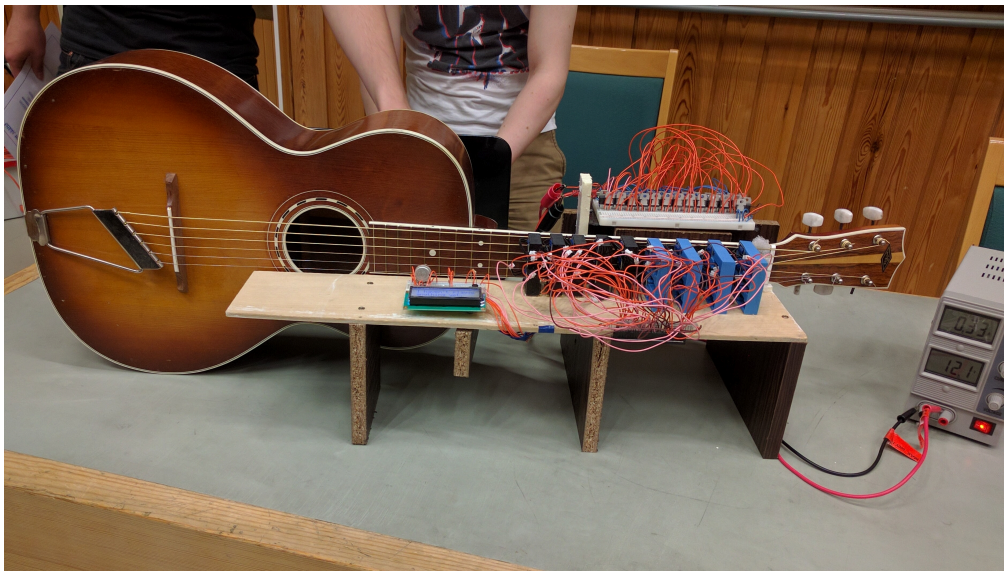
Ryhmä 20 - Solenoidikitara "Swanky Migelo-Bombul"

Solenoidikitara "Swanky Migelo-Bombul"

Ryhmä 20: Aki Kivivirta, Akseli Partanen, Henrik Romppainen, Paavo Räihä

Ohjaaja: Svantte Hyvämäki



Sisällys

- Idea
- Komponentit
- Avainosat
- Toimintaperiaate
- Kytkentäkaavio
- Sävellajin tunnistusalgoritmi
- Teensy-koodi
- Haasteita
- Lopputulos
    - Videoita
    - Kuvia
    - Kehittämisajatukset

Idea

Ryhmässämme on kaksi kitaristia, ja yhteistuumin totesimme, että kitaran soittaminen itse on vaivalloista. Niinpä päätimme antaa nykytekniikan auttaa ja soittaa puolestamme!

Alkuperäinen ideamme oli oikeaa, fyysistä akustista kitaraa soittava Arduino-mikrokontrolleri. Arduino lukisi laajalti kitaristien käytössä olevaa Guitar Pro -tiedostoformaattia, ja soittaisi sen "livenä". Tuhannet ja taas tuhannet kappaleet löytyvät Internetistä Guitar Pro -tiedostoiksi käännettynä.

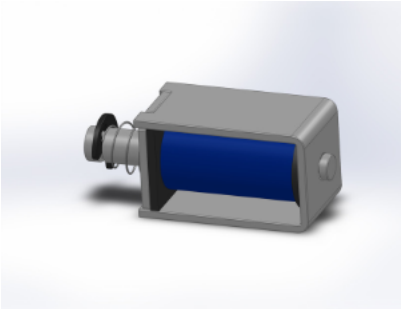
Lopullisessa versiossa Arduino vaihtui Teensy 2.0:aan ja se osaa lukea MIDI-syötettä, joka voi tulla ei pelkästään Guitar Pro -ohjelmasta, vaan mistä tahansa MIDI:ä tuottavasta ohjelmasta. Lisäsimme myös LCD-näytön, joka kertoo kappaleen sävellajin. Alun perin projektissa piti olla servoilla toimivat plektrat jokaiselle kielelle, mutta kokeiluissa selvisi, että solenoidit iskevät kieliä niin kovaa, että kieli alkaa soida jo niiden vaikutuksesta. Näin ollen plektraratkaisua ei näy lopullisessa versiossa.

Komponentit

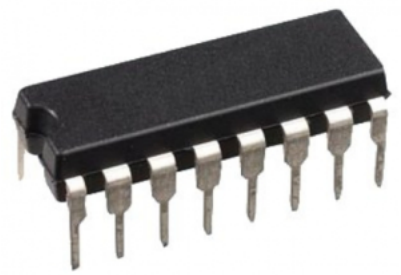
- Teensy 2.0
- (todennäköisesti ROB-11015) solenoideja (27 kpl)
- SN74HC595N siirtorekistereitä (4 kpl)
- STB55NF06 transistoreita (27 kpl)
- 3D-tulostetut kehikot solenoideille
- Hitachi HD44780 -yhteensopiva 2x16 LCD-näyttö
- painonappi
- 1000 ja 220 vastukset ja P160KNP5E potentiometri LCD-näytön kytkemiseen
- Virtalähde (projekti toimi hyvin 12 voltilla ja suurehkoilla virroilla)
- Tietokoneen IDE-kaapeli
- Vaahtomuovia kieltensammuttimeksi
- Leipälauta ja jumpperijohtoja. Paljon jumpperijohtoja.
- Kehikko vanerista ja lastulevystä pitämään kaikki osat yhdessä

Avainosat

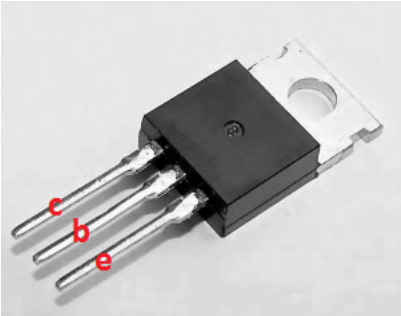
Solenoidi on komponentti, jossa kehikon sisällä on käämi sekä rautatappi. Kun käämiin johdetaan (tasa)sähkövirtaa, käämin indusoiman magneettikentän takia rautatappi työntyy ulos kehikosta. Käytimme solenoideja kielten painamiseen otelautaa vasten.



