Sustav za upravljanje zvukom i njegovim svojstvima s pomoću pokreta

Palčić, Martin

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 412

SUSTAV ZA UPRAVLJANJE ZVUKOM I NJEGOVIM SVOJSTVIMA S POMOĆU POKRETA

Martin Palčić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 412

SUSTAV ZA UPRAVLJANJE ZVUKOM I NJEGOVIM SVOJSTVIMA S POMOĆU POKRETA

Martin Palčić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 412

Pristupnik:	**Martin Palčić (0119042614)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentor:	prof. dr. sc. Davor Petrinović

Zadatak: **Sustav za upravljanje zvukom i njegovim svojstvima s pomoću pokreta**

Opis zadatka:

U sklopu diplomskog rada potrebno je osmisliti i realizirati sustav s pomoću kojeg je moguće parametrima ljudskog pokreta upravljati digitalnim generatorom zvuka, tj. karakteristikama generiranog zvuka, kao što su osnovna frekvencija i boja zvuka, njegova amplituda i pripadajuća vremenska ovojnica, karakteristika filtra kojim se signal uobličuje, odnosno upravljati frekvencijom niskofrekvencijskog generatora kojim se provode raznovrsne modulacije tog zvuka. Provesti analizu i odabir pogodnog sintetizatora zvuka kojeg je moguće upravljati s vanjskim digitalnim signalima. U svrhu mjerenja signala pokreta potrebno je odabrati i specificirati odgovarajuće senzore poput akcelerometra, žiroskopa i slično te projektirati i realizirati odgovarajući ugradbeni računalni sustav koji će provoditi mjerenja pokreta s navedenih senzora u stvarnom vremenu. Izlazne izmjerene podatke s ovog ugradbenog sustava potrebno je obraditi na odgovarajući način da budu pogodni za direktno upravljanje sustavom za sintezu zvuka. Demonstrirati rad razvijenog sustava u stvarnoj primjeni upravljanja zvukom.

Rok za predaju rada: 28. lipnja 2024.

Hvala obitelji na podršci i motivaciji.

Sadržaj

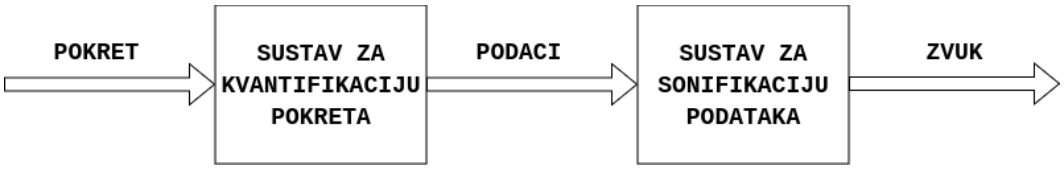
1. Uvod

Cilj ovog rada je prikazati ostvarenje sustava za upravljanjem zvukom i njegovim svojstvima s pomoću pokreta, te objasniti matematičku, programsku i hardversku pozadinu tog ostvarenja. Sustav koji je ostvaren sastoji se od mikrokontrolerskog sustava s pripadajućim senzorima i programskom podrškom te programskog sustava koji signale s mikrokontrolera na konfigurabilan način pretvara u signale prikladne za sonifikaciju softverskim alatima, ili u MIDI poruke pomoću kojih sustav može kontrolirati širok spektar softvera ili fizičkih uređaja. Ovaj se sustav, uparen sa nekom metodom generacije zvuka npr. alatom Supercollider ili nekim MIDI-sposobnim sintetizatorom zvuka, može se smatrati elektroničkim glazbenim instrumentom pogonjenim pokretom.

Prvi takav instrument bio je theremin. Razvio ga je sovjetski inženjer Leon Theremin u dvadesetim godinama prošlog stoljeća. Theremin je radio pomoću dvije antene koje su u kombinaciji s rukama svirača sačinjavale kondenzator čiji je kapacitet ovisio o udaljenosti ruke od antene. Jedna ruka kontrolirala je frekvenciju sinusnog oscilatora koji je proizvodio izlazni signal, a druga njegovu amplitudu.

Slika 1.1. Theremin i theremin

Ostvarenje theremina temelji se na istom principu kao i sustav obrađen u ovom radu. Taj princip je slijedeći: fizičkim se mjerenjem kvantificira ljudski pokret pa se vrijednosti dobivene tom kvantifikacijom sonificiraju pomoću nekog zasebnog sustava. Za razliku od tradicionalnih glazbenih instrumenata, čin sviranja i generacija zvuka odvojeni su. Umijeće i vještina sviranja sadržani su u ljudskom pokretu, ali zvuk koji sustav proizvodi ovisi o njegovoj implementaciji, bila ona jednostavni sinusni oscilator, ili moćan programski alat.



Slika 1.2. Model električnog instrumenta pogonjenog pokretom

Danas je na tržištu dostupno par varijanti sustava koji su sposobni upravljati zvukom pomoću pokreta. Prva varijanta radi na bazi računalnog vida. Primjer takvog sustava je Leap Motion Controller [1], kojeg je razvila tvrtka Ultraleap. Pomoću infracrvenih kamera, kontroler snima ruke korisnika i koristeći algoritme računalnog vida sposoban je prepoznati poziciju ruka i neke geste koje ruke rade. Ove informacije mogu se pomoću aplikacija treće strane, npr. MidiPaw, pretvoriti u MIDI signale, kojima se može kontrolirati zvuk. Problem s ovom varijantom sustava je to što ograničava korištenje samo na situacije gdje se ruke nalaze u vidnom polju kamere.

Slika 1.3. Leap motion controller

Druga varijanta radi na bazi inercijskih senzora. Primjer takvog sustava su MiMU-Gloves rukavice[2]. One se sastoje od senzora savijanja i inercijalne mjerne jedinice čija kombinacija daje informacije o položaju zgloba i savijenosti prstiju. Rukavice se koriste u kombinaciji s programskom podrškom Glover iste firme te pretvaraju mjerenja s rukavica u zvuk ili MIDI/OSC signal. Prednost ovakvog sustava je u tome da je sadržan na

samoj rukavici, a time neovisan o položaju i orijentaciji izvođača ili o osvjetljenju. Mana konkretnog sustava MiMUGloves je u njegovoj visokoj cijeni.

Slika 1.4. MiMUGloves rukavice

Sustav opisan u ovom radu može se svrstati u drugu skupinu. On koristi inercijsku mjernu jedinicu, senzor savijanja, akcelerometre i gumb. Informacija o poziciji ruke dobiva se obradom senzorskih informacija iz inercijske mjerne jedinice pomoću proširenog Kalmanovog filtra[3], a ostali senzori služe za dobivanje drugih korisnih senzorskih podataka.

2. Elementi sustava

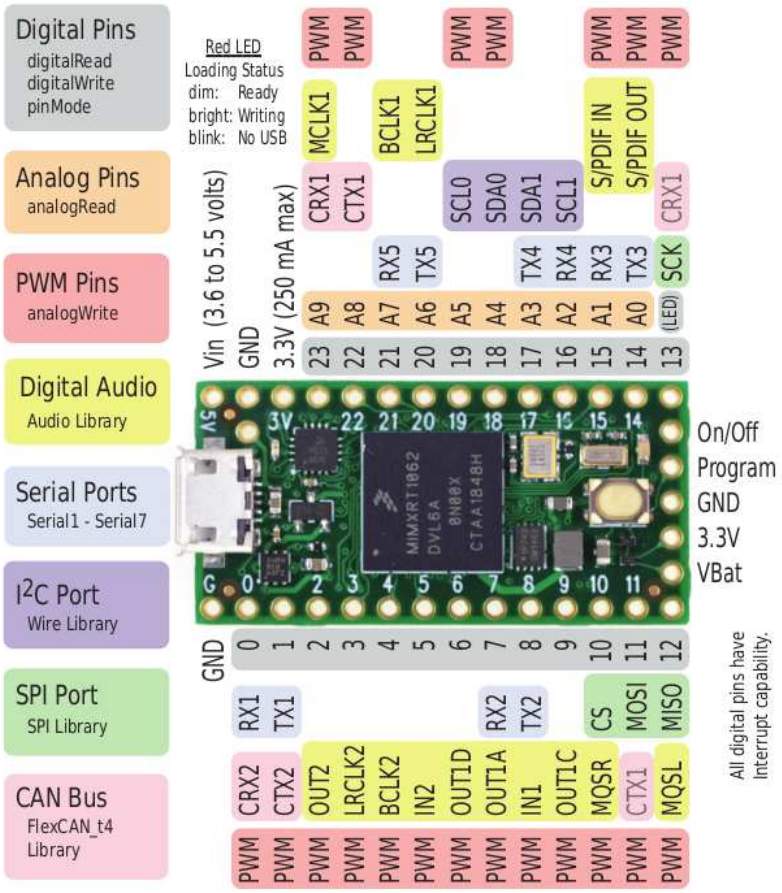
U ovom poglavlju dati će se pregled teorijskih, hardverskih i softverskih elemenata sustava.

2.1. Teensy 4.0 Development Board

Teensy USB Development Board[4] je obitelj kompletnih mikrokontrolerskih razvojnih sustava malog otiska, na kojima je moguće implementirati razne tipove projekata. Razvija i prodaje ih PJRC.com, LLC.,bazirani u saveznoj državi Oregon u SAD-u. Svo programiranje sustava vrši se preko njihovog USB sučelja. Teensy 4.0 i 4.1 su najmoćnije inačice, napravljene oko potentnog ARM Cortex-M7 procesora sa maksimalnom brzinom sata od 600 MHz. Teensy 4.1 ima ethernet sučelje, veću flash memoriju i veću količinu ulazno/izlaznih pinova, no pri biranju mikorokontrolerskog sustava za ovaj projekt, odlučeno je da je ono što nudi Teensy 4.0, u kombinaciji s činjenicom da je fizički manji, i više nego dovoljno.

Osim već navedenog moćnog procesora, Teensy 4.0 ima jedinicu za aritmetiku brojeva s pomičnim zarezom, 1024K RAM-a, 3 I2C i 3 SPI portova za komunikaciju sa perifernim uređajima poput senzora i drugo[4]. Pri originalnom planiranju projekta u obzir se uzelo i to da Teensy 4.0 ima i 2 I2S/TDM porta i 1 S/PDIF digitalni audio port, no oni se na kraju nisu koristili u finalnom projektu.

Što se tiče programske podrške za Teensy, bitno je naglasiti da su svi Teensy sustavi kompatibilni sa Arduino programskim bibliotekama i razvojnim alatnim lancem. Osim toga postoje i mnoge programske biblioteke napisane specifično za Teensy sustave. Te biblioteke uključuju i močnu audio biblioteku s pripadnim grafičkim alatom za razvoj, usporedivim s alatima Pure Data[5] ili MAX/MSP[6], ali i biblioteku za rad s MIDI stan-



Slika 2.1. Teensy 4.0 i njegovi priključci

dardom. Iako se te biblioteke nisu koristile u konačnom projektu, pokazuju da se programska podrška za Teensy sustave razvijala sa projektima vezanima uz zvuk na umu. Također je bitno naglasiti da je podršku pri razvoju za Teensy sustave moguće pronaći na web stranicama proizvođača, uključujući forum u kojem na pitanja odgovara i sam dizajner sustava Paul Stoffregen.

2.2. MPU6050 inercijska mjerna jedinica

MPU6050 inercijska mjerna jedinica tvrtke InvenSense je uređaj koji spaja akcelerometar i žiroskop u senzorski sklop koji omogućava mjerenje pokreta u 6 stupnjeva slobode. Za komunikaciju koristi I^2C. Namjenjena je za korištenje u kontekstima koji zahtjevaju nisku cijenu, nisku potrošnju, ali relativno visoke performanse, primjerice kod pametnih telefona, tableta i nosivih senzora. Za ovaj projekt je odabrana zbog velike dostupnosti dokumentacije i uputa za projekte koji uključuju Arduino ili Teensy sustave. Iako je za MPU6050 proglašen end-of-life i zamjenjen je već s više generacija novijih jedinica, on

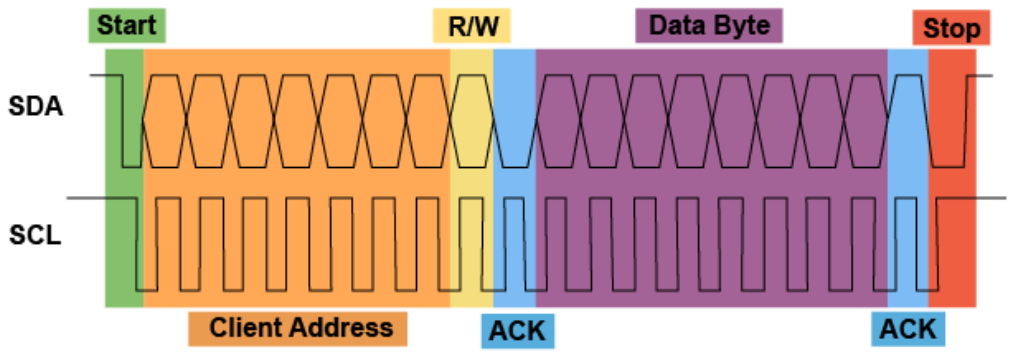
se i dalje može pronaći na internetu zbog ogromnog volumena originalne proizvodnje.

