Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jože Kulovic

Sistem za prikaz predvajane glasbe s pomočjo led diod

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00089/2013
Datum: 08.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JOŽE KULOVIC**

Naslov: **SISTEM ZA PRIKAZ PREDVAJANE GLASBE S POMOČJO LED DIOD**
SYSTEM TO DISPLAY PLAYING MUSIC WITH LED DIODE.

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V diplomski nalogi izdelajte sistem za prikaz predvajane glasbe s pomočjo led diod. Sistem naj bo sestavljen iz krmilnika Arduino Mega ADK, lastnega modula za prikaz (LED trakovi in LED žarnice) ter programov, ki omogočajo krmiljenje. Krmilnik, ki naj z računalnikom komunicira preko USB vodila, naj na podlagi prejetih podatkov prižiga in ugaša posamezen led trak ter spreminja intenziteto svetlosti led žarnic.
Programska aplikacija naj bo sestavljena iz dveh delov. Prvi del, program napisan v programskem jeziku C#, naj omogoča zajemanje zvočnih signalov s podanega vhoda, obdelavo prejetih podatkov s pomočjo FFT, ter pretvorbo rezultata v zaporedje ukazov za krmilnik. C# aplikacija naj služi tudi kot mini strežnik za povezavo z Andriod napravo. Drugi del, javanski program napisan za operacijski sistem Android, naj omogoča povezavo z mini strežnikom ter krmiljenje programa Aimp za predvajanje glasbe.

Mentor:

doc. dr. Tomaž Dobravec

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jože Kulovic, z vpisno številko **63100247**, sem avtor diplomskega dela z naslovom:

Sistem za prikaz predvajane glasbe s pomočjo led diod

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobraveca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Tomažu Dobravcu, ki me je sprejel pod svoje mentorstvo ter mi dajal koristne napotke in nasvete pri izdelavi diplomskega dela.

Posebna zahvala gre mojim staršem, ki so me tekom študija finančno podpirali. Zahvaljujem se jim tudi za moralno podporo, ki so mi jo namenili, saj sem sam na vse večkrat gledal preveč pesimistično, starši pa so me s svojo podporo vedno držali pokonci in me vzpodbujali.

Zahvalil bi se tudi vsem sošolcem za pomoč tekom študija, tako pri učenju in domačih nalogah kot tudi pri skupnih projektih, v katerih smo sodelovali.

Ne smem pozabiti tudi na Tejo Cetinski, ki je lektorirala mojo diplomsko nalogo in popravila vse moje nenamerno storjene napake.

Zahvaljujem pa se tudi vsem drugim, ki so mi na kakršen koli način pomagali tekom študija.

Kazalo

Seznam uporabljenih kratic

USB - Univerzalno Serijsko Vodilo (Universal Serial Bus)

PWM - Pulzno širinska modulacija (Pulse Width Modulation)

Vin - Vhodna napetost (Input Voltage)

GND - Ozemljitev (Ground)

LED - Svetleča dioda (Light Emitting Diode)

IDE - Integrirano razvojno okolje (Integrated Development Environment)

IP - Internetni Protokol (Internet Protocol)

PORT - Vrata

TCP/IP - Internetni protokol za nadzor prenosa (Transmission Control Protocol)

WiFi - Brezžično računalniško omrežje (Wireles Fidelity)

I/O - Vhod/Izhod (Input/Output)

Povzetek

V diplomski nalogi smo si zadali cilj, da bomo krmilil LED-diode po glasbi. Naš cilj smo uresničili tako, da smo z računalnika zajemali zvočne signale. Nato smo iz vrednostne oblike s pomočjo hitre Fourierjeve transformacije pretvorili v frekvenčno obliko. Iz te oblike pa smo lahko že posredovali podatke naprej h krmilniku. Za krmiljenje LED-diod smo uporabili krmilnik Arduino Mega ADK, ki je bil ravno pravšnji za naše potrebe, saj ima veliko število izhodov. Tako nismo potrebovali dodatnih čipov za vezavo LED-diod na krmilnik. Ko smo imeli ta del diplomske naloge narejen, smo se lotili izdelave Android aplikacije, s pomočjo katere lahko krmilimo predvajalnik glasbe Aimp. Za krmiljenje Aimpa smo uporabil razširitev Aimp Wep Control, ki nam omogoča krmiljenje predvajalnika preko spleta.

Ključne besede

Arduino, C# aplikacija, Android, LED trakovi, krmiljenje, AIMP, glasba

Abstract

This thesis is about how to control led strip with music. We realized our goal with sampling the audio signal on the computer. After that we converted from values in frequency spectrum with Fast Fourier transformation. From the frequency form we passed data to the controller. To control led strips we used controller Arduino Mega ADK which was the right one for our requirements, because it has a lot of inputs and outputs. So we did not need to buy additional chips for connecting led strips on controller. When we had that part of thesis done, we started working on Android application for controlling music player Aimp. For controlling Aimp we used Aimp Wep Control extension which provides us using controlling player through web.

Key words

Arduino, C# aplication, Android, LED strip, control, AIMP, music

Poglavje 1

Uvod

Vedno sem si želel, da bi lučke utripale v ritmu glasbe in tako sem dobil idejo, da bi sam naredil nekaj v tej smeri ter si ustvaril lastne svetlobne efekte. Prvič sem se lotil izdelave domačih svetlobnih efektov, ko sem bil še v osnovni šoli, in sicer tako, da sem preprosto priklopil nekaj novoletnih luči na izhod glasbenega stolpa. Luči so utripale, dokler nisem dal glasbenega stolpa bolj naglas, nakar so pregorele. To me ni ustavilo, zamenjal sem luči z novimi in poskušal biti bolj pazljiv, vendar sem pomotoma ustvaril kratek stik in glasbeni stolp je nehal delovati. Starši so ga odpeljali na servis na popravilo, jaz pa sem za nekaj časa prenehal z nadaljnjimi aktivnostmi z lučkami.

Ko sem bil na faksu ter se že naučil kakšno stvar in ko mi je sošolec povedal za krmilnik Arduino, sem se lotil izdelave lastnih svetlobnih efektov, kar je tudi tema moje diplomske naloge. Za svoj projekt sem se odločil, da bom krmilili dva sklopa LED-luči.Prvi sklop sta dve LED-luči, ki sta stranski in jima uravnavam jakost svetilnosti (v nadaljevanju levi ter desni modul). Torej ob šibkih tonih glasbe ti dve luči utripata z zelo majhno svetilnostjo, ob močnejših tonih pa bolj svetlo. Svetlost krmilim z nivojem izhodne napetosti krmilnika. Za nadziranje napetosti sem uporabil izhode PWM, ker drugačne podpore za določanje analognih izhodnih nivojev napetosti izbran krmilnik nima.

Drugi sklop luči je mišljen za sredino, v nadaljevanju sredinski modul. Na sredinskem modulu sem se odločil, da bom imel zaporedje 41 LED-trakov, ki se prižigajo v odvisnosti od efektov, razloženih v nadaljevanju. V splošnem na sredinskem modulu lahko hkrati sveti največ 28 LED-trakov, ki jih preko krmilnika Arduino le prižigam in ugašam, ne morem pa jim določiti, s kakšno svetilnostjo bodo svetili, kakor lahko to storim na stranskih dveh modulih. Po uspešnem programu na prototipni ploščici sem se odločil za resnejšo realizacijo. Sprva sem imel namen krmiliti le LED-diode, ki sem jih pritrdil v okroglo cev. Projekt sem želel realizirati čim ceneje. Ker sem cev našel že doma, mi je preostal le še nakup LED-diod ter vodnikov, kar pa ni tako drago. Vendar sem šele po praktični realizaciji ugotovil, da je ta način zelo nepraktičen, saj sem izbral za ta namen majhno cev in je bilo spajkanje uporov k diodam ter sámo povezovanje zelo problematično. Na koncu sem sicer res dobil delujoč izdelek, vendar pa z njim nisem bil zadovoljen, saj je bila verjetnost, da bi kakšen spoj odpovedal, zelo velika. Med sámo montažo se mi je to tudi zgodilo, a sem imel pri roki vse orodje, tako da sem to lahko popravil. Ta izdelek je bil zelo težko prenosljiv, saj sem moral biti pri nošenju zelo pazljiv, da ga ne bi preveč pretresel ali iztaknil kašnega vodnika. Zaradi vseh teh razlogov sem se odločil, da bom ta prvotni izdelek poimenoval prototip in se lotil izgradnje malo večjega, bolj resnega izdelka. Zato sem moral svoja denarna sredstva tudi povečati za okoli 300 evrov. V diplomski nalogi si bomo sprva ogledali uporabljeno strojno opremo in uporabljene tehnologije za izdelavo projekta, nato pa še potek realizacije.

Poglavje 2

Uporabljena strojna oprema

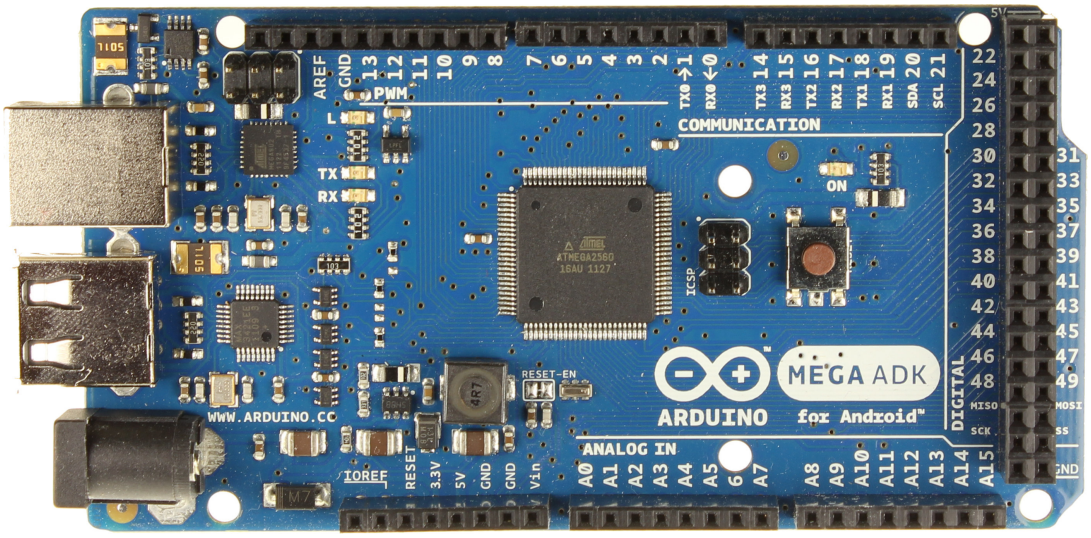
2.1 Krmilnik Arduino

Arduino je mikrokrmilnik, s pomočjo katerega lahko ustvarimo skoraj vse, kar želimo za domačo uporabo. Od krmiljenja robotov, avtomobilov pa vse do umetniških projektov. Ustvarjalci Arduina ga definirajo kot "an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software". Kar pomeni, da je odprtokodna elektronska prototipna platforma, ki temelji na prilagodljivi ter preprosti strojni in programski opremi. Arduino Mega ADK temelji na mikrokontrolerju ATmega2560 s to posebnostjo, da ima na plošči integriran USB host controller, ki je kompatibilen z Google ADK [1]. To pomeni, da ga lahko enostavno povežemo z napravami, ki imajo nameščen operacijski sistem Android. Tako lahko s pomočjo teh naprav krmilimo Arduina oziroma spremljamo rezultate, ki jih Arduino posreduje napravi.

