Mladi za napredek Maribora 2019
36. srečanje

MIDI DMX krmilnik

Raziskovalno področje: **ELEKTROTEHNIKA, ELEKTRONIKA**

Raziskovalna naloga

PROSTOR ZA NALEPKO

Avtor: KEVIN BARON LAH, MARK VEZJAK, MATIC ŠULC

Mentor: IVANKA LESJAK
Šola: SREDNJA ELEKTRO-RAČUNALNIŠKA ŠOLA MARIBOR
Število točk: 154
Mesto: 7
Priznanje: srebrno

Maribor, februar 2019

Mladi za napredek Maribora 2019
36. srečanje

MIDI DMX krmilnik

Raziskovalno področje: **ELEKTROTEHNIKA, ELEKTRONIKA**

Raziskovalna naloga

PROSTOR ZA NALEPKO

Maribor, februar 2019

KAZALO VSEBINE

KAZALO SLIK

KAZALO GRAFOV

1. ZAHVALA

Zahvaljujemo se vsem, ki so pripomogli pri izdelavi te raziskovalne naloge. Sprva mentorici, za vso pomoč in podporo pri pisanju naloge. Posebna zahvala gre tudi podjetju Laser Tehnik, za pomoč in izdelavo ohišja za krmilnik. Predvsem pa gre zahvala tudi šoli, za vso finančno podporo, brez katere bi bila realizacija naloge verjetno neuspešna.

2. POVZETEK

Cilj naše raziskovalne naloge je izdelava DMX (digitalno multipleksiranje) krmilnika, ki bo na lahko komuniciral z računalniškimi programi preko MIDI (Musical Instrument Digital Interface) signalov. Krmilnik bomo izdelali s pomočjo Arduino mikrokrmilnika, ki bodo poganjali jedro sistema. Za uporabniški vnos pa bomo uporabljali stikala(MX Cherry), linearne potenciometre in rotacijske kodirnike. Ohišje krmilnika bo izdelano po meri, pri podjetju Laser Tehnik.

Za to raziskovalno nalogo smo se odločili, saj so naše izkušnje z večjimi proizvajalci bolj kot ne negativne. Večji problem krmilnikov je njihova prenosljivost in cena, saj nizkocenovni proizvodi (npr. Showtech ipd.) po navadi ne ponujajo takšne fleksibilnosti glede dela, saj nas omejuje glede funkcionalnosti. Na drugi strani pa se pojavljajo dražji ponudniki(npr. MA Lighting, Avolites, ETC Lighting), ki po navadi potrebujejo napravo, ki poganja zunanjo programsko opremo ali pa niso prenosni zaradi svoje teže.

Po mnenju uporabnikov je večji problem tudi cena, saj se krmilniki takšne velikosti prodajajo od 300€ in vse do 90.000€ pri večjih ponudnikih, zato si manjše korporacije ali posamezniki tega ne morejo privoščiti.

Največja prednost našega krmilnika je, da je kompatibilen z vsemi programi ki za svoje delovanje uporabljajo MIDI signale. Sprva smo se mislili omejiti le na osvetlitev, a smo med pisanjem algoritmov ugotovili, da se ga lahko uporablja tudi kot samostojna MIDI naprava, npr. glasbeni instrument.

3. HIPOTEZE

Hipoteza 1: Celoten izdelek bo cenovno ugoden za cilje uporabnike
Hipoteza 2: Program bo napisan v jeziku C#
Hipoteza 3: Uporabili bomo zunanjo bazo podatkov za prijavo v sistem
Hipoteza 4: Ohišje za izdelek bomo lahko izdelali sami

4. VSEBINSKI DEL

V vsebinskem delu naše raziskovalne naloge bomo opisali in predstavili sestavne dele našega izdelka ter prikazali bomo programsko kodo, ki je odgovorna za delovanje le tega krmilnika.

4.1 MIDI Krmilniki

MIDI krmilniki so naprave, ki posameznikom omogočajo pošiljanje digitalnih impulzov, katere programski vmesnik v računalniku lahko pretvori v glasbo ali druge signale po naših željah.

Ohišje našega izdelka je specifično prilagojeno za uporabo pri osvetlitveni tehniki, a z nekaj modifikacijami ga lahko uporabnik spremeni v praktično karkoli.

Ker se tudi sami ukvarjamo s osvetlitveno tehniko se zavedamo problemov, ki nastajajo pri obstoječih krmilnikih: cena in teža. Redki proizvajalci so uspeli na trg dostaviti prenosljivost, a so večinoma za to postavili ceno, ki si jo posameznik redko lahko sam privošči.

Naš MIDI krmilnik omogoča uporabniku, da se prosto sprehaja po prostoru, kjer dela. S tem pridobi veliko prednosti, npr. pogled kot občinstvo, premikanje luči kjerkoli se nahaja s katerokoli mobilno napravo...

4.2 Princip delovanja

Po tehtnem premisleku smo se odločili, da postavimo jedro našega sistema v razvojne ploščice Arduino Mega. To nam omogoča sprotno programiranje samega sistema, ter olajšano pisanje samih algoritmov, saj nam Arduino ponuja mnogo pripravljenih knjižnic.

Vsak Arduino ima na sebe priključen tudi »network shield«, kar nam omogoča povezavo preko UTP kabla in implementacijo omrežja.

Na vsaki napravi je nameščena modificirana verzija »Firmata« protokola, ki omogoča komunikacijo z zunanjimi napravami in dostop do pin-ov brez posredovanja mikrokrmilnika. S tem mnogo izboljšamo procesne čase, saj so zunanje procesne naprave mnogo hitrejše od teh razvojnih ploščic.

Ko smo vsaki napravi določilo svoj »MAC« naslov in vrata, na katerih naj posluša/oddaja pakete, smo bili pripravljeni na razvoj strežnika, ki bo vse skupaj poganjal.
Odločili smo se za uporabo ogrodja Node.JS, ker omogoča implementacijo mnogih protokolov.

Za spletno aplikacijo uporabljamo strežnik Express, ki je implementiran vanjo. Spletna stran je izdelana po meri, izključno v HTML5, CSS in JavaScript-u.
Ko uporabnik premakne potenciometer na krmilniku se izvede t.i »websocket« metoda ko pošlje podatke k uporabniku na spletno stran. To se v obratni smeri izvede tudi, ko uporabnik vnese nekaj preko same spletne strani,
Vse animacije se izvajajo dinamično in sočasno na vseh napravah, ki so povezane v omrežje.

4.3 Arduino MEGA

Arduino, ali drugače imenovan tudi Genuino, je razvojna ploščica, s katero lahko upravljamo različno vrsto naprav. Arduino ponuja več vrst razvojnih ploščic, kot npr. Uno za začetnike ali Mega za napredno uporabo. V našem primeru, bo Arduino Mega prišel prav, saj moramo priključiti več stikal in priključkov.