2.3. ADXL345 akcelerometar

ADXL345 akcelerometar je 3-osni akcelerometar malih dimenzija kojeg je razvila tvrtka Analog Devices. Za komunikaciju koristi I^2C ili SPI. Za ovaj projekt je, poput MPU6050, odabran zbog velike dostupnosti dokumentacije i uputa za projekte koji uključuju Arduino ili Teensy.

2.4. I^2C

I^2C (engl. Inter-Integrated Circuit) je sinkrona, multi-kontroler/multi-target, serijska komunikacijska sabirnica s jednim krajem koju je 1982. izumio Philips Semiconductors. Ima široku primjenu u spajanje perifernih integriranih sklopova manje brzine na procesore i mikrokontrolere u komunikaciji unutar istog sklopa na kratkim udaljenostima.

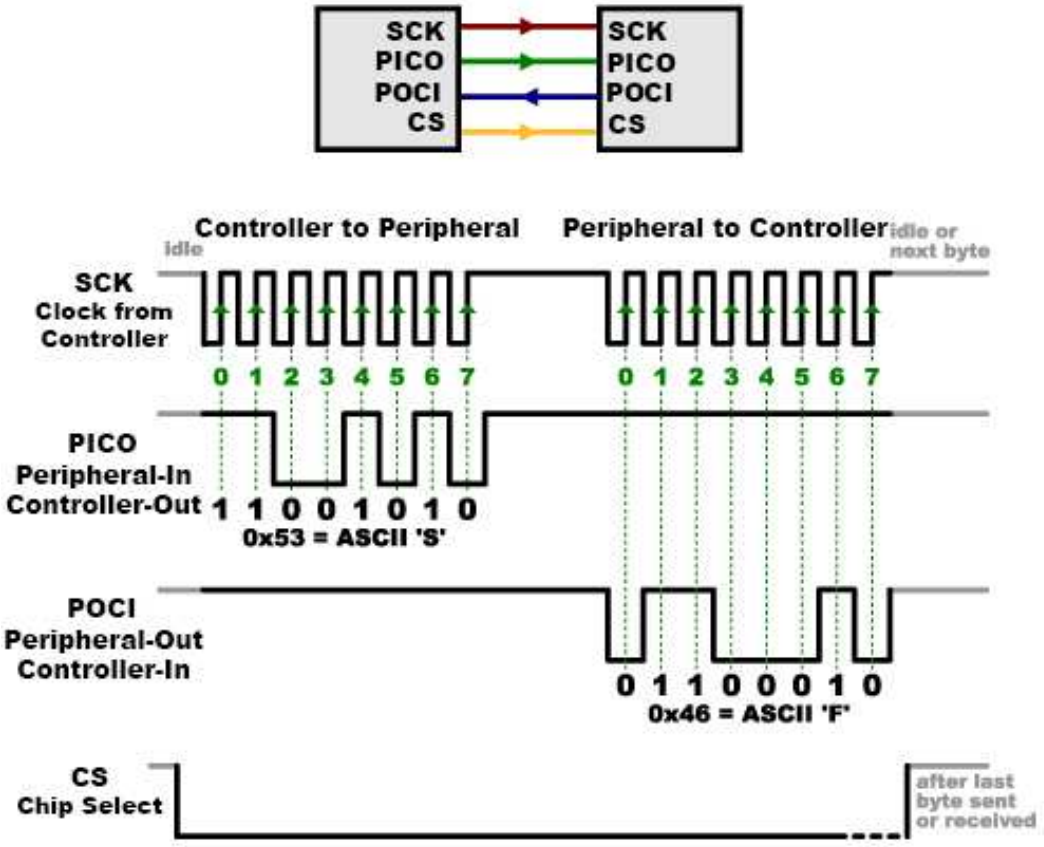


Slika 2.2. I^2C protokol

2.5. SPI

SPI (engl. Serial Peripheral Interface) smatra se industrijskim standardom u za sinkronu serijsku komunikaciju kratkog dometa, primarno u ugradbenim računalnim sustavima.

Originalna Motorolina specifikacija uključuje 4-žični prijenos za *full duplex* komunikaciju, ali postoje i 3-žične verzije za 2 *half duplex*.



Slika 2.3. SPI protokol

2.6. Arduino IDE i Teensy Loader

Za izgradnju riješenja i njegovo spuštanje na Teensy sustav koristio se Arduino razvojni alatni lanac integriran u Arduino IDE (engl. integrated development environment) u kombinaciji sa programom Teensy Loader. Arduino IDE je softver otvorenog koda, koji je razvila tvrtka Arduino za rad s njihovom serijom mikrokontrolera. On pruža funkcionalnosti uređivača teksta, *debugger*-a, upravitelja razvojnih pločica i drugo. Također, on dopušta da se u njega kao *plugin* doda softver za baratanje ne-Arduino sustavima, poput Teensy-ja. Time je omogućeno to da se pokretanjem naredbe za izgradnju i *flash*-anje kad završi prevođenje automatski pokrene program Teensy Loader koji služi za spuštanje kreiranih .hex datoteka na sam sustav. Treba uzeti u obzir, međutim, da ako se koriste ne-Arduino sustavi, ne može se koristiti funkcionalnost *debugger*-a integriranog u Arduino IDE.

Velika prednost korištenja Arduino IDE-a je u tome da olakšava pregled serijske komunikacije između pločice i računala korištenog za razvoj. Osim tipičnog tekstualnog monitora serijske komunikacije, Arduino IDE pruža i automatsko grafiranje varijabli, ukoliko su one ispisivane u razumno formatiranom obliku. Iako se Arduino IDE nije koristio kao glavni uređivač teksta za ovaj projekt, funkcionalnost ispisa i grafiranja podataka sa serijskog porta pokazala se veoma vrijednom.

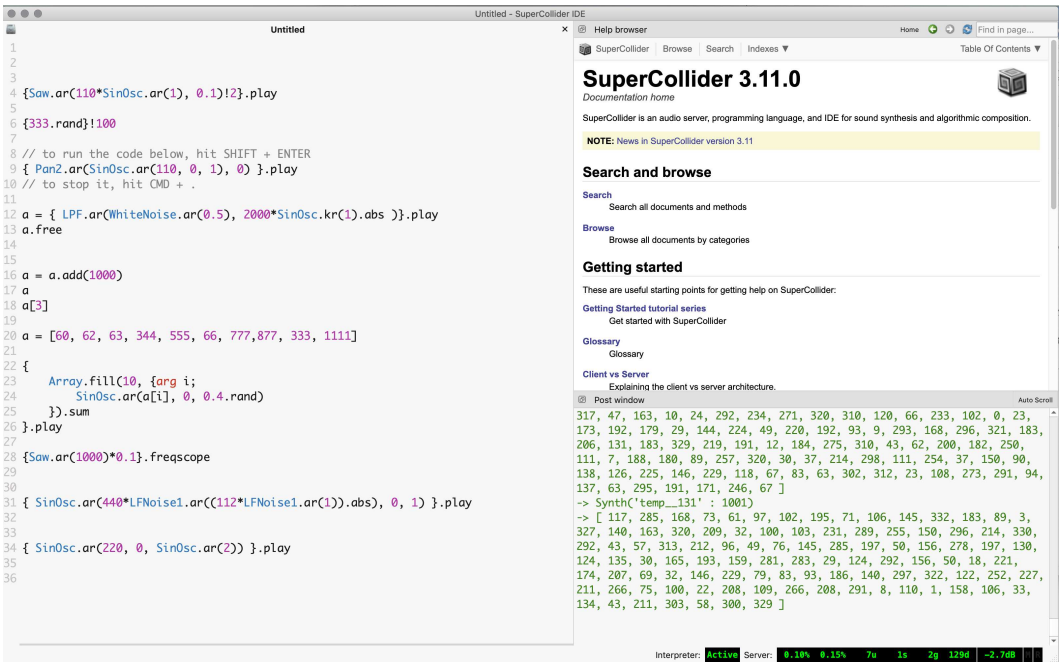
2.7. Supercollider

Supercollider[7] je okruženje i programski jezik dizajniran za audio sintezu i algoritamsku kompoziciju. Osim primjena u umjetnosti, koristi se i kao programski okvir za istraživanja u polju akustike. Supercollider je softver otvorena koda kojeg je 1996. razvio James McCartney. Arhitektura Supercollider-a temelji se na podjeli na server, *scsynth* i klijent, *sclang*. Ove dvije komponente komuniciraju pomoću OSC[8] protokola.

Scsynth je audio server zaslužan za svu generaciju zvuka u okruženju Supercollider. Kontrolira ga se poglavito OSC protokolom kroz Supercollider programski jezik, ali ga se može pokretati i kao program naredbenog retka. Podržava implementaciju korisničkih *plugin*-ova putem C i C++ API-ja, bilo koji broj ulaznih i izlaznih audio i kontrolnih kanala, čime omogučava masivno multikanalne konfiguracije i sustav virtualnih audio sabirnica koji dozvoljava dinamičnu promjenu toka signala.

Sclang je druga komponenta Supercollider okruženja. To je dinamički tipiziran, objektno orijentiran i funkcionalan programski jezik sa sintaksom sličnom C-u. Programski jezik i pripadni interpretator čine arhitekturu za kreiranje i brisanje objekata na *scsynth* serveru i njihovu manipulaciju na razne načine. Jezik je dizajniran da potiče eksperimentaciju. To postiže fleksibilnosti svoje sintakse, omogučavajući apstraktno povezivanje funkcionalnih cjelina u blokove koda koji, ako su uspješno interpretirani, generiraju zvuk kao povratnu informaciju. Iz toga se rađa tipični ciklus razvoja nekog Supercollider *patch*-a: pisanje koda, slušanje rezultata, prilagodba koda.

Supercollider IDE je razvojno okruženje dizajnirano za rad sa Supercollider-om. Ono uključuje uređivač teksta, prozor za povratnu informaciju, npr. o greškama u kodu i drugo[7]. Također, IDE uključuje i veoma dobro promišljen sustav integrirane doku-



Slika 2.4. Supercollider IDE

mentacije sa primjerima koda koji se mogu interpretirati direktno u dokumentacijskom prozoru i rezultati te interpretacije slušati.

2.8. MIDI

MIDI[9] (engl. Musical Instrument Digital Interface) je tehnički standard koji opisuje komunikacijski protokol, digitalno sučelje i električne priključke sa sposobnosti povezivanja širokog spektra elektroničkih glazbenih instrumenata, računalnog softvera, sustava za kontrolu rasvjeta i mnogih drugih uređaja. MIDI je razvijen početkom osamdesetih godina prošlog stoljeća zbog potrebe za digitalnim standardom kojim bi se osigurala interoperabilnost u komunikaciji između elektroničkih glazbenih instrumenata različitih proizvođača. Projekt je, na čelu s gigantom u svijetu elektroničke glazbene opreme - Roland-om, ali i mnogim drugim etabliranim imenima poput Yamaha, Korg, Kawai, Moog i dr., izrodio prvi MIDI standard. Iako je MIDI originalno namjenjen da se koristi pomoću 5-pinskih MIDI sučelja i pripadnih kablova, u devedesetim godinama prilagođen je za korištenje i sa novijim standardima povezivanja kao što su USB ili tada aktualan FireWire. Utjecaj MIDI standarda na glazbenu industriju i industriju elektroničkih glazbenih instrumenata bio je monumentalan, što potvrđuje to da ne samo da se on i dalje koristi preko 40 godina kasnije, nego je i standard industrije za kojeg se očekuje da će

svaki imalo ozbiljan proizvod imati implementaciju.

Dio MIDI standarda relevantan ovom projektu zasigurno je binarni komunikacijski protokol. Taj se protkol sastoji od asinkronog niza MIDI poruka prenesenih serijskom komunikacijom s *baud rate*-om od 31250. MIDI poruke su instrukcije koje kontroliraju neki aspekt ciljnog uređaja. Sastoje se od statusnog okteta, koji određuje tip poruke i zatim dva podatkovna okteta koji sadrže parametre poruke. Postoji pet vrsta MIDI poruka, a to su: *Channel Voice*, *Channel Mode*, *System Common*, *System Real-Time* i *System Exclusive*. Za ovaj projekt relevantno je razumijevanje samo prvog tipa - *Channel Voice* poruke.

Postoji šest tipova *Channel Voice* poruka. To su *Note On*, *Note Off*, *Control Change*, *Program Change*, *Aftertouch* i *Pitch Bend Change*. *Note On* i *Note Off* poruke služe, kao što se da pretpostaviti po njihovim nazivima, za pokretanje i zaustavljanje tonova odnosno glasova. *Note On* i *Note Off* poruke sastoje se od statusnih okteta `0x9N`, odnosno `0x8N`, gdje je `N` broj MIDI kanala[9] na koji se šalju, koje sljede 2 okteta koji definiraju broj MIDI note i *velocity*. MIDI note su numeričke vrijednosti u rasponu od 0 do 127 koje se pripisuju frekvencijama tonova u standardnoj 12-tonskoj ljestvici. Tako primjerice ton C4 ima ima MIDI vrijednost od 60. *Velocity* je vrijednost čija interpretacija je ostavljena ciljnom instrumentu, no uglavnom, što je veća vrijednost, to je neki efekt uzrokovan pokretanjem ili zaustavljanjem tona izraženiji. Treći tip *Channel Voice* poruke koji je relevantan za ovaj projekt je *Control Change* poruka. *Control Change* poruke su generične poruke koje se sastoje od kontrolnog broja i vrijednosti. Primajući MIDI uređaj može interpretirati *Control Change* poruke na razne, ponekad i konfigurabilne, načine. *Control Change* poruke sastoje se od statusnog okteta `0xBN`, gdje je `N` broj kanala, zatim okteta sa kontrolnim brojem pa okteta sa vrijednosti. Ostale poruke se ne koriste u sklopu ovog projekta.

2.9. Kalmanov filtar

Kalmanov filtar[3], također poznat kao i linearna kvadratna procjena (LKP), algoritam je koji koristi niz mjerenja koja sadrže statistički šum i druge nepreciznosti, a daje procjene nepoznatih varijabli koje su u prosjeku točnije od onih temeljenih samo na jednom mjerenju. Algoritam je dobio ime po mađarsko-američkom znanstveniku Rudolfu

E. Kálmánu. Kalmanov filtar ima brojne primjene u tehnologiji, ponajviše u robotici i automatici. U tom polju koristi se za vođenje, navigaciju i kontrolu vozila, uglavnom zrakoplova, dronova i svemirskih letjelica. Osim u tim poljima, Kalmanov filtar ima primjene i u obradi signala, ekonomiji i medicini. U sklopu ovog projekta Kalmanov filtar se koristi za fuziju senzora i dobivanje relativno preciznih i stabilnih pretpostavki o stavu sustava u tri dimenzije.

Algoritam radi u dva koraka. U koraku predviđanja Kalmanov filtar daje procjene varijabli trenutnog stanja, zajedno s njihovim nesigurnostima. Jednom kada se dobije ishod sljedećeg mjerenja (nužno opterećen s određenom količinom pogreške, uključujući slučajni šum), te se procjene ažuriraju, u drugom koraku algoritma, pomoću ponderiranog prosjeka, s tim da se procjenama s većom sigurnosti daje veća težina u sumi. Algoritam je rekurzivan. Može se pokretati u stvarnom vremenu, koristeći samo trenutna ulazna mjerenja i prethodno izračunato stanje i njegovu matricu nesigurnosti, nisu potrebne dodatne informacije o prošlim stanjima sustava.

Produženi Kalmanov filtar, korišten u ovom projektu, modifikacija je Kalmanovog filtra za rad s nelinearnim sustavima.

3. Tok projekta

U ovom poglavlju prikazan je tok razvoja projekta i pojašnjene odluke u njegovoj implementaciji.

3.1. Inicijalno uspostavljanje komunikacije s Teensy-jem

Inicijalna predispozicija početka rada na projektu je uspostavljanje komunikacije sa Teensy razvojnim sustavom. Da bi se to postiglo, potrebno je pratiti upute dostupne na web sjedištu proizvođača[4]. Za korisnike operacijskog sustava Linux, za koji je rađen ovaj projekt, potrebno je kopirati *udev* pravila dostupna na web sjedištu proizvođača u `/etc/udev/rules.d/`. To će omogućiti Linux sustavu da prepozna Teensy sustav prilikom povezivanja na USB sučelje računala. To je moguće napraviti pomicanjem u direktorij gdje se nalaze preuzeta udev pravila i pokretanjem slijedeće naredbe:

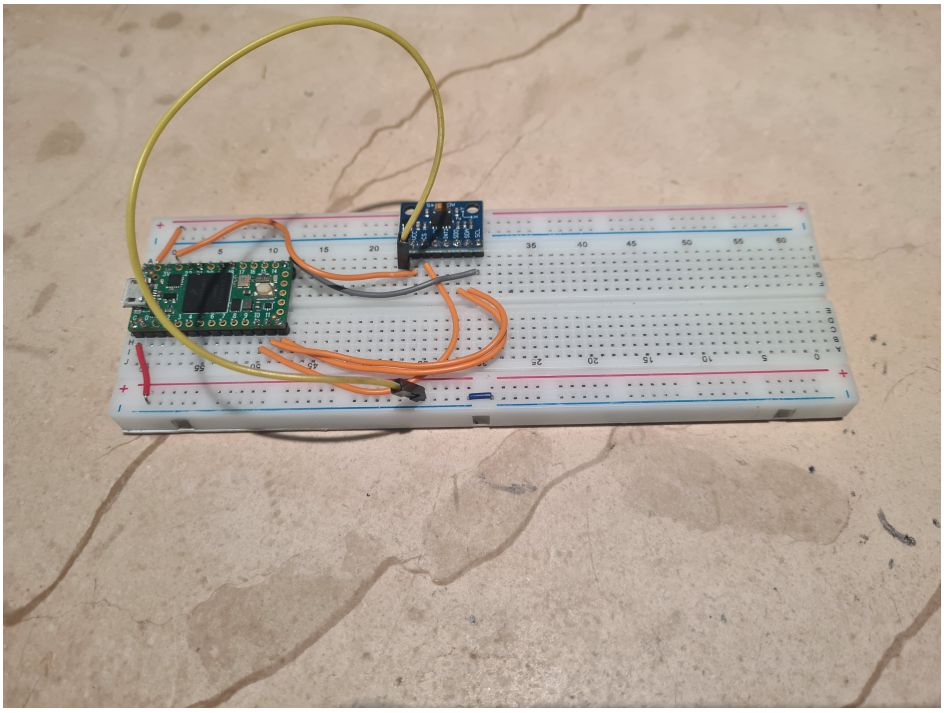
```
$ sudo cp 00-teensy.rules /etc/udev/rules.d/
```

Idući korak je preuzeti najnoviju verziju Arduino IDE-a. Ona je za korisnike Linux distribucija baziranih na distribuciji Arch dostupna na AUR repozitoriju i može se instalirati pokretanjem naredbe:

```
$ yay -S arduino-ide-bin
```

Ukoliko se koristi druga distribucija ili upravljač korisničkim repozitorijem (npr. trizen), Arduino IDE može se dohvatiti praćenjem uputa na stranicama proizvođača [4]. Jednom kad se u Arduino IDE instalira *board support package* za Teensy proizvode, Arduino IDE bi trebao automatski prepoznati Teensy uređaj spojen na računalo. Ukoliko se to ne događa, moguće je da je USB kabel koji se koristi samo strujni, a potreban je podatkovni. Kad je Teensy uspješno povezan sa Arduino IDE-om može se napraviti jednostavni test

ispisa na serijsko sučelje da se potvrdi funkcioniranje sustava.



Slika 3.1. Prva iteracija sustava

Ultimativno, sustav će izgledati kao rukavica sa Teensy-jem i senzorima integriranima na njoj, ali dok je sustav još u razvoju koristit će se eksperimentalna pločica, odnosno *breadboard*. Prva iteracija sustava, prikazana na slici 3.1., uključuje Teensy pločicu povezanu SPI sučeljem na ADXL345 akcelerometar. Bitno je primjetiti da je potrebno sve pinove koji se koriste zalemiti na njihova sjedišta, jer čak i ako se čine kao da između njih postoji kontakt, dobra je šansa da ne postoji. U dokumentaciji za senzor ADXL345 moguće je naći mapu njegovih internih registara, koji služe za konfiguraciju sklopa i koji se, radi jednostavnosti i jasnoće kasnije implementacije mogu prenjeti u programski kod pomoću direktiva `define`. Nakon toga, čitanje s akcelerometra je jednostavno implementirati kodom u C-u, uz korištenje jedne C++ funkcionalnosti, a to je tip `bool`, koristeći pritom dostupnu SPI.h biblioteku.

```
1 // Add the SPI library so we can communicate with the ADXL345
      sensor
2 #include <SPI.h>
3 #include <stdint.h>
4 // Assign the Chip Select signal to pin 10.
5 int CS = 10;
6
7 // ADXL345 Register Addresses
8 ...
9 #define POWER_CTL 0x2D // Power Control Register
```

```
#define INT_ENABLE 0x2E // Interrupt Enable Control
#define INT_SOURCE 0x30 // Source of interrupts
#define DATA_FORMAT 0x31 // Data format control
#define DATAX0 0x32 // X-Axis Data 0
#define DATAX1 0x33 // X-Axis Data 1
...

uint8_t values[10];
int ax, ay, az;
float axg, ayg, azg;

void adxl_writeRegister(char registerAddress, unsigned char
   value)
{
   // Set Chip Select pin low to signal the beginning of an
      SPI packet.
   digitalWrite(CS, LOW);
   // Transfer the register address over SPI.
   SPI.transfer(registerAddress);
   // Transfer the desired register value over SPI.
   SPI.transfer(value);
   // Set the Chip Select pin high to signal the end of an
      SPI packet.
   digitalWrite(CS, HIGH);
}

void adxl_readRegister(char registerAddress, int numBytes,
   unsigned char* values)
{
   // Since we're performing a read operation, the most
      significant bit of the register address should be set.
   char address = 0x80 | registerAddress;
   // If we're doing a multi-byte read, bit 6 needs to be set
       as well.
   if (numBytes > 1)
      address = address | 0x40;

   // Set the Chip select pin low to start an SPI packet.
   digitalWrite(CS, LOW);
   // Transfer the starting register address that needs to be
       read.
   SPI.transfer(address);
   // Continue to read registers until we've read the number
      specified, storing the results to the input buffer.
   for (int i = 0; i < numBytes; i++)
   {
      values[i] = SPI.transfer(0x00);
   }
   // Set the Chips Select pin high to end the SPI packet.
   digitalWrite(CS, HIGH);
}

int16_t adxl_toDecimal(uint16_t x)
```

```
{
   bool negative = (x & (1 << 9)) != 0;
   if (negative)
      return x | ~((1 << 10) - 1);
   return (int16_t) x;
}

void setup()
{
   Serial.begin(115200); // Create a serial connection to
      display the data on the terminal.
   SPI.begin();      // Initiate an SPI communication instance
      .

   // Configure the SPI connection for the ADXL345.
   SPI.setDataMode(SPI_MODE3);
   SPI.beginTransaction(SPISettings(4000000, MSBFIRST,
      SPI_MODE3));

   pinMode(CS, OUTPUT); // Set up the Chip Select pin to be
      an output from the Arduino.
   digitalWrite(CS, HIGH); // Before communication starts,
      the Chip Select pin needs to be set high.

   writeRegister(DATA_FORMAT, 0x00); // Put the ADXL345 into
      +/- 2G range
   writeRegister(INT_ENABLE, 0x00); // disable interrupts
   writeRegister(POWER_CTL, 0x08); // Measurement mode
   readRegister(INT_SOURCE, 1, values); // Clear the
      interrupts from the INT_SOURCE register.
}

void loop()
{
   for (;;)
   {
      readRegister(DATAX0, 6, values); // Read acceleration
         values into the values buffer

      // Combine bytes to get 10-bit acceleration values
      ax = adxl_toDecimal((((uint16_t) values[1] << 8) | (
         uint16_t) values[0]) & 1023);
      ay = adxl_toDecimal((((uint16_t) values[3] << 8) | (
         uint16_t) values[2]) & 1023);
      az = adxl_toDecimal((((uint16_t) values[5] << 8) | (
         uint16_t) values[4]) & 1023);

      // Convert accelerometer values to G's
      axg = ax * 0.00390625;
      ayg = ay * 0.00390625;
      azg = az * 0.00390625;

      Serial.print(axg);
```

```
97        Serial.print(", ");
98        Serial.print(ayg);
99        Serial.print(", ");
100       Serial.println(ayg);
101
102       delay(100);
103    }
104 }
```

Kod za rad sa ADXL345 akcelerometrom baziran je na kodu github korisnika *hamaluik*[10].