Krmilnik Arduino Mega ADK lahko napajamo preko USB-povezave ali pa preko zunanjega napajanja. Preklapljanje med viri napajanja se dogaja avtomatično; če bo imel Arduino na razpolago zunanji vir napajanja, ga bo uporabil, sicer pa se bo napajal preko USB-povezave. Zunanje napajanje mu lahko dovedemo preko adapterjev ali pa nanj priklopimo baterije.

Če uporabljamo baterije, lahko za vezavo uporabimo temu namenjena pina Vin ter GND. Pri uporabi adapterja pa imamo na plošči že privarjen 2.1 mm vtičnik, tako da ga lahko na enostaven način priklopimo. Če uporabljamo zunanje napajanje in uporabljamo za to napajalni vtič, lahko na pinu Vin dobimo našo vhodno napetost ter jo uporabimo kje drugje, če jo potrebujemo. Naš električni izvor mora biti zmožen zagotavljati 1.5 A izhodnega toka, saj ga naprava Android porabi 750 mA, ostalih 750 mA pa se porabi za sam Arduino ter vse njegove senzorje, ki so priključeni na plošči. Arduino Mega ADK deluje na napetosti od 5.5 V pa do 16 V. Če je napetost manjša od 5 V, lahko pride do nestabilnega delovanja.

Arduinova verzija ADK ima 256 KB bliskovnega pomnilnika, ki je namenjen shranjevanju programske kode. Od teh 256 KB se 8 KB uporablja za zaganjalnik (bootloader), ki se zažene ob priklopu na napajanje in pritisku na reset gumb. Ima tudi 8 KB SRAMa ter 4 KB EEPROMa, do katerega lahko dostopamo s pomočjo dodatne knjižnice EEPROM, ki jo moramo vključiti v našo programsko kodo.



Slika 2.1: Krmilnik Arduino ADK MEGA

Arduino Mega ADK ima 16 analognih vhodov (pinov) ter 54 digitalnih pinov. Analogni pini zajemajo podatke v 10-bitni natančnosti, torej imajo lahko vhodni podatki 1024 različnih vrednosti. Vhodna napetost je od 0 do 5 V, vendar pa lahko s pomočjo uporov to območje spremenimo. Digitalne pine lahko uporabimo kot vhod ali pa kot izhod. Za izbor, ali bo določen pin vhod ali izhod, ter za branje in pisanje na določene pine imamo na razpolago posebne metode, kot so `pinMode`, `digitalWrite` in `digitalRead`. Ti digitalni pini delujejo na nivoju napetosti 5 V, vsaki izmed njih lahko proizvede do 40 mA izhodnega toka. Od 54 pinov je 14 izhodov PWM, kar pomeni, da lahko dobimo iz njih katerokoli napetost na območju od 0 V do 5 V.

Nekateri izmed teh 54 pinov imajo posebne funkcije. Nekateri so, na primer, uporabljeni za serijsko komunikacijo. Zato moramo pri izboru pinov upoštevati, katerih pinov ne smemo uporabiti. Če bomo uporabljali serijsko komunikacijo preko USB-povezave, ne smemo uporabiti pinov 0 in 1, ker se ne bosta odzvala na naše ukaze.

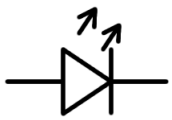
Povzetek specifikacij mikrokrmilnika Arduino mega ADK:

Mikrokontroler	ATmega2560
Delovna napetost	5 V
Vhodna napetost (priporočena)	7-12 V
Vhodna napetost (mejne)	6-20 V
Digitalno vhodno/izhodno pini	54 (od katerih 15 omogoča izhod PWM)
Analogno vhodni pini	16
Izhodni tok digitalnih pinov	40 mA
Flash spomni	256 KB
EEPROM	4 KB
Frekvenca ure	16 MHz

Slika 2.2: Povzetek specifikacij krmilnika

2.2 Led trak

LED-trak je sestavljen iz več skupaj povezanih LED-diod. Na enem LED-traku imamo torej več LED-diod, količina in vrsta le-teh pa sta odvisni od izbire LED-traku. Nekateri LED-trakovi imajo tudi zaščito za zunanjo uporabo, torej so vodoodporni. LED-diode so polprevodniški elektronski elementi; to pomeni, da prevajajo električni tok le v eno smer. Njihove električne karakteristike so zelo podobne karakteristikam navadnih polprevodniških diod, s to razliko, da LED-dioda sveti, kadar prevaja tok. LED-diod je veliko vrst, med sabo pa se razlikujejo po barvi, obliki, velikosti in električnih karakteristikah. LED-diode imajo v primerjavi z navadno žarnico daljšo življenjsko dobo, ki znaša okoli 50.000 ur, ter boljši električni izkoristek [2].



Slika 2.3: Električni simbol led diode

2.3 Upori

Upor je pasivni in najpogosteje uporabljeni električni element, ki ga uporabljamo za omejevanje električnega toka [3].

$$R = \frac{U}{I} \tag{2.1}$$

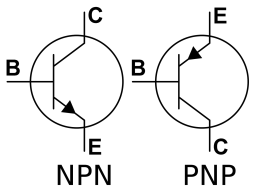
Iz Ohmovega zakona (2.1) sledi, da večji, ko je upor, manjši tok bomo imeli. Ravno tako lahko iz enačbe vidimo, da je tudi napetost odvisna od upornosti. Upori se glede na izvedbo delijo na masne, plastne in žične upore. Masni upori so v obliki valja, zgrajenega najpogosteje iz ogljenega praška.

Njihova dovoljena izgubna moč je relativno majhna, vendar so bili za naš projekt, zato smo te upore uporabili. Žični upori imajo visoko časovno stabilnost in jih je možno izdelati za poljubno moč. Plastni upori pa se uporabljajo najpogosteje, saj imajo veliko uporovno območje ter majhne tolerance.

Slika 2.4: Električni simbol upora

2.4 Tranzistorji

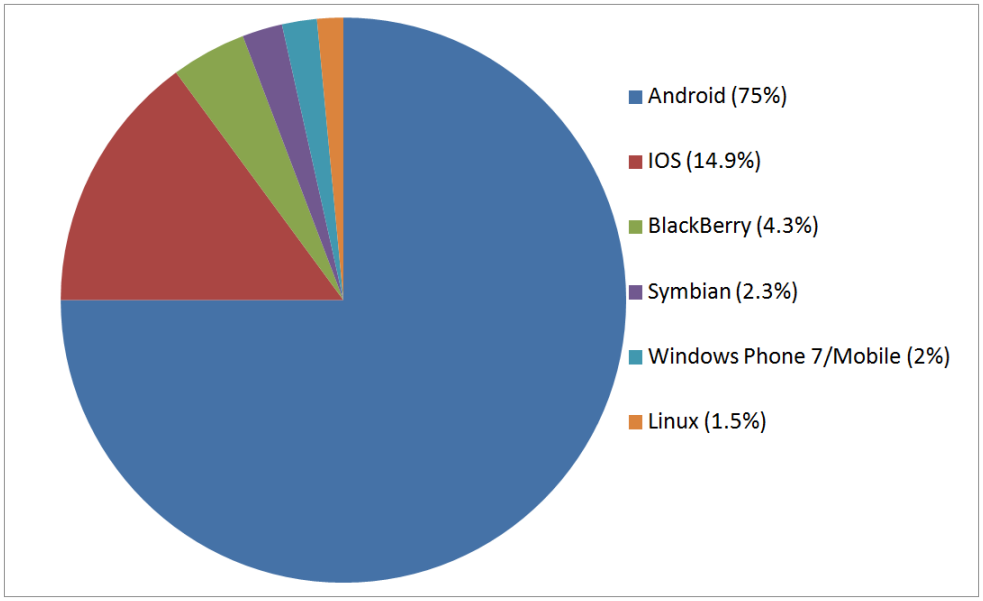
Tranzistor je polprevodniški element s tremi priključki. Uporablja se za ojačanje ter preklapljanje električnih signalov. Priključki se imenujejo baza, emitor in kolektor. S pomočjo baznega pina lahko sklenemo ali pa ne električni tok, ki poteka skozi pina kolektor in emitor. V našem projektu imamo krmilnik Arduino z izhodnimi pini napetosti 5 V ter LED-trakove potrebne napetosti 12 V. Če torej želimo krmiliti trakove z našim krmilnikom, nujno potrebujemo tranzistorje, s katerimi bomo vklapljali in izklapljali LED-trakove. V splošnem se tranzistorji delijo na tipa NPN in PNP. Razlika je le v tem, kdaj bo električni tok sklenjen. Pri tipu PNP je električni tok sklenjen ob nizkem stanju baze, pri NPN pa ob visokem. V našem projektu bomo uporabili tranzistorje tipa NPN; ko bo krmilnik dal na bazni vhod napetost 5 V, se bo na tranzistorju sklenil električni krog med emitorjem in kolektorjem in tako bo ob pravilni vezavi zasvetil LED-trak.



Slika 2.5: Električni simbol NPN ter PNP tranzistorja

2.5 Mobilni telefon Android

Android je operacijski sitem za mobilne naprave, ki temelji na Linuxovem jedru. Razvit je s strani podjetja Google in je odprtokoden z licenco Apache [4]. To omogoča cenejše in lažje razvijanje aplikacij, ravno tako pa se proizvajalcem mobilnih naprav ni več potrebno ukvarjati z razvojem operacijskega sistema, saj lahko uporabijo operacijski sistem Android. Operacijski sistem Android ima tudi urejeno sinhronizacijo z Googlovimi storitvami, kot so koledar, Gmail, Picasa in ostale, ravno tako pa ima trgovino Google Play (licenčna aplikacija podjetja Google, preko katere lahko prenašamo ali pa nalagamo aplikacije na njihove strežnike), s katere je bilo prenešenih že 48 miljard aplikacij. Operacijski sistem Android ima sedaj že 75-odstotni tržni delež, torej je najbolj razširjen operacijski sistem na pametnih mobilnih napravah [5]. Iz tega razloga in ker si sam lastim mobilni telefon z Androidom, smo se odločili, da bomo aplikacijo za mobilno napravo napisali za operacijski sistem Android.



Slika 2.6: Tržni delež operacijskih sistemov na pametnih mobilnih napravah

Poglavje 3

Uporabljene tehnologije

3.1 Android

Vsa orodja za razvijanje aplikacije operacijskega sistema Android so zbrana v eni datoteki `.zip`, ki jo lahko prenesemo z njihove uradne strani [6]. Vsebino datoteke skopiramo v poljuben direktorij in že lahko pričnemo z razvijanjem aplikacij. Ker se operacijski sistem Android ves čas razvija, moramo najprej izbrati, za katero verzijo bomo razvijali našo aplikacijo, ter dodatno prenesti dodatne stvari, ki so značilne za to verzijo operacijskega sistema. Lahko pa prenesemo tudi vse verzije, vendar to ni potrebno. Android vse svoje verzije poimenuje po slaščicah; njegova najnovejša verzija se imenuje Jelly Bean s stopnjo API-ja 17. Pri izbiri verzije je predvsem pomembno preveriti stopnjo API-ja, kajti nekatere verzije imajo lahko isto ime, vendar različne stopnje API-ja, ki se med sabo nekoliko razlikujejo. Trenutno je še vedno najbolj razširjena verzija z imenom Gingerbread s stopnjo API-ja 10. Ta stopnja je tudi mejna za našo aplikacijo, ki posreduje podatke krmilniku, saj je komaj v tej verziji mogoče povezati napravo Android s krmilnikom Arduino.

Ko imamo izbrano verzijo operacijskega sistem, za katero bomo razvijali aplikacijo, lahko v razvijalnem okolju Eclipse, ki smo ga prenesli skupaj z ostalimi orodji v datoteki `.zip`, pričnemo razvijati aplikacijo. Aplikacije se razvija v programskem jeziku Java, lahko pa uporabljamo tudi domorodne

(native) knjižnice, napisane v programskem jeziku C ali katerem drugem, in jih potem kličemo znotraj javanske kode. Med razvijanjem lahko aplikacijo direktno testiramo na naši mobilni napravi tako, da le-to priključimo na računalnik in izberemo development mode, ali preko emulatorja, ki pa deluje počasneje, kakor deluje aplikacija v realnosti na katerikoli drugi napravi Android.

3.2 Microsoft .NET C#

Microsoft .NET je okolje za razvijanje programske opreme za operacijske sisteme Windows. V razvojnem okolju .NET lahko aplikacije pišemo v različnih programskih jezikih, ki se nato prevedejo le za operacijski sistem Windows [7]. Diplomsko nalogo sem napisal v programskem jeziku C#, ki je objektno usmerjen ter zelo podoben programskemu jeziku Java. V razvojnem okolju .NET se uporabniške vmesnike lahko izdela tako, da vse gradnike, ki jih želimo uporabiti, le povlečemo z miško na našo površino. Nato lahko pričnemo s pisanjem kode v ozadju, ki bo določila, kako posamezen gradnik reagira na določeno akcijo uporabnika.

3.3 Razvojno okolje Arduino

Razvojno okolje Arduino se uporablja za programiranje krmilnikov Arduino. Programira se v programskem jeziku C z nekaterimi dodatnimi metodami, ki nam olajšajo programiranje, kot so `map`, `analogWrite`, `digitalRead`, `digitalWrite` in podobne. Za testiranje delovanja programa moramo imeti na računalniku priklopljen krmilnik Arduino, saj lahko testiramo le tako, da program dejansko naložimo na krmilnik. S krmilnika lahko tudi pošiljamo podatke nazaj preko serijskega vmesnika USB in jih v razvojnem okolju nato tudi izpišemo. Lahko pa tudi iz razvojnega okolja med samim delovanjem krmilnika pošiljamo podatke po USB-vodilu.

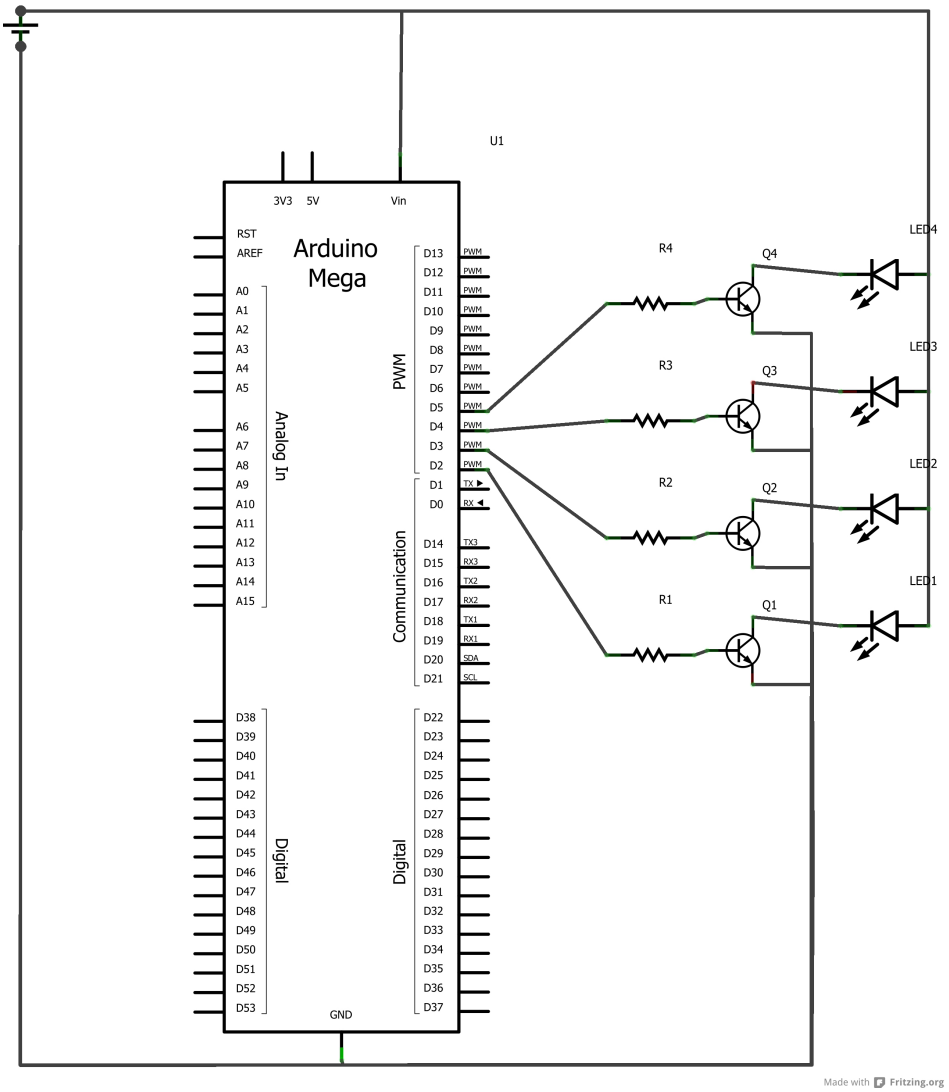
Poglavje 4

Zasnova in realizacija

4.1 Priključevanje elementov

Krmilnik Arduino smo priključil na 12-voltni napajalnik moči 150 W, kar je njegova zgornja priporočena meja. Čeprav bi bil za takšen projekt dovolj 100-vatni napajalnik, smo se odločili za večjega zaradi možnosti kasnejše nadgradnje projekta. Uporabljen krmilnik ima veliko število izhodov, katerih izhodna napetost je 5 V, tok pa 40 mA. Za projekt smo uporabili skoraj vse izhode, ki so nam bili na voljo. Ker imajo vsi moduli, ki smo si jih zamislili, potrebno napetost 12 V, nismo mogli nobenega od njih direktno krmiliti z izhodom krmilnika. Iz tega razloga smo vse module vezali preko tranzistorjev, tako da je krmilnik odpiral in zapiral tranzistor, ki je prepuščal napetost 12 V ali pa ne. Levi in desni modul sta imela dovolj vsak po en izhod PWM, torej dva tranzistorja. Sredinski modul, ki je sestavljen iz 41 LED-trakov, pa potrebuje tudi 41 izhodov od krmilnika in 41 tranzistorjev. Sredinski modul ima možnost, da sveti v zeleni barvi ali pa v treh barvah, med katerimi je večji del zelene. Zato imamo na sredinskem modulu 41 LED-trakov, sveti pa jih le 28 - katerih, lahko izberemo preko programa. Na voljo za izbiranje imamo torej, ali svetijo le zeleni LED-trakovi ali pa barvni. Vsi uporabljeni LED-trakovi so enobarvni. Razlog, da se nismo odločili za barvne trakove je cena, saj bi s tem za en LED-trak na krmilniku potrebovali 3 izhodne pine, torej bi

poleg dražjih LED-kablov morali kupiti še dodatne čipe, da bi vse to lahko povezali, saj bi na krmilniku zmanjkalo izhodov. Za krmiljenje barvnih LED-trakov namreč potrebujemo izhode PWM, če želimo res imeti vse barve, ker pa jih je na našem krmilniku le 14, zadostuje za 5 barvnih LED-trakov. Tako smo se odločili, da bomo raje podvojili zgornjo polovico in s tem izkoristili vse izhode krmilnika, ki jih imamo na razpolago. Na spodnji sliki je zaradi preglednosti shema vezave le štirih LED-trakov sredinskega modula.



Slika 4.1: Priključitev LED-trakov

Na enak način se veže ostale trakove, ravno tako tudi levi in desni modul, le da pri njiju uporabimo manjši upor, ker je posamezni modul močnejši od posameznega LED-traku. Tranzistor, ki sem ga izbral, ima v povprečju 40-kratno ojačanje, kar je potrebno upoštevati pri izbiri ustreznega predupora za vezavo. Izbran LED-trak ima na moji dolžini 1.6 W moči.

$$I = \frac{P}{U} = \frac{1.6W}{12V} = 133mA \tag{4.1}$$

Po enačbi (4.1) dobimo, da je potreben tok za posamezni LED-trak 133 mA.

$$I_b = \frac{I}{Ojacanje} = \frac{133mA}{40} = 3.3mA \tag{4.2}$$

Zaradi 40-kratnega ojačanja je potem potreben tok na bazi vsaj 3.3 mA.

$$R = \frac{U - U_{be}}{I_b} = \frac{5V - 1V}{3.3mA} = 1894\Omega \tag{4.3}$$

Torej mora biti upor 1894 Ω ali manjši. Mi smo uporabili upor 900 Ω. Na enak način smo vezali stranska dva modula, le da imamo drugačne vrednosti v enačbi, saj ima en stranski modul 6 W moči.

$$I = \frac{P}{U} = \frac{6W}{12V} = 500mA \tag{4.4}$$

Po enačbi (4.4) dobimo, da je potreben tok za posamezni led trak 500 mA.

$$I_b = \frac{I}{Ojacanje} = \frac{500mA}{40} = 12mA \tag{4.5}$$

Zaradi 40-kratnega ojačanja je potem potreben tok na bazi vsaj 12 mA.

$$R = \frac{U - U_{be}}{I_b} = \frac{5V - 1V}{12mA} = 320\Omega \tag{4.6}$$

Potreben upor je torej 320 Ω, mi pa smo uporabili 200-ohmske upore, kar privede do izhodnega toka 20 mA od krmilnika, kar pa ni problematično, saj naši izhodi premorejo 40 mA.

4.2 Program v C# za krmiljenje krmilnika Arduino

Za zajem zvoka z računalnika smo napisali program v programskem jeziku C#. Razlog za C# je ta, da sem od vsega začetka nanj najbolj navajen in sem, ko sem se lotil pisanja takšnega programa, pomislil samo nanj. Kasneje smo program poizkusili implementirati v Javi, vendar smo dobili bolj popačene rezultate - neodzivne, kar je bila morda tudi posledica izbire knjižnice, ki nam je vrnila vzorce zvoka. Pri programiranju smo morali ves čas paziti na hitrost in odzivnost programa. Naš celoten program je zelo objektno razdeljen, zato se nam je zgodilo, da se med analiziranjem zvoka drsniki na uporabniškem vmesniku niso več premikali gladko. Problem smo rešili tako, da smo vse parametre katere nismo spreminjali v klicanih metodah prenašali po naslovih (referencah). Ta sprememba je močno vplivala na odzivnost uporabniškega vmesnika.

4.2.1 Uporabniški vmesnik

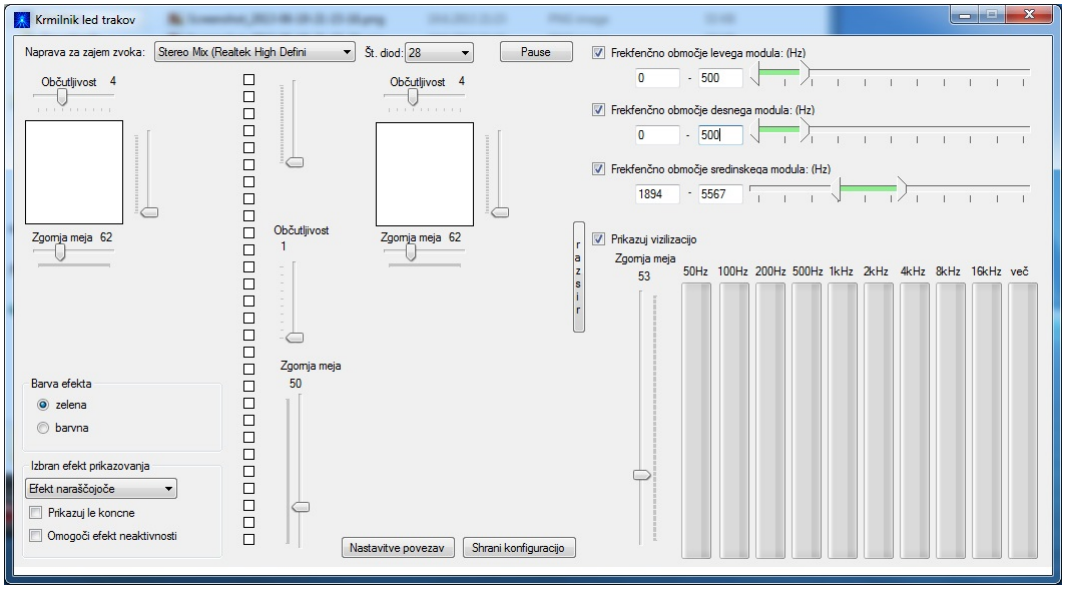
Ob odprtju programa izgleda uporabniški vmesnik nekoliko zapleten, vendar se ga že v kratkem času navadimo in ga obvladamo. V njem si lahko s pomočjo spustnega seznama (combobox) izberemo, s katere naprave bomo zajemali zvok ter število LED-trakov, ki jih imamo v sredinskem modulu. Število je omejeno na 12, 20 in 28. Razlog za to je učinkovito pošiljanje podatkov naprej h krmilniku Arduino. Na desni strani uporabniškega vmesnika so prikazane tudi jakosti posameznih frekvenc trenutne skladbe, ki se predvaja; tako nam je lažje pri določevanju frekvenc za posamezne module. Frekvenčno območje lahko izbiramo s pomočjo dveh drsnikov, ki potujeta od 0 Hz pa do 22 kHz. Za bolj natančni izbor frekvenc lahko uporabnik tudi ročno vpiše želeno območje. Uporabniški vmesnik je prilagojen za krmiljenje treh modulov. Za vsak modul lahko nastavimo želen izbor frekvenc in občutljivost. Poleg teh dveh stvari lahko kontroliramo tudi zgornjo mejo.

Zgornja meja je meja, pri kateri modul sveti 100-odstotno. Stranski moduli pri 100 % svetijo s 100-odstotno svetilnostjo, sredinskem modulu pa pri 100 % svetijo vsi LED-trakovi. Razlog, da nismo to fiksno določili je ta, da zajemamo različno močne podatke tako iz mikrofona kakor iz Windows zvočne mešalke. Če damo skladbo v predvajalniku bolj naglas, bodo tudi prejeti podatki večji, zato smo dodali možnost ročnega nastavljanja zgornje meje.

Občutljivost določi, kako hitro bo določen modul reagiral na vhod. Izračunana je preprosto po formuli (4.7).

$$noviDelez = delez^{obcutljivost} \tag{4.7}$$

Pri tem delež zavzema vrednost med 0 in 1. Na podlagi te se določi svetilnost stranskih modulov in prižgane trakove sredinskega modula. Iz tega sledi, da se ob zelo veliki občutljivosti vmesna stanja izgubijo in tako lahko na posameznem modulu dosežemo le stanje 0 ali 1. Torej ali posamezni modul ne sveti ali pa sveti v celoti (100 %).



Slika 4.2: Uporabniški vmesnik

4.2.2 Zajem zvoka

Za zajem zvoka smo uporabil Microsoftovo knjižnico Naudio, ki je izdana pod Microsoft Public License (Ms-PL). S pomočjo te knjižnice smo najprej prebrali naprave, ki so na voljo za zajem zvoka, ter jih prikazali uporabniku, da si lahko želeno napravo izbere. Naprave, ki so uporabniku na voljo, smo prikazali v spustnem seznamu, kjer se lahko napravo tudi zamenja med samim analiziranjem zvoka. Ob zamenjavi se bo avtomatično prenehalo z analizo, preklopilo napravo ter ponovno začelo z analizo. Vse to se zgodi tako hitro, da uporabnik ne opazi, da se je analiza v bistvu končala in ponovno začela. Ko je naprava izbrana in je uporabnik kliknil na gumb Play, se začne zajemanje zvočnih signalov.

Z našim programom zajemamo mono avdio signal, ker se nam ni zdelo pomembno, da bi zajemali stereo avdio signal. Predvsem smo opazovali, koliko s tem pridobimo in koliko moramo za to plačati. Ugotovili smo, da s zajemanjem stereo avdio signalov ne bi dobili dosti boljših efektov, vendar pa bi to lahko močno vplivalo na časovno zahtevnost procesiranja signalov v nadaljevanju, saj bi imeli dvakrat toliko dela, kar pa je za analiziranje v realnem času veliko. Frekvenco vzorčenja smo nastavili na 44100 Hz na sekundo s 16-bitno natančnostjo, frekvenčni razpon je torej od 0 pa do 22050 Hz.

Listing 4.1: Metoda za pričetek zajemanja zvočnega signala

```
public void start(){
      wavein = new NAudio.Wave.WaveIn();
      wavein.DataAvailable += new
         EventHandler<NAudio.Wave.WaveInEventArgs>(waveinData);
      wavein.WaveFormat = new
         NAudio.Wave.WaveFormat(frekfencaSimpleranja,
         kanalov);
      wavein.BufferMilliseconds = 30;
      wavein.DeviceNumber = forma1.comboBox1.SelectedIndex;
      wavein.StartRecording();
}
```

Iz predhodne metode je razvidno, da bo knjižnica Naudio ob dogodku prejema signala prožila metodo `waveinData`. Knjižnica nam vrne podatke v bajtih, za en vzorec se porabita 2 bajta, ker imamo 16-bitno natančnost za en vzorec. Zato bajte najprej združimo po dva skupaj. Kar vrne metoda, so velikosti (amplitude) napetosti v določenem času. Kar knjižnica torej naredi, je to, da zajema vzorce časovnih intervalih, kot jih določimo s frekvenco vzorčenja.

Listing 4.2: Združevanje bajtov v eno vrednost

```
void waveinData(object sender, NAudio.Wave.WaveInEventArgs
    e) {
        indeksVzorcev = 0;
        short sample;
        float sample32;
        for (int index = 0; index < e.BytesRecorded; index
            += 2){
                sample = (short)((e.Buffer[index + 1] << 8) |
                    e.Buffer[index + 0]);
                sample32 = sample / 32768f;
                obdelajVzorec(ref sample32);
        }
}
```

Ko bajte združimo dva pa po dva skupaj v vzorce jih nato pošljemo naprej v obdelavo, kjer na vsakih 1024 vzorcev analiziramo prejete vzorce, torej na vsakih 23 ms.

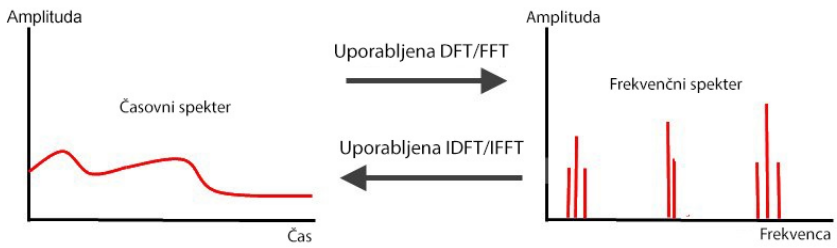
$$T = \frac{st.\ \ vzorcev\ \ analiziranja}{st.\ \ vzorcev\ \ na\ \ sekundo} = \frac{1024}{44100} = 23ms \tag{4.8}$$

Dodatna analiza je potrebna, ker nam brez analize ti podatki ne povedo kaj dosti. Povedo nam lahko le glasnost oziroma amplitudo, ki smo jo dobili v določenem času. Zato podatke analiziramo s pomočjo hitre Fourierjeve transformacije.

4.2.3 Procesiranje zvoka

Kot smo omenili že prej, je sedaj naša naloga, da pretvorimo prejete vzorce iz časovnega v frekvenčni spekter. Za to pretvorbo smo uporabili hitro Fourierjevo transformacijo (FFT) na kompleksnih številih. Hitra je predvsem zaradi potrebe analiziranja podatkov v realnem času, torej med samim zajemanjem vzorcev.

Fourierjeva transformacija naše osnovne signale, ki jih zajamemo s pomočjo knjižnice Naudio, pretvori v osnovne v sinusoide, tako da se lahko v nadaljevanju namesto na celotno osredotočimo le na eno frekvenčno območje [8].



Slika 4.3: Fourierjeva transformacija

Hitra Fourierjeva transformacija za razliko od diskretne Fourierjeve transformacije izkorišča prednost dejstva, da če narišemo v graf vse sinusoide, se nam le-te prekrivajo. Skupnih točk nam torej ni potrebno računati (množiti) večkrat, ampak je dovolj, da to storimo le enkrat. Ko neko vrednost potrebujemo znova, jo le prepišemo in si tako prihranimo nekaj časa. Hitra Fourierjeva transformacija računa transformacijo rekurzivno; v vsak naslednji krog rekurzije dá polovico manj podatkov. Hitro Fourierjevo transformacijo je zato potrebno začeti s številom vzorcev, ki je enako potenci števila 2, da se celotna transformacija na koncu lepo izide. V našem primeru delamo transformacijo na vsakih 1024 vzorcev, kar je enako 2 na 10.

4.2.4 Efekti

Po procesiranju zvoka imamo delež posameznih frekvenc, na podlagi izbranih nastavitev pa izračunamo tudi delež za posamezni modul. Če uporabnik na primer izbere frekvenčno območje od 300 Hz do 500 Hz za sredinski modul, poiščemo maksimalno frekvenco v tem območju ter si zapišemo njeno apmlitudo. Ta amplituda bo predstavljala sredinski modul. Na enak način dobimo amplitudi za stranska modula. Amplitudo spremenimo v delež tako, da jo delimo z zgornjo mejo, ki jo ima uporabnik nastavljeno pri posameznem modulu. Teh deležev še ne pošljemo krmilniku, ampak jih ustrezno obdelamo, tako da imamo lahko več efektov. Še posebej se to opazi pri sredinskem modulu.

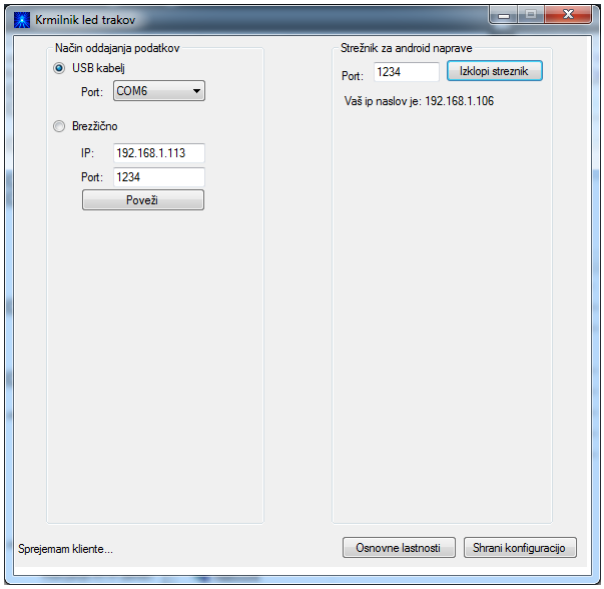
Primer efekta je to, da se lahko pri sredinskem modulu LED-trakovi prižigajo od zgoraj navzdol: to pomeni, da bi se najprej prižgal zgornji trak, ob večjem deležu drugi trak od zgoraj in tako naprej. Lahko se prižigajo od spodaj navzgor ali pa od sredine navzven ter še na veliko drugih načinov. Nekaj teh efektov smo zakodirali v sámo programsko kodo, dodali pa smo tudi možnost dinamičnega nalaganja efektov ob zagonu programa. V novem projektu lahko napišemo nov efekt, ga prevedemo v datoteko `.dll` in to datoteko damo k ostalim datotekam programa. Ko se bo program zagnal, bo avtomatično prebral vse datoteke in dinamično naložil efekte v program.

Listing 4.3: Koda, ki dinamično naloži vse efekte v program

```
foreach (FileInfo f in fil){
      types = kniznica.GetTypes().Where(t =>
         t.IsClass&&t.IsSubclassOf(tipNadrazreda)).ToList();
      foreach (Type type in types){
            Object o = Activator.CreateInstance(type);
            vsiEfekti.Add(new
               RazredZaPredstavitevEfekta(type,o));
      }
}
```

4.2.5 Strežnik za sprejem Androida

V aplikacijo za krmiljenje LED-modulov smo vgradili tudi strežnik TCP/IP, s pomočjo katerega se lahko na aplikacijo povežemo preko omrežja. Namen strežnika je, da bi krmilil določene nastavitve utripanja ter upravljal tudi določene dele računalnika. Preko strežnika, ki smo ga realizirali, lahko poleg nastavitev lastne aplikacije upravljamo tudi z miško ter tipkovnico. Ko se naprava poveže na strežnik, strežnik čaka na zahteve v asinhroni metodi. Ko dobi zahtevo, se nanjo primerno odzove. Na primer, če dobi ukaz za premik miške v levo, se bo izvedla posebna metoda za to akcijo. Če pa naprava zahteva vse efekte, ki so aplikaciji na razpolago, strežnik te podatke vrne vsem odjemalcem, ki so trenutno povezani. Tako imajo vsi odjemalci ves čas najnovejše podatke o stanju aplikacije.



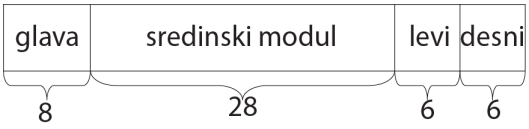
Slika 4.4: Uporabniški vmesnik za pošiljanje krmilniku ter postavitev strežnika

4.3 Pošiljanje podatkov krmilniku Arduino

Za pošiljanje podatkov smo sprva uporabljali le povezavo USB, kasneje pa smo dodali tudi možnost povezave preko omrežja. Napisali smo še eno aplikacijo za Android, ki deluje kot strežnik, na katero se poveže C#-aplikacija. Ko je ta povezana na aplikacijo Android, poteka prenos podatkov enako kakor preko USB-povezave. Androidna aplikacija po prejemu podatkov te pošlje naprej preko USB-ja do krmilnika Arduino. Tako tudi Arduino te podatke obravnava, kakor da bi prišli direktno preko C#-aplikacije po USB-vodilu. Uporabnik lahko izbere, po kateri povezavi bo pošiljal podatke krmilniku, tako da klikne na gumb "Nastavitve povezav". Ob kliku se uporabniku odpre vmesnik, v katerem izbere način pošiljanja ter določi nastavitve, ki so specifične za posamezni način. Pri USB-povezavi nastavi vrata, na katerih je priklopljen krmilnik, pri povezavi TCP/IP pa IP naslov in vrata, na katerih posluša Android aplikacija, povezana s krmilnikom Arduino.

Povezava med C#-aplikacijo ter krmilnikom se vzpostavi ob zagonu aplikacije oziroma ko uporabnik klikne na gumb. Če povezava ni vzpostavljena in aplikacija želi poslati podatke krmilniku, se bo v vsakem 50. poizkusu pošiljanja povezava poizkusila vzpostaviti ponovno. Razlog, da povezava še ni vzpostavljena, je lahko ta, da uporabnik preprosto ni priklopil krmilnika na računalnik, druga možnost pa je, da se aplikacija na napravi Android še ni zagnala. Lahko gre tudi za napačne nastavitve, ki pa jih bo moral uporabnik nastaviti sam. Razlog, da smo za ponovno vzpostavljanje povezave izbrali vsak 50. poskus pošiljanja, je, da vzpostavitev oz. poizkus vzpostavitve brezžične povezave traja nekaj časa. Sprva, ko smo vzpostavljali povezavo ob vsakem poizkusu pošiljanja, se je odzivnost same aplikacije močno zmanjšala. Krmilniku zaradi optimizacije pošiljamo nove podatke le, če so različni od predhodnih. V C#-aplikaciji tako preverjamo tudi, ali so se podatki od prejšnjega poizkusa spremenili, kar pa je majhna cena za veliko razbremenitev tako aplikacije kot tudi krmilnika.

Podatke pred pošiljanjem še posebej stisnemo. V splošnem pošiljamo tri podatke. Dva izmed njih sta jakosti stranskih modulov, tretji pa predstavlja posamezen LED-trak na sredinskem modulu. Za 28 LED-trakov na sredinskem modulu je potrebno 28 bitov podatkov, za vsak LED-trak posebej. Imamo dva stranska modula s po eno LED-diodo ter sredinski modul z 28 LED-trakovi; poslati moramo dve jakosti za stranska modula plus 28 bitov za vsak posamezni trak. Odločili smo se, da je za jakost dovolj 6-bitna natančnost; tako imamo za posamezno jakost 64 različnih vrednosti svetlosti, za oba stranska modula pa porabimo skupaj 12 bitov. Tem bitom dodamo še 28 bitov, ki predstavljajo posamezen LED-trak v sredinskem modulu. Vse podatke ene analize signalov tako pošljemo s 5 bajti.



Slika 4.5: Format podatkov, ki se pošiljajo

Pred vsemi temi podatki dodamo še glavo, ki sporoči krmilniku, za katere podatke gre. Ko krmilnik prejme podatke, najprej preveri glavo podatkov, da ugotovi, koliko podatkov mora počakati, in preveri, če so podatki veljavni. Če krmilnik sprejme podatke z glavo, ki je ne pozna, bo trenutno glavo preskočil in vzel naslednji podatek v vrsti za glavo. To bo ponavljal, dokler bo imel na voljo podatke oziroma dokler ne bo dobil pravilne glave podatkov. Tako je rešen tudi problem preklopa med USB- in TCP/IP-povezavo, ko poteka analiza signalov, saj se lahko medtem, ko se pošiljajo podatki, povezava prekine in se ne uspejo poslati celotni podatki. Tako krmilnik le počaka na pravilno glavo in ugotovi, kje je začetek novih podatkov. Ko krmilnik sprejme podatke s pravilno glavo, ima zaradi predhodne priprave podatkov enostavno obdelavo, saj je zamikanje bitov enostavna in hitra operacija.

4.3.1 Program na Androidu za posredovanje

Kot že omenjeno, smo izdelali še eno androidno aplikacijo, ki deluje kot posrednik med C#-aplikacijo ter krmilnikom Arduino. Na začetku, ko se aplikacija zažene, izberemo vrata, na katerih želimo sprejemati podatke od C#-aplikacije, nato pa lahko te podatke le izpisujemo na zaslon ali pa jih posredujemo naprej krmilniku, čemur je aplikacija tudi namenjena. Podatke lahko pošiljamo tudi C#-aplikaciji, vendar je to bolj za testne namene.

Aplikacija je sestavljena iz treh delov. Prvi del skrbi za sprejemanje odjemalcev, ki se lahko na aplikacijo (vu je mišljena C#-aplikacija) povežejo preko povezave TCP/IP. Ravno tako skrbi za sprejemanje podatkov/sporočil od posameznih odjemalcev. Ta del aplikacije poteka v svoji niti. Drugi del aplikacije skrbi za sprejemanje podatkov od krmilnika, ki ravno tako poteka v svoji niti. Tretji del aplikacije pa skrbi za celotno povezavo obeh delov z uporabniškim vmesnikom ter za posredovanje podatkov po USB-vodilu do krmilnika.

Slika 4.6: Strežnik Android

4.4 Krmiljenje LED-modulov

Podatke lahko v krmilnik prejmemo preko USB-povezave računalnika ali pa USB-povezave androidne naprave. Krmilnik celotni čas preverja, če ima na kateri od teh povezav na voljo nove podatke (bajte). Če jih ima, jih shrani v vrsto k ostalim, ki jih je že prejel. Ko se vrsta napolni s tremi bajti, se ti podatki pošljejo v nadaljnjo obdelavo.

Sprva se gleda le prvi bajt podatkov (glava), ki pove, za katero kategorijo so namenjeni podatki. Lahko so namenjeni krmiljenju 12, 20 ali pa 28 LED-trakov; od tega je odvisna tudi potrebna dolžina podatkov. Podatki pa so lahko namenjeni tudi preklapljanju med različnima načinoma predstavitve efektov, torej če svetijo le zeleni LED-trakovi ali pa barvni.

Listing 4.4: Koda, ki sprejema podatke v krmilniku

```
if (Serial.available() > 0) {
      bufferVrsta.push(Serial.read());
      if(bufferVrsta.count()>=3){
            obdelajNaslednjePodatke();
      }
}
byte data[256];
if(acc.isConnected()){
      int len = acc.read(data,sizeof(data),1);
      if(len > 0){
            for(int i=0; i<len; i++){
                  bufferVrsta.push(data[i]);
            }
            if(bufferVrsta.count()>=3){
                  obdelajNaslednjePodtke();
            }
      }
}
```

Ko krmilnik na podlagi prvega bajta podatkov ugotovi, da mora krmiliti na primer 28 LED-trakov, ve, da mora biti v vrsti skupno vsaj 6 bajtov podatkov; v nasprotnem primeru se metoda za obdelovanje podatkov zaključi, krmilnik pa počaka, da dobi nove podatke preko računalnika oziroma naprave Android. Če pri obdelavi krmilnik naleti na bajt podatkov, ki ne predstavlja nobene lastnosti, ga zavrže in nadaljuje z obdelavo naslednjih bajtov.