Arduino Mega je razvojna plošča z mikrokrmilnikom, ki temelji na ATmega2560. Ima na voljo do zdaj največje število digitalnih vhodov in izhodov – 54, 16 analognih vhodov, 4 strojne serijske priključke, 16Mhz oscilator, USB in napajalni priključek.

Za naš izdelek smo uporabili 4 razvojne ploščice, saj jih potrebujemo za povezavo vseh stikal in priključkov.

Slika 1: Arduino Mega (vir: Arduino Store)

4.4 Vhodne enote

Za vhodne enote našega izdelka smo izbrali linearne potenciometre, stikala MX Cherry blue in rotacijske kodirnike, saj slednje uporablja tudi večina orodij za delo v tej stroki.

4.4.1 Cherry MX Blue stikala

Na trgu obstaja mnogo različnih vrst stikal, ki jih proizvajalci uporabljajo pri izdelavi tipkovnic. Najbolj pogosti sta mehanično in membransko. Membranska stikala, najdena v večini cenejših tipkovnicah imajo namesto mehanskega stikala le prevodno črnilo, ki ob stiku z vezjem sklene potrebne kontakte. Mehanska stikala pa so zgrajena iz vzmeti, plastike in dveh pozlačenih metalnih kontaktov, ki se skleneta ob kliku gumba. Delimo jih na dve večji skupini, linearna in taktilna. Pri taktilnih občutimo, kdaj je tipkovnica klik registrirala, pri linearnih pa ne. Mehanske tipkovnice se največkrat uporabljajo za igranje iger ali pisanje daljših besedil, saj omogočajo večjo učinkovitost.

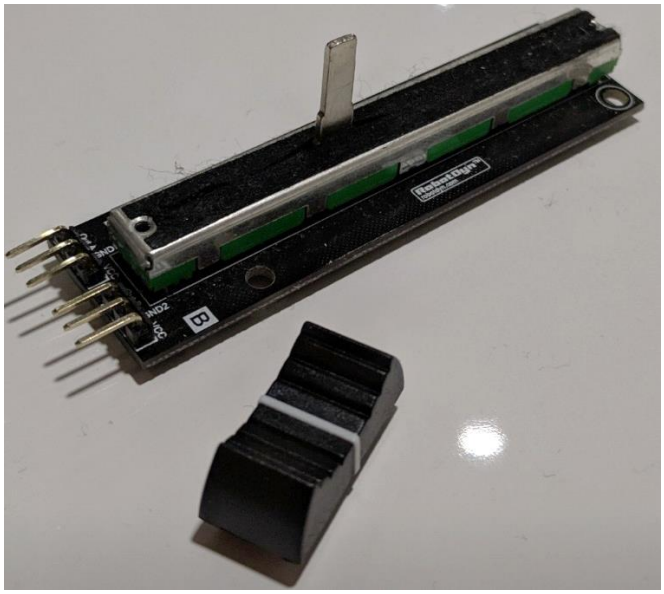
Največje podjetje, ki izdeluje mehanična stikala je MX Cherry. Vsa njihova stikala imajo 4mm hoda in se aktivirajo pri 2mm.
Najbolj razširjena so:

- Črna (linearna, uporabljena med »gamerji«)
- Rjava (taktilna, tišja, kombinacija med stikali za gaming in pisanje)
- Modra (taktilna, najglasnejša, najprimernejša za pisanje)
- Prozorna (podoben občutek membranskim, redko uporabljena)
- Rdeča(linearna, najpogosteje uporabljena v »gaming« tipkovnicah)

Slika 2: Mehansko stikalo CHERRY MX Blue (vir: Avtor naloge)

4.4.2 Linearni potenciometer

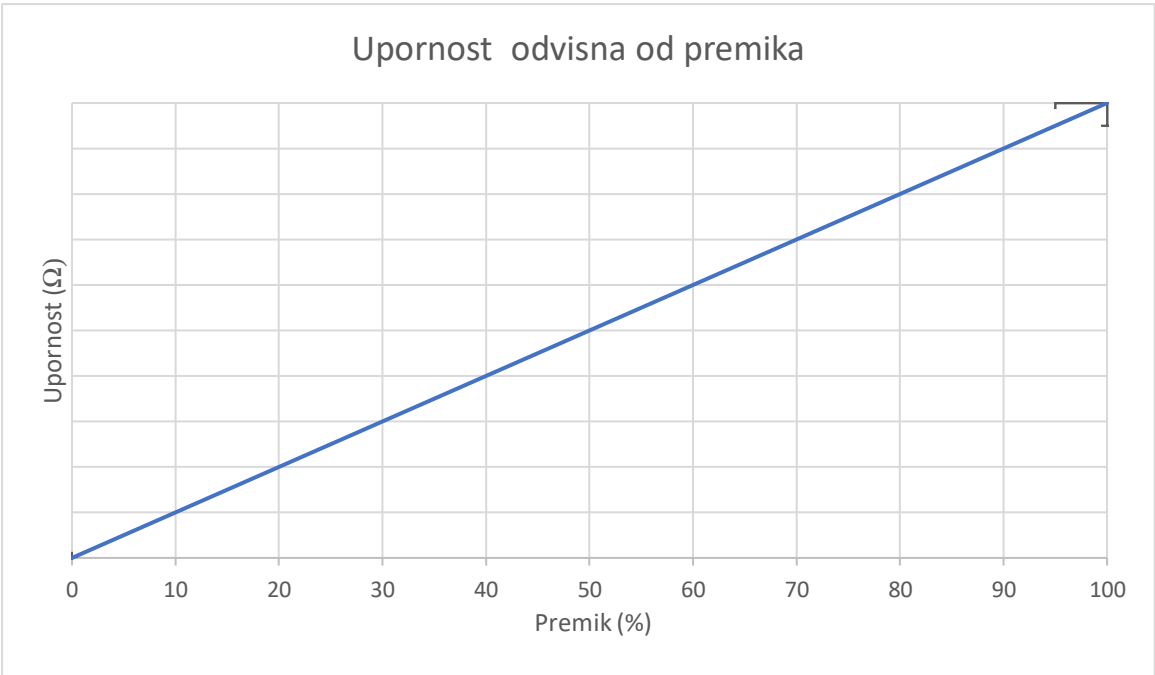
Potenciometer je delilnik napetosti s tremi terminali, ki je lahko ali drseči ali pa rotacijski. Ker bo potrebno pošiljati MIDI signale, smo v našem primeru uporabili **linearne potenciometre**.



