Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Benjamin Polovič

Svetlomer za iOS naprave

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Peter Peer

Ljubljana 2014

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: 01950 / 2013
Datum: 5.9.2013

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

Kandidat: **BENJAMIN POLOVIČ**

Naslov: **SVETLOMER ZA IOS NAPRAVE**
LIGHT METER FOR IOS DEVICES

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Za svetlomer, ki se priključi na audio izhod iOS naprave, naredite načrt in implementacijo programske opreme. Razložite celoten postopek izdelave aplikacije z glavnim poudarkom na komunikaciji med svetlomerom in iOS napravo: napajanje svetlomera, sprejemanje podatkov, razumevanje podatkov, ogrodje za uporabo komunikacije, uporabniški vmesnik. Naredite tudi primerjavo celotne rešitve s sorodnimi izdelki.

Mentor:
doc. dr. Peter Peer

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Benjamin Polovič, z vpisno številko **63080124**, sem avtor diplomskega dela z naslovom:

Svetlomer za iOS naprave

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Petra Peera,
- 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 14. marca 2014 Podpis avtorja:

Zahvaljujem se doc. dr. Petru Peeru za vodenje pri izdelavi diplomskega dela in celotni ekipi, ki je sodelovala pri izdelavi svetlomera in je projekt spravila k življenju.

Rad bi se zahvalil tudi staršema za podporo skozi študij.

Kazalo

Povzetek

V fotografiji je pogosta uporaba svetlomera za natančno merjenje svetlobe, ki fotografu omogoči posneti pravilno osvetljeno fotografijo. Na tem področju se kar nekaj časa ni veliko dogajalo, profesionalni svetlomeri so še vedno veliki in dragi. Zato smo se odločili narediti bolj dostopen svetlomer za uporabo z iOS napravo. Svetlomer je majhnih dimenzij, priključi se na audio izhod iOS naprave in omogoča enako natančno merjenje svetlobe kot profesionalni svetlomeri. Diplomsko delo se osredotoča na programsko opremo, razložen je celoten postopek izdelave aplikacije z glavnim poudarkom na komunikaciji med svetlomerom in iOS napravo. Naša rešitev, tako kot profesionalni svetlomer, fotografu omogoča posneti pravilno osvetljeno fotografijo, z boljšim uporabniškim vmesnikom ter dodatnimi funkcionalnostmi, kot je shranjevanje beležk s podatki o uporabljenih foto parametrih.

Ključne besede: svetlomer, fotografija, foto parameter, aplikacija, iOS, Apple, Objective-C

Abstract

A light meter is often used in photography to ensure precise light measurement, which enables the photographer to capture a perfectly exposed photo. There was no major breakthrough in this area for a long time, the professional light meters are still large and expensive. That is why we decided to make a more affordable light meter for use with iOS devices. This light meter has a small footprint, connects through the audio jack fo the iOS device and provides the same precise light measurement as a professional light meter. This thesis focuses on the software, it describes the whole process of the developement of the application and concentrates on the communication between the light meter and the iOS device. Just like a professional light meter, our solution enables the photographer to capture a perfectly exposed photo, with a better user interface and additional functionalities, which include saving a note with information about used photo parameters.

Keywords: light meter, photography, photo parameter, application, iOS, Apple, Objective-C

Poglavje 1

Uvod

V zadnjem času se v svetu pojavlja trend majhnih pripomočkov, ki jih uporabljamo v povezavi s pametnim telefonom. Square, čitalec bančnih kartic [1], ki se preko audio priključka priklopi na pametni telefon in omogoča izvedbo transakcij, je povzročil pravo malo revolucijo na tem področju, saj je omogočil manjšim trgovinam, da lahko enostavno nadomestijo blagajno in terminal. Obstaja tudi termometer Thermodo [2], ki ga priklopimo na enak način in lahko z njim natančno izmerimo temperaturo prostora. Popularne postajajo tudi naprave, ki služijo za to, da določenega predmeta ne izgubimo. Obesimo jo na ta predmet in nato lahko s pametnim telefonom preko Bluetooth-a to napravo najdemo v omejenem območju. Lep primer je slovenski Chipolo [3].

Vse te naprave obstajajo tudi v samostojni izvedbi. Razlogi, da postajajo popularne kot pripomočki za pametne telefone pa so tudi razlogi, zaradi katerih s seboj vedno nosimo pametne telefone: ker imajo velike in lepe zaslone, ponujajo ogromno funkcionalnosti, najboljši uporabniški vmesnik, povezavo z internetom in ker so dovolj majhni, da jih lahko imamo vedno s seboj v žepu.

Ob vseh teh pripomočkih pa smo opazili področje, ki že dolgo ni bilo deležno večje pozornosti – svetlomeri. Na področju svetlomerov, ki jih uporabljajo fotografi, se že približno 10 let ni dogajalo nič razburljivega. Zato smo

se odločili, da bomo naredili pameten svetlomer – svetlomer za pametne iOS naprave.

V diplomskem delu je opisana izdelava aplikacije za iOS naprave od zasnove in načrtovanja do implementacije. Navedena so tudi osnovna načela uporabe svetlomera ter kratek opis strojne opreme svetlomera. Glavni poudarek je na sami aplikaciji in izdelavi komunikacije med svetlomerom in iOS napravo, kot jo vidimo iz aplikacije. Opisana je tudi izdelava uporabniškega vmesnika ter uporaba nekaterih zanimivih ogrodij in knjižnic za iOS.

Poglavje 2

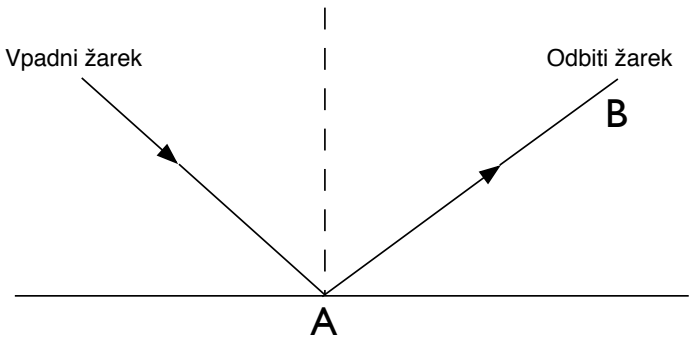
Definicija problema

2.1 Zakaj svetlomer?

V fotografiji se že od nekdaj pojavlja problem določanja svetlosti scene. Da lahko zajamemo sliko, ki ni preveč ali premalo osvetljena, moramo poznati kako svetla je scena. Zelo talentiran fotograf lahko mogoče ugane dovolj dober približek osvetlitve, ki mu omogoči dober posnetek, veliko bolj zanesljiva in natančna pa je uporaba svetlomera.

Svetlomer je torej naprava, ki izmeri količino svetlobe. Fotografi za zajem slike s pravilno osvetlitvijo upravljajo s tremi parametri: občutljivost filma (ISO), čas zajema (koliko časa traja zajem svetlobe) in odprtost zaslonke (velikost odprtine skozi katero pride svetloba). Svetlomer ponavadi vsebuje tudi računalnik, ki fotografu pomaga izračunati enega od foto parametrov: čas ali zaslonko, ki ga uporabi za dosego optimalne osvetlitve slike pri uporabi določenega filma oziroma ISO vrednosti [4].

Svetlomeri se uporabljajo tudi pri snemanju filmov in postavljanju scen pri določanju optimalne količine svetlobe za sceno. Uporabljajo se tudi za ponovno postavitev scene, da se lahko simulirajo isti svetlobni pogoji, zato da prostor na posnetku vedno izgleda enako. Uporabljajo se na splošnem področju osvetlitve, kjer lahko služijo kot pripomoček za zmanjševanje svetlobne onesnaženosti v gospodinjstvih ali zunanjih prostorih in pri gojenju



Slika 2.1: Vpadna in odbita svetloba.

rastlin za zagotavljanje ustrezne stopnje svetlosti. Fotografi (ki uporabljajo studijske luči) in video snemalci uporabljajo ročne svetlomere, da natančno izmerijo svetlobo, ki pada na različne dele subjekta, nato pa ustrezno popravijo osvetlitve za dosego želene stopnjo osvetljenosti.

Obstajata dve vrsti svetlomerov: merilci odbite svetlobe in vpadne svetlobe. Merilci odbite svetlobe merijo svetlobo, ki se odbija od scene, katero slikamo. Vsi merilci vgrajeni v kamere so takšne narave. Merilci odbite svetlobe so narejeni tako, da prikažejo ustrezen nivo osvetlitve za povprečno sceno. Prizor na katerem prevladujejo svetle barve ima višjo odbojnost. Na takem prizoru merilec odbite svetlobe nepravilno oceni razliko odbite svetlobe, kar vodi do premajhne osvetlitve fotografije. Pod-osvetljene slike sončnih zahodov so lep primer tega efekta: svetloba zahajajočega sonca zavede svetlomer kamere in če fotograf tega ročno ne popravi, bo slika močno pod-osvetljena in monotona.

Razliko med vpadno in odbito svetlobo lahko vidimo na poenostavljeni sliki 2.1, ki prikazuje vpadni in odbiti žarek. Če imamo na mestu A subjekt, ki ga želimo fotografirati, in na mestu B postavljen fotoaparat, potem merilec vpadne svetlobe uporabimo na mestu subjekta A, merilec odbite svetlobe pa na mestu fotoaparata B [5].

Da se izognemo pomanjkljivostim merilcev odbite svetlobe, uporabimo

merilec vpadne svetlobe, ki izmeri količino svetlobe, ki pada na subjekt s pomočjo zbiralne sfere (ponavadi prosojna plastična polkrogla), ki je postavljena na vrh svetlobnega senzorja. Ker je vpadna svetloba neodvisna od odbojnosti subjekta, ponavadi ni problemov z napačnimi osvetlitvami subjektov z nenavadno odbojnimi površinami. Primer fotografskega svetlomera podjetja Konica Minolta, ki meri vpadno svetlobo, lahko vidimo na sliki 2.2. Merilec vpadne svetlobe uporabljamo tako, da ga postavimo na mesto subjekta in ga usmerimo proti kameri [6].

Slika 2.2: Fotografski svetlomer.

2.2 Opis naprave

Naprava, ki smo jo razvili, je merilec vpadne svetlobe. Za to vrsto svetlomera smo se odločili, ker je veliko bolj splošno uporaben in ker je zahteva po takih merilcih večja. Zaradi izbire te vrste merilca je bila v veliki meri določena tudi njegova oblika, saj tak merilec za pravilno merjenje uporablja zbiralno sfero oziroma hemisferični difuzor. Ta je pri nas plastična polkrogla, ki je nameščena na enostavno aluminijasto ohišje. Pomembna stvar pri ohišju je še audio priključek, ki se nahaja na spodnjem aluminijastem delu ohišja.

Odločili smo se, da bo naprava s telefonom komunicirala preko zvoka in da se bo priklopila na audio priključek oziroma priključek za slušalke. Tak način komunikacije je sicer nekakšna bližnjica, saj ni najbolj običajen način za priklop dodatne opreme na iOS napravo. Če bi se odločili, da bomo razvili napravo, ki se bo priključila na polnilni/podatkovni priključek naprave, bi pridobili višjo hitrost, večjo električno moč, večjo programsko podporo ipd., a vendar nam je priklop na audio priključek povsem zadoščal. Pravtako nam ni bilo potrebno plačati za certifikat uradne strojne opreme iOS naprav [7]. Naša uporaba je dovolj enostavna in tudi hitrost komunikacije, ki smo jo dosegli, nam ustreza.

Svetlomer preko audio vhoda z iOS napravo komunicira in se preko nje tudi napaