VEGOVA | ELEKTROTEHNIŠKO-RAČUNALNIŠKA STROKOVNA ŠOLA IN GIMNAZIJA LJUBLJANA

Klasifikacija in prepoznava gibov roke na podlagi podatkov iz senzorja gibanja

Informatika in računalništvo

Raziskovalna naloga

Mentor: Aleksandar Lazarević, univ. dipl. inž

Avtor: Bojan Dogandžić, R 4. B

Domžale, marec 2022

Klasifikacija in prepoznava gibov roke na podlagi podatkov iz senzorja gibanja

Informatika in računalništvo

Raziskovalna naloga

2022

Kazalo

Kazalo slik

Povzetek

Za raziskovalno nalogo sem izdelal napravo, ki prepoznava gibe roke na podlagi podatkov iz senzorja gibanja. Izdelal sem dva vezja, enega za testiranje in drugega, ki je končna naprava. Za krmiljenje sem uporabil mikrokrmilnik ESP32. Za prepoznavo giba sem najprej moral narediti program, ki bi podatke shranil v bazo. Tukaj sem naredil dva programa, enega v programskem jeziku Arduino in drugega v programskem jeziku python. Arduino program izpisuje podatke iz senzorja gibanja na serijski monitor, python program pa jih nato shrani v podatkovno bazo. Naslednji korak je bil klasifikacija gibov, tukaj sem s pomočjo knjižnic v pythonu testiral tri različne klasifikatorje in ustvaril header datoteko v kateri je klasifikator v programskem jeziku C. Za uporabo tega klasifikatorja pa sem naredil še četrti program, ki je podoben prvemu Arduino program, samo da namesto izpisovanja podatkov na serijski monitor, izpisuje prepoznane gibe na zaslon na podlagi klasifikatorja. Zadnja faza pa je bila testiranje naprave.

Abstract

I made a circuit that recognizes hand movements based on data from the motion sensor. I made two circuits, one for testing and the other that is the final product. I used an ESP32 microcontroller. To identify the movement, I first had to make a program that would store the data in a database. Here I made two programs, one in the Arduino programming language and the other in the python programming language. The Arduino program prints data from the motion sensor to a serial monitor, and python program then stores that data in a database. The next step was the classification of movements, here, with the help of python libraries, I tested three different classifiers and created a header file containing the classifier in C programming language. To use this classifier, I made a fourth program, that is similar to the first Arduino program, except that instead of printing data to a serial monitor, it displays the identified movements on the screen. The last phase was testing the circuit.

Ključne besede: Arduino, ESP32, klasifikacija, python, strojno učenje

1 Uvod

V svetu prav tako pa tudi v moji bližnji okolici je veliko starejših in bolnih oseb, ki so tudi v glavnem sami in v svoji bližini nimajo osebe, ki bi jih opozarjala da jemljejo zdravila. Za raziskovalno nalogo sem se tako odločil narediti napravo, ki bi prepoznala gib roke med jemanjem zdravila, glede na podatke iz senzorja gibanja. Za klasifikacijo giba sem uporabil strojno učenje in med različnimi algoritmi izbral najboljšega. Glede na to, da naprava prepoznava gib roke, je izdelana v obliki ročne ure in jo ima lahko oseba ves čas na roki. Ta naprava bi bila lahko uporabna tudi v zdravstvenih ustanovah in sicer bolnicah, domovih za starejše občane…

Hipoteze

- Ali je možna implementacija klasifikatorja na nizkocenovnem mikrokrmilniku za klasifikacijo na podlagi senzorskih podatkov v realnem času.
- Naprava lahko klasificira več kot en gib.

2 Strojna oprema in orodje

2.1 Testno vezje

Izdelal sem testno vezje in vezje za končni napravo. Pri testnem vezju sem za krmiljenje najprej uporabil mikrokrmilnik Arduino Uno, ampak sem se nato odločil za mikrokrmilnik ESP32 Wemos D1 R32, ker ima večji pomnilni prostor od Arduinota, ki ga potrebujem zaradi velikosti datoteke za klasifikacijo giba. Prav tako pa ima ESP32 tudi hitrejšo frekvenco ure in sicer 240MHz, medtem ko ima Arduino Uno frekvenco ure 16MHz, s tem sem lahko zajel tudi več podatkov za en gib, kar naredi klasifikacijo natančnejšo. Na spodnji primerjalni tabeli so prikazane še ostale specifikacije mikrokrmilnika.

		Uno	ESP32
General	Dimensions	2.7" x 2.1"	2" x 1.1"
	Pricing	$20-23	$10-12
Connectivity	I/O Pins	14	36
	PWM Pins	6	16
	Analog Pins	6	Up to 18 *
	Analog Out Pins (DAC)		2
Computing	Processor	ATMega328P	Xtensa Dual Core 32-bit LX6 microprocessor
	Flash Memory	32 kB	4 MB
	SRAM	2 kB	520 kB
	EEPROM	1 kB	-
	Clock speed	16 MHz	Upto 240 MHz
	Voltage Level	5V	3.3V
	USB Connectivity	Standard A/B USB	Micro-USB

Slika 1:Primerjalna tabela med Arduino Uno in ESP32

Za merjenje pospeška in rotacije sem uporabil senzor MPU6050, ki ima vgrajen merilnik pospeška in žiroskop. MPU6050 deluje prek I2C komunikacije in ima naslov 0x68, ki se ga da spremeniti na 0x69 če pin AD0 postavimo na 1. Za povezovanje komponent sem uporabil moške žičke in vse povezal na eksperimentalno ploščo. MPU6050 ima 8 pinov od tega sem jih uporabil 4 in sicer VCC, kjer priklopimo 3,3V, GND, ki je masa ter SCL in SDA, ki se uporabljata za I2C komunikacijo. Z MPU6050 lahko merimo tudi temperaturo ampak tega nisem potreboval. Testno vezje sem napajal preko USB vhoda na računalniku z micro USB kablom.