Slika 3: Linearni potenciometer (vir: Avtor naloge)

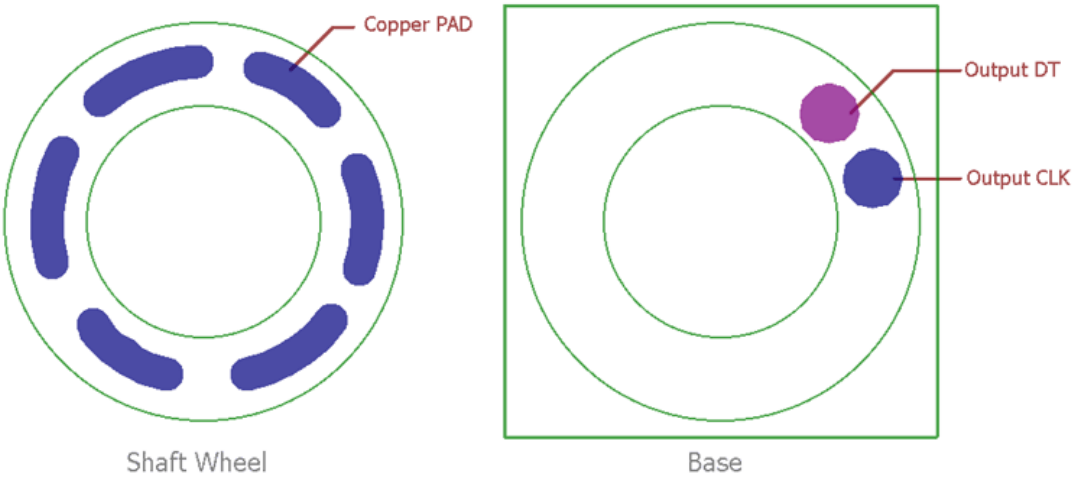
Linearni potenciometer se uporablja za detekcijo linearnih premikov. Uporablja se tam, kjer je zaželena napetost na izhodu, ki je linearno odvisna od premika.

Graf 1: Delovanje linearnega potenciometra(vir: Avtor naloge)

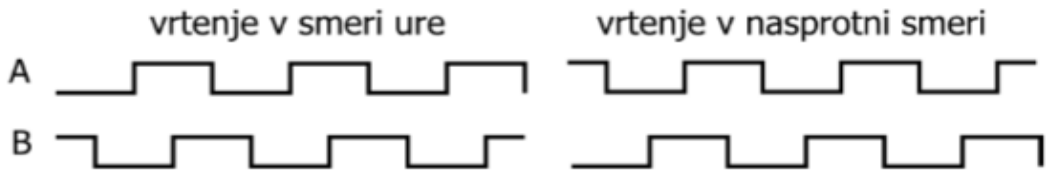


4.4.3 Rotacijski kodirnik

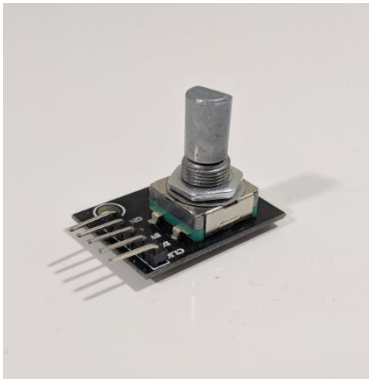
Rotacijski kodirnik je elektro-mehanska naprava, ki pretvarja fizično kotno pozicijo elementa v analogne oz. digitalne signale. Na koleščku so nameščene bakrene podlage, katere imajo stik z bazo(base), ki se nahaja pod njim. Na bazi se nahajata dva priključka po katerih kolešček oz. bakrene podlage drsajo kadar ga obračamo.



Slika 4: Sestava rotacijskega kodirnika (vir: circuitdigest.com)

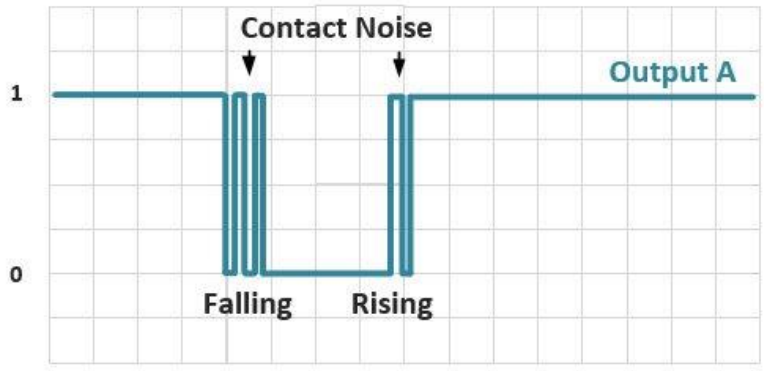


Slika 5: Signali rotacijskega kodirnika (vir: lniv.fe.uni-lj.si)



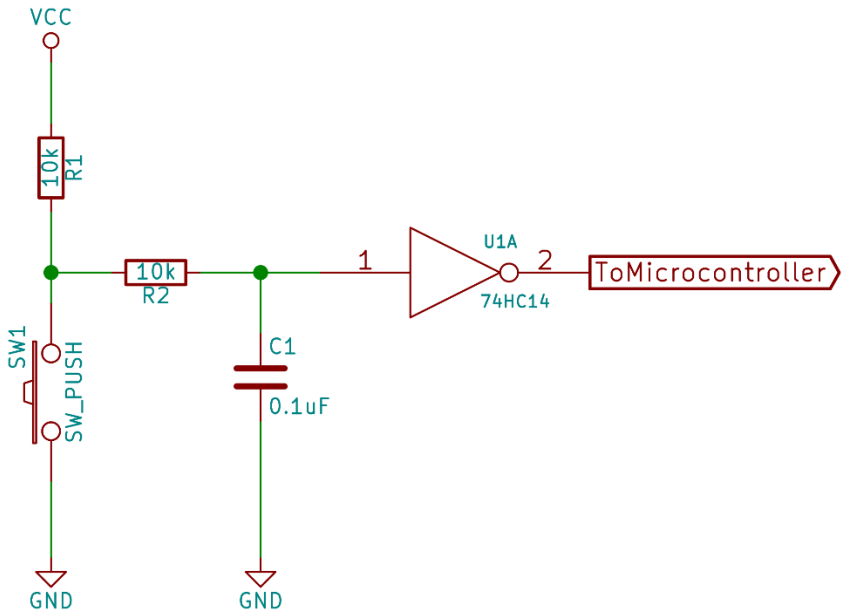
Slika 6: Rotacijski kodirnik (vir: Avtor naloge)

Ko smo prvič preizkusili nas rotacijski kodirnik smo ugotovili, da izhodni signal ni bil enakomeren. Kmalu smo ugotovili, da ima to težavo večina rotacijskih kodirnikov. Pri vrtenju gumba pride do popačenega signala oz. tako imenovanega "Contact Noise". To lahko preprečimo s tako imenovanim "Signal Debouncing". Ko zavrtimo gumb, prejmemo signal, ki izgleda popačeno.



Slika 7: Primer popačenega signala (vir: allaboutcircuits.com)

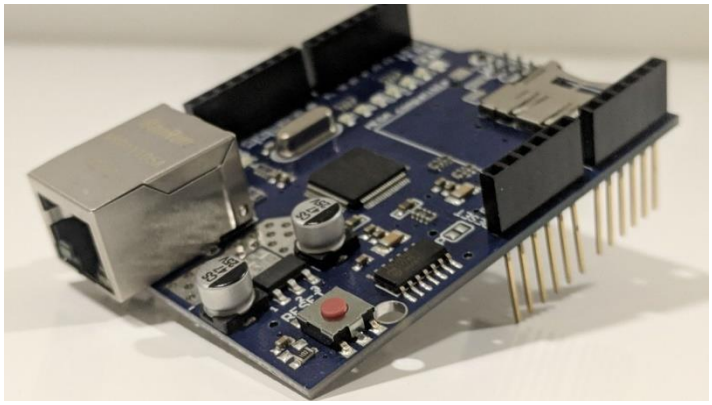
To težavo je možno odpraviti sistemsko, vendar ker smo uporabili Firmato kot nas protokol za komuniciranje smo si to pot onemogočili. Zato smo se odločili to težavo odpraviti strojno z "hardware debouncing circuit". Uporabili smo tako imenovani "hex inverting" Schmitt trigger (74HC14), kondenzator in dva upora.



Slika 8: Odprava popačenih signalov na rotacijskem kodirniku (vir: hackday.com)

4.5 Povezava z napravami

Za povezavo z napravami smo uporabili EtherPort, ki ga lahko uporabimo skupaj z Ethernet modulom za Arduino. Uporabljamo ga lahko s protokolom za komunikacijo »Firmata«, mi smo se odločili za uporabo modificirane verzije, ki omogoča več vrst povezav.
Vsaki napravi kot argument lahko določimo ID, vsaka EtherPort povezava pa deluje na različnih lokalnih vratih, ki jih določimo ko nalagamo kodo na samo napravo.



Slika 9: Modul "Ethernet Shield" za Arduino (vir: Avtor naloge)

```
var portsDict = [
{ id: "A", repl: false, port: new EtherPort(3030)},
{ id: "B", repl: false, port: new EtherPort(3031)},
{ id: "C", repl: false, port: new EtherPort(3032)},
{ id: "D", repl: false, port: new EtherPort(3033)}
];


new five.Boards(portsDict).on("ready", function(){
```

Slika 10: Povezava na Arduino (vir: Avtor naloge)

Na posamične naprave smo v okolju Arduino naložili modificirano verzijo protokola »Firmata«, ki smo jo pridobili s spletne strani http://firmatabuilder.com/.
Pridobljeno kodo smo morali malce spremeniti, saj smo uporabljali 4 različne naprave, ki bi lahko povzročale motnje med sabo.

V kodi smo morali fizično določiti MAC naslove naših naprav z namenom lažje prepoznavnosti v omrežju. Odločili smo se za naslove AA:AA:AA:AA:AA:(00 do 04). Vsaki napravi smo morali določiti tudi port(vrata) na katerem je komunicirala s strežnikom, v našem primeru so to od 3030 do 3034. Za konec smo vsaki napravi morali vpisati lokalni IP naslov računalnika, na katerem se je izvajala koda.

```
const byte mac[] = {0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0x00};
int remotePort = 3030;
IPAddress remoteIp(129, 156, 12, 176);
EthernetClientStream stream(client, IPAddress(0, 0, 0, 0), remoteIp, NULL, remotePort);
```

Slika 11: Modificirana verzija protokola Firmata (vir: Avtor naloge)

4.6 Uporaba zunanjih zbirk za vključitev elementov

Z namenom skrajšanja kode in s tem doseči čim krajše izvedne čase, smo se lotili izziva, da bomo gumbe in potenciometre v kodo vključevali z zunanjimi datotekami in jim tudi tam določali argumente.

```
"use strict";
var btn_call = function (boards) {

var buttonDict = [
//{pin: 0 , board: boards.byId("A"), id: 11}, serial rx
//{pin: 1 , board: boards.byId("A"), id: 12}, serial tx
//{pin: 2 , board: boards.byId("A"), id: 13},
{pin: 3 , board: boards.byId("A"), id: 14},
//{pin: 4 , board: boards.byId("A"), id: 15}, for shield
{pin: 5 , board: boards.byId("A"), id: 16},
{pin: 6 , board: boards.byId("A"), id: 17},
{pin: 7 , board: boards.byId("A"), id: 18},
{pin: 8 , board: boards.byId("A"), id: 19},
{pin: 9 , board: boards.byId("A"), id: 20},
```

Slika 12: Zunanji klic na zbirko gumbov (vir: Avtor naloge)

Odločili smo se za uporabo zunanjega klica na funkcijo, ki bo vsebovala vse potrebne elemente, ter sprejela argument, ki nam bo povedal na kateri ploščici leži fizični element(»boards«). Zaradi uporabe »network shield«-ov, smo morali odpisati pine 0, 1(serijska komunikacija) ter 4, 10, 50, 51, 52 in 53. Le te vrstice smo za lažjo prepoznavnost komentirali iz kode.

```
 ];
 return buttonDict;
};
module.exports.btn_call = btn_call;
```

Slika 13: Zunanji klic na zbirko gumbov (vir: Avtor naloge)

Funkcijo smo izvozili iz dokumenta z uporabo module.exports vgrajenega v Node.js in kot njen rezultat vračali celotno zbirko.

```
var pot_call = function(boards){
var potsDict = [
{pin: "A0",board: boards.byId("A"),threshold: 6, id: 1 }, //Treshold smo postavili na 6, saj so potenciometri včasih pošiljali manjše signale
{pin: "A1",board: boards.byId("B"),threshold: 6, id: 2 },
{pin: "A2",board: boards.byId("A"),threshold: 6, id: 3 },
{pin: "A3",board: boards.byId("A"),threshold: 6, id: 4 },
{pin: "A4",board: boards.byId("A"),threshold: 6, id: 5 },
{pin: "A5",board: boards.byId("A"),threshold: 6, id: 6 },
{pin: "A6",board: boards.byId("A"),threshold: 6, id: 7 },
{pin: "A7",board: boards.byId("A"),threshold: 6, id: 8 },
{pin: "A8",board: boards.byId("A"),threshold: 6, id: 9 },
{pin: "A9",board: boards.byId("A"),threshold: 6, id: 10}
];
return potsDict;
};
module.exports.pot_call = pot_call;
```

Slika 14: Zunanji klic na zbirko potenciometrov (vir: Avtor naloge)

Enako smo storili tudi za potenciometre. Vsem smo določili šifro z imenom »id«, da smo z njimi lažje upravljali kasneje.

```
var buttons = require("./dict.js").btn_call;
var pots = require("./dict.js").pot_call;
```

Slika 15: Klic zunanje zbirke elementov (vir: Avtor naloge)

Kasneje smo v osnovni skripti z uporabo »require« poklicali funkcije, ki smo jih izvozili. V nadaljevanju smo lahko s spremenljivko »buttons« klicali celotno knjižnico gumbov in jih uspešno inicializirali.