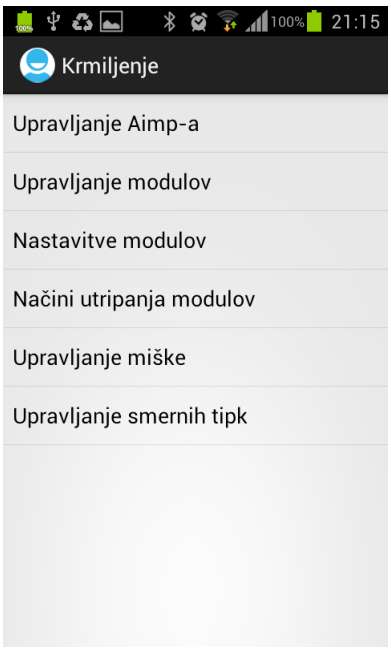
Ko krmilnik sprejme dovolj podatkov s pravilno glavo, razreže prve štiri bajte na posamezne bite tako, da dobi ustrezne bite, ki predstavljajo posamezne LED-trakove sredinskega modula. Za krmiljenje stranskih modulov pa združi zadnjih 6 ter predzadnjih 6 bitov ter jih pretvori v celoštevilsko število. Tako dobi intenziteto za stranska modula. Za krmiljenje posameznih LED-trakov iz sredinskega modula uporabljamo metodo `digitalWrite(pin, value)`, pri čemer je pin celoštevilsko število, ki predstavlja številko pina, value pa je lahko ena izmed celoštevilskih konstant `HIGH/LOW`. Za krmiljenje stranski modulov uporabimo metodo `analogWrite(pin, vrednost)`. Vrednost lahko zavzame katerokoli celo število na območju med 0 in 255. Predhodno moramo vse pine, ki jih bomo uporabili, inicilizirati. To storimo z metodo `pinMode(pin, mode)`. Mode je lahko ena izmed celoštevilskih konstant `INPUT/OUTPUT`.

Ker je v sredinskem modulu skupno 41 LED-trakov, ki so razdeljeni v dve skupini, so njihovi pini, s katerimi jih krmilimo, shranjeni v dvodimenzionalni tabeli. Tako lahko zelo hitro dostopamo do njih. LED-trakovi niso zaporedno priključeni na pine, ker krmilnik potrebuje posamezne pine za druge stvari, zato nismo mogli uporabiti le odmika, ampak je bilo potrebno za vsak trak posebej shraniti pin, ki ga krmili. Ravno tako so le na zgornjem delu sredinskega modula podvojeni LED-trakovi, tako da so spodnji LED-trakovi v obeh skupinah enaki, tako v skupini zelenega kakor v skupini barvnega efekta. Ko krmilnik te podatke obdela, jih iz vrste zavrže in začne s prejemanjem novih podatkov ter z njihovo obdelavo.

4.5 Androidna aplikacija za krmiljenje predvajalnika Aimp

Za povezavo androidne aplikacije ter C#-aplikacije smo uporabili TCP/IP-povezavo. Ob zagonu se odpre vmesnik, preko katerega uporabnik vnese potrebne podatke. Uporabnik mora vnesti IP-naslov računalnika, na katerem je postavljen strežnik s C#-aplikacijo in zagnan Aimp, in številki vrat za posamezni aplikaciji. Nato pa se lahko v naslednjem meniju izbira med posameznimi možnostmi aplikacije. Povezave se vzpostavljajo ob trenutkih, ko jih potrebujemo, zato uporabniku ni potrebno vnašati vseh podatkov, če želi krmiliti le predvajalnik Aimp. Če se vmes povezava prekine, se bo poskušala ponovno vzpostaviti ob trenutkih, ko jo bomo potrebovali.

(a) Androidna aplikacija - začetni prikaz

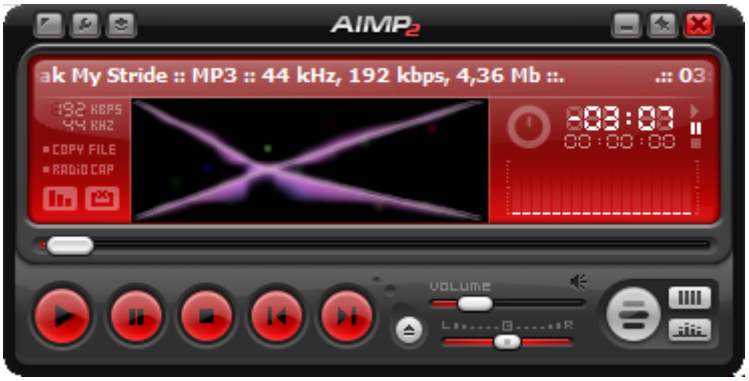


(b) Meni androidne aplikacije

4.5.1 Aimp

Aimp je brezplačni predvajalnik glasbe, ki je zelo podoben predvajalniku Winamp. V Aimp lahko dodamo tudi nekatere razširitve, ki so na voljo za predvajalnik Winamp. Ena izmed njih, ki jo uporabljam, je DFX Audio Enhancer. Ta razširitev omogoča dodatno nastavljanje dinamike, nizkih tonov in še nekaj drugih nastavitev, s katerimi lahko nastavljamo predvajanje po občutku. Tudi sam Aimp brez razširitev omogoča razne efekte ter prilaganje skladb lastnim željam (hitrost, tempo, količina tonov ...).

Glavna razloga, da sem izbral Aimp, sta preprostost in hitrost. Ostali predvajalniki glasbe, ki sem jih preizkusil in so bili dokaj sprejemljivi na pogled in uporabo, so za nalaganje večjega števila skladb namreč potrebovali veliko več časa kakor Aimp. Predvidevam, da je razlog za Aimpovo hitrost to, da Aimp ne bere celotne skladbe in vseh njenih podatkov, ko skladbo naložimo, temveč prebere le naslove datotek, potem pa začne z branjem podatkov o skladbah. Medtem ko bere podatke o posameznih skladbah, lahko mi že upravljamo z njim in predvajamo glasbo. Poizkusili smo tudi z nekaterimi predvajalniki na platformi Linux, vendar so bili rezultati še slabši. Nekateri predvajalniki so se med predvajanjanjem skladb nepravilno zaustavili(»zrušili«), drugi pa so odnehali že med samim nalaganjem večjega števila skladb na seznam predvajanj.



Slika 4.8: Predvajalnik Aimp

Za predvajalnik Aimp je spisana tudi razširitev AIMP Web Control, ki nam omogoča krmiljenje predvajalnika preko spletnega brskalnika s pomočjo zahtev HTTP. Ta razširitev je napisana v programskem jeziku C++ in je izdana pod licenco BSD 3-Clause. To razširitev uporablja tudi nekaj drugih aplikacij, zraven same razširitve pa je tudi program s katerim lahko takoj preko brskalnika krmilimo naš predvajalnik glasbe [9]. Razširitev in njeno dokumentacijo, kjer je njena uporaba ponazorjena s primeri, lahko prenesemo s spletne strani:

https://code.google.com/p/aimp-web-ctl/downloads/list

Naložimo jo tako, da vsebino prenesene datoteke .zip skopiramo v direktorij predvajalnika Aimp. Če smo pri nameščanju predvajalnika na računalnik pustili privzeti direktorij, je pot do direktorija:

```
C:\Program Files (x86) \AIMP2 \PlugIns
```

V nasprotnem primeru moramo vsebino skopirati v spremenjen direktorij, v katerega smo namestili Aimp. Lahko prenesemo datoteko .exe, ki predvajalnik namesti na pravilno lokacijo in nam glede direktorija ni potrebno skrbeti. Privzeta vrata za dostop do razširitve so 38475, drugi varnostni ukrepi pa niso nastavljeni. Zraven same razširitve `aimp_web_ctl.dll` je tudi direktorij `aimp_web_ctl`, v katerem je konfiguracijska datoteka `config.cfg`. S pomočjo te datoteke lahko spremenimo številko vrat, preko katerih dostopamo do razširitve predvajalnika. Ravno tako lahko nastavimo geslo, ki ga je potrebno vnesti pred krmiljenjem. Poleg teh dveh lahko v konfiguracijski datoteki nastavimo še nekaj drugih stvari po lastnih željah, če nam privzete nastavitve ne ustrezajo.

4.5.2 Pošiljanje zahtev HTTP

Za pošiljanje zahtev HTTP ob kliku na gumbe generiramo novo instanco razreda, ki je razširjen iz razreda AsyncTask. Ta razred vsebuje metodi `doInBackround(String... uri)` in `onPostExecute(String r)`. Razred AsyncTask nam omogoča, da v drugi niti ustvarimo vzporedni proces, ki izvede zahtevo in počaka na odgovor. Tako dosežemo, da je naša aplikacija še vedno odzivna na uporabnikove iterakcije, medtem ko se izvaja zahteva. Vzporedni proces začne teči z metodo `doInBackround`, kjer izvedemo zahtevo, zaključi pa se z metodo `onPostExecute`, ko od zahteve prejmemo odgovor. Pri tem lahko nastavimo tudi čas, po katerem se avtomatično zaključi, če od predvajalnika ne prejme odgovora. Ravno tako lahko dodamo v razred metodo `onProgressUpdate(Integer... pr)`, s pomočjo katere lahko uporabniku sporočamo vmesne rezultate. Če uporabljamo te tri metode, nam ni potrebno skrbeti za njihovo sinhronizacijo med vzporednimi procesi, saj za to poskrbi že razred AsyncTask.

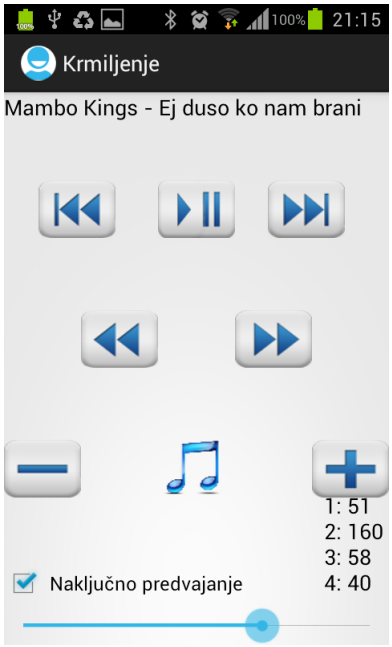
Listing 4.5: Metoda za izvrševanje HTTP-zahtev

```
public void executeCommand(String com){
   try{
      BackHttpRequest backTask=new BackHttpRequest(this);
      backTask.execute("http://"+ipAndPort+"/?action="+com);
   }
   catch(Exception e){
      Log.e("napaka", "Napaka pri http zahtevi."+e);
      }
}
```

Odgovor predvajalnika Aimp prejmemo v razredu, ki je razširjen AsyncTask, do razreda, ki je podal zahtevo, pa ga prenesemo s pomočjo sporočil `Message`, kjer iz njega izluščimo potrebno informacijo. Informacija, ki jo pridobimo, je lahko skladba, ki se trenutno predvaja, glasnost, seznam predvajanja ali pa katera druga lastnost predvajalnika, ki smo jo zahtevali.