2.2 Končno vezje

Ker bi bil ESP32 Wemos D1 za ohišje prevelik, sem pri končnem vezju namesto tega uporabil ESP32 MH-ET LIVE D1 mini. Za ohišje končnega vezja sem uporabil črno plastično ohišje velikosti 58x57x28 mm. Senzor za merjenje pospeška in rotacije sem uporabil isti kot pri testnem vezju in sicer MPU6050.

Uporabil sem tudi OLED zaslon SSD1306 I2C 0,91inc 128x32, ki enako kot MPU6050 deluje prek I2C komunikacije in ima naslov 0x3C. Zaslon lahko prikazuje samo modro barvo in ima 4 pine: VCC, kjer priklopimo 3,3V, GND, ki je masa ter SCK (SCL) in SDA, ki se uporabljata za I2C komunikacijo.

Za napajanje sem uporabil 3,6V 1/2AA litijevo baterijo Xcell ER14250 z kapaciteto 1200 mAh, ki ima nameščena kontakta v nasprotni smeri (Z oblika). Med vpisovanjem podatkov v bazo pa sem napajal preko USB vhoda na računalniku z micro USB kablom. Za vklapljanje naprave sem uporabil eno-polno stikalo velikosti 10x15mm.

Uporabil sem tudi dvostransko PCB ploščo velikosti 8x12cm. Za pas sem uporabil trak, ki je bil na ščitniku.

Vezju sem dodal tudi aktivni piskač in Reset tipko, ker je krmilnik na začetku potrebno resetirati in do Reset tipke na krmilniku ne bi mogel dostopati, ker je zaprt v ohišju.

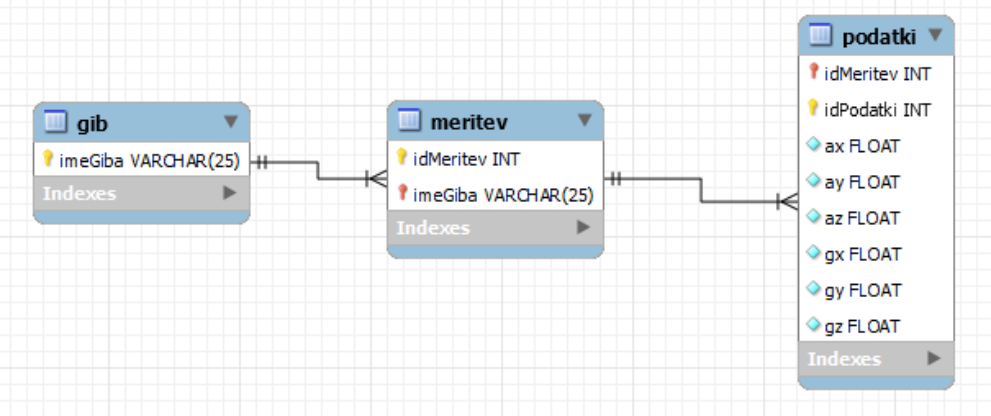
Uporabil sem tudi 2 ženski letvici s tremi pini in eno žensko letvico s osmimi pini, ter 2 moški letvici z tremi pini in 2 moški letvici z desetimi pini. Komponente pa sem povezoval z spajkanjem povezovalnih žičk.

3 Izdelava naloge

3.1 Priprava naloge

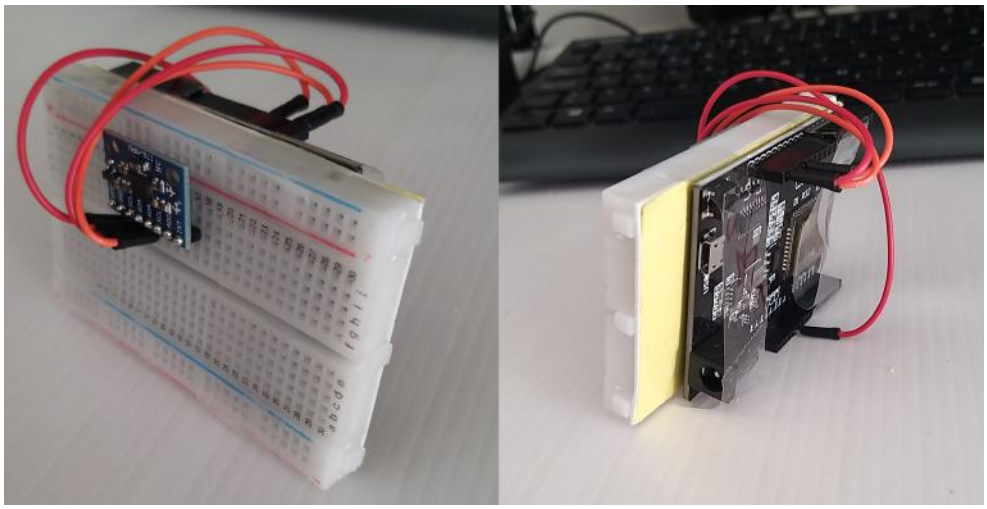
Od programske opreme sem za izdelavo raziskovalne naloge uporabljal razvojno okolje Arduino IDE za pisanje Arduino programa, razvojno okolje PyCharm za pisanje python programa in Xampp MariaDB za izdelavo podatkovne baze. Arduino programi imajo dve funkciji, setup in loop. Setup funkcija se izvede samo enkrat na začetku programa. V njej inicializiramo spremenljivke, nastavimo komunikacijo (npr. serijsko…), nastavimo način pinov (vhod/izhod), itn. Loop funkcija pa je glavna zanka programa.