Käyttämämme tyyppinen **siirtorekisteri** on komponentti, johon syötetään data- ja kellosignaalin avulla 8-bittinen tavu, jonka arvot vastaavat rekisterin ulostulopinneissä kunkin solenoidin tilaa (päälle/pois). Siirtorekisteriä käytimme 27 solenoidin ohjaamiseen ainoastaan kolmella Teensyn pinnillä.

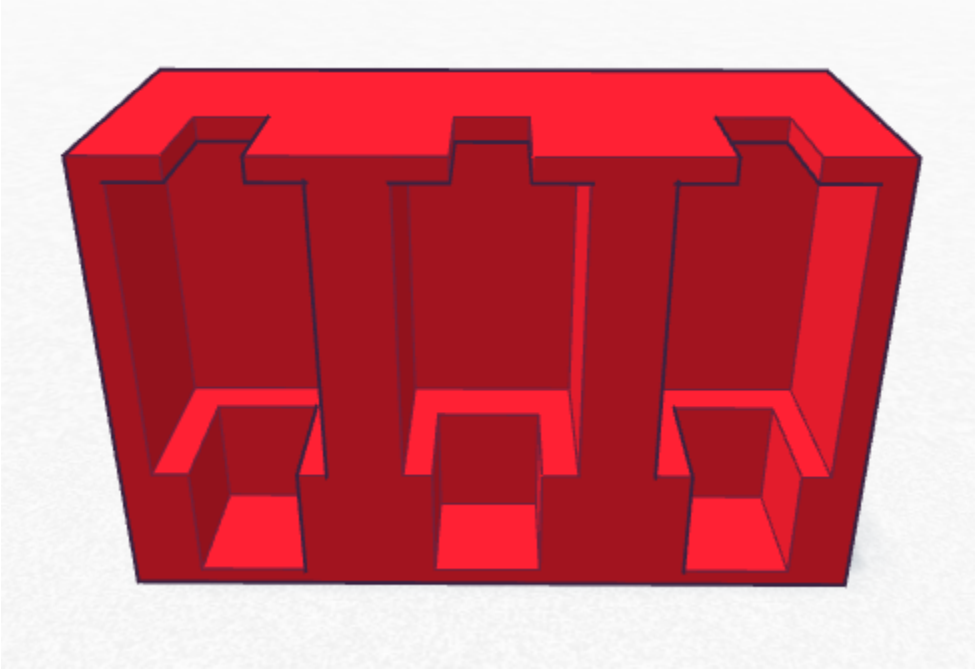


Transistori on komponentti, joka päästää kollektorin (c) ja emitterin (e) välistä virtaa vain, kun kantaan (b) syötetään ohjausvirtaa. Käytimme transistoreita siirtorekisterien heikon ohjausvirran vahvistamiseen solenoideja varten.



Itse **kitara** oli vanhahko, suomalainen, teräskielinen akustinen kitara, johon ei tehty mitään fyysisiä muutoksia. (Kuvassa vaahtomuovisammutin satulan vieressä kielten alla)

3D-tulostetut **solenoidikehikot** mallinnettiin ilmaisella, selainpohjaisella TinkerCad-ohjelmistolla. Tulostuksessa meni useita tunteja.



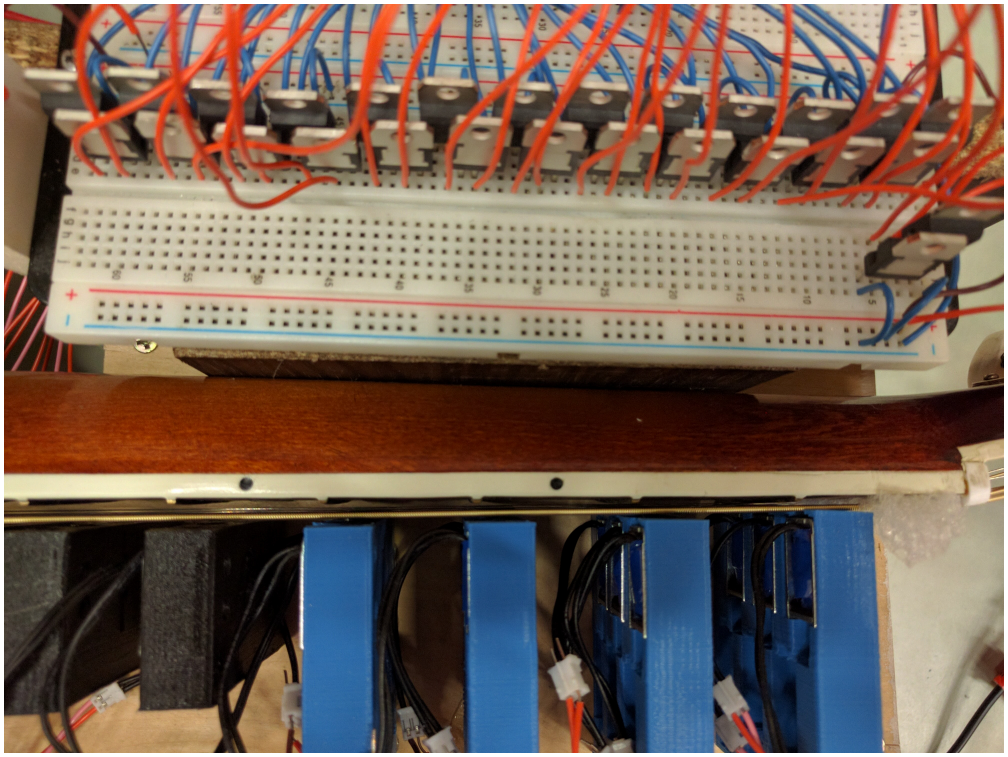
MIDI on laajalti käytetty tiedonsiirtojärjestelmä, joka on suunniteltu välittämään viestejä tietokoneiden ja sähköisten musiikkilaitteiden välillä. MIDI-syöte sisältää tiedon nuoteista, niiden sävelkorkeuksista, kestoista, voimakkuudesta, efekteistä ja monista muista tarpeellisista asioista. Projektissamme kirjoittamamme koodi kuitenkin käyttää syötteestä ainoastaan "note on"- ja "note off"-käskyjä sekä tietoa sävelkorkeuksista.