4.7 Delovanje spletnega vmesnika

Naš spletni vmesnik je v celoti postavljen na strežniku »Express« v ogrodju node.js. Za delovanje le tega, potrebujemo knjižnico »Express«. Knjižnico lahko v programu kličemo tako, da ji dodamo spremenljivko.

```
var express = require('express');
var app = express();
```

Slika 16: Inicializacija vtičnika "Express" (vir: Avtor naloge)

Strežnik deluje na privzetih vratih 80. Če želimo uporabljati zunanje datoteke, jim moramo določiti statično pot, mi smo jim dodelili pot z imenom /public, zato vse shranjujemo v le to mapo.

```
app.get('/',function(req,res) {
  res.sendFile('index.html', { root: __dirname }); //nastavi index.html za privzeto stran ob zagonu
});

app.use(express.static(__dirname + '/public/')); //statična pot za zunanje datoteke(css, js...)
app.listen(80, () => console.log('Listening on port 80!')); //uporaba vrat 80 za strežnik
```

Slika 17: Vklop spletnega strežnika (vir: Avtor naloge)

Ko se spletna stran inicializira, se vklopi tudi naš vtičnik »socket.io«, ki skrbi za prenos podatkov od strežnika do uporabnikov. Dodamo mu spremenljivko, da ga lažje uporabljamo v nadaljnjem kodiranju.

```
var io = require('socket.io').listen(8081);
```

Slika 18: Inicializacija vtičnika socket.io na vratih 8081 (vir: Avtor naloge)

```
<script src="http://192.168.0.100:8081/socket.io/socket.io.js"></script>
```

Slika 19: Uporaba socket.io na spletni strani (vir: Avtor naloge)

```
var socket = io('http://192.168.0.100:8081');
```

Slika 20: Povezava na strežnik (vir: Avtor naloge)

Do spletnega vmesnika lahko nato dostopamo z lokalnim naslovom računalnika, na katerem se izvaja program. V našem primeru je to "localhost".
Ko pridemo na vstopno stran, nas pričaka prijavni obrazec.
Prijavimo se lahko samo z uporabniškim imenom in geslom, ki smo ga pridobili od lastnika orodja. V primeru napake nam računalnik ustavi skripto ali pa nam izpiše, da so podatki napačni.

Ko uporabnik spremeni vrednost(npr. premakne potenciometer, klikne gumb) se le ta zazna preko integrirane funkcije v našem ogrodju »johnny-five« ter izvede želene ukaze.

```
var pot = new five.Sensors(pots(this));

pot.on("change", function(pots) {
    console.log("Pot " + pots.pin + " has a value of: " + pots.scaleTo(0, 127) + " id: " + pots.id);
    midi_out.sendMessage([176,pots.pin,pots.fscaleTo(0, 127)]); //pošlje MIDI signal
    io.emit('pot_change', {pin: pots.pin, value: pots.fscaleTo(0, 127), id: pots.id}); //pošlje signal preko socket.io na druge naprave
  });
```

Slika 21: Zaznava fizičnega premika potenciometra (vir: Avtor naloge)

Za prepoznavo klika gumbov smo uporabili dve funkciji, saj smo naleteli na težavo, pri kateri bi uporabnik držal gumb več sekund, bi prišlo do podvajanja vrednosti.

```
var btn = new five.Buttons(buttons(this));

 btn.on("press", function(buttons) {
        console.log( "Button with id", buttons.id, "has a value of:",   buttons.value );
        io.emit('button_change', {id: buttons.id, value: 1}); //pošlji na računalnike
        if(data.id <= 50 )
            midi_out.sendMessage([190,buttons.id , 1]); //prvi arduino
        else if(data.id > 50 && data.id <= 100)
            midi_out.sendMessage([191,buttons.id , 1]); //drugo arduino
        else if(data.id > 100 && data.id <= 150)
            midi_out.sendMessage([192,(buttons.id-100), 1]); //tretji arduino
        else if(data.id > 150)
            midi_out.sendMessage([193,(buttons.id-100), 1]); //četrti arduino
  });

 btn.on("release", function(buttons) {
        console.log( "Button with id", buttons.id, "has a value of:",   buttons.value );
        io.emit('button_change', {id: buttons.id, value: 0}); //pošlji na računalnike
        if(data.id <= 50 )
            midi_out.sendMessage([190,buttons.id, 0]); //prvi arduino
        else if(data.id > 50 && data.id <= 100)
            midi_out.sendMessage([191,buttons.id, 0]); //drugi arduino
        else if(data.id > 100 && data.id <= 150)
            midi_out.sendMessage([192,(buttons.id-100), 0]); //tretji arduino
        else if(data.id > 150)
            midi_out.sendMessage([193,(buttons.id-100), 0]); //četrti arduino
  });
```

Slika 22: Zaznava klika gumbov (vir: Avtor naloge)

Ko strežnik pošlje podatek z ukazom »socket.emit« ga moramo na strani uporabnika sprejeti. To storimo z ukazom socket.on (»ime_podatka«, function(data) { ... }); kjer lahko v skupku podatkov »data« izbiramo posamezne, ki smo jih poslali z uporabo selektorja ».«, npr. data.id, data.value...

```
socket.on('pot_change', function(data){ //data je skupek podatkov, ki smo jih poslali z ukazom socket.emit
    console.log("Potentiometer with pin: ", data.pin, "has a value of: ", data.value, " and id: ", data.id);
    document.getElementById(data.id).value = data.value; //nastavi prejeto vrednost za potenciometer, ki smo ga premaknili
});
socket.on('button_change', function(data){
    console.log('Button with id: ', data.id, ' has a value of: ' , data.value);
    document.getElementById(data.id).value = data.value;
    if(data.value == 1){
    document.getElementById(data.id).style.color = "red"; //če je gumb pritisnjen, je vrednost 1 in se pobarva rdeče
    }
    else{
    document.getElementById(data.id).style.color = "white"; //če gumb spustimo dobi vrednosti 0 in pobarva se belo
    }
});
```

Slika 23: Prikaz podatkov na spletnem vmesniku (vir: Avtor naloge)

