću nje prikupljeni su podaci o gestama. Za to je bila izrađena aplikacija u programskom jeziku Processing, koja tražili podatke od mikrokontrolera te dodaje odgovarajuću oznaku geste na te podatke. Podaci su zatim zapisani u .csv datoteku. Neuronska mreža koja je implementirana je unaprijedna neuronska mreža, a algoritam za učenje je algoritam propagacije pogreške unatrag. Nakon što smo istrenirali mrežu i dobili model koji prepoznaje geste, a taj model je ugrađen u aplikaciju za prepoznavanje u stvarnom vremenu.

Ključne riječi: unaprijedne neuronske mreže, mikrokontroler, prepoznavanje, geste, digitalni filter, algoritam propagacije unatrag, gradijentni spust, klasifikacija

Hand gesture detection system based on microcontroller data and neural network

Abstract

This paper covers the entire process of making system using neural networks. The Arduino Nano, microcontroller board, with the MPU-6050 sensor, was used as a source of data. Using it gestures data was collected. For this, an application was developed in the Programming language Processing, that asked for data from the microcontroller and added the corresponding gesture label to data. The data was then written to a .csv file. The neural network that was implemented is a feedforward neural network, and the algorithm for learning is the backpropagation. After we've trained the network and got a model for recognizing gestures, this model was embedded in the real-time recognition app.

Keywords: feedforward neural network, microcontroller, recognition, gesture, digital filter, backpropagation, gradient descent, classification