Toimintaperiaate

Kitaran kielet viritettiin vireeseen Eb A C G A E (lähtien matalimmasta), mutta käytössä olivat vain kielet Eb, C ja korkeampi A. Solenoideja asennettiin kitaran 9. nauhaan asti. Näin käytössä oli 27 eri nuottia, joiden ääniala oli hiukan yli kaksi oktaavia (kitaran koko ääniala oli yli kolme oktaavia). Projektimme kykenee soittamaan kolmea nuottia yhtä aikaa (yksi kultakin kieleltä).

Projektissa ei käytetty lainkaan vapaita kieliä, vaan kielten vapaa sointi estettiin ensimmäisen solenoidikehikon ja kitaran satulan väliin asennetulla vaahtomuovisella sammuttimella. Sammutin esti samalla kielien satulanpuoleisen osan soimisen, kun solenoidi iski kieliä.

Solenoidikehikot aseteltiin kielien eteen aivan otelaudan nauhojen taakse:

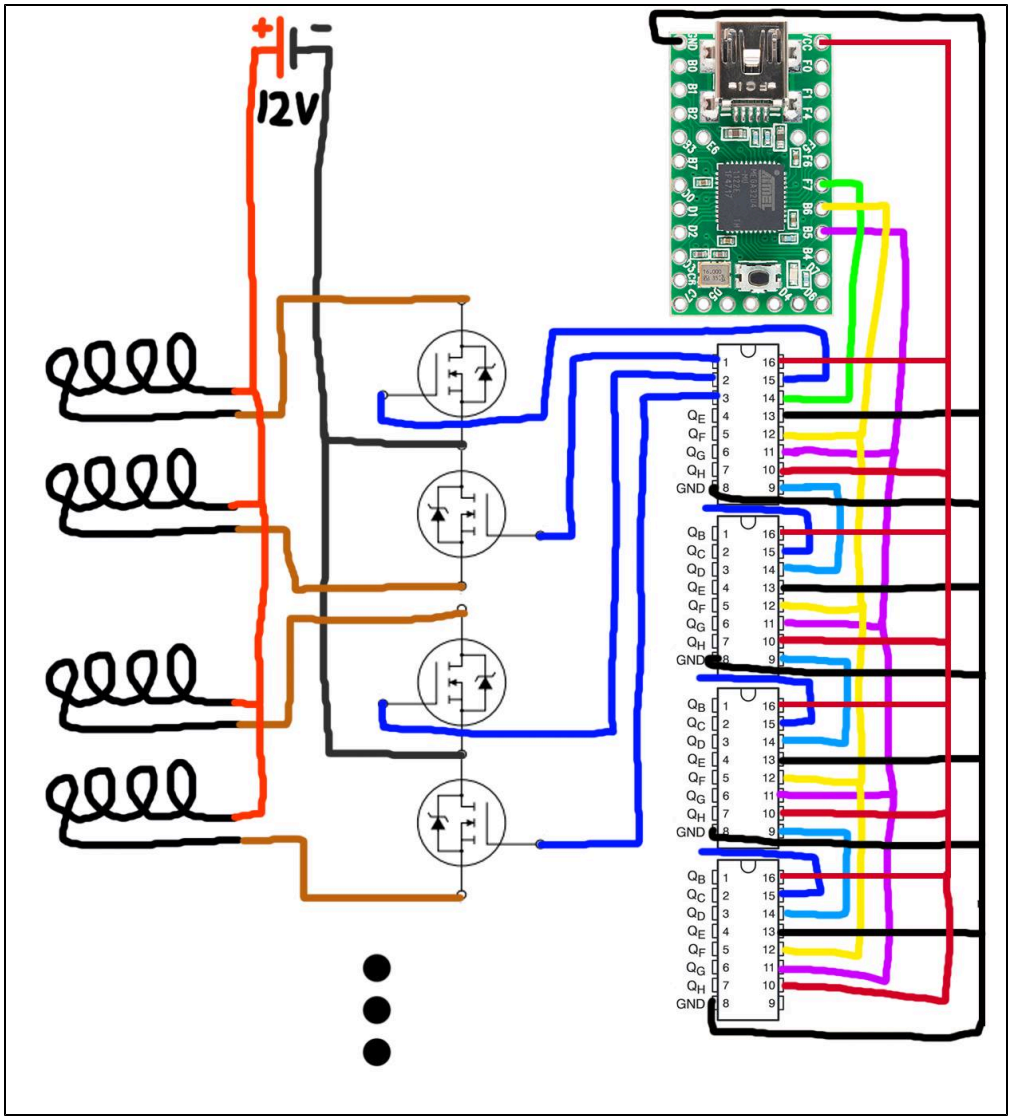


Projektimme aivot ovat Teensy 2.0, jonka itse kirjoittamamme koodi lukee MIDI-syötettä. Kun syötteessä tulee "note on"-käsky tietyllä sävelkorkeudella, Teensy kytkee siirtorekisterin kautta sitä säveltä vastaavan solenoidin päälle painamaan kieltä. Transistorit vahvistavat rekisterin ohjausvirran solenoidien käyttövirraksi. Koodissa tietty nuotti vastaa tiettyä solenoidia, ja käytettyyn skaalaan kuulumattomat nuotit eivät soi. Solenoidi pysyy päällä niin kauan, kunnes syötteessä tulee "note off"-käsky, ja Teensy kytkee sen pois päältä.

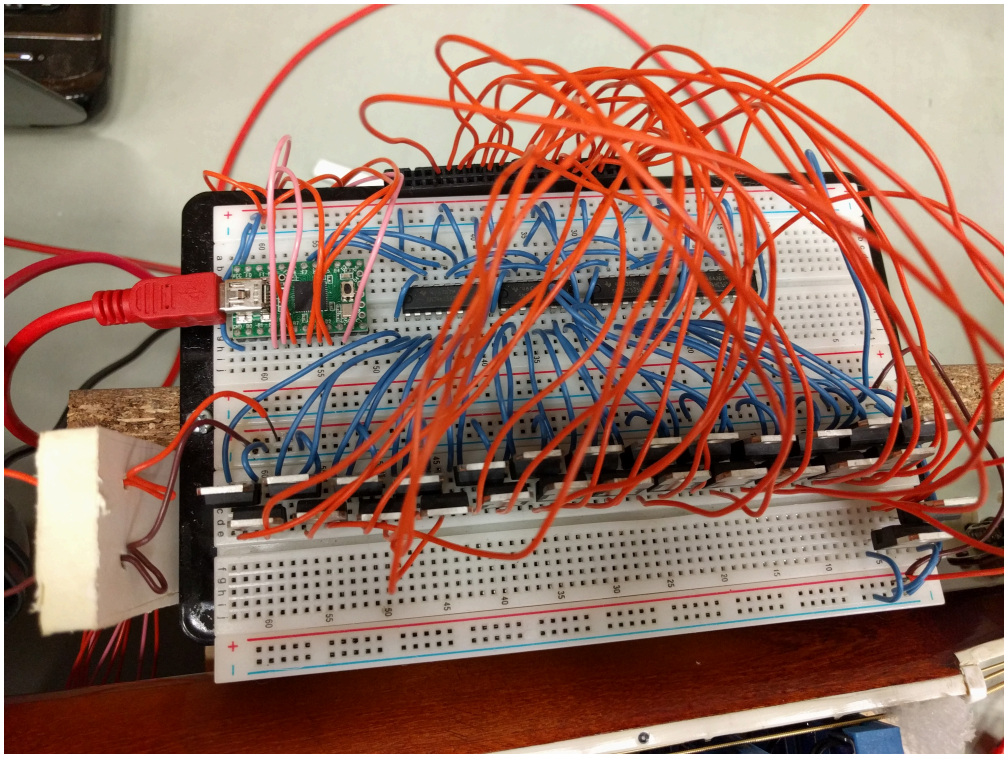
Kytkentäkaavio

Kaaviossa tilan säästämiseksi vain neljä solenoidia. Loput kytkettiin siirtorekistereihin vastaavalla tavalla (tummansiniset johdot).