3.2. Čitanje sa serijskog porta u Supercollider

Nakon uspješne inicijalizacije razvojnog sustava i uspostavljene komunikacije sa senzorom ADXL345, idući korak je pokušati razviti jednostavnu komunikaciju između sustava i okruženja Supercollider. Budući da je trenutna ideja učitavati podatke u Supercollider direktno sa Teensy-ja, čitati će se sa serijskog sučelja. Informacije o postizanju takve komunikacije, kao i vodič kroz općenito korištenje Supercollider-a, mogu se naći u serijalu videozapisa na portalu YouTube koje kreira Eli Fieldsteel, profesor kompozicije i glazbene teorije na Sveučilištu u Chicagu [11]. Kod za postići jednostavnu komunikaciju između Teensy sustava i Supercollider okruženja:

```
1  SerialPort.devices;
2
3  ~port = SerialPort.new("/dev/ttyACM0", 115200);
4  (
5  ~charArray = [ ];
6  ~getValues = Routine.new({
7   var ascii;
8   var i = 2;
9   {
10    ascii = ~port.read.asAscii;
11    if(ascii != $,,
12      {
13        ~charArray = ~charArray.add(ascii);
14
15      },
16      {
17        i = (i + 1) % 3;
18        if(i == 0,
19          {
20            "X = ".postln;
21            ~valX = ~charArray.join.asFloat.postln;
22          };
```

```
      );
      if(i == 1,
        {
          "Y = ".postln;
          ~valY = ~charArray.join.asFloat.postln;
        };
      );
      if(i == 2,
        {
          "Z = ".postln;
          ~valZ = ~charArray.join.asFloat.postln;
        };
      );
      ~charArray = [ ];
    };
  );
}.loop;
}).play;
)

~port.close;

Routine.stop;
```

Ovaj isječak kreira *handler* za serijsko sučelje otvoreno na `/dev/ttyACM0` (lokacija gdje se automatski otvara serijsko sučelje na korištenom Linux sustavu), čita liniju po liniju poslanu na otvoreno sučelje, dijeli linije na elemente odvojene zarezima i zatim te elemente parsira kao realne brojeve i sprema u globalne varijable. Te se globalne varijable zatim mogu koristiti da aktivno mijenjaju parametre generatora zvuka, kao u idućem isječku.

```
(
SynthDef.new(\saw, {
  arg cutoff=1000, freq = 440;
  var sig;
  sig = Saw.ar([freq*1.05,freq]);
  sig = RLPF.ar(sig, cutoff.lag(0.02), 0.25, 0.2);
  Out.ar(0, sig);
}).add;
)

~synth = Synth(\saw, [\cutoff, 200]);

(
~control = Routine.new({
  {
    ~synth.set(\freq, ~valX.linexp(-512, 511, 80, 4000));
    ~synth.set(\cutoff, ~valZ.linexp(-512, 511, 30, 5000));

```

```
19     0.01.wait;
20   }.loop;
21 }).play;
22 )
23
24 ~synth.free;
```

Ovaj isječak kreira Synth objekt naziva `saw` i dodaje ga na *scsynth* audio server. Synth objekt ima deklarirane argumente `freq` i `cutoff`, koji kontroliraju frekvenciju oscilatora i graničnu frekvenciju niskopropusnog filtra. Zatim se kreira instanca tog objekta, `synth`, i rutina u kojoj se parametri `freq` i `cutoff` mapiraju na kontinuirano čitane vrijednosti `valX` i `valY`, koje predstavljaju akceleracije na X i Z osima senzora. Rezultat nije naročito zanimljiva zvuka, ali pruža obrazac za direktnu komunikaciju Teensy sustava i Supercollider okruženja.

3.3. Problem sa računanjem položaja kroz dvostruku integraciju akcelerometarskih podataka

Da bi sustav mogao biti korišten kao relativno predvidivi glazbeni instrument potrebno je u njega dodati funkcionalnost koja bi omogućavala neki stabilni indikator položaja, kojim bi se mogli kontrolirati aspekti zvuka za koje je bitno da mogu biti stabilni, primjerice frekvencija tona i njegova glasnoća. Prvi pokušaj ostvarivanja takve funkcionalnosti bio je kroz dvostruku integraciju akcelerometarskih podataka da se dobije položaj akcelerometra. To je ostvareno numeričkom integracijom metodom kumulativne sume trapezoida kao što je implementirano u slijedećem isječku.

```
1 ...
2 #define USECONDS_IN_SECOND 1000000
3 #define A_SAMPLERATE 1000 // approximately 1000 accelerometer
      measurements per second
4 ...
5
6 int velocity_from_a_buffer(float *aBuf, int bufsize, float *
     v_new, float *v_prev)
7 {
8    *v_new = *v_prev;
9
10   for (int i = 0; i < bufsize; ++i)
11   {
12      *v_new += aBuf[i] * A_SAMPLEPERIOD; // gives velocity
           in m/s
```

```
13    }
14
15    *v_prev = *v_new;
16
17    return 0;
18 }
19
20 int position_from_velocity(float *v_new, float *v_prev, float
       *p_new, float *p_prev)
21 {
22    *p_new = *p_prev + *v_new * A_SAMPLEPERIOD * A_BUFSIZE;
23    *p_prev = *p_new;
24    return 0;
25 }
26
27 ...
28
29 void loop()
30 {
31
32    // read A_BUFSIZE measurements with a sample rate of
         A_SAMPLERATE
33    while (a_index < A_BUFSIZE)
34    {
35       readRegister(DATAX0, 6, values);
36       aBuf[a_index++] = adxl_toDecimal((((uint16_t) values[1]
             << 8) | (uint16_t) values[0]) & 1023) *
37                         0.03832f; // gives acceleration in m/s^2
38       delayMicroseconds(USECONDS_IN_SECOND /
39                         A_SAMPLERATE); // microseconds in second
                             / samplerate = sample period
40    }
41    a_index = 0;
42
43    velocity_from_a_buffer(aBuf, A_BUFSIZE, &f_vx, &f_vxp);
44    position_from_velocity(&f_vx, &f_vxp, &f_px, &f_pxp);
45    ...
46 }
```

Pokazalo se, međutim, da u ovom smjeru razmišljanja postoji veliki problem. ADXL345, kao i svaki relativno jeftini MEMS[12] senzor podložan je davati šumovita i neprecizna mjerenja. Te nepreciznosti same po sebi ne stvaraju veliki problem jer za većinu primjena, pa tako i primjenu u ovom projektu, apsolutna preciznost senzorskih mjerenja nije potrebna. Problem nastaje kad se šumovita mjerenja kontinuirano dvostruko integriraju. U tom slučaju greške u mjerenjima postaju kvadratne i zbog kontinuiranosti mjerenja stvara se kumulativni drift u vrijednostima. Primjerice, prilikom mjerenja pomaka u X osi koristeći metodu dvostruke integracije događa se situacija ilustrirana u slici

3.2. .

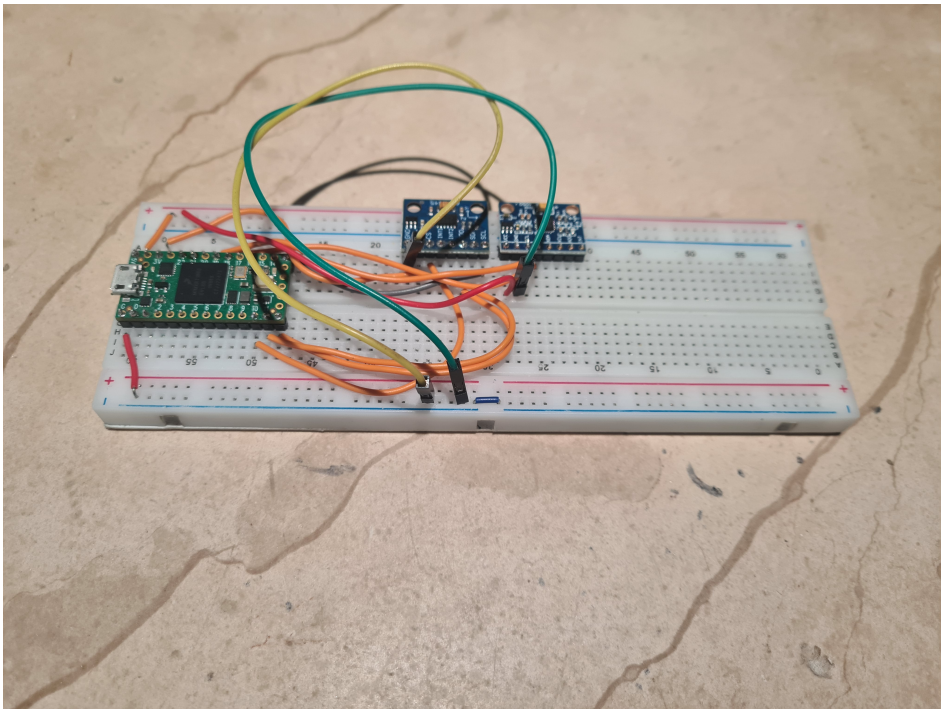
Slika 3.2. Drift uzrokovan dvostrukom integracijom

Pomak razvojnog sustava u smjeru x osi uzrokuje akceleraciju, na grafu `Value 1`, da se popne na neku pozitivnu vrijednost dok sustav ubrzava, zatim padne na negativnu vrijednost dok sustav usporava te se zatim vrati na nulu kad sustav stane. Brzina, `Value 2`, raste s rastom akceleracije pa zatim opada prema nuli kako se sustav zaustavlja. Problem je, naime, da brzina ne opadne potpuno na nulu, nego na neku vrijednost blizu nule, a to rezultira time da u položaju, `Value 3`, postoji konstantan drift zbog akumulacije greške u brzini. Korištenje preciznijih metoda integracije i uzorkovanja senzora bi donekle popravilo ovaj problem, ali on bi i dalje postojao. Pokazuje se da je ovo netrivijalni *open-loop* kontrolni sustav[13] i kao takav nema zadovoljavajuće rješenje. Čak i ako je korisnik sustava element koji zatvara otvorenu petlju, prilagođavajući svoje pokrete ponašanju sustava, rješenje je i dalje pre nestabilno za primjenu u ovom projektu. Potencijalna metoda kojom se i dalje može koristiti dvostruka integracija je tako da se uz nju koristi primjerice gumb, čije držanje aktivira integraciju vrijednosti, ali čim se on

otpusti, vrijednosti se vraćaju na inicijalne postavke. Na taj način mogu se dobiti kratkotrajni relativni pomaci, bez da se greška stigne jako akumulirati. Zbog ograničenosti i potencijalne nestabilnosti takvih rješenja, dvostruka integracija kao koncept se napustila za jezgru analize pokreta. Umjesto nje odabran je produženi Kalmanov filtar.

3.4. Implementacija produženog Kalmanovog filtra

Da bi se Kalmanov filtar mogao koristiti za dobivanje informacija o poziciji ruke, u mikrokontrolerski sustav treba se integrirati i žiroskop. Mjerenje akceleracije i kutne brzine za potrebe obrade Kalmanovim filtrom vršit će se pomoću MPU6050 inercijske mjerne jedinice. Iako su mjerenja akceleracije već dostupna sa ADXL345 akcelerometra, poželjno je dobivati mjerenja akceleracije i kutne brzine sa istog integriranog sklopa, djelomično zbog toga što je bitno da ta mjerenja dolaze iz što bliže točke u prostoru, a djelomićno zbog toga što se onda ADXL345 akcelerometar može koristiti za nešto drugo, negdje drugdje.



Slika 3.3. Druga iteracija sustava, ADXL345 i MPU6050

Kalmanov filtar je relativno apstraktan koncept neiniciiranima, pa ima smisla okoristiti se materijalima dostupnima na internetu koji olakšavaju njegovo razumijevanje i korištenje. Naime, postoje web-stranice stvorene specifično sa ciljem objašnjavanja Kalmanovog filtra[14] i bile su korisne u njegovoj demistifikaciji. Također, kao koris-

tan materijal, koji je služio kao baza za implementaciju produženog Kalmanovog filtra u ovom radu bio je videozapis[15] autora *Phil's Lab*, dostupan na portalu YouTube, u kojem autor implementira produženi Kalmanov filtar na sustavu baziranom na STM32 mikrokontroleru.

Dva ključna koraka Kalmanovog filtra, korak predviđanja i korak ažuriranja, implementirani su kao dvije funkcije - `EKF_Predict()` i `EKF_Update()`.