Najprej sem naredil podatkovno bazo s pomočjo Xampp MariaDB strežnika, kjer sem ustvaril entitetne tipe: gib, meritev in podatki. V entitetnem tipu gib shranjujem imena različnih gibov(npr. zajem zdravila, gib v levo, desno), v entitetnem tipu meritev sem shranjeval primere/meritve teh gibov. V entitetnem tipu podatki pa shranjujem podatke vseh treh osi (x,y,z) merilnika pospeška in vseh treh osi žiroskopa, za vsako meritev giba 350. Bazo sem najprej sestavil brez entitetnega tipa »gib«, ampak sem se nato odločil dodati še »gib«, ker lahko tako v bazo shranjujem več gibov. Spodaj je ER model baze.



Slika 2: ER-model baze

Po tem sem izdelal testno vezje, tako da sem MPU6050 postavil v eksperimentalno ploščo, na drugo stran eksperimentalno plošče, pa sem postavil ESP32 in ga zalepil z lepilnim trakom. MPU6050 sem povezal z moškimi žičkami tako da sem pin VCC na MPU6050 povezal z 3,3V pinom na ESP32, GND z GND pinom ter SCL in SDA pina z SCL in SDA pinoma na ESP32.



Slika 3: Testno vezje

Ko sem sestavil testno vezje, sem za testiranje vezja uporabil program iz interneta, ki izpisuje vrednosti senzorja MPU6050 na serijski monitor in opazoval kako se te vrednosti spreminjajo. Program je uporabljal samo knjižnico Wire.h, ki omogoča I2C komunikacijo. Ta program sem spremenil tako da sem dodal še knjižnici Adafruit_MPU6050.h in Adafruit_Sensor.h za lažje zajemanje podatkov iz senzorja. Po tem ko sem testiral kako deluje senzor MPU6050, sem naredil 4 programe: 2 programa za Vpisovanje podatkov v bazo in 2 programa za prepoznavo giba.

3.2 Vpisovanje podatkov v bazo

3.2.1 Arduino

Prvi program je napisan v programskem jeziku Arduino, v njem se zbirajo podatki za vpis v bazo in izpisujejo na serijski monitor. Pri tem programu sem uporabil iste knjižnice kot prej. V setup funkciji program vzpostavi serijsko komunikacijo z funkcijo Serial.begin(), v katero pošlje kot argument hitrost prenosa podatkov, ki je v tem programu 115200 bitov na sekundo. Prav tako je v setup-u vzpostavljena povezava z zaslonom in senzorjem gibanje, ter ostale nastavitve za njiju.

Program ima funkcijo »kalibracija« za kalibracijo podatkov iz senozrja. Potrebna je zato, da imajo vse osi ob mirovanju približno vrednost 0. Deluje tako da zajame podatke za vse osi med mirovanjem (začetni položaj) in jih shrani v tabelo »baseline« in nato ko začne zajemati podatke, v funkciji »zajemPodatkov«, za vpisovanje v bazo, od vsake osi odšteje ustrezno vrednost iz tabele baseline. Vrednost iz senzorja so tudi omejene med -20 in 20 z funkcijo constrain. Kalibracija se izvede ob vklopu naprave ali ob pritisku tipke reset.

```
for (int i = 0; i < steviloPodatkov; i++) {
  mpu.getEvent(&a, &g, &temp);
  int index = i * steviloOsi;
  features[index] = constrain(a.acceleration.x - baseline[0], -Meja, Meja);
  features[index + 1] = constrain(a.acceleration.y - baseline[1], -Meja, Meja);
  features[index + 2] = constrain(a.acceleration.z - baseline[2], -Meja, Meja);
  features[index + 3] = constrain(g.acceleration.x - baseline[3], -Meja, Meja);
  features[index + 4] = constrain(g.acceleration.y - baseline[4], -Meja, Meja);
  features[index + 5] = constrain(g.acceleration.z - baseline[5], -Meja, Meja);
}
```

Slika 4: Zajem podatkov in shranjevanje v tabelo features

Program deluje tako da čaka da uporabnik vnese vrednost »y« in ko vnese to vrednost, začne preverjati ali se je zgodil premik. To preverja s funkcijo »zaznanPremik«, ki preverja če je vsota absolutnih vrednosti vseh treh osi merilnika pospeška večja od nastavljene praga (ang. treshold), ki je v mojem primeru nastavljen na 5.

```
bool zaznanPremik(float ax, float ay, float az) {
  return (abs(ax) + abs(ay) + abs(az)) > THRESHOLD;
}
```

Slika 5: Funkcija zaznanPremik

Ko zazna premik začne zajemati podatke iz senzorja in jih shranjevati v tabelo »features«. V tabelo shrani 350 podatkov posamezne osi oz. 2100 podatkov. Ko shrani vse te podatke v tabelo jih izpiše na serijski monitor z funkcijo Serial.begin.