4.5.3 Krmiljenje Aimpa

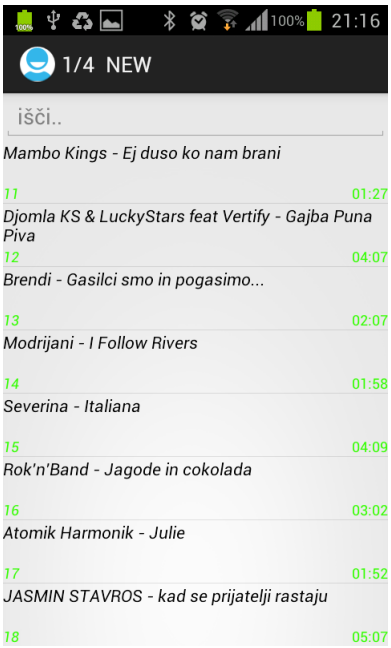
Ko na aplikaciji preidemo v meni upravljanje Aimpa, se nam ob pravilni nastavitvi naslova IP in vrat prenesejo naslov trenutne skladbe in splošne nastavitve, kot je glasnost. Po tem pa se začnejo prenašati seznami skladb.



Slika 4.9: Meni upravljanja Aimpa

Seznam skladb prenašamo po 1000 skladb hkrati; na začetku naredimo zahtevo za prvih 1000 skladb in ko dobimo odgovor, naredimo novo zahtevo za naslednjih 1000 skladb. To ponavljamo, dokler predvajalnik vrača skladbe oziroma dokler se ne prenesejo celotni seznami predvajanj. Glavni razlog za to je, da smo to aplikacijo razvijali na računalniku na emulatorju. Emulator je, kot vemo, zelo počasen in manj zmogljiv od dejanske naprave Android. Če smo poizkusili prenesti več kot 1000 skladb, je prišlo do napake pri prenosu in aplikacija na emulatorju se ni več odzivala. Kasneje smo spoznali, da na mobilnem telefonu to ni težava in lahko hkrati prenesemo vse skladbe. Tako smo preizkusili oba načina - prenos seznama po 1000 skladb naenkrat ter prenos celotnega seznama.

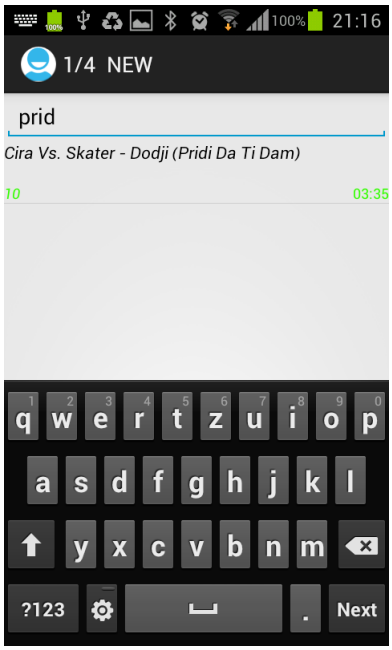
Ugotovili smo, da v času prenosa skladb ni bistvene razlike; ta je bila manjša od 5%. Zato smo se odločili, da bomo uporabili nekoliko počasnejši način, torej prenos po 1000 skladb, ker ima uporabnik tako pregled nad stanjem aplikacije. Med samim prenosom seznamov skladb lahko sami krmilimo predvajalnik z osnovnimi ukazi, kot so: sprememba glasnosti, stop, predvajaj, premor, previjanje naprej ter nazaj. Ko se z računalnika prenesejo vse skladbe (vsi seznami predvajanj), lahko preko aplikacije dostopamo do vseh seznamov predvajanj (gumb v obliki note med gumboma – in +). Seznam nas postavi na trenutno skladbo, ki je izbrana v predvajalniku Aimp.



Slika 4.10: Seznam predvajanj v androidni aplikaciji

Med posameznimi seznami lahko preklapljamo preko menijske tipke, lahko pa tudi preprosto drsamo s prstom levo ali desno, ob pogoju da imamo tudi v predvajalniku več seznamov predvajanj. Pri realizaciji zaznavanja drsenja levo in desno je nastopil problem. Če smo uporabil privzete metode, ki zaznajo premik, je to zelo upočasnilo samo drsenje navzdol in navzgor po skladbah. Zato smo morali implementirati lastno metodo, ki se proži ob dotiku na seznam skladb, ter ustrezno reagirati na dotik.

Za prikaz vseh pesmi smo uporabili `ListActivity`, v katerem smo generirali lastni adapter za prikaz. Adapter je vmesnik med podatki in pogledom, uporabljamo ga za prikaz skladb v seznamu. Na vrhu seznama smo dodali tudi polje za vnos, s pomočjo katerega lahko iščemo želeno skladbo po naslovu. Ob pisanju naslova se nam v realnem času osvežujejo podatki.



Slika 4.11: Iskanje skladb po seznamu

Listing 4.6: Metoda, ki skrbi za osveževanje seznama ob tipkanju naslova

```
private TextWatcher filterTextWatcher = new TextWatcher() {
      public void afterTextChanged(Editable s) { }
      public void beforeTextChanged(CharSequence s, int
         start, int count,int    after) { }
      public void onTextChanged(CharSequence s, int start,
         int before,int count)  {
            myAdapter.getFilter().filter(s);
      }
};
```

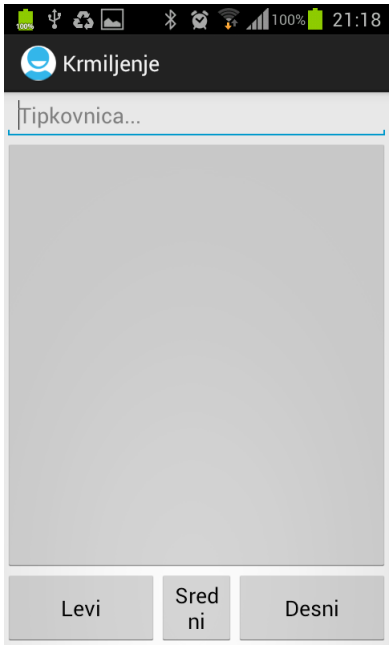
Iz predhodne kode je razvidno, da je za osvežitev v večji meri odgovoren adapter, ki izvede filtriranje. Razlog, da smo ustvaril lasten adapter, je hitrost. Ker smo želeli, da sta poleg naslova prikazana tudi id skladbe ter njena dolžina, za to nismo mogli več uporabiti najenostavnejšega adapterja, ki nam je ponujen, temveč naprednejšega, ki ima na razpolago več polj. Posamezno filtriranje se je sicer zelo upočasnilo, tako da smo morali po tem, ko smo vtipkali eno črko, počakati sekundo ali dve, da se je seznam osvežil. Razlog za to je bil, da je ta, naprednejši adapter filtriral po vseh podanih poljih, kar pa za nas ni bilo primerno, saj se je iskanje močno upočasnilo. Tako smo ustvarili lasten adpater, ki je imel še vedno tri polja, a je ob iskanju želene skladbe iskal le po naslovu skladbe, ne pa tudi po id-ju in dolžini, kakor je to delal privzeti naprednejši adapter.

Ob kliku na skladbo v seznamu predvajanja se ta nemudoma začne predvajati. Dodali smo tudi dogodek na dolgi klik, ki odpre dialog, v katerem lahko postavimo želeno skladbo v vrsto. To pomeni, da se bo ta skladba začela predvajati za trenutno skladbo oziroma za skladbami, ki smo jih predhodno že dodali v vrsto predvajanja. Ravno tako pa lahko skladbo odstranimo iz vrste predvajanja. Vrste predvajanja ne shranjujemo v aplikaciji, ravno tako je ne moremo dobiti iz predvajalnika. Razlog, da je ne shranjujemo, je ta, da ne moremo vedeti, kdaj je šla skladba naprej oziroma kdaj je bila v predvajalniku odstranjena iz vrste. Zato smo se raje odločili, da te možnosti ne realiziramo, čeprav jo včasih pogrešamo.

Ob testiranju aplikacije smo spoznali, da lahko ob iskanju želenih skladb po seznamu uporabnik ponesreči klikne na skladbo, medtem ko se želi pomakniti navzgor oziroma navzdol po seznamu predvajanja. Ob kliku se izbrana skladba nemudoma začne predvajati in prekine trenutno predvajano skladbo. Da bi to preprečili, smo v meni dodali možnost, ki onemogoči hitro izbiranje. Tako se s klikom na skladbo ne začne predvajati izbrana skladba, lahko pa še vedno dodajamo skladbe v vrsto z dolgim klikom. Če želimo klikanja zopet omogočiti, v meniju ponovno izberemo opcijo "omogoči hitro izbiranje".

4.5.4 Upravljanje tipkovnice in miške

Pri C#-aplikaciji smo že omenili, da smo v aplikacijo dodali tudi strežnik. Ta strežnik uporabljamo pri krmiljenju same aplikacije ter pri upravljanju miške in tipkovnice. Vsi prenosi med C#-aplikacijo ter androidno aplikacijo potekajo po eni sami povezavi. Za simuliranje tipkovnice smo uporabili element EditText, na katerega smo dodali dogodek, ki se proži ob spremembi besedila. V tem dogodku ob vsaki novi vrstici pošljemo trenutne podatke C#-aplikaciji ter jih pobrišemo iz trenutnega elementa EditText.



(a) Upravljanje miške in tipkovnice

(b) Upravljanje smernih tipk

Listing 4.7: Koda, ki se odziva na vneseno besedilo v vnosnem polju

```
if(s.length()>1&&s.charAt(s.length()-1)=='\n') {
        Charset charset = Charset.forName("UTF-8");
        byte[] bytes = s.toString().getBytes(charset);
        Beginig.povezava.poslji(bytes.length,22,bytes);
        tipkovnica.setText("");
}
```

Ko aplikacija v C# prejme sporočilo, ki predstavlja znake, ki jih mora simulirati, to stori s pomočjo simulatorja Windows Input. Do njega lahko dostopamo ob predhodni vključitvi knjižnice WindowsInput:

Listing 4.8: Ukaz, ki simulira tipkovnico v C#-aplikaciji

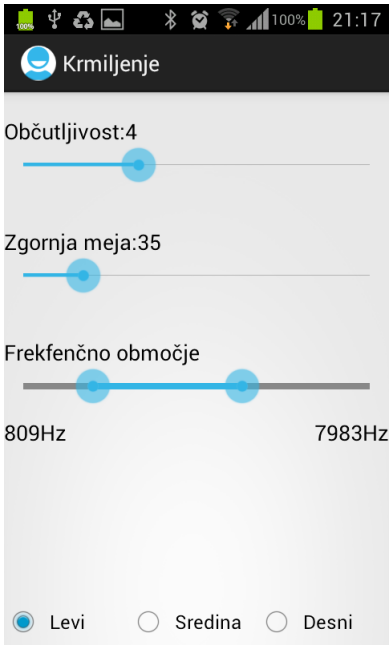
```
WindowsInput.InputSimulator.SimulateTextEntry(
      ((char)podatki[j + i + 2]).ToString());
```

Ker preko elementa EditText ne moremo pošiljati smernih tipk, smo za ta namen dodali novo aktivnost (activity), ki vsebuje 4 gumbe. Ti ob kliku pošljejo ukaz za posamezno smerno tipko, ki jih simuliramo na enak način kot ostale znake s tipkovnice. Aplikacijama smo dodali tudi možnost upravljanja miške. V aplikaciji Android smo za ta namen dodali 4 gumbe. Trije so splošni levi, desni in sredinski klik, četrti gumb pa smo uporabili za ponazarjanje drsenja miške, ki smo ga raztegnili čez celotni preostali del zaslona. Za osnovne premike/dotike, kot so drsenje z enim prstom, enojni klik in dvojni klik, smo uporabili metode iz razreda GestureDetector.