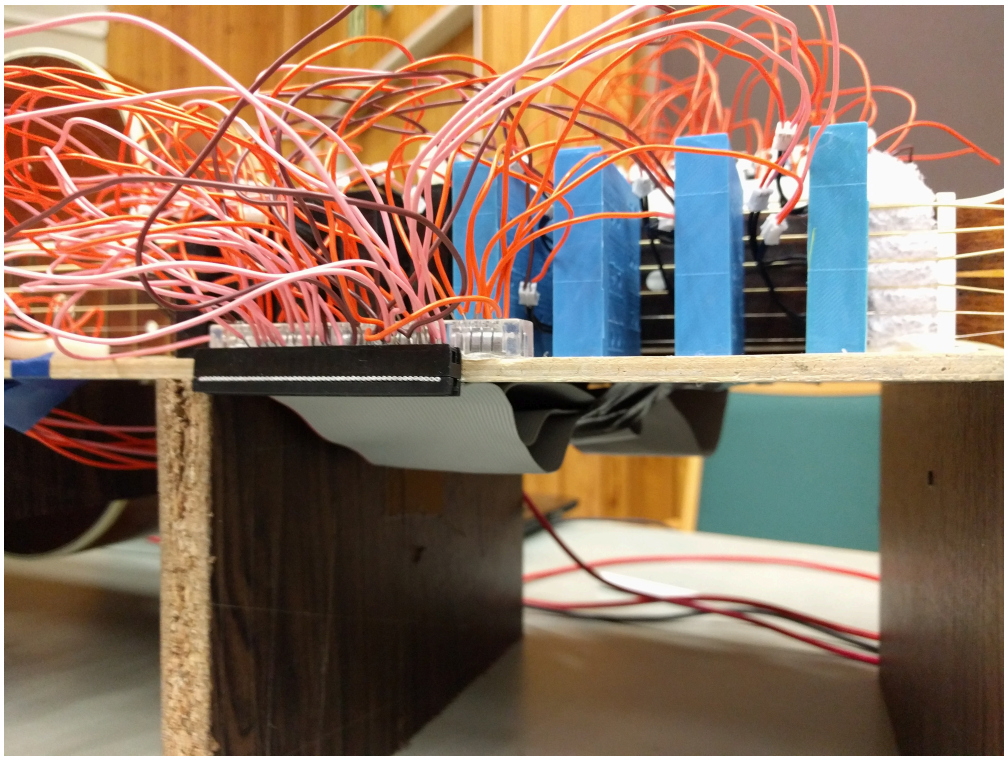
Kytkentäkaavio



Piiri käytännössä. Oranssit johdot eivät mene suoraan solenoideihin, vaan yläreunassa näkyvän IDE-kaapelin kautta kitaran alta.



IDE-kaapeli (harmaa)



Sävellajin tunnistusalgoritmi

Sävellajin tunnistamiseksi käytettiin Krumhansl-Schmuckler -algoritmia. Tämä algoritmi pystyy arvaamaan kappaleen sävellajin nuottien pituuksien perusteella. Aina kun nuotti tulee vastaan, meidän laite lisää nuotin kestoajan datalistaan. Jokaisella nuotilla on oma alkio listassa. Sitten kun painiketta painaa, laite laskee datalistan korrelaation jokaiseen sävellajiprofiiliin. Sitten se vertailee jokaista eri korrelaatiota, ja tulostaa suurimman korrelaation ja sitä vastaavan sävellajin LCD-näyttöön. Carol Krumhansl ja Mark Schmuckler ovat keksineet sävellajiprofiilit algoritmin lisäksi.

http://rnhart.net/articles/key-finding/

Teensy-koodi

Itse kirjoittamamme koodi lukee tietokoneelta tulevan MIDI-syötteen "note on"- ja "note off"-käskyjä reaaliajassa ja kertoo kitaraa soittaville solenoideille, minkä niistä tarvitsee kulloinkin toimia.

Koodi myös tilastoi kahden napinpainalluksen välillä soitetut nuotit, ja sen perusteella laskee välin sävellajin, joka tulostetaan LCD-näyttöön.

Käytimme LCD-näytön ohjaamiseen valmista LiquidCrystal.h -kirjastoa.