3.2.2Python

Za vpisovanje podatkov v podatkovno bazo sem uporabil programski jezik python. Program deluje tako da s pomočjo knjižnice pyserial bere podatke iz serijskega vhoda z funkcijo readline() in te podatke nato dekodira z funkcijo decode(). Za vzpostavitev serijske komunikacije sem moral vpisati serijski vhod, ki je v mojem primeru COM8 in baudrate, ki je enaka hitrosti prenosa podatkov pri serijski komunikaciji v Arduino programu, ki sem jo podal kot argument v funkcijo Serial.begin(), torej 115200. Za vpisovanje podatkov v podatkovno bazo pa sem uporabil knjižnico mysql-connector, kjer sem rabil za vzpostavitev povezavo z podatkovno bazo podati podatke za strežnik, ime uporabnika, njegovo geslo in ime podatkovne baze.

```
import serial
import mysql.connector

arduino = serial.Serial(port='COM8', baudrate=115200, timeout=.1)
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="",
    database="zaznavagiba"
)
```

Slika 6: Povezovanje z podatkovno bazo v python-u

Program deluje tako da najprej uporabnik vpiše ime giba, ki ga program vnese v podatkovno bazo in nato čaka da uporabnik vpiše vrednost »y« in ko vpiše to vrednost, jo zakodira in pošlje krmilniku preko serijske komunikacije, kjer izvede prej navedeni Arduino program in nato python program prebere preko serijske komunikacije, podatke, ki jih je krmilnik izpisal na serijski monitor. Ko zaključi z branjem podatkov iz serijskega vhoda in vpisovanjem teh podatkov v podatkovno bazo, spet začne čakati na vrednost »y«. Če vnesemo vrednost »n« se program zaključi.

Ko sem končal s pisanjem tega programa sem v podatkovno bazo s pomočjo teh dveh programov vnesel 30 meritev za gib jemanje zdravila. Nato sem z SQL ukazom na sliki 7, ustvaril .csv datoteko v kateri so vse meritve giba »zdravilo«. Na enak način sem ustvaril .csv datoteko mirovanje v kateri so meritve, med mirovanjem, ker bi brez tega program vedno zaznal da je bil prepoznan gib »jemanje zdravila«. Meritve za mirovanje niso popolno mirovanje ampak so zelo rahli premiki.

```
SELECT ax,ay,az,gx,gy,gz
FROM podatki p inner join meritev m ON (p.idMeritev=m.idMeritev)
where imeGiba='zdravilo'
order by p.idMeritev,idPodatki
INTO OUTFILE 'C:/Users/novid/Google Drive/Izdelek Matura/ZaznavaGiba/PrepoznavaGiba/data/zdravilo.csv'
FIELDS ENCLOSED BY ''
TERMINATED BY ','
ESCAPED BY ''
LINES TERMINATED BY '\r\n';
```

Slika 7: SQL ukaz za shranjevanje poizvedbe v .csv datoteko

3.3 Prepoznava giba

3.3.1 Algoritmi za klasifikacijo giba

Ena od metod strojnega učenja je klasifikacija. Naloga klasifikatorja je razvrstiti v razrede primere, opisane z množico atributov. To pomeni da iz vhodnih podatkov, ki so množica že klasificiranih primerov, zgradi pravilo, ki ga lahko kasneje uporabimo za klasifikacijo novih primerov. Za klasifikacijo giba sem testiral in poskusil uporabiti 3 različne algoritme za klasifikacijo, in sicer: Naključni gozd, Odločitveno drevo in GaussianNB.

Odločitveno drevo (ang. Decision tree) je zgrajeno v obliki drevesne strukture iz notranjih vozlišč, listov in vej. Deluje tako da iz vhodnih podatkov primerov ustvari da/ne vprašanja (notranje vozlišče) in to dela dokler do konca ne izločimo vse atribute

iz množice. Vozlišča so torej atributi. Prvo vozlišče imenujemo koren. Vsakič, ko ustvari novo vprašanje, ustvari tudi veje drevesa, ki so podmnožice vrednosti atributa. Na koncu vsake veje pa so listi, ki predstavljajo razrede. Ko je drevo zgrajeno, klasificiramo nov primer tako da potujemo od korena navzdol po ustreznih vejah.

Naključni gozd (ang. Random forest) je zgrajen iz več posameznih odločitvenih dreves. Za vsako drevo iz osnovne množice naredijo novo množico, kjer iz osnovne množice izberejo enako število primerov, kot jih je v osnovni množici, s tem da se primeri lahko ponavljajo. Iz teh množic se nato gradijo drevesa, ki pa namesto izbiranja najboljšega atributa med vsemi množicami, izbirajo najboljšega med naključno izbranimi. Vsako drevo nato glasuje kateri razred je najustreznejši in glede na število glasov, klasificira primer z razredom, ki ima največ glasov.

GaussianNB (Gausov Naivni Bayes) temelji na Bayesovem teoremu. Imenuje se naivni, ker predvideva da vhodni podatki niso odvisni eden od drugega, tako da sprememba enega podatka ne bo vplivala na ostale. Za vsak razred po naivni Bayesovi formuli izračuna verjetnost da primer pripada nekemu razredu in primer klasificira v razred z največjo vrednostjo. GaussianNB temelji na Gausovi distribuciji.

3.3.2Python

Ko sem zbral podatke za oba giba, sem naredil drugi python program, ki generira kodo za klasifikacijo giba. Uporabil sem knjižnice **Numpy**, ki je knjižnica za tabele, **os.path**, **scikit-learn**, kjer sem za strojno učenje uporabil klasifikator Naključni gozd in knjižnico **micromglen**, ki se uporablja za prevajanje scikit-learn kode za klasifikacijo v jezik C. Program najprej podatke iz .csv datotek skupaj shrani v tabelo »features« in doda vsaki množici podatkov index giba kateremu pripadajo, imena gibov, ki ga dobi iz imena .csv datoteke, pa shrani v tabelo »classmap« in jim doda index giba. Kot sem prej omenil sem za klasifikacijo uporabil klasifikator Naključni gozd. Elemente v tabeli features se najprej razdelijo tako da se v tabelo X shranijo podatki o meritvah, v tabelo y pa indexe vsake množice podatkov. Klasifikatorju sem kot parameter podal da uporabi 50 dreves z maksimalno globino 20.

```
X, y = features[:, :-1], features[:, -1]
return RandomForestClassifier(50, max_depth=20).fit(X, y)
```

Slika 8: Funkcija scikit-learn klasifikatorja Naključni gozd

Ko zgenerira klasifikator, ga najprej prevede v jezik C s pomočjo knjižnice micromglen in nato ustvari datoteko »model.h« v katero prekopira prevedeno kodo v jeziku C.

3.3.3Arduino

Zadnji program je končni program, ki prepozna gib roke na podlagi ustvarjenega klasifikatorja v datoteki »model.h«. Program je napisan v programskem jeziku Arduino in je podoben prvemu programu za zajem podatkov za vpisovanje v bazo. Razlikuje se v tem da tukaj ne čaka da uporabnik vnese vrednost »y« ampak čaka samo, da bo zaznan premik roke (funkcija zaznanPremik). Ko zazna premik, enako kot pri prejšnjem programu zajame 350 podatkov vseh šestih osi in nato te podatke pošlje v funkcijo »klasifikacija«, ki na podlagi klasifikatorja v datoteki »model.h« določi kateri gib je bil prepoznan in ga izpiše na serijski monitor.

```
void klasifikacija() {
  Serial.print("Detected gesture: ");
  String gesture = classifier.predictLabel(features);
  if (gesture == "zdravilo") {
    Serial.println(gesture);
    oledDisplayCenter(gesture);
  }
  else if (gesture == "levo") {
    Serial.println(gesture);
    oledDisplayCenter(gesture);
  }
  else if (gesture == "desno") {
    Serial.println(gesture);
    oledDisplayCenter(gesture);
  }
  else {
    Serial.println("None");
    oledDisplayCenter("None");
  }
  delay(2000);
  Serial.print("\nDetect new gesture: ");
}
```

Slika 9: Funkcija klasfikacija

3.4 Končno vezje

Ker bi bila PCB plošča za ohišje prevelika sem od plošče odrezal kos velikosti 50x45 mm. Na spodnji strani ohišja sem naredil dve luknji skozi kateri sem dal trak, kjer sem

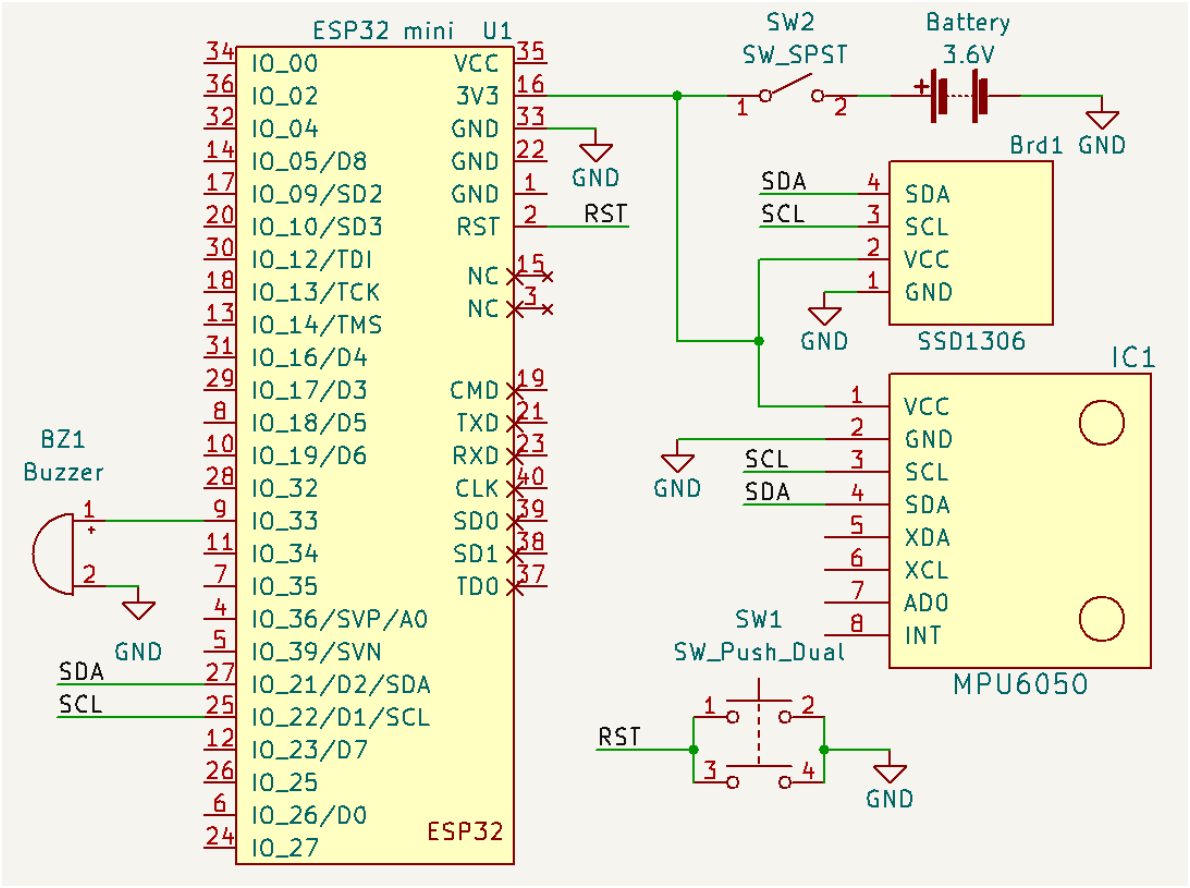
nato oba konca traka zašil skupaj. Na ohišju sem naredil še luknjo za stikalo in luknjo za micro USB kabel.

Na PCB ploščo sem zaspakjal moške letvice, na katere sem v višini luknje za micro USB zaspajkal ESP32 MH-ET LIVE D1 mini. Po tem sem na ploščo zaspajkal še 2 ženski letvici s 3 pini in 1 žensko letvico z 8 pini. Na letvici s 3 pini sem nataknil še dve moški letvici s 3 pini na kateri sem zaspajkal gumb. Na letvico z osmimi pa sem nataknil senzor MPU6050. En kontakt gumba sem povezal z RST pinom, drugi kontakt pa z GND pinom, tako da sem žice zaspajkal na spodnji strani PCB plošče.

Na pokrovu od ohišja sem naredil luknjo skozi katero gre gumb in luknjo skozi katero gredo kontakti zaslona SSD1306. Na notranjo stran pokrova sem kontakte zaslona zaspajkal na majhen košček ploščice velikosti cca. 5x15mm, ki sem ga odrezal od prvotne PCB plošče.

MPU6050 in zaslon imata oba VCC,GND,SCL (na zaslonu SCK) in SDA pine, tako da sem pri obeh zaspajkal žice tako da sem VCC povezal z 3,3V na krmilniku, GND pin z GND pinom krmilnika, SCL/SCK in SDA pa z pinoma GPIO22 (SCL/SCK) in GPIO21 (SDA) na krmilniku. Žice senzorja MPU6050 sem napeljal na spodnjo stran PCB plošče medtem, ko sem žice zaslona najprej napeljal od spodnje strani pokrova do spodnje strani PCB plošča in nato do ustreznega kontakta krmilnika.

Piskač sem zaspajkal direktno na ESP32 MH-ET LIVE D1 mini in sicer en kontakt na GPIO33 pin drugi pa na GND pin. Na zgornji strani plošče sem zaspajkal minus baterije, ki je povezan z žico, ki je na spodnji strani plošče, na GND pin mikrkokrmilnika. Nato sem ploščo pritrdil v ohišje z 2,5x6mm vijaki. Pozitivni kontakt baterije sem nato zaspajkal na en kontakt stikalo, na drugi kontakt stikala pa žico, ki sem jo pred pritrditvijo plošče povezal z 3,3V pinom krmilnika.



Slika 10: Shema končnega vezja

Slika 11: Končna naprava

Ker sem na končnem vezju dodal še piskač in zaslon sem moral tudi malo spremeniti obe Arduino kodi. Piskač začne piskati, ko se začnejo in ko se nehajo zajemati podatki za klasifikacijo. Za zaslon sem uporabil knjižnice SPI.h, Adafruit_GFX.h in Adafruit_SSD1306.h. Dodal sem dve funkciji za izpis besedila na zaslon. Ena funkcija sprejme kot argument eno beseda druga funkcija pa dve besedi. Funkciji delujeta tako da besedilo postavita na sredino zaslona. Na zaslon se pri programu za vpisovanje v bazo izpisuje, kdaj lahko začneš vpisovati v bazo in kdaj se zbirajo podatki za vpis v

bazo. Pri končnem programu za prepoznavanje giba pa se izpisuje kdaj lahko narediš nov gib, kdaj se začnejo zbirati podatki za klasifikacijo in kateri gib je bil prepoznan.

```
void oledDisplayCenter(String text) {
  int16_t x1, y1;
  uint16_t width,height;

  display.getTextBounds(text, 0, 0, &x1, &y1, &width, &height);
  display.clearDisplay();
  Serial.println(height);
  display.setCursor((SCREEN_WIDTH - width) / 2, (SCREEN_HEIGHT - height) / 2 + height);
  display.println(text);
  display.display();
  display.clearDisplay();
}
void oledDisplayCenter(String text1, String text2) {
  int16_t x1, y1, x2, y2;
  uint16_t width1, height1, width2, height2;

  display.getTextBounds(text1, 0, 0, &x1, &y1, &width1, &height1);
  display.getTextBounds(text2, 0, 0, &x2, &y2, &width2, &height2);
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - width1) / 2, (SCREEN_HEIGHT - (height1 + height2)) / 2 + height1);
  display.println(text1);
  display.setCursor((SCREEN_WIDTH - width2) / 2, (SCREEN_HEIGHT - (height1 + height2 - 3)) / 2 + height1 + height2 - 3);
  display.println(text2);
  display.display();
  display.clearDisplay();
}
```

Slika 12: Funkciji za izpis na OLED zaslon

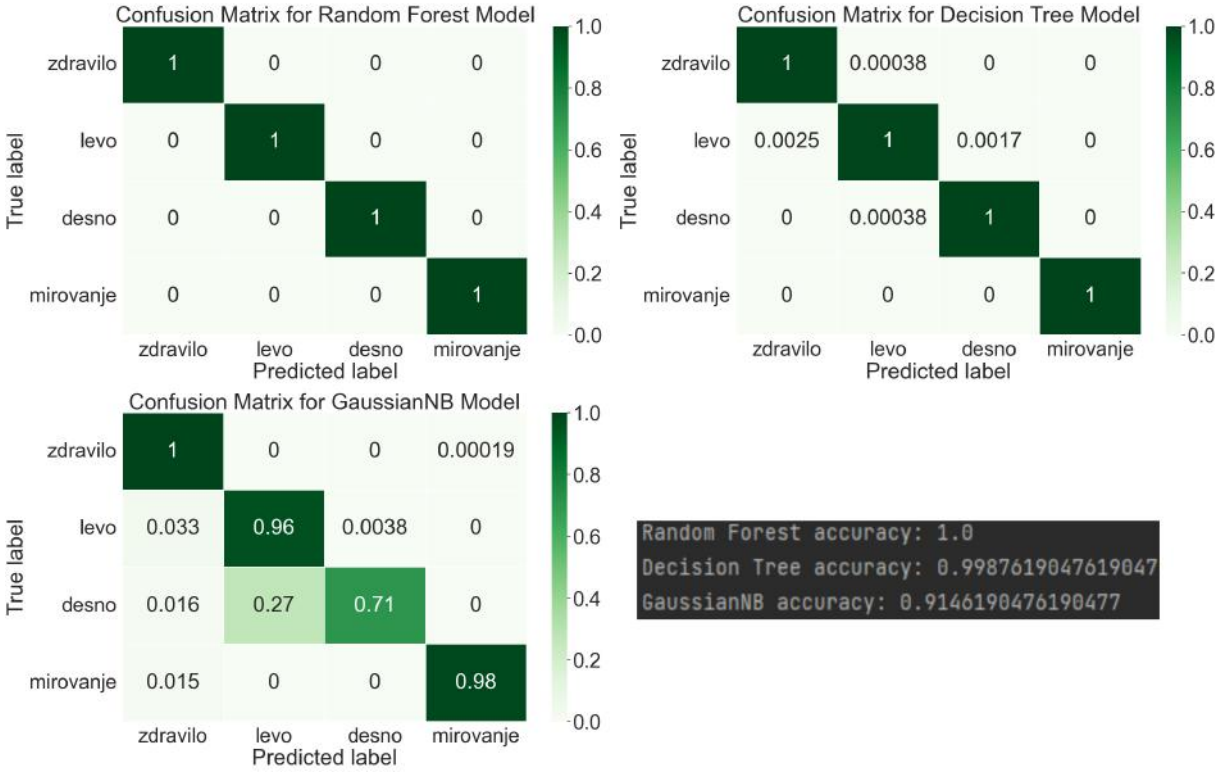
4 Testiranje in ugotovitve

Po končanem vezju sem na novo vpisal podatke v bazo za gib jemanje zdravila in mirovanje, za testiranje če naprava deluje tudi za druge gibe, pa sem dodal še gib v levo in gib v desno. Preden sem dodal ta dva giba je program skoraj vedno pravilno klasificiral gib jemanje zdravila, ko pa sem dodal še ta dva giba sem opazil da velikokrat, ko sem naredil gib jemanja zdravila je napačno klasificiral, kot da sem naredil gib v levo. Enako se je dogajalo, ko sem naredil gib v desno, kjer je tudi dostikrat klasificiral gib v levo. V kodi sem nato opazil napako, kjer sem pri odštevanju od vrednosti iz senzorja z tabelo »baseline« odšteval z napačnimi vrednostmi. To sem popravil in ponovno vnesel nove podatke v baze. Z novimi podatki je bilo veliko manj napak, včasih namesto giba v desno, ne prepozna nobenega giba za katere sem vnesel podatke in včasih namesto giba jemanje zdravila prepozna gib v levo, ampak se to zgodi malokrat.

Poskusil sem tudi namesto klasifikatorja Naključni gozd uporabiti klasifikator Odločitveno drevo in GaussianNB. Odločitveno drevo je delovalo približno enako dobro kot Naključni gozd, GaussianNB pa nekoliko slabše. Velikokrat je ko sem naredil

gib v levo prepoznal gib v desno in obratno, in ko sem naredil nek gib za katerega nisem vnesel podatke npr. premik roke navzdol, prepozna kot da je bil premik v desno, medtem ko pri Naključnem gozdu in Odločitvenem drevesu prepozna »gib mirovanje« in izpiše da ni bilo prepoznanega nobenega giba.

Nato sem še programsko preveril natančnost klasifikacije. Tukaj sem uporabil dve funkciji »accuracy_score« in »confusion_matrix« iz knjižnice scikit-learn. Najprej sem za testiranje uporabil samo podatke, ki sem jih vnesel v bazo in ji razdelil na polovico. Polovico podatkov za treniranje modela in polovico za testiranje, 15 meritev enega giba za treniranje in 15 za testiranje. Dobil sem naslednje rezultate.