Ker pa GestureDetector ne zazna akcij z več prsti, smo pred klicem metod iz GestureDetector dodali lastno metodo, s pomočjo katere zaznamo, ali se je uporabnik dotikal gumba z več kot enim prstom. Če uporabnik uporablja gumb z enim prstom, prepustimo nadaljnjo obravnavo dogodka GestureDetectorju, sicer pa jo prevzamemo sami. Če ugotovimo, da uporabnik upravlja z dvema prstoma in da z obema drsi navzgor oziroma navzdol, pošljemo C#-aplikaciji sporočilo za vrtenje koleščka navzgor oziroma navzdol. Tako upravljamo s koleščkom na enak način kakor recimo na sledilnih ploščicah prenosnih računalnikov. Ko pridejo podatki do C#-aplikacije, zopet nanje le ustrezno reagiramo, tako da prožimo dogodke nad miško (klik, dvojni klik, premik ...) na podoben način kakor pri simuliranju tipkovnice.

4.5.5 Upravljanje nastavitev C#-aplikacije

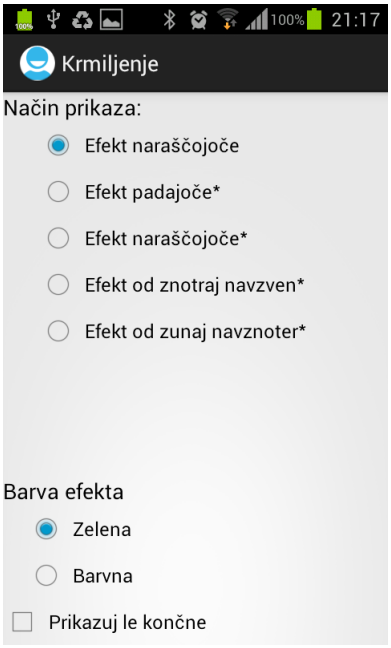
Do možnosti upravljanja nastavitev C#-aplikacije pridemo tako, da v osnovnem meniju aplikacije kliknemo na "Nastavitve modulov". Ob kliku se odpre vmesnik, preko katerega lahko spreminjamo nastavitve za vsak modul posebej.



Slika 4.13: Upravljanje nastavitev C#-aplikacije

Preko androidne aplikacije lahko nastavljamo občutljivost, zgornjo mejo ter frekvenčno območje posameznih modulov C#-aplikacije. Ko izberemo modul, ki ga želimo nastavljati, aplikacija pošlje C#-aplikaciji zahtevo za podatke. C#-aplikacija nanjo odgovori ustreznimi podatki, ki se prikažejo na zaslonu. Za prikaz območja frekvenc smo uporabili element RangeSeekBar avtorjev Stephan Titte, Peter Sinnottm, Thomas Barrasso, saj privzeti elementi ne podpirajo grafične možnosti izbiranja območja na nekem intervalu. Na vse elemente smo dodali dogodke, tako da se ob kakršnikoli spremembi s strani uporabnika prožijo različni dogodki. Če uporabnik spremeni določeno nastavitev, se ta pošlje C#-aplikaciji, ki jo nato upošteva.

V androidno aplikacijo smo dodali tudi možnost izbire efektov. Do nje dostopamo tako, da v osnovnem meniju kliknemo na "Način utripanja modulov".



Slika 4.14: Spreminjanje efektov

Ob vstopu na to možnost se vsi efekti (imena) prenesejo na androidno aplikacijo in se prikažejo kot radijski gumbi, gumb trenutnega efekta, ki ga upoštevamo, pa je izbran. Če uporabnik izbere drugega, se ta sprememba pošlje C#-aplikaciji in se tam ustrezno upošteva.

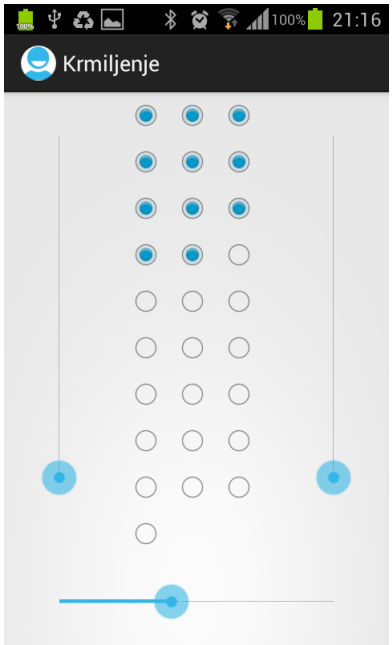
Listing 4.9: Izsek kode za dinamično dodajanje radijskih gumbov

```
skupinaGumbov.removeAllViews();
for(int i=0; i<sporocilo.efekti.length; i++) {
      RadioButton gumb= new RadioButton(trenutnaAktivnost);
      gumb.setText(sporocilo.efekti[i]);
      gumb.setId(i);
      skupinaGumbov.addView(gumb);
      ..........................
```

4.5.6 Krmiljenje LED-modulov

V androidni aplikaciji smo omogočili tudi krmiljenje posameznih LED-modulov na podoben način kakor v C#-aplikaciji. Tudi uporabniški vmesnik je podoben tistemu v C#-aplikaciji. Tako lahko stranska modula krmilimo z drsniki, sredinskemu modulu pa lahko poleg drsnika, ki predstavlja delež, krmilimo tudi posamezni LED-trak posebej. Format pošiljanja podatkov je popolnoma enak formatu, ki ga Arduinu pošilja C#-aplikacija. Tako C#-aplikacija to sporočilo le posreduje naprej in nanj ne reagira.

Za isti format podatkov smo se odločili iz razloga, da bi mogoče v nadaljevanju direktno povezali androidno aplikacijo z drugo androidno aplikacijo, ki bi bila na mobilni napravi, priključeni direktno na krmilnik Arduino. Tako bi lahko z mobilnim telefonom brezžično preko povezave wifi prižigali in ugašali posamezne module.



Slika 4.15: Krmiljenje LED-trakov

Poglavje 5

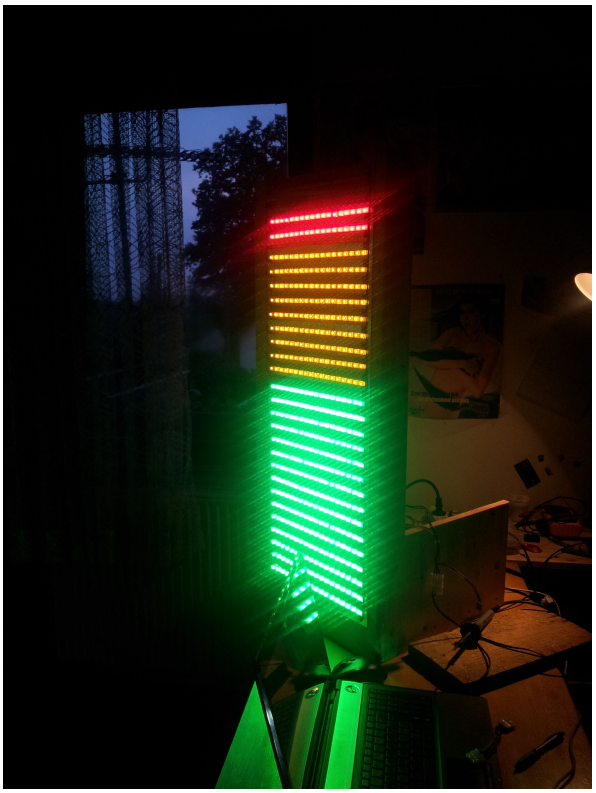
Zaključek

Cilj diplomske naloge je bil krmiljenje LED-trakov v ritmu glasbe. Ta cilj sem tudi dosegli, ker pa sem ga dosegel precej hitro, sem si zastavil dodatni cilj - krmiljenje predvajalnika Aimp. Pri realizaciji projekta sem naletel na kar nekaj težav. Pri izdelavi programske opreme za krmiljenje LED-trakov ni bilo omembe vrednih težav v C#-aplikaciji, pri programski kodi na krmilniku Arduino pa sem naletel na težavo, ki je nisem uspel odpraviti; po odgovorih na spletu sodeč se je odpraviti niti ne da. Težava je v tem, da če želim svoj mobilni telefon uporabljati kot strežnik, tako da ga priključim na krmilnik Arduino, se ne bo uspel povezati, če bo naložen operacijski sistem, novejši od različice 4.03. Ta problem naj bi bil značilen le za moj mobilni telefon, Samsung Galaxy S2, na ostalih napravah pa naj bi povezava delovala tudi na novejših operacijskih sistemih.

Na teževe sem naletel tudi pri sami izgradnji modulov. Kot že omenjeno, sem jih sprva izdelal tako, da nisem smel ničesar premikati. Kasneje, ko sem se odločil izdelati resnejšo verzijo, sem prvih 40 tranzistorjev prispajkal preblizu; tako sem jih moral ponovno kupiti skupaj s ploščami, na katere sem spajkal, ter z upori, zaradi česar sem za kratek čas izgubil zagon za nadaljevanje.

Na večino težav pa sem naletel ob pisanju androidne aplikacije, nekatere izmed njih sem v diplomski nalogi že omenil. Kljub vsem težavam pa mi je uspelo uspešno realizirati zastavljene cilje in na koncu sem dobil delujoč izdelek, na katerega sem precej ponosen.

Celoten izdelek bi se dalo še izboljšati oziroma nadgraditi. V mislih imam dodatne module drugačne barve, kar bi izdelek še popestrilo. Ravno tako bi lahko na krmilnik priklopili fotosenzor ali pa senzor gibanja, na podlagi katerih bi lahko izvajali razne efekte, saj je na krmilniku ostalo vseh 15 analognih vhodov. Možnost za dodatne efekte pa sem že podprl, tako da lahko vsak napiše svoj efekt in tega ni potrebno pisati v izvorno kodo C#-aplikacije.



Slika 5.1: Izdelan sredinski modul

Slike

Literatura

[1] Arduino adk.
Dostopno na: *http://arduino.cc/en/Main/ArduinoBoardADK*

[2] W. F. Tom Harris. How light emitting diodes work.
Dostopno na: *http://electronics.howstuffworks.com/led3.htm*

[3] Resistor.
Dostopno na: *http://en.wikipedia.org/wiki/Resistor*

[4] Android (operating system).
Dostopno na: *http://goo.gl/CCBF9K*

[5] Z. Whittaker. Android accounts for 75 percent market share.
Dostopno na: *http://goo.gl/aiHxQO*

[6] Developer tools.
Dostopno na: *http://developer.android.com/tools/index.html*

[7] .net framework.
Dostopno na: *http://en.wikipedia.org/wiki/.NET_Framework*

[8] Fast fourier transform.
Dostopno na: *http://en.wikipedia.org/wiki/Fast_Fourier_transform*

[9] V. Dyatlov. Aimp web control plug-in.
Dostopno na: *https://code.google.com/p/aimp-web-ctl/*