Teensy-koodi

```
#include <LiquidCrystal.h>


LiquidCrystal lcd(7, 6, 5, 4, 3, 2);


float majorp[] = {6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39,
3.66, 2.29, 2.88};
float minorp[] = {6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98,
3.69, 3.34, 3.17};
float corr1 = 0;
float corr2 = 0;
unsigned long ontime[108];
float data[12];
float shift[12];
char key1[] = {' ', ' '};


const int pbutton = 9;
boolean buttonpos = LOW;


const int SER_pin = 16;
const int RCLK_pin = 15;
const int SRCLK_pin = 14;


boolean registers[27];


void OnNoteOn(byte channel, byte note, byte velocity) {
  ontime[note] = millis();


  byte solenoid = convertsol(note);
  byte string;
  if (solenoid != 100)  {
    string = convertstr(solenoid);
    closesols(string);
    registers[solenoid] = HIGH;
    writereg();
```

```
  }
}


void OnNoteOff(byte channel, byte note, byte velocity) {
  float Time = ((float)(millis() - ontime[note]))/1000;
  int realnote = getnote(note);
  data[realnote] = data[realnote] + Time;
  //Serial.println(String("note=") + realnote + ", time=" + Time);


  byte solenoid = convertsol(note);
  if (solenoid != 100)  {
     registers[solenoid] = LOW;
     writereg();
     //Serial.println(String("note=") + note + ", solenoid=" + solenoid);
  }
  else  {
    //Serial.println(String("note=") + note + ", out of bounds");
  }
}


void setup()  {
  //Serial.begin(57600);
  usbMIDI.setHandleNoteOff(OnNoteOff);
  usbMIDI.setHandleNoteOn(OnNoteOn);
  lcd.begin(16, 2);
  pinMode(pbutton, INPUT);
  pinMode(SER_pin, OUTPUT);
  pinMode(RCLK_pin, OUTPUT);
  pinMode(SRCLK_pin, OUTPUT);
  writereg();
}


void loop() {
  if (digitalRead(pbutton) == HIGH && buttonpos == LOW)  {
    //printarray(data, 12);
    int major = findkey(majorp, &corr1);
    int minor = findkey(minorp, &corr2);
    lcd.clear();


    if (corr1 < corr2)  {
      //Serial.println(minor + String(" Minor"));
      tostring(minor, key1);
      lcd.setCursor(0, 0);
      lcd.print(key1[0]);
      lcd.print(key1[1]);
      lcd.print(" minor");
      lcd.setCursor(0, 1);
      lcd.print(corr2);
```

```
    }
    else if (corr1 > corr2) {
      //Serial.println(major + String(" Major"));
      tostring(major, key1);
      lcd.setCursor(0, 0);
      lcd.print(key1[0]);
      lcd.print(key1[1]);
      lcd.print(" major");
      lcd.setCursor(0, 1);
      lcd.print(corr1);
    }


    else  {
      lcd.setCursor(0, 0);
      lcd.print("I haven't played");
      lcd.setCursor(0, 1);
      lcd.print("anything yet.");
    }
    //Serial.println(corr1);
    //Serial.println(corr2);
    for (int i = 0; i < 12; i++)  {
      data[i] = 0;
    }
  }
  buttonpos = digitalRead(pbutton);
  usbMIDI.read();
}


void closesols(int string)  {
  int solenoid;
  for (int i = 0; i < 9; i++)  {
    solenoid = 3*i + string;
    if (registers[solenoid] == HIGH)  {
      registers[solenoid] = LOW;
      writereg();
    }
  }
}


byte convertstr(byte solenoid) {
  if (solenoid % 3 == 0)  {
    return 0;
  }
  if ((solenoid - 1) % 3 == 0)  {
    return 1;
  }
  if ((solenoid - 2) % 3 == 0)  {
    return 2;
  }
  return 4;
```

```
}
void tostring(int note, char* key){
  if (note == 0)  {
    key[0] = 'C';
    key[1] = ' ';
  }
  else if (note == 1)  {
    key[0] = 'C';
    key[1] = '#';
  }else if (note == 2)  {
    key[0] = 'D';
    key[1] = ' ';
  }
  else if (note == 3)  {
    key[0] = 'D';
    key[1] = '#';
  }
  else if (note == 4)  {
    key[0] = 'E';
    key[1] = ' ';
  }
  else if (note == 5)  {
    key[0] = 'F';
    key[1] = ' ';
  }
  else if (note == 6)  {
    key[0] = 'F';
    key[1] = '#';
  }
  else if (note == 7)  {
    key[0] = 'G';
    key[1] = ' ';
  }
  else if (note == 8)  {
    key[0] = 'G';
    key[1] = '#';
  }
  else if (note == 9)  {
    key[0] = 'A';
    key[1] = ' ';
  }


  else if (note == 10)  {
    key[0] = 'A';
    key[1] = '#';
  }
  else  {
    key[0] = 'B';
    key[1] = ' ';
  }
}
```

```
byte convertsol(byte note){
  switch (note){
          case 40:
            return 0;
          case 41:
            return 3;
          case 42:
            return 6;
          case 43:
            return 9;
          case 44:
            return 12;
          case 45:
            return 15;
          case 46:
            return 18;
          case 47:
            return 21;
          case 48:
            return 24;
          case 49:
            return 1;
          case 50:
            return 4;
          case 51:
            return 7;
          case 52:
            return 10;
          case 53:
            return 13;
          case 54:
            return 16;
          case 55:
            return 19;
          case 56:
            return 22;
          case 57:
            return 25;
          case 58:
            return 2;
          case 59:
            return 5;
          case 60:
            return 8;
          case 61:
            return 11;
          case 62:
            return 14;
          case 63:
            return 17;
          case 64:
            return 20;
```

```
          case 65:
            return 23;
          case 66:
            return 26;
  }
  return 100;
}


void writereg() {
  digitalWrite(RCLK_pin, LOW);
  for (int i = 30; i >= 0; i--){
    digitalWrite(SRCLK_pin, LOW);
    digitalWrite(SER_pin, registers[i]);
    digitalWrite(SRCLK_pin, HIGH);
  }
  digitalWrite(RCLK_pin, HIGH);
}


int findkey(float* array, float* point)  {
  float corr = -1;
  int key = 0;
  for (int i = 0; i < 12; i++) {
    shiftdata(array, i);
    if (correlation(shift) > corr) {
      corr = correlation(shift);
      key = i;
    }
  }
  *point = corr;
  return key;
}


float correlation(float* array) {
  long numerator = 0;
  long denominator1 = 0;
  long denominator2 = 0;
  float dataave = average(data, 12);
  float arrayave = average(array, 12);
  for (int i = 0 ; i < 12 ; i++) {
    numerator = numerator + (array[i] - arrayave)*(data[i] - dataave);
    denominator1 = denominator1 + pow((array[i] - arrayave), 2);
    denominator2 = denominator2 + pow((data[i] - dataave), 2);
  }
  float R = numerator / pow((denominator1*denominator2), .5);
  return R;
}


void shiftdata(float* array, int numb)  {
  if (numb <= 12)  {
```

```
    for (int i = 0; i < 12; i++)  {
      if (i - numb < 0){
        shift[(i - numb) + 12] = array[i];
      }
      else  {
        shift[i - numb] = array[i];
      }
    }
  }
}


float average(float* array, int len)  {
  long sum = 0L ;
  for (int i = 0 ; i < len ; i++) {
    sum += array[i] ;
  }
  return  ((float) sum) / len ;
}


byte getnote(byte note) {
  for (byte i = 0; i < 12; i++) {
    if ((note - i) % 12 == 0)  {
      return i;// C:0 C#:1 D:2 D#:3 E:4 F:5 F#:6 G:7 G#:8 A:9 A#:10 B:11
    }
  }
  return 12;
}


void printarray(float* array, int len)  {
  Serial.print(String("[ "));
  for (int i = 0; i < len; i++) {
      Serial.print((int)array[i] + String(" "));
    }
  Serial.println(String("]"));
}
```

Haasteita

Alusta asti suurin haasteemme oli solenoidien leveys - kuusi solenoidia rinnakkain oli leveämpi kuin kuusi kitaran kieltä. Halusimme käyttää kuutta kieltä, ja yritimmekin ensiksi valmistaa sellaisen solenoidikehikon, jossa solenoidit olivat limittäin siten, että kolme niistä oli askeleen taaempana ja niille annettiin jatkopaloilla lisäulottuvuutta. Emme kuitenkaan saaneet jatkopaloja pysymään kiinni kunnolla, ja päädyimme lopulta kolmen kielen ratkaisuun.

Jotta solenoidien voima riittäisi kielten painamiseen, niille täytyi antaa reilusti suosituksia suurempi käyttöjännite. Se ei vahingoittanut solenoideja, mutta aiheutti solenoidien ja transistorien selvää kuumenemista.