Funkcija `EKF_Predict()` poziva se svakih 10 milisekundi i ona na temelju mjerenja žiroskopa i prošlog stanja sustava daje predviđanja o poziciji sustava.

```

 /* Pre-compute trigonometric quantities */
  float sp = sinf(ekf->phi_r);
  float cp = cosf(ekf->phi_r);
  float tt = tanf(ekf->theta_r);

  /* Compute state transition function dx/dt = f(x,u) */
  float dphidt = p_rps + tt * (q_rps * sp + r_rps * cp);
  float dthetadt = q_rps * cp - r_rps * sp;

  /* Update state estimates (x(n+1) = x(n) + T * dx/dt) */
  ekf->phi_r += sampleTime_s * dphidt;
  ekf->theta_r += sampleTime_s * dthetadt;
```

Funkcija texttt`EKF_Update()` poziva se svakih 105 milisekundi i ona ažurira stanje sustava, uključujući i *Kalman gain*, težine u ponderiranom prosjeku.

```

/* Compute Kalman gain K = P * C' * (R + C * P * C ')^-1 in
   steps (note that C[0][0] = 0!) */

  /* P * C'*/
  float PCt[2][3] = . . .

  /* R + C * P * C' */
  float RCPCt[3][3] = . . .

  /* inv(R + C * P * C') */
  float detMatInv = 1.0f / . . .

  float matInv[3][3] = . . .

  for (int i = 0; i < 3; i++)
     for (int j = 0; j < 3; j++)
        matInv[i][j] *= detMatInv;

  /* C' * inv(R + C * P * C') */
```

```
20    float CtmatInv[2][3] = . . .
21
22    /* K = P * C' * inv(R + C * P * C') */
23    float K[2][3] = {{ekf->P[0][0] * CtmatInv[0][0]
24              + ekf->P[0][1] * CtmatInv[1][0],
25                  ekf->P[0][0] * CtmatInv[0][1]
26                   + ekf->P[0][1] * CtmatInv[1][1],
27                  ekf->P[0][0] * CtmatInv[0][2]
28                   + ekf->P[0][1] * CtmatInv[1][2]},
29                 {ekf->P[1][0] * CtmatInv[0][0]
30                  + ekf->P[1][1] * CtmatInv[1][0],
31                  ekf->P[1][0] * CtmatInv[0][1]
32                   + ekf->P[1][1] * CtmatInv[1][1],
33                  ekf->P[1][0] * CtmatInv[0][2]
34                   + ekf->P[1][1] * CtmatInv[1][2]}};
```

Općenito, implementacija produženog Kalmanovog filtra u C++-u prati matematičke formulacije tog algoritma, pa je samo pitanje uspješnog prevođenja matematičke terminologije u strukture podataka i operatore dostupne u programskom miljeu.

Podaci koji se dobivaju kroz Kalmanov filtar relativno precizno i stabilno opisuju *pitch* i *roll* položaja sustava prilikom njegovog kretanja u prostoru, kao što je prikazano na slici 3.4.

3.5. Osmišljanje arhitekture sustava

Sada kad je jezgra sustava, produženi Kalmanov filtar, implementirana, fokus je prebačen natrag na širu sliku - arhitekturu cijelog sustava. Trenutno stanje sustava je slijedeće: sustav se sastoji od dvije komponente, a to su mikrokontrolerski sustav, koji uključuje Teensy i senzore, te okruženje Supercollider.

Budući da se od sustava očekuje da ima sposobnost kontrolirati MIDI uređaje, a i bilo bi dobro da vrijednosti koje se šalju u Supercollider budu prikladnije, tj. da se njima mora što manje manipulirati u samom Supercollider-u, jasno je da treba postojati neki međukorak između prikupljanja senzorskih podataka i njihove sonifikacije. Taj međukorak služio bi tome da na konfigurabilan način vrijednosti dobivene mjerenjem pokreta skalira i mapira na proizvoljan broj izlaznih podatkovnih kanala. Osim toga, taj međukorak trebao bi biti sposoban nekom metodom pretvarati podatke u MIDI poruke.

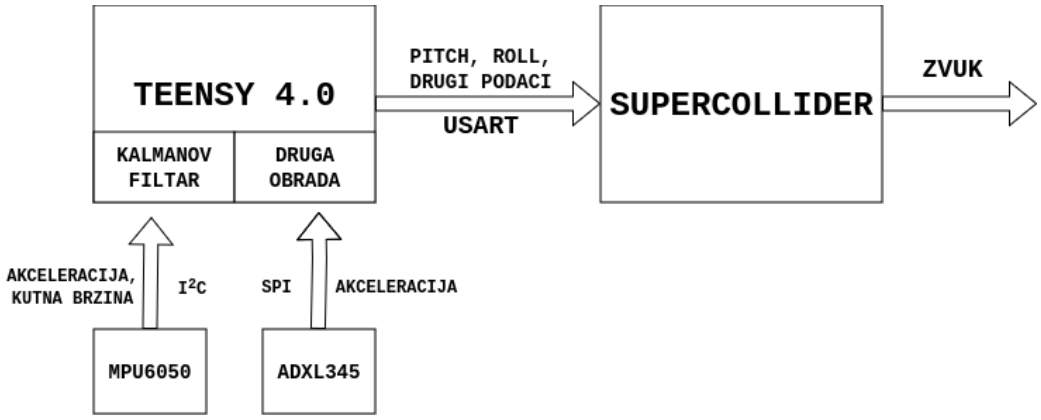
Konfiguracija tog međukoraka vrši se pomoću konfiguracijske datoteke u `.json` formatu[16].

Slika 3.4. Pitch i roll tokom micanja sustava

Odlučeno je da se konfiguracijska datoteka sastoji od čelija slijedećeg oblika:

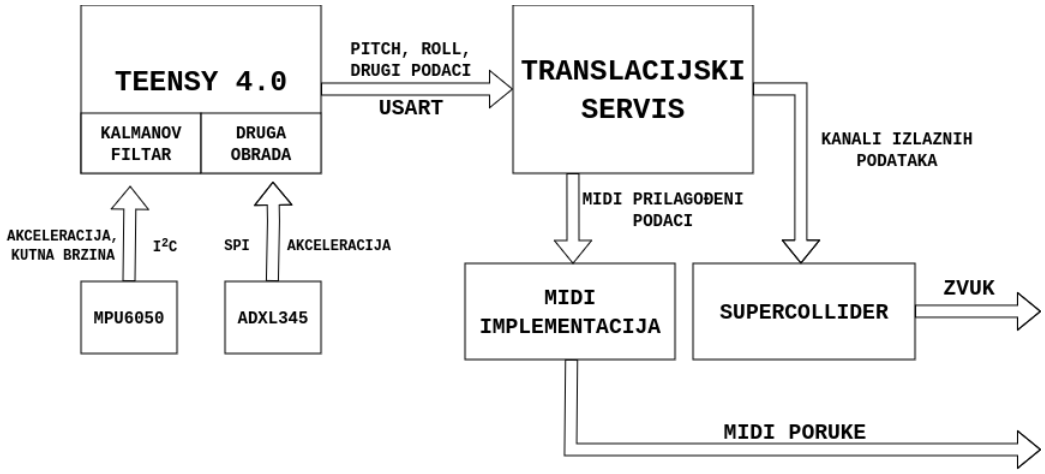
```
"pitch":
{
   "map_channel" : 1,
   "min" : 200,
   "max" : 8000,
   "log" : true,
   "quant": "12et",
   "midi" : "n"
}
```

Čelija u isječku govori o postupcima koji se trebaju napraviti nad podacima o nagibu (engl. pitch). Oni se trebaju mapirati na prvi izlazni kanal, skalirati na eksponencijalnu razdiobu od 200 do 8000, kvantizirati na vrijednosti frekvencija 12-tonske ljestvice, tzv. 12-tone equal temperament - *12et* i koristiti za MIDI *Note On* poruke. Iste takve čelije u konfiguraciji postoje za sve vrijednosti koje izlaze iz Teensy sustava.



Slika 3.5. Trenutna arhitektura sustava

Konfiguraciju bi prilikom pokretanja čitao translacijski servis, a zatim u beskonačnoj petlji čitao podatke sa Teensy-ja preko serijskog sučelja, nad njima vršio potrebnu obradu i ispisivao ih na standardni izlaz, koji će se potom hvatati u zadnjoj komponenti sustava - okruženju Supercollider ili servisu za MIDI.



Slika 3.6. Ciljna arhitektura sustava

3.6. Translacijski servis

Translacijski servis implementiran je u programskom jeziku C, a preveden i izgrađen korištenjem alata CMake[17]. Za parsiranje `.json` datoteke korištena je biblioteka otvorenog koda Jansson[18]. Parsiranje konfiguracijske datoteke vrši se pri pokretanju servisa, tako da ako se u njoj naprave izmjene, nije potrebno ponovno graditi servis, nego ga samo ponovno pokrenuti. Princip parsiranja konfiguracijske datoteke prikazan je u slijedećem isječku:

```
1 #include <jansson.h>
```

```
2
3  #define NUM_PARAMS 6
4  enum Params
5  {
6     PITCH,
7     ROLL,
8     ...
9  };
10
11 enum Quantization
12 {
13    Q_NONE,
14    Q_12ET
15 };
16
17 typedef struct
18 {
19    uint8_t channel;
20    uint32_t min;
21    uint32_t max;
22    char log;
23    char quant;
24    char midi[8];
25 } ValueConfig;
26
27 ValueConfig CONFIG[NUM_PARAMS];
28
29 int Config_Parse(FILE *cfg)
30 {
31    json_t *root;
32    json_error_t error;
33
34    json_t *pitch;
35    json_t *roll;
36    ...
37
38    json_t *channel;
39    json_t *min;
40    json_t *max;
41    json_t *log;
42    json_t *quant;
43    json_t *midi;
44
45    root = json_loadf(cfg, 0, &error);
46    if (!root)
47    {
48       fprintf(stderr, "json error: on line %d: %s\n", error.
             line, error.text);
49       return 1;
50    }
51
52    pitch = json_object_get(root, "pitch");
53    if (!json_is_object(pitch))
```

```
54    {
55       fprintf(stderr, "error: pitch is not an object\n");
56       json_decref(root);
57       return 1;
58    }
59    else
60    {
61       channel = json_object_get(pitch, "map_channel");
62       if (!json_is_integer(channel))
63       {
64          fprintf(stderr, "error: channel is not an integer\n"
                );
65          json_decref(root);
66          return 1;
67       }
68       else
69       {
70          CONFIG[PITCH].channel = json_integer_value(channel);
71       }
72
73       ...
74
75       quant = json_object_get(pitch, "quant");
76       if (!json_is_string(quant))
77       {
78          fprintf(stderr, "error: quant is not a string\n");
79          json_decref(root);
80          return 1;
81       }
82       else
83       {
84          if (!strcmp(json_string_value(quant), "none"))
85          {
86             CONFIG[PITCH].quant = Q_NONE;
87          }
88          else if (!strcmp(json_string_value(quant), "12et"))
89          {
90             CONFIG[PITCH].quant = Q_12ET;
91          }
92       }
93
94       midi = json_object_get(pitch, "midi");
95       if (!json_is_string(midi))
96       {
97          fprintf(stderr, "error: quant is not a string\n");
98          json_decref(root);
99          return 1;
100      }
101      else
102      {
103         strcpy(CONFIG[PITCH].midi, json_string_value(midi));
104      }
105   }
```

```
106
107     roll = json_object_get(root, "roll");
108
109     ...
110
111     return 0;
112 }
113
114 int main(int argc, char *argv[])
115 {
116     // PARSE CONFIG FILE
117
118     FILE *config = fopen("/path/to/config/config.json", "r");
119     Config_Parse(config);
120
121     ...
122 }
```

Funkcija `Parse_Config` prima deskriptor datoteke koja sadrži konfiguraciju i koristeći pokazivaće na `json_t` *handler*-e definirane u biblioteci Jansson čita datoteku i sprema njene vrijednosti u polje `CONFIG` struktura tipa `ValueConfig`. Radi jednostavnosti i čitljivosti, toj se strukturi pristupa pomoću enumeracije `Params`.

Jednom kad je konfiguracija učitana u polje `CONFIG`, potrebno je uspostaviti komunikaciju sa Teensy-jem preko serijskog sučelja. To se ostvaruje pomoću biblioteke otvorenog koda termios i prikazano je u slijedećem isječku:

```
1  #include <termios.h>
2
3  #define SERIAL_PORT "/dev/ttyACM0"
4
5  ...
6
7  int Open_Serial()
8  {
9      // OPEN SERIAL PORT
10
11     struct termios tty;
12
13     int serial_port = open(SERIAL_PORT, O_RDWR);
14     if (serial_port < 0)
15     {
16         fprintf(stderr, "couldn't open serial_port %s\n",
               SERIAL_PORT);
17     }
18
19     if (tcgetattr(serial_port, &tty) != 0)
20     {
21         fprintf(stderr, "Error %i from tcgetattr: %s\n", errno,
```

```
            strerror(errno));
    }

    tty.c_cflag &= ~PARENB;        // Clear parity bit, disabling parity (most common)
    tty.c_cflag &= ~CSTOPB;        // Clear stop field, only one stop bit used in communication (most common)
    tty.c_cflag |= CS8;            // 8 bits per byte (most common)

    ...

    cfsetispeed(&tty, B115200); // input BAUD
    cfsetospeed(&tty, B115200); // output BAUD

    // Save tty settings, also checking for error
    if (tcsetattr(serial_port, TCSANOW, &tty) != 0)
    {
        fprintf(stderr, "Error %i from tcsetattr: %s\n", errno,
            strerror(errno));
    }

    return serial_port;
}

...

int main(int argc, char *argv[])
{
    ...
    // OPEN SERIAL PORT

    int serial_port = Open_Serial();

    ...

    return 0;
}
```

Serijsko sučelje funkcionira otvoreno sa najuobičajenijim postavkama, tako da je funkcija `Open_Serial` mogla biti većinom kopirana iz dokumentacije biblioteke.

Glavna funkcionalnost translacijskog servisa je skaliranje i kvantizacija. Oni su implementirani u datoteci `mutils.c` i korišteni preko zaglavlja `mutils.h`.

```
#include "mutils.h"
#include <math.h>

int Quantize_To_Midi(float freq)
{
```

```
6     return (int) 12 * log2(freq / 440) + 69;
7  }
8
9  float Midinote_To_Freq(int midinote)
10 {
11    float m_norm = (float) midinote - 69;
12    return pow(2.0, m_norm / 12) * 440;
13 }
14
15 float Value_To_Range(float in_value, float in_min, float
      in_max, float out_min, float out_max, char exp)
16 {
17    float in_range;
18    float out_range = out_max - out_min;
19    float out_value;
20
21    if (in_min < 0 && in_max > 0)
22    {
23       in_value += fabs(in_min);
24       in_max += fabs(in_min);
25       in_range = in_max;
26       in_min = 0.0f;
27
28    }
29    else
30       in_range = in_max - in_min;
31
32    if (exp)
33    {
34       float exponent = ((in_value / in_range) + in_min) * (
            log2(out_max) - log2(out_min)) + log2(out_min);
35       out_value = pow(2, exponent);
36    }
37    else
38    {
39       out_value = (in_value / in_range + in_min) * out_range
            + out_min;
40    }
41
42    return out_value;
43 }
```

Kvantizacija na 12-tonsku ljestvicu vrši se na slijedeći način. Prvo se iz nekvantizirane frekvencije $freq$ dobiva midi nota $mnote$ formulom

$$mnote = \lfloor 12 * \log_2(\frac{freq}{440}) + 69 \rfloor \tag{3.1}$$

Zatim se midi nota ponovno pretvara u frekvenciju, ovaj put kvantiziranu, $freq_q$, po-

moću formule

$$freq_q = 2^{\frac{midinote-69}{12}} * 440 \tag{3.2}$$

Dakle, ako se frekvencija treba kvantizirati, to se radi slijedećim pozivom:

```
freq_q = Midinote_To_Freq(Quantize_To_Midi(freq));
```

Nadalje, druga bitna funkcionalnost, skaliranje u zadani raspon, postiže se u funkciji `Value_To_Range`. Ta funkcija kao argumente prima ulaznu vrijednost, njene trenutne ekstreme, željene izlazne ekstreme i zastavicu koja određuje treba li izlazni raspon biti raspoređen linearno ili eksponencijalno. Funkcija radi tako da raspon ulaznih vrijednosti prvo normalizira, odnosno pretvori u raspon kojem je minimum u 0, a zatim skalira linearno ili eksponekcijalno pomoću slijedećih formula.

$$outval_l = (\frac{inval - inmin}{inrange}) * outrange + outmin \tag{3.3}$$

$$outval_e = 2^{(\frac{inval}{inrange}+inmin)*(\log_2(outmax)-\log_2(outmin))+\log_2(outmin)} \tag{3.4}$$

Glavna radna petlja translacijskog servisa vidi se u slijedećem isječku.

```
int CSV_Split(char *csv, int length, float *values)
{
   csv[length] = 0;
   int j = 0;
   char *token = strtok(csv, ", ");
   while (token != NULL)
   {
      values[j++] = atof(token);
      token = strtok(NULL, ", ");
   }

   return 0;
}

int main(int argc, char *argv[])
{
   ...
   MIN[PITCH] = -5.50;
   MAX[PITCH] = 5.50;
   ...

   // MAIN LOOP

   char readbuf[32];
   float values[NUM_PARAMS];
```

```
26    float OUT[NUM_CHANNELS + 1];
27    char CHANNEL_ACTIVE[NUM_CHANNELS + 1];
28    int MIDIVAL[NUM_PARAMS];
29    memset(CHANNEL_ACTIVE, 0, sizeof(CHANNEL_ACTIVE));
30
31    for (;;)
32    {
33       memset(&readbuf, '\0', sizeof(readbuf));
34       memset(OUT, 0, (NUM_CHANNELS + 1) * sizeof(float));
35       memset(MIDIVAL, 0, NUM_PARAMS * sizeof(int));
36       int i = 0;
37       int num_bytes = 0;
38
39       for (;;)
40       {
41          num_bytes += read(serial_port, readbuf + i, 1);
42          if (readbuf[i] == '\n')
43             break;
44          ++i;
45       }
46
47       CSV_Split(readbuf, num_bytes, values);
48
49       for (int p = 0; p < NUM_PARAMS; ++p)
50       {
51          if (values[p] > MAX[p])
52             values[p] = MAX[p];
53          if (values[p] < MIN[p])
54             values[p] = MIN[p];
55       }
56
57       for (int p = 0; p < NUM_PARAMS; ++p)
58       {
59          CHANNEL_ACTIVE[CONFIG[p].channel] = 1;
60
61          float val = Value_To_Range(values[p], MIN[p], MAX[p
             ], CONFIG[p].min, CONFIG[p].max, CONFIG[p].log);
62
63          switch (CONFIG[p].quant)
64          {
65          case Q_12ET: val = Midinote_To_Freq(Quantize_To_Midi
             (val)); break;
66          case Q_NONE: break;
67          }
68
69          OUT[CONFIG[p].channel] += val;
70
71          // midi:
72          if (isalpha(CONFIG[p].midi[0])) // not CC
73          {
74             switch (CONFIG[p].midi[0])
75             {
76             case 'n': // note
```

```
77                MIDIVAL[p] = Quantize_To_Midi(val);
78                break;
79             case 'p': // pitchbend TODO
80                break;
81             default: break;
82             }
83          }
84          else // CC
85          {
86             MIDIVAL[p] = Value_To_Range(values[p], MIN[p],
                  MAX[p], 1, 127, 0); // 0-127 linear for MIDI
87          }
88       }
89
90       // supercollider
91       // channel:value, channel:value, ... , channel:value\n
92       if (!strcmp(argv[1], "-s"))
93       {
94          for (int c = 1; c <= NUM_CHANNELS; ++c)
95          {
96             if (CHANNEL_ACTIVE[c])
97             {
98                printf("%d:%f, ", c, OUT[c]);
99             }
100         }
101         printf("\n");
102         fflush(stdout);
103      }
104      // midi:
105      // n:value, <cc_number>: value, ... \n
106      else if (!strcmp(argv[1], "-m"))
107      {
108         for (int p = 0; p < NUM_PARAMS; ++p)
109         {
110            if (CONFIG[p].midi[0])
111            {
112               printf("%s:%d, ", CONFIG[p].midi, MIDIVAL[p]);
113            }
114         }
115
116         printf("\n");
117         fflush(stdout);
118      }
119   }
120
121   return 0;
122 }
```

Kao prvo, potrebno je u polja `MIN` i `MAX` upisati minimalne i maksimalne ulazne vrijednosti za sve fizičke parametre. Te se vrijednosti koriste prilikom skaliranja vrijednosti funkcijom `Value_Range`. Zatim se inicijalizira polje `values`, koje će sadržavati parsi-

rane ulazne vrijednosti, polje `OUT`, koje će sadržavati izlazne vrijednosti mapirane na izlazne kanale, polje `CHANNEL_ACTIVE`, koje sadržava zastavice aktivnosti kanala, i polje `MIDIVAL` koje će sadržavati broj midi note ili vrijednost *Control Change* poruke. Unutar petlje, relevantna polja se nuliraju, zatim se čita redak sa serijskog sučelja, parsira pomoću funkcije `CSV_Split` i posprema u polje `values`. Vrijednosti polja `values` zatim se odrežu na njihovim određenim ekstremima. Nakon toga, nad svakim se ulaznim parametrom vrši skaliranje i eventualna kvantizacija metodama opisanima ranije i dobivena vrijednost se pribraja u određeni kanal, koji se označuje kao aktivan. Bitno je primjetiti da se više parametara može mapirati na isti kanal, u tom slučaju se vrijednosti zbrajaju.

Midi poruke se dijele na dva tipa, *Note On* poruke, u konfiguraciji označene sa "n" i *Control Change* poruke, u konfiguraciji označene sa *CC* brojem na kojeg poruka utječe. U slučaju *Note On* poruke, vrijednost parametra interpretira se kao frekvencija i pretvara se u broj midi note pomoću funkcije `Quantize_To_Midi`, a u slučaju `Control Change` poruke, vrijednost se skalira u linearni raspon 0–127.

Idući korak je ispisati dobivene vrijednosti. Ispis se djeli na dva slučaja, ovisno o traženom *end point*-u. Ako se podaci koriste kroz okruženje Supercollider, program se pokreće sa zastavicom "-s" a ispis je formatiran kao red parova `kanal:vrijednost` odvojenih zarezima. Ako se podaci koriste za generiranje MIDI poruka, ispis je formatiran kao red parova `n:midinote` ili `ccnum:ccval`, ovisno o tipu poruke. Naredba `fflush(stdout)` koristi se za instantni ispis `stdout` međuspremnika, što je potrebno za glatko dohvaćanje izlaza servisa u okruženjima u kojima će se on pokretati. Bitno je napomenuti da će se proces translacijskog servisa pokretati ili iz Supercollider okruženja ili iz generatora MIDI poruka, nikad samostalno, tako da je cijeli proces prerade i ispisa podataka krajnjem korisniku apstrahiran.

3.7. MIDI skripta

Generiranje MIDI poruka postignuto je koristeći programsku biblioteku rtmidi. Rtmidi je biblioteka napisana u jeziku C++, ali postoji *wrapper*[19] za nju koji omogućava njeno korištenje u programskom jeziku python. MIDI implementacija u ovom projektu je na *proof-of-concept* razini, ali ostvaruje bazu na kojoj se mogu izgraditi kompleksnije i sve-

obuhvatnije implementacije. MIDI skripta, implementirana u programskom jeziku python sastoji se od tri koraka i prikazana je u slijedećem isječku koda.

```
import time
import rtmidi
import subprocess

# SELECT AND OPEN MIDI PORT
midiout = rtmidi.MidiOut()
available_ports = midiout.get_ports()
print("available MIDI ports:")
i = 1
for p in available_ports:
   print(str(i) + " : " + str(p))
   i += 1

midi_port = input("SELECT DESIRED PORT: ")
midiout.open_port(int(midi_port) - 1)

if(not available_ports):
   print("NO AVAILABLE MIDI PORTS")

# START UNIX PROCESS AND CAPTURE PROCESS stdout
p = subprocess.Popen(["/path/to/TranslationService", "-m"],
   stdout = subprocess.PIPE, universal_newlines = True)

# PARSE INPUT AND SEND MIDI MESSAGES
lastmnote = 0
for stdout_line in iter(p.stdout.readline, ""):
   mappings = {}
   allvalues = stdout_line.split(',')
   allvalues.pop()
   for v in allvalues:
      valstr = v.split(':')
      mappings[valstr[0]] = valstr[1]

   for k in mappings.keys():
      if k.isalpha(): # znaci da nije CC
         if k == 'n':
            #note on
            mnote = int(mappings[k])
            note_off = [0x80, lastmnote, 0]
            midiout.send_message(note_off)
            note_on = [0x90, mnote, 127]
            midiout.send_message(note_on)
            lastmnote = mnote
            print("note on " + mappings[k])
         else:
            pass
      else: # CC
         val = int(mappings[k])
         cc = [0xB0, int(k), val]
         midiout.send_message(cc)
```

```
50          print("cc " + k + " " + mappings[k])
51
52    #time.sleep(0.01)
53
54 del midiout
```

Prvi korak je pretražiti dostupne MIDI priključke, prikazati ih korisniku i zatražiti da odabere uređaj kojem želi slati poruke. Da bi fizički MIDI instrument bio dostupan, potrebno je da je njegovo USB MIDI sučelje *USB Class Compliant*, što u suštini znači da radi bez potrebe za *driver*-om. Ako jest, MIDI priključak tog instrumenta će se pojaviti u popisu dostupnih. Što se tiče softverskih MIDI uređaja, primjerice VST uređaja[20], potrebno je koristiti audio server, primjerice Jack[21], za usmjeriti MIDI poruke prema određenom softverskom MIDI uređaju. U tom slučaju, u popisu dostupnih MIDI priključaka vidjet će se MIDI input audio servera, te se onda njegovim korištenjem poruke usmjeravaju prema uređaju.

Drugi korak je pomoću biblioteke `subprocess` pokrenuti proces translacijskog servisa sa prikladnom zastavicom "-m" i hvatati njegov izlaz.

Treći i najbitniji korak je parsiranje izlaza translacijskog servisa redak po redak, spremanje podataka u obliku ključeva i vrijednosti u *dictionary* `mappings`, i zatim od podataka kreiranje MIDI poruka. U sklopu biblioteke rtmidi, MIDI poruke sastoje se od liste koja sadrži 3 broja. Prvi broj određuje tip MIDI poruke i MIDI kanal na koji se šalje, drugi i treći brojevi su vrijednosti koje se šalju. U slučaju *Note On* poruka, ta dva broja su broj midi note i *velocity*. Primjerice, ako se želi poslati *Note On* poruka na nulti MIDI kanal za notu C4 sa maksimalnim *velocity*-jem, ta poruka bi izgledala ovako:

```
1 note_on = [0x90, 60, 127]
```

Poruka se zatim šalje na ciljni MIDI uređaj naredbom:

```
1 midiout.send_message(note_on)
```

Trenutna implementacija podržava korištenje MIDI uređaja na monofoničan način, tj. samo jedan glas se čuje u jednom trenutku. Za ostvariti to, prije slanja *Note On* poruke, šalje se *Note Off* poruka za prošlu poslanu notu na sličan način.

Control Change poruke, prvi broj u listi određuje da se radi o tom tipu poruke, drugi

broj je CC vrijednost, a treći broj veličina koja se šalje. Primjerice, ako se želi na CC broj 7, koji u većini implementacija određuje *Master Volume* MIDI uređaja, poslati vrijednost 50, poruka bi izgledala ovako:

```
cc = [0xB0, 7, 50]
```

3.8. Čitanje iz UNIX procesa u okruženju Supercollider

Budući da je implementiran međukorak između Teensy sustava i Supercollider-a, potrebno je promjeniti način na koji se u Supercollider učitavaju podaci, budući da se više ne radi o čitanju sa serijskog sučelja nego iz standardnog izlaza UNIX procesa. To se postiže korištenjem klase `Pipe`.

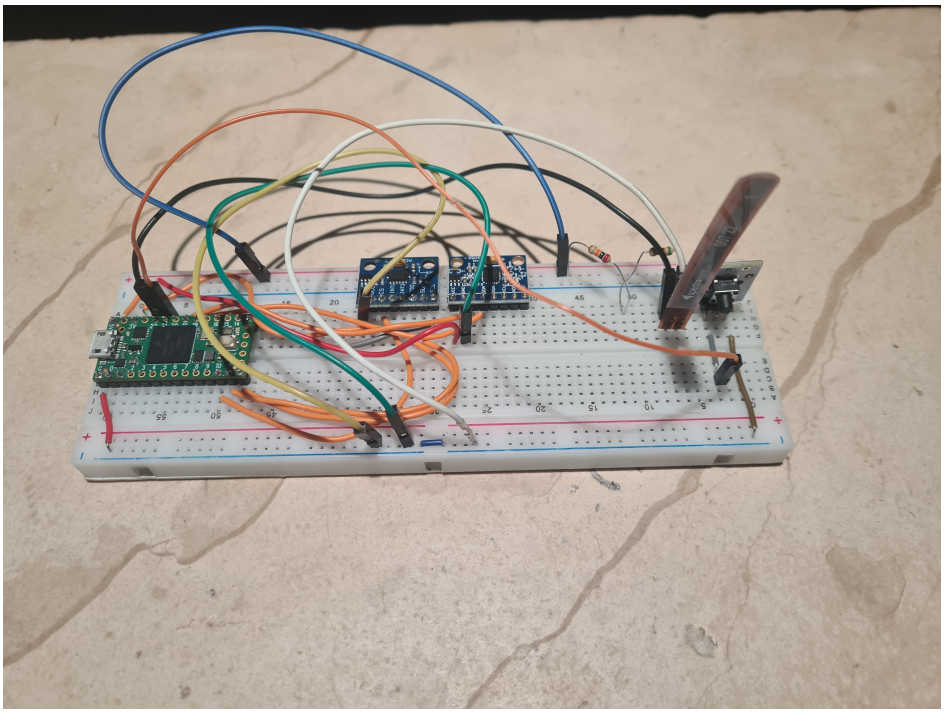
```
s.boot
(
d = SynthDef(\SimpleSine, {|freq = 440, amp = 0.5, out| Out.
   ar(out, SinOsc.ar(freq, 0, amp)) });
d.add;
)
~x = Synth(\SimpleSine);

(
var p, l, arr;
p = Pipe.new("/path/to/TranslationService -s", "r");
l = p.getLine;
while(
  {l.notNil},
  {
    arr = l.split($,).collect(_.asFloat).postln;
    ~freq = arr.at(0);
    ~amp = arr.at(1);
    ~x.set(\freq, ~freq);
    ~x.set(\amp, ~amp*0.01);
    l = p.getLine;
  }
);
p.close;
)
```

Do vrijednosti u listi kanala dolazi se kroz indekse liste `arr`, pa se one mogu mapirati na proizvoljne elemente generatora zvuka korištenjem naredbe `set`.

3.9. Dodavanje dodatnih funkcionalnosti u sustav

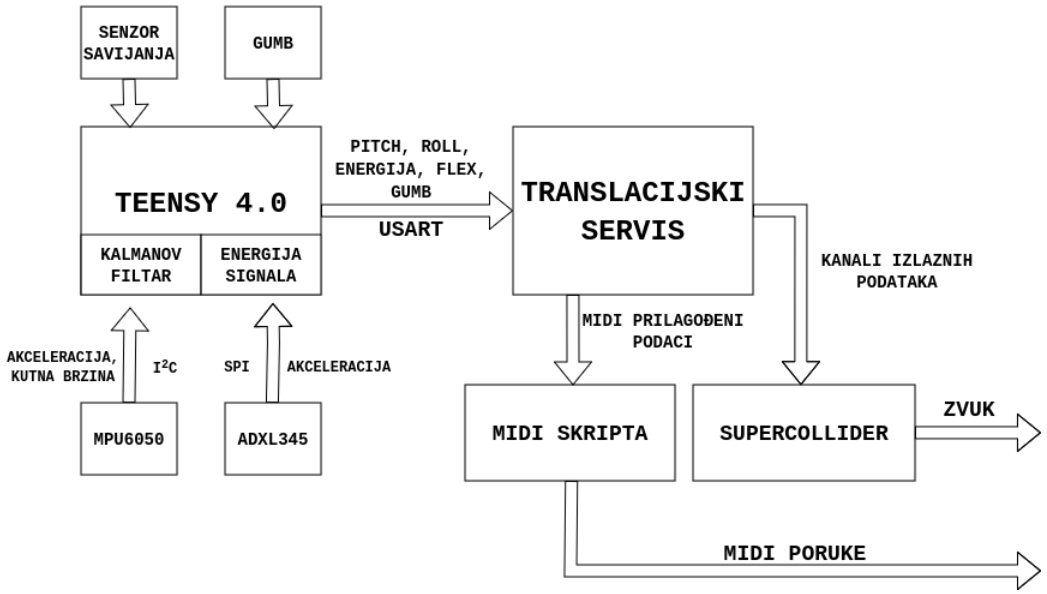
Sada kad je ostvaren cjelokupni podatkovni cjevovod sustava, fokus se vraća na širenje fizičkih funkcionalnosti sustava - tj. dodavanje novih ulaznih fizičkih parametara. Dodaju se senzor savijanja (engl. flex sensor) i gumb, a funkcija ADXL345 akcelerometra se mijenja.



Slika 3.7. Zadnja iteracija sustava na eksperimentalnoj pločici

Senzor savijanja funkcionira kao potenciometar, kad je ravan ima minimalan otpor, a kad je savijen njegov otpor raste. Potrebno je spojiti ga tako da jedan njegov priključak vodi do nule, a drugi je spojen na neki analogni pin Teensy sustava i preko nekog otpora na 5V napajanje dostupno na Teensy sustavu. Nakon isprobavanja par vrijednosti otpora, pokazalo se da otpor od 30 kΩ funkcionira na prihvatljivoj razini. Čitanje sa senzora je jednostavno i vrši se preko funkcije `analogRead()`. Dobivaju se vrijednosti od otprilike 750 do 1023, što daje dovoljnu zrnatost za skaliranje na ciljne vrijednosti preko konfiguracijske datoteke i translacijskog servisa.

Gumb ima tri priključka, jedan je spojen na nulu, drugi direktno na napajanje a treći na neki digitalni pin Teensy sustava. Originalna ideja bila je da gumb ima dvojaku funkcionalnost, da služi kao *trigger* za neke predodređene događaje, ovisne o MIDI/Supercollider implementaciji, ali da ima i *tap-tempo* funkcionalnost, odnosno da se periodičkim



Slika 3.8. Konačna arhitektura sustava

stiskanjem gumba određuje neka frekvencija koja se, opet oviso o implenentaciji, može koristiti za vrijeme kašnjenja *delay* efekta, za frekvenciju niskofrekventnog oscilatora koji kontrolira *tremolo*, *vibrato* ili nešto drugo. Međutim, kašnjenje između pristizanja svakog mjerenja, koje uključuje i informaciju o tome je li gumb stisnut ili ne, uzrokovano neoptimalnosti koda i značajne količine obrade potrebne za izračunavanje Kalmanovog filtra, dovoljno je da nije moguće odrediti takvu frekvenciju do zadovoljavajuće preciznosti. To bi bilo moguće korištenjem višedretvenosti, koja je, za razliku od Arduino sustava, Teensy sustavima dostupna kroz korištenje biblioteke `TeensyThreads`[22].

Što se tiče akcelerometra, u planu je bilo koristiti ga na drugoj ruci za kontroliranje parametara korištenjem gravitacije da se odredi orijentacija šake[23], ali takvo ostvarenje zahtjevalo bi korištenje I²C ili SPI protokola za udaljenost koju za koju ti protokoli nisu dizajnirani, a postoji i logistički problem stabilnog ožičavanja takvog ostvarenja. Zbog toga, akcelerometar će se koristiti za mjerenje energije signala, E_s, preko formule

$$E_s = \langle x(n), x(n) \rangle = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{3.5}$$

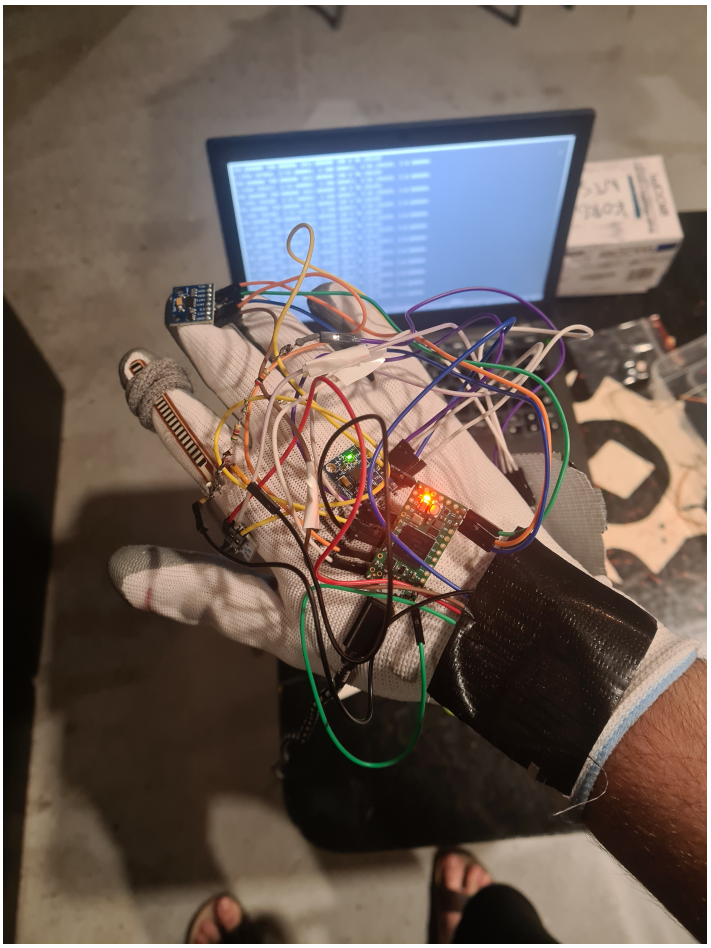
Ta vrijednost daje informaciju o količini gibanja akcelerometra. On će se, ultimativno, staviti na dio sustava, primjerice vrh prsta, koji se može slobodno micati bez velikog utjecaja na gibanje MPU6050, a time i bez da značajno utječe na rezultate Kalmanovog filtra, pa se njegovim gibanjem može dodati još jedan sloj ekspresije bez gubitka stabilnosti.

3.10. Konačni izgled sustava

Konačni zahtjev za sustav je njegova fizička implementacija na rukavicu. Pozicije elemenata sustava odabrane su tako da maksimiziraju svoju korisnost i jednostavnost korištenja. Pričvršćeni su za rukavicu ljepilom. Teensy sustav pozicioniran je tako da bude dovoljno blizu svim elementima sustava, radi osiguravanja funkcije komunikacijskih protokola, MPU6050 pozicioniran je tako da može kvalitetno mjeriti kretnje zapešća, ADXL345 tako da mjeri gibanje srednjaka i prstenjaka, senzor savijanja na kažiprst radi njegove relativne neovisnosti od ostalih prstiju, a gumb na mjesto gdje ga se može pritisnuti palcem bez obzira na poziciju ruke i prstiju.

Slika 3.9. Pozicije elemenata sustava na rukavici

Sustav je ožičen pomoću kablova za eksperimentalnu pločicu, uz lemljenje po potrebi. Nakon ožičenja, sustav je isproban.



Slika 3.10. Ožičen i aktivan sustav na rukavici

4. Zaključak

Cilj ovog rada bio je osmisliti i realizirati sustav s pomoću kojeg je moguće parametrima ljudskog pokreta upravljati digitalnim generatorom zvuka. Za postizanje tog cilja odabrana je arhitektura sačinejena od od tri velike funkcionalne cjeline. Prva cjelina je mikrokontrolerski sustav koji se sastoji od Teensy 4.0 razvojnog sustava i pripadnih senzora. Ova cjelina služi prikupljanju senzorskih podataka i njihovoj obradi na način da izlazni podaci odražavaju izabrane elemente pokreta. Ključan element prve cjeline je korištenje Kalmanovog filtra da se iz senzorskih mjerenja akceleracije i kutne brzine dobiju relativno precizna i stabilna predviđanja o poziciji šake. Osim toga, drugim senzorima dobiva se informacija o savijenosti i gibanju prstiju, a moguće je i slanje diskretnih *trigger*-a niz podatkovni cjevovod pomoću gumba. Druga cjelina je programski servis, implementiran na operacijskom sustavu Linux, uglavnom u jeziku C, koji služi preradi podataka dobivenih sa mikrokontrolerskog sustava preko serijskog sučelja. Servis služi tome da se podaci, na konfigurabilan način, mogu skalirati na linearne ili eksponencijalne raspone proizvoljnih granica i zatim slati na proizvoljan broj izlaznih kanala za sonifikaciju. Osim toga, servis pruža mogućnost kvantizacije podataka na frekvencije 12-tonske ljestvice. Treća cjelina predstavlja kraj podatkovnog cjevovoda. To su dvije zasebne implementacije, jedna koja koristi okruženje Supercollider, druga koja koristi python skriptu, koje pretvaraju podatke u zvuk ili u MIDI poruke, koje zatim mogu kontrolirati neki eksterni generator zvuka. Konkretne implementacije u Supercollider-u ili hardkodiran MIDI ne smatraju se djelom rada, zbog toga što je ostvaren sustav zamišljen kao kontroler, a detalji implementacije sonifikacije dobivenih podataka su prerogativ njegovog korisnika.

Što se tiče mogućih poboljšanja sustava - postoje mnoga. Kao prvo, prikupljanje senzorskih podataka može se događati u zasebnim dretvama od dretve za obradu i izlazne

dretve za slanje podataka dalje niz cjevovod. Na taj način, izlazna dretva može većom brzinom slati podatke, bez da mora čekati na obradu. Osim toga, može se smisliti metoda za interaktivnu konfiguraciju, tj. da se parametri translacijskog servisa mogu mijenjati dok je on aktivan, na taj način bi se sustav mogao "naštimavati" bez potrebe da se gasi i ponovno pokreće. Također, može se osmisliti neki krovni sustav, idealno sa grafičkim sučeljem, u kojem se može vršiti konfiguracija translacijskog servisa i birati koji se Supercollider program pokreće, možda i imati način za interaktivnu zamjenu Supercollider programa. Nadalje, moguće je napraviti mnogo poboljšanja u MIDI implementaciji sustava, primjerice implementirati veći broj tipova poruka i smisliti pametniji način za *trigger*-anje *Note On* i *Note Off* poruka, koje se trenutno okidaju svaki put kad se pojavi novi niz podataka, često ponavaljajući istu poruku više puta zaredom.

Na tržištu postoje sustavi koji su sposobni ostvariti istu ili sličnu funkcionalnost kao sustav realiziran u sklopu ovog rada, ali ograničeni su mogućnostima ili cijenom. Primjerice *MiMUGloves* sustav isto radi na bazi Kalmanovog filtra i sustava senzora za savijanje integriranih na rukavici, ali cijena tog sustava je u redu veličine par tisuća eura, dok je sustav ostvaren u ovom radu imao ukupnu cijenu materijala od pedesetak eura. Ta se velika razlika u cijeni, jasno, javlja zbog toga što je *MiMUGloves* sustav robusniji i kvalitetniji proizvod, ali i činjenice da je dizajniran za relativno malu nišu profesionalnih umjetnika, a mali volumen proizvodnje iziskuje visoku cijenu. Usprkos tome, sustav ostvaren u ovom radu nije namjenjen da u trenutnom obliku bude komercijalni sustav, nego predstavlja okvir pomoću kojeg se može ostvariti elektronički glazbeni instrument pogonjen pokretom po pristupačnoj cijeni. Osim toga, usprkos činjenici da izgleda kao košmar žica i ljepljive trake, subjetkivno mišljenje autora rada je da sustav otvara vrata relativno neistraženom modalitetu ekspresije u elektroničkoj glazbi i da to radi na zabavan način.

Literatura

[1] Y. Du, S. Liu, L. Feng, M. Chen, i J. Wu, "Hand gesture recognition with leap motion", 2017.

[2] Z. Kanga, "The cyborg hand: Gesture, technology, disability and interdisciplinarity in whatever weighs you down", *Contemporary Music Review*, sv. 42, br. 3, str. 319–338, 2023. https://doi.org/10.1080/07494467.2023.2279357

[3] G. Welch i G. Bishop, "An introduction to the kalman filter", University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, teh. izv. 95-041, 1995. [Mrežno]. Adresa: http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html

[4] P. Stoffregen, "Teensy® 4.0 development board", 2014., accessed February, 2024. [Mrežno]. Adresa: https://www.pjrc.com/store/teensy40.html

[5] M. Puckette, "Pure data: another integrated computer music environment", 02 1970.

[6] Zicarelli.D, "Max/msp", 2002. [Mrežno]. Adresa: http://www.cycling74.com/products/maxmsp.html

[7] S. Wilson, D. Cottle, i N. Collins, *The SuperCollider Book*. The MIT Press, 2011.

[8] A. Fraietta, "Open sound control: Constraints and limitations", 06 2008.

[9] M. M. Association, *Complete MIDI 1.0 Detailed Specification*, 1999/2008. [Mrežno]. Adresa: http://www.midi.org/techspecs/gm.php

[10] K. Hamaluik, "How to use the adxl345 accelerometer with a teensy (through spi)." 2015. [Mrežno]. Adresa: https://gist.github.com/hamaluik/b1903a8353dc1ec57da8

[11] E. Fieldsteel. Supercollider tutorials. Youtube. [Mrežno]. Adresa: https://www.youtube.com/playlist?list=PLPYzvS8A{_}rTaNDweXe6PX4CXSGq4iEWYC

[12] D. K. Shaeffer, "Mems inertial sensors: A tutorial overview." *IEEE Communications Magazine*, sv. 51, br. 4, str. 100–109, 2013. [Mrežno]. Adresa: http://dblp.uni-trier.de/db/journals/cm/cm51.html#Shaeffer13

[13] Wikipedia contributors, "Open-loop controller — Wikipedia, the free encyclopedia", 2024., [Online; accessed 28-June-2024]. [Mrežno]. Adresa: https://en.wikipedia.org/w/index.php?title=Open-loop{_}controller&oldid=1213570349

[14] W. Franklin. The kalman filter explained simply. thekalmanfilter.com. [Mrežno]. Adresa: https://thekalmanfilter.com/kalman-filter-explained-simply/

[15] P. Lab. Extended kalman filter software implementation. Youtube. [Mrežno]. Adresa: https://www.youtube.com/watch?v=7HVPjkWOrLE

[16] Wikipedia contributors, "Json — Wikipedia, the free encyclopedia", 2024., [Online; accessed 28-June-2024]. [Mrežno]. Adresa: https://en.wikipedia.org/w/index.php?title=JSON&oldid=1231214366

[17] ——, "Cmake — Wikipedia, the free encyclopedia", 2024., [Online; accessed 28-June-2024]. [Mrežno]. Adresa: https://en.wikipedia.org/w/index.php?title=CMake&oldid=1216921214

[18] "C library for encoding, decoding and manipulating json data". [Mrežno]. Adresa: https://jansson.readthedocs.io/en/latest/

[19] "A python binding for the rtmidi c++ library implemented using cython". [Mrežno]. Adresa: https://pypi.org/project/python-rtmidi/

[20] G. Tanev i A. Božinovski, "Virtual studio technology inside music production", u *ICT Innovations 2013*, V. Trajkovik i M. Anastas, Ur. Heidelberg: Springer International Publishing, 2014., str. 231–241.

[21] Wikipedia contributors, "Jack audio connection kit — Wikipedia, the free encyclopedia", 2023., [Online; accessed 28-June-2024]. [Mrežno]. Adresa:

https://en.wikipedia.org/w/index.php?title=JACK{_}Audio{_}Connection{_}Kit&oldid=1186500871

[22] F. Trias, “Teensythreads”, 2022. [Mrežno]. Adresa: https://github.com/ftrias/TeensyThreads

[23] J. Fox i J. Carlile, “Sonimime: Movement sonification for real-time timbre shaping”, u *Proceedings of the International Conference on New Interfaces for Musical Expression*, Vancouver, BC, Canada, 2005., str. 242–243. https://doi.org/10.5281/zenodo.1176741

Sažetak

Sustav za upravljanje zvukom i njegovim svojstvima s pomoću pokreta

Martin Palčić

U sklopu rada osmišljen je i realiziran sustav s pomoću kojeg je moguće parametrima ljudskog pokreta upravljati digitalnim generatorom zvuka. Odabrani su prikladni senzori, poput akcelerometra, žiroskopa i drugih, te je njihovom integracijom sa mikrokontrolerskim razvojnim sustavom Teensy 4.0 ostvaren ugradbeni sustav sposoban mjeriti ljudski pokret. Testirane su metode obrade senzorskih podataka te je kao jezgra obrade odabran Kalmanov filtar. Također, ostvaren je programski servis koji podatke priprema za sonifikaciju njihovim skaliranjem i mapiranjem na proizvoljan broj kanala za sonifikaciju. Sonifikacija, odnosno pretvaranje podataka u zvuk, vrši se ili pomoću okruženja Supercollider, ili putem MIDI uređaja.

Ostvaren sustav na zadovoljavajućoj razini funkcionira kao elektronički glazbeni kontroler, koji u kombinaciji s nekom metodom generacije zvuka, primjerice onima testiranima u radu, čini elektronički glazbeni instrument pogonjen ljudskim pokretom. Ponuda sličnih sustava na današnjem tržištu je oskudna i ograničena visokom cijenom ili mogućnostima, pa ovaj rad pruža okvir za ostvarenje takvog instrumenta pomoću dostupnih komponenti pristupačnih cijena.

Ključne riječi: Sonifikacija; interakcija čovjeka i računala; glazbeni kontroler

Abstract

English title

Martin Palčić

In this master's thesis, a controller system with the ability to control a digital sound generator with the parameters of human movement was designed and implemented. Suitable sensors, such as accelerometers, gyroscopes and other sensors, were selected, and by using them in conjunction with the Teensy 4.0 development board, an embedded system capable of measuring human hand and fist movement was created. Various sensor data processing methods were tested, and the Kalman filter was selected as the core of the processing. Also, a software service was created with the intention of preparing the sensor data for sonification by scaling and mapping it to an arbitrary number of sonification channels. Sonification, i.e. converting data into sound, is done either using the Supercollider environment, or via a MIDI device.

The realized system functions at a satisfactory level as an electronic musical controller, which, in conjunction with some method of sound generation, such as those tested in the thesis, forms an electronic musical instrument powered by human movement. The number of similar systems on today's market is scarce and these systems are limited by their capabilities or their high price, so this paper provides a framework for the realization of such an instrument using components widely available at affordable prices.

Keywords: Sonification; Human Computer Interaction; Musical Controller