Za večjo funkcionalnost same aplikacije smo z nekaj truda omogočili tudi pošiljanje podatkov v obeh smereh, torej lahko uporabnik uporablja elemente tudi na spletnem vmesniku. Za to smo ponovno uporabili socket.io, le določili smo drugačna imena za ukaze, da ne bi trčili drug v drugega.

```
<div class="slidecontainer">
<input class="slider" id="1" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="2" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="3" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="4" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="5" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="6" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="7" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="8" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="9" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
<input class="slider" id="10" type="range" min="0" max="127" step="1" oninput="socket.emit('pot_change_html', {id: id, value: this.value});">
</div>
```

Slika 24: Pošiljanje podatkov ob premiku potenciometra na spletnem vmesniku

Sprva smo mislili za zaznavo spremembe potenciometra uporabiti JQuery, a po krajšem testiranju smo odkrili, da vključen element »onchange« deluje kot smo želeli in ga uporabili.

```
$(":button").on({
  mousedown: function(){
    console.log('BROWSER button ', this.id, 'clicked');
    socket.emit('button_click_html', {id: this.id, value: 1} );
    document.getElementById(this.id).style.color = "red";
  },
  mouseup: function(){
console.log('BROWSER button ', this.id, 'released');
    socket.emit('button_click_html', {id: this.id, value: 0} );
    document.getElementById(this.id).style.color = "white";
  },
  touchstart: function(){
    console.log('BROWSER button ', this.id, 'clicked');
    socket.emit('button_click_html', {id: this.id, value: 1} );
    document.getElementById(this.id).style.color = "red";
  },
  touchend: function(){
console.log('BROWSER button ', this.id, 'released');
    socket.emit('button_click_html', {id: this.id, value: 0} );
    document.getElementById(this.id).style.color = "white";
  }
});
```

Slika 25: Pošiljanje podatkov ob kliku gumba na spletnem vmesniku (vir: Avtor naloge)

Ob testiranju pri pošiljanju za gumbe smo naleteli na napako, saj smo ugotovili, da brskalniki ločujejo med klikom na zaslon na dotik in miško, zato smo morali uporabiti JQuery. Uporabljena je funkcija z argumentom »:button«, ki zadeva vse elemente imena »button« na spletni strani. Štiri gnezdene funkcije pa omogočajo prepoznavo klika z miško ali pa na zaslonu na dotik.

```
socket.on('button_click_html', function(data) {
    console.log('button with id ', data.id, 'has value of: ' , data.value );
    if(data.value == 0){
        io.emit('button_change', {id: data.id, value: 0} ); //pošlji na vse naprave
        if(data.id <= 50 )
            midi_out.sendMessage([190,data.id, 0]); //prvi arduino
        else if(data.id > 50 && data.id <= 100)
            midi_out.sendMessage([191,data.id, 0]); //drugi arduino
        else if(data.id > 100 && data.id <= 150)
            midi_out.sendMessage([192,(data.id-100), 0]); //tretji arduino
        else if(data.id > 150)
            midi_out.sendMessage([193,(data.id-100), 0]); //četrti arduino
    }
    else if(data.value == 1){
        io.emit('button_change', {id: data.id, value: 1} ); //pošlji na vse naprave
        if(data.id <= 50 )
            midi_out.sendMessage([190,data.id , 1]); //prvi arduino
        else if(data.id > 50 && data.id <= 100)
            midi_out.sendMessage([191,data.id , 1]); //drugo arduino
        else if(data.id > 100 && data.id <= 150)
            midi_out.sendMessage([192,(data.id-100), 1]); //tretji arduino
        else if(data.id > 150)
        midi_out.sendMessage([193,(data.id-100), 1]); //četrti arduino
}

});
```

Slika 26: Izvedena dejanja ob kliku gumba (vir: Avtor naloge)

Ko strežnik prejme socket z imenom »button_click_html« se sproži algoritem, ki preverja na kateri ploščici je posamezni gumb. Deluje na podlagi id-jev posameznih gumbov, ki smo jih

določili z zunanjih zbirkah podatkov in nam pomaga pri pošiljanju MIDI signalov. Predem se strežnik loti pošiljanja le teh signalov pa še pošlje dodaten socket z imenom »button_change« na vse računalnike, saj samo v tem primeru prikaže vsem uporabnikom, da je nekdo pritisnil gumb na spletnem vmesniku.

Izkazalo se je, da so najzahtevnejši elementi, ki jih bomo implementirali rotacijski kodirniki. Za prepoznavo njihovih premikov smo uporabili že vnaprej pripravljene knjižnice z imenom »Encoder«, ter z njihovo pomočjo pošiljali MIDI signale. V pomoč nam je bilo dejstvo, da naš program zanje uporablja le 2 vrednosti: premik v smeri urinega kazalca(UP) in premik v nasprotni smeri urinega kazalca(DOWN).

4.8 Uporaba rotacijskih kodirnikov v programu

Naši rotacijski kodirniki imajo 3 priklopne pin-e, ki jih priklopimo na digitalne vhode krmilnika. Dva izhoda uporabimo za prepoznavo strani v katero ga vrtimo, tretji pa služi kot gumb, če kodirnik pritisnemo. Vsak izhod kodirnika je tako nastal »gumb« v končnem programu.

```
var upButton_01 = new five.Button({pin: 2, board: this.byId("A"), id: 13});
var downButton_01 = new five.Button({pin: 3, board: this.byId("A"), id: 14});
var pressButton_01 = new five.Button({pin: 5, board: this.byId("A"), id: 16});
```

Slika 27: Inicializacija gumbov na rotacijskih kodirnikih (vir: Avtor naloge)

Po raziskavi smo na spletu našli že pripravljene knjižnice za kodirnike, zato smo uporabili le te. Zanje smo uporabili skripte v zunanjih datotekah, ter jih klicali s pomočjo funkcije.

```
  rotaryEncoder_01({
  upButton_01,
  downButton_01,
  pressButton_01,
  onUp_01: () => {
      console.log('Encoder on pin', upButton_01.pin, 'going UP');
      output.sendMessage([178,upButton_01.pin,1]);
  },
  onDown_01: () => {
      console.log('Encoder on pin', downButton_01.pin, 'going DOWN');
      output.sendMessage([178,downButton3.pin,1]);
  },
  onPress_01: () => {
      console.log('Encoder on pin', upButton3.pin, 'PRESSED');
      output.sendMessage([178,pressButton3.pin,1]);
  },
});
```

Slika 28: Klic funkcije rotacijskega kodirnika (vir: Avtor naloge)

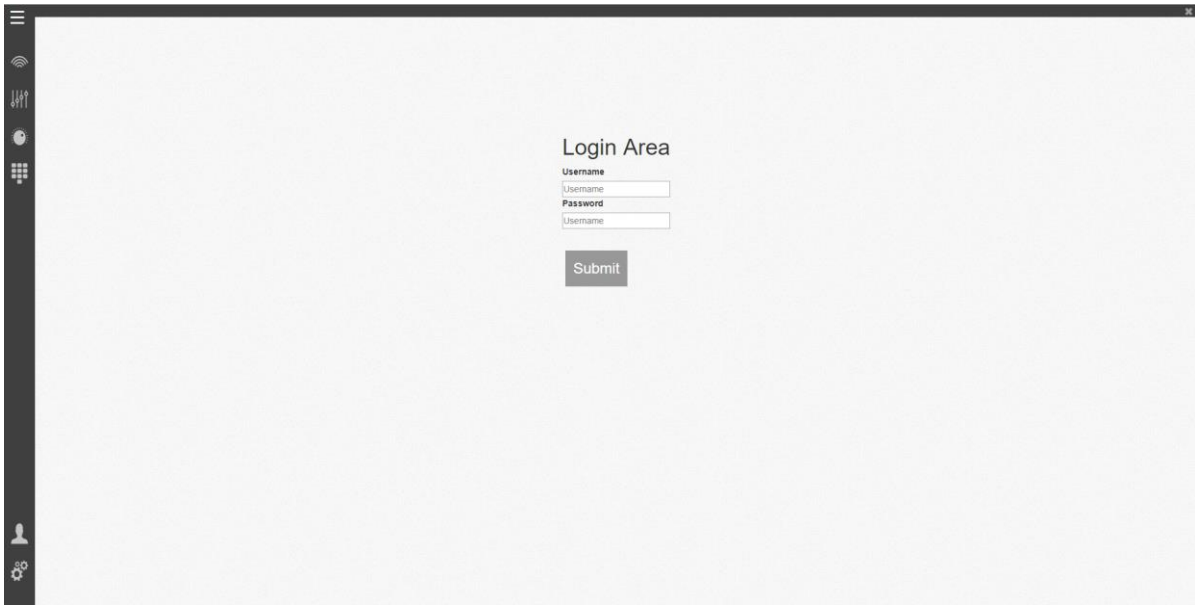
Sprva smo se želeli lotiti združitve skripte, da bi omogočila uporabo več kodirnikov z eno datoteko, a nam to ni uspelo. Zato smo uporabili 3 različne zunanje datoteke in 3 funkcije v glavnem programu.

```
module.exports = function rotaryEncoder_01({
  upButton_01,
  downButton_01,
  pressButton_01,
  onUp_01,
  onDown_01,
  onPress_01,
}) {
  let waveform = '';
  let waveformTimeout;

  upButton_01.on('up', () => {
    waveform += '1';
    handleWaveform();
  });

  downButton_01.on('up', () => {
    waveform += '0';
    handleWaveform();
  });

  pressButton_01.on('up', () => {
    onPress_01();
  });

  function handleWaveform() {
    if (waveform.length < 2) {
      waveformTimeout = setTimeout(() => {
        waveform = '';
      }, 8);
      return;
    }

    if (waveformTimeout) {
      clearTimeout(waveformTimeout);
    }

    if (waveform === '01') {
      onUp_01();
    } else if (waveform === '10') {
      onDown_01();
    }

    waveform = '';
  }
}
```

Slika 29: Skripta za prepoznavo smeri, v katero vrtimo kodirnik (vir: Avtor naloge)

4.9 Prijava v program

Ko se povežemo na stran se nam pojavi obrazec, v katerega vnesemo svoje uporabniško ime in geslo, ki smo ga pridobili od administratorja.



Slika 30: Prijava v program (vir: Avtor naloge)

Ob vpisu uporabniškega imena in gesla program najprej preveri ali je uporabniško ime in geslo v bazi podatkov. Če sta podatka pravilna se nam odklenejo vse potrebne funkcionalnosti programa.

Program se sprva poveže z zunanjo podatkovno bazo in izvede t.i. query, s katerim primerja naš vnos na spletni strani z vnosi v bazi z uporabo jezika SQL. Če so podatki pravilni nam strežnik pošlje »socket« z imenom »uspesna_prijava«, ki spletno stran odklene. V nasprotnem primeru pa nam spletna stran javi napako.

```
app.post('/index2.html', function(request, response) {
    var username = request.body.username;
    var password = request.body.password;
    if (username && password) {
        connection.query('SELECT * FROM accounts WHERE username = ? AND password = ?', [username, password], function(error, results, fields) {
            if (results.length > 0) {
                request.session.loggedin = true;
                request.session.username = username;
                request.session.password = password;
                console.log(results[0].email);
                gmail = results[0].email;
                response.redirect('/index.html');
                console.log("dela");
                io.emit('uspesna_prijava', {msg: request.session.username, pas: request.session.password, mail: gmail, unlock: 1});

            } else {
                response.send('Incorrect Username and/or Password!');
                console.log("nedela");
                io.emit('neuspesna_prijava', {msg: "Uporabnisko ime ali geslo je napacno"});
            }
            response.end();
        });
    } else {
        response.send('Please enter Username and Password!');
        console.log("connection error.");
        response.end();
    }
});
```

Slika 31 Koda za preverjanje uporabniških imen in gesel (vir: Avtor naloge)

```html
<div class="userpage" id="usersp">
    <div id="loginwrap" class="loginwrap">
        <h1>Login Area</h1>
            <div id="loginform" class="loginform">
                <form>
                    <input name="iebugaround" type="hidden" value="1">
                    <label class="logininginfo">
                        Username
                    </label>

                    <fieldset class="fieldset2">
                        <input type="text" name="username" class="requiredField" placeholder="Username" required>
                    </fieldset>

                    <label class="logininginfo">
                        Password
                    </label>

                    <fieldset class="fieldset2">
                        <input type="password" name="password" class="text requiredField subject" placeholder="Username" required>
                    </fieldset><br>

                    <p id="incorectpasswordusername" style="color:red;"></p>

                    <fieldset>
                        <input name="submit" id="submit" value="Submit" class="flashswoopbtn" type="submit"/>
                    </fieldset>
                </form>
            </div>
```

Slika 32: Prijavni obrazec v HTML (vir: Avtor naloge)