Useat kappaleet aiheuttivat haasteita siinä, että jotkin niiden nuoteista olivat liian matalia tai korkeita käyttämäämme skaalaan nähden. Meillä ei ollut muuta vaihtoehtoa kuin jättää sellaiset muotit soittamatta. Joissain kappaleissa myös kolme yhtäaikaista nuottia oli liian vähän, ja soinnut olivat siis epätäydellisiä. Näistä seikoista johtuen osa kappaleista kuulosti oudoilta tai kokonaisia osuuksia jäi kokonaan soittamatta. Nämä olivat kuitenkin projektin puitteissa ylitsepääsemättömiä fyysisiä rajoitteita, ja pyrimme valitsemaan soitettavat kappaleet nämä rajoitteet huomioon ottaen.

Lopputulos

Projekti saatiin toimimaan hyvin ja soittamaan kappaleita tunnistettavasti niin pajalla kuin loppunäytöksessäkin. Soittaminen onnistui MIDI-tiedostojen lisäksi myös tietokoneen näppäimistöllä ja MIDI-koskettimistolla (LMMS-ohjelman kautta). Jotkin kappaleet kuulostivat oudoilta projektin fyysisistä rajoitteista johtuen (katso kohta *Haasteita*).

Vaikka nuotit olivatkin kuuluvia ja selkeitä, ei kitaran äänensävy ollut kovin kaunis, vaan terävä ja metallinen, erään kuuntelijan mukaan "cembalomainen". Lisäksi solenoidit aiheuttivat metallisia kilahduksia kytkeytyessään päälle ja pois. Osa solenoideista ei yltänyt painamaan kieliä täysin kitaran otelautaa vasten, jolloin solenoidin pää toimi ikään kuin nauhana. Tämän tuloksena niiden solenoidien nuotit olivat hieman alavireisiä.

Kuitenkin projekti loisti nopeissa juoksutuksissa ja "tilutuksessa". Testauksissa projekti soitti nuotteja jopa vain 10 ms välein, mihin ei ihminen enää kykene (ääni tosin kuulosti kauhealta).

Kitaran alta kulkevan IDE-kaapelin ansiosta kitara oli helppo ottaa pois ja asettaa takaisin kehikkoon. (Tämä ei kuitenkaan mahdollista minkä tahansa kitaran käyttöä, sillä kitaroissa kielten välinen etäisyys ja nauhojen välinen etäisyys vaihtelevat jonkin verran.)

Videoita

Nikolai Rimsky-Korsakov - Kimalaisen lento

Led Zeppelin - Stairway to Heaven

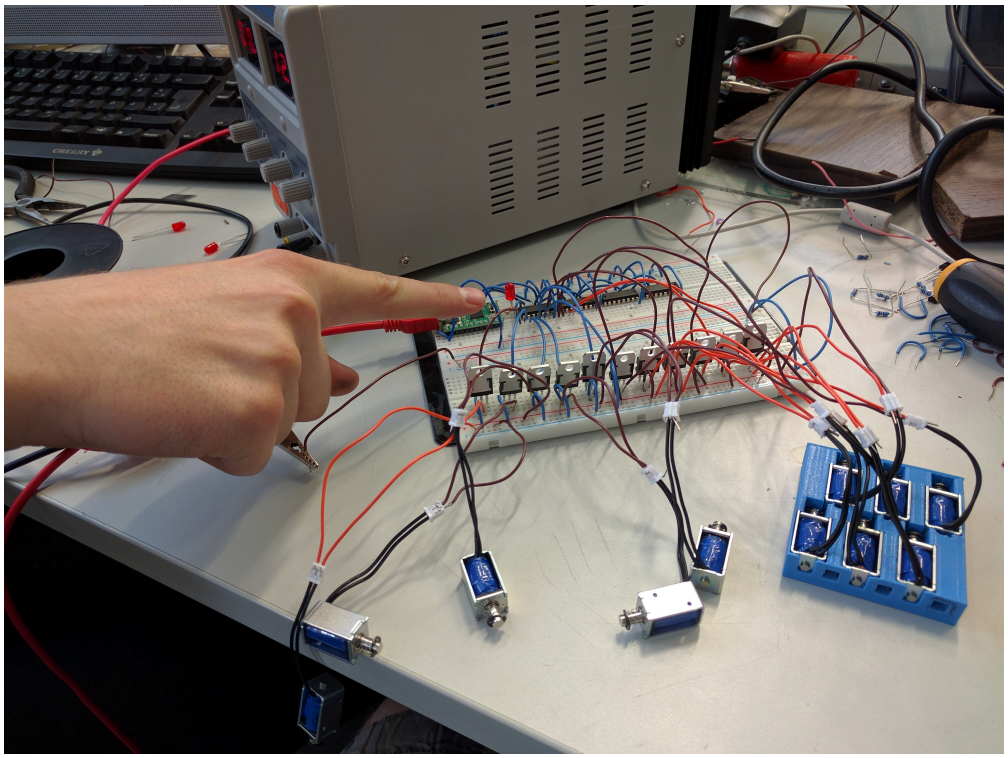
Scott Joplin - The Entertainer

"Soittakaa Paranoid!"

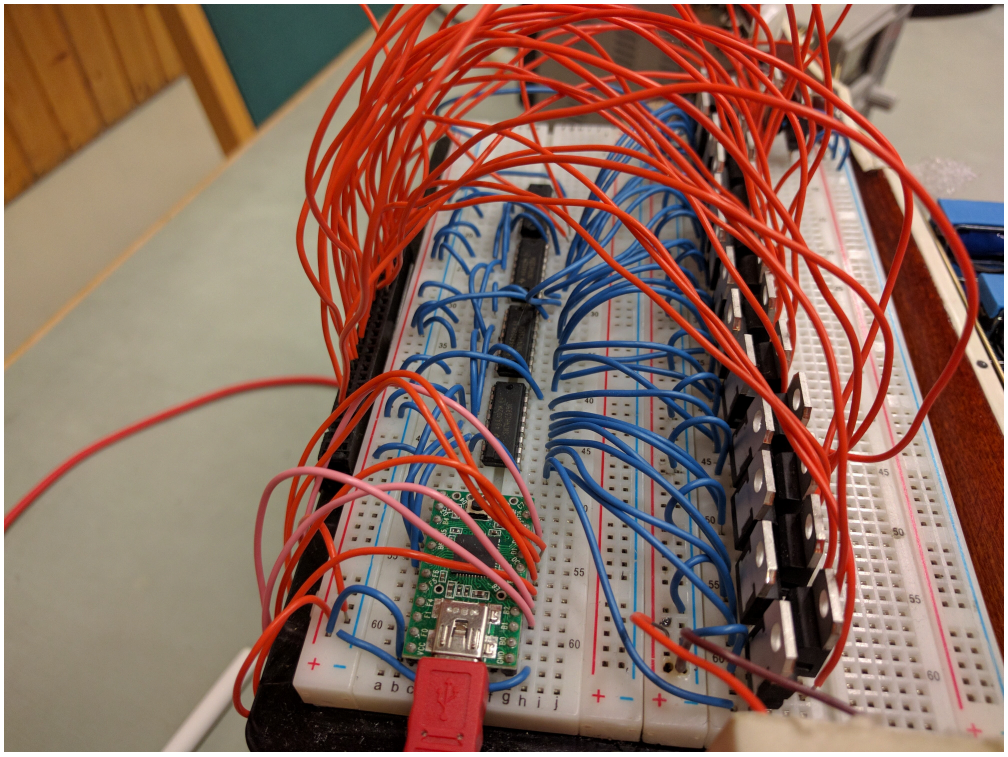
Kuvia

LCD-näytön kytkentä

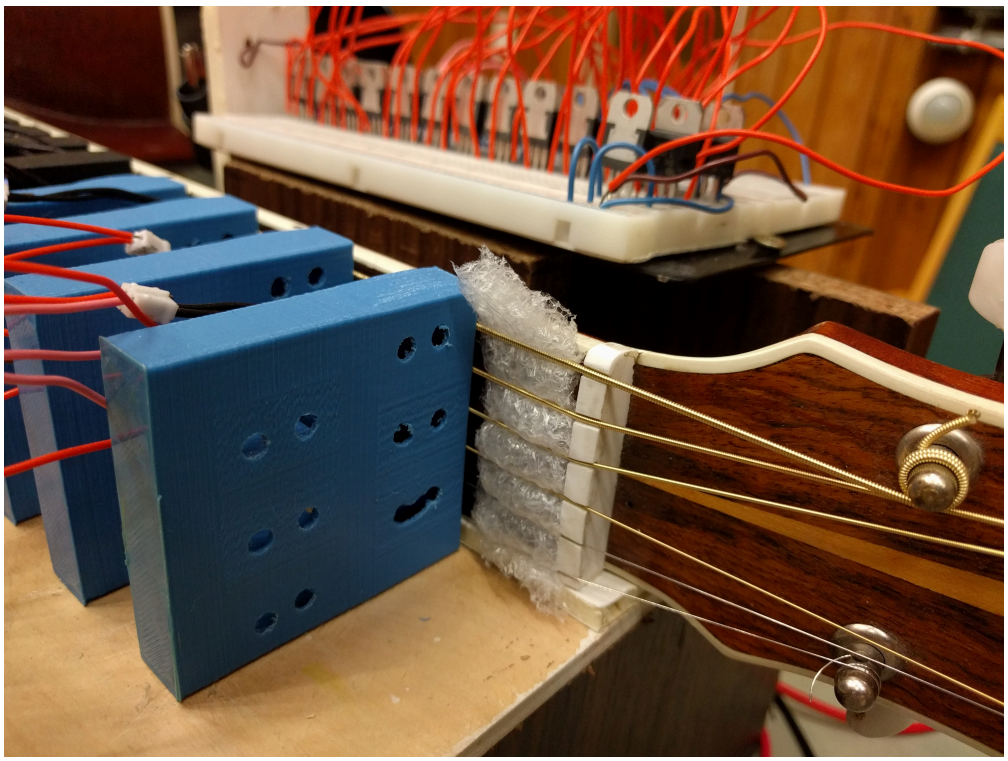
Solenoidien testausta kehikossa



Valmis piiri toisesta kulmasta



Vaahtomuovisammutin kielten alla



Kehittämisajatukset

Jos haluaisimme parantaa projektia tulevaisuudessa, me hankkisimme kapeampia ja ulottuvampia solenoideja, joilla saisimme kaikki kuusi kieltä käyttöön (ratkaisuna tähän pohdittiin myös olemassaolevien solenoidien vinottaista asettelua). Ottaisimme myös käyttöön kitaran nauhat ainakin 12. asti. Kehittäisimme myös jonkin ratkaisun pehmentämään solenoidien metallisia kilahduksia kieliä vasten.

Projektin ulkonäkö jäi myöskin keskeneräisen näköiseksi johtolinnunpesäksi. Kuitenkin sen ansiosta oli mahdollista työtä vilkuilemalla päästä

selville toimintaperiaateesta. Jonkinlainen kotelointi parantaisi ulkonäköä.

Soitimme Paranoidin!!