Slika 13: Test natančnosti in matrika zamenjav

Točnost Naključnega gozda in Odločitvenega drevesa sta bila zelo visoka (100%), medtem ko je bila točnost GaussianNB okrog 91%. Kot se lahko vidi na matriki zamenjav (ang. confusion matrix) je imel GaussianNB največ težav z klasifikacijo giba v desno.

Ker so se mi te številke zdele previsoke sem v bazo dodal še 30 zapisov za vsak gib, pri katerih sem poskušal dodati tudi manj idealne gibe, ker sem pri prvem vpisovanju v bazo poskušal dodati čim bolj idealne gibe. Ko sem dodal še teh 30 zapisov za vsak gib sem jih združil z ostalimi podatki in ponovno zagnal program.

Slika 14: Test natančnosti in matrika zamenjav 2

Vrednosti so bile malo slabše kot prvič, Naključni gozd in Odločitveno drevo sta bila ponovno približno enaka (100%), GaussianNB pa ponovno najslabši ampak približno enako kot prej (91%). Tako kot prej je imel največ težav z klasifikacijo giba v desno.

5 Zaključek

Že na začetku izdelovanja raziskovalne naloge sem predvideval, da klasifikacija giba ne bo 100% natančna. Glede na testiranje, fizično in programsko, se mi zdi da naprava dosti dobro prepoznava različne gibe. Ocena natančnosti pri programskem testiranju se mi sicer zdi previsoka, ampak je to ocena natančnosti samo za podatke, ki so v bazi, nek gib pa lahko naredimo na različne načine in samo majhno spremembo v gibu lahko naprava klasificira čisto drugače, kot dejansko je. Natančnost naprave glede na fizično testiranje, bi za vse tri klasifikatorje, ki sem jih testiral ocenil približno 5% nižje od programske ocene, se pravi okrog 95% za klasifikator Naključni gozd. Pri nalogi sem potrdil obe hipotezi:

- Potrdil sem da je možna implementacija klasifikatorja na nizkocenovnem mikrokrmilniku za klasifikacijo na podlagi senzorskih podatkov v realnem času. Uporabil sem nizkocenovni mikrokrmilnik ESP32, ki ne presega 15€ in klasifikacija poteka v realnem času. Klasifikacija ni 100% natančna, saj kot je zgoraj navedeno lahko nek gib naredimo na neskončno različnih načinov in se lahko zgodi, da bo klasifikacija napačna.
- Potrdil sem tudi da bo naprava lahko klasificirala več kot en gib, s tem da sem dodal še gib roke v levo in desno.

Med izdelovanjem naloge sem na največ težav naletel pri izdelavi končnega vezja. Prva težava je bila napajanje, najprej sem uporabil gumbne baterije, ker so majhne in bi jih lahko spravil v ohišje, ampak so se kar hitro praznile. Po krajšem iskanju po internetu sem našel 1/2AAA litijevo baterijo s kapaciteto 1200mAh, jo testiral koliko časa bi zdržala in sem bil s časom zadovoljen. Z izbiro te baterijo pa se je pojavila nova težava, in sicer velikost ohišja, ker je ta baterija zanj prevelika, zato sem moral uporabiti malo večje ohišje. Z ostalimi stvarmi nisem imel večjih težav.

Naprava opozori samo uporabnika da je zdravilo vzel, v nadaljevanju pa bi lahko to napravo izpopolnil tako, da bi osebo, ki ga uporablja tudi opozorila kdaj mora oziroma ob kateri uri mora zdravilo vzeti, prav tako bi lahko oseba, ki skrbi za bolnika oziroma starostnika prejemala informacijo, o tem ali je ta oseba vzela zdravilo.

6 Viri

[1] Gesture Detection Using Machine Learning (ESP8266), dostopno na: https://www.hackster.io/Neutrino-1/gesture-detection-using-machine-learning-esp8266-a00716, 22.3.2022

[2] Interface MPU6050 Accelerometer and Gyroscope Sensor with Arduino, dostopno na: https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/, 22.3.2002

[3] Guide for I2C OLED Display with Arduino, dostopno na: https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/, 22.3.2022

[4] Surbhi Arora: Decision Tree vs Random Forest in Machine Learning, dostopno na: https://www.aitude.com/decision-tree-vs-random-forest-in-machine-learning, 22.3.2022

[5] Anja Zupančič: Odločitvena drevesa, dostopno na: https://repozitorij.uni-lj.si/Dokument.php?id=105660&lang=slv, 22.3.2022

[6] Tony Yiu: Understanding Random Forest, dostopno na: https://towardsdatascience.com/understanding-random-forest-58381e0602d2, 22.3.2022

[7] Matjaž Balon: Ekstremno naključni kvantilni gozdovi, dostopno na: https://core.ac.uk/download/pdf/77923336.pdf, 22.3.2022

[8] Naivni Bayesov klasifikator, dostopno na: http://kt.ijs.si/PetraKralj/UNGKnowledgeDiscovery/Bayes.pdf, 22.3.2022

[9] Vihar Kurama: Introduction to Naive Bayes: A Probability-Based Classification Algorithm, dostopno na: https://blog.paperspace.com/introduction-to-naive-bayes/, 22.3.2022

[10] Rohan Vats: Gaussian Naive Bayes: What you Need ti Know?, dostopno na: https://www.upgrad.com/blog/gaussian-naive-bayes/, 22.3.2022