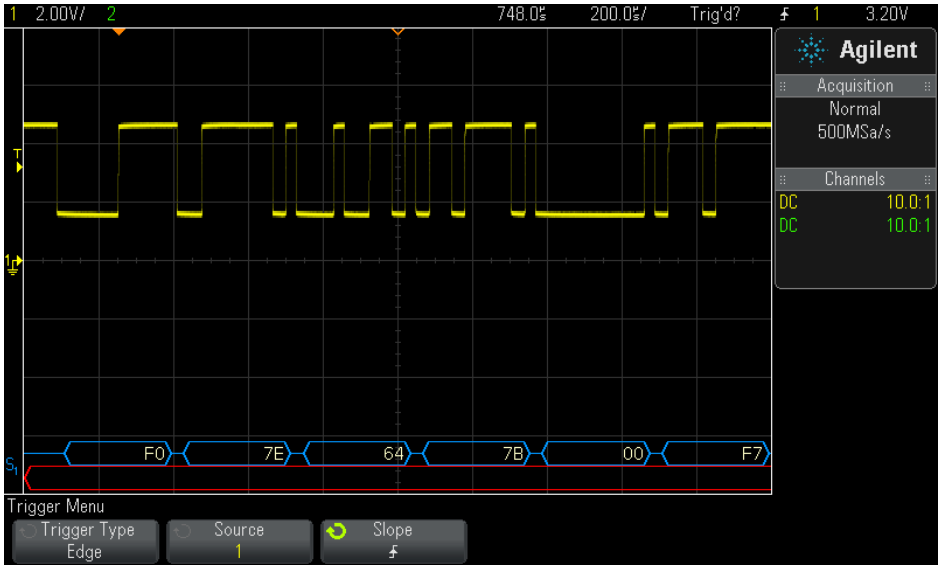
S pomočjo JavaScript-a in AJAX-a smo omogočili pošiljanje podatkov in primerjavo z bazo brez osveževanja strani, saj nam naš program ni omogočal uporabe več HTML datotek.

```javascript
$(document).ready( function () {
    $('form').submit( function () {
        var formdata = $(this).serialize();
        $.ajax({
            type: "POST",
            url: "index2.html",
            data: formdata,
        });
        return false;
    });
});
```

Slika 33: Pošiljanje podatkov brez osveževanja strani z uporabo AJAX (vir: Avtor naloge)

4.10 MIDI

»MIDI, oziroma **M**usical **I**nstrument **D**igital **I**nterface je elektronski standardni protokol, s pomočjo katerega komunicirajo različne elektronske glasbene naprave, kot so npr. mešalna miza, sintetizator zvoka in računalnik.«(Wikipedija, 2019)
Sam protokol MIDI ne vsebuje nobenega zvočnega zapisa, temveč sprejema skupke podatkov ki jih pretvarja v točno določene signale ali tone.



Slika 34: MIDI signal (vir: frank-buss.de)

Pri pošiljanju MIDI sporočil poznamo točno določene parametre, med katerimi lahko izbiramo. To so: Note ON(pritisk note), Note OFF(spust note), Note Number(katera tipka je v uporabi), Velocity(moč pritiska na tipko), Program Number(spomin – memorija), Pitch Bend(barva in globina zvoka).

Posamezno sporočilo je sestavljeno iz treh bajtov. Prvi bajt(imenovan tudi statusni bajt) je sestavljen iz tipa sporočila(parameter) in kanala(channel) ki ga bomo uporabili. Drugi bajt vsebuje noto(note), ki jo bomo poslali(med 1 in 127), tretji bajt pa hitrost(velocity).
Maksimalna hitrost in nota lahko imata vrednost 127, saj ima ta bajt le 7 uporabnik bitov, posledično lahko izračunamo, da je $2^7 = 127$.

Status Byte		Data Byte 1	Data Byte 2
1001	0000	00111100	01111111
Note On	Channel 0	Note Number 60 (C4)	Note Velocity 127

Slika 35: Sestava MIDI sporočila (vir: Ramp me up, Scotty!)

5. ZAKLJUČEK

Pri izdelavi raziskovalne naloge smo se naučili mnogo novega o programiranju v okolju Arduino. Izpopolnili smo tudi znanje programiranje v drugih jezikih, ter pridobili dodatne kompetence pri sodelovanju z drugimi podjetji.

Ob začetku raziskovanja smo si postavili tudi nekaj hipotez, ki smo jih potrdili ali ovrgli med samim raziskovanjem.

Hipoteza 1: Celoten izdelek bo cenovno ugoden za ciljne uporabnike.
To hipotezo lahko deloma potrdimo, saj je končni produkt v skladu s svojo uporabnostjo res cenejši od svoje konkurence.

Hipoteza 2: Program bo napisan v jeziku C#
To hipotezo smo v celoti zavrgli, saj smo zaradi našega pomanjkljivega znanja iz C# nalogo napisali v JavaScript-u.

Hipoteza 3: Uporabili bomo zunanjo bazo podatkov za prijavo v sistem.
To hipotezo smo potrdili, saj smo se zaradi večje varnosti res odločili za zunanjo podatkovno bazo uporabnikov, ki jo urejamo samo.

Hipoteza 4: Ohišje za izdelek bomo lahko izdelali sami.
To hipotezo smo ovrgli, saj smo iz podjetja Laser Tehnik prejeli ponudbo za izrez ohišja po meri.

5. DRUŽBENA ODGOVORNOST

Naš izdelek, MIDI DMX krmilnik, smo zgradili z namenom, da bi delo lučkarskih tehnikov hitreje in bolj učinkovito. Za primer lahko vzamemo program Titan One, ki ga je razvilo podjetje Avolites. Je eden najbolj naprednih programov za delo z osvetlitveno tehniko, a glede na število funkcij ki jih ima, je bilo podjetje primorano skriti določene funkcije v menije.
Z uporabo našega izdelka bo lahko uporabnik dostopal do teh funkcij z uporabo posebnih gumbov, brez potrebnega vložka v izdelek, ki bi deloval samo z eno programsko opremo.

6. VIRI

Arduino (5.2.2019), pridobljeno iz: https://www.arduino.cc/

Avolites (20.12.2018), pridobljeno iz: https://www.avolites.com/

EtherPort (5.2.2019), pridobljeno iz: https://github.com/rwaldron/etherport

Firmata (20.12.2018), pridobljeno iz: https://www.arduino.cc/en/reference/firmata

Firmata Builder (30.1.2019), pridobljeno iz: http://firmatabuilder.com/

Johnny-Five (5.2.2019), pridobljeno iz: http://johnny-five.io/

Johnny-Five Encoder (10.2.2019), pridobljeno iz: https://github.com/akinnee/johnny-five-rotary-encoder

MIDI (1.2.2019), pridobljeno iz: https://en.wikipedia.org/wiki/MIDI

MX Cherry (10.1.2019), pridobljeno iz: https://en.wikipedia.org/wiki/Cherry_(keyboards)

Node.js (1.2.2019), pridobljeno iz: https://nodejs.org/en/

Rotary encoder (28.1.2019), pridobljeno iz: https://en.wikipedia.org/wiki/Rotary_encoder

Socket.io (1.2.2019), pridobljeno iz: https://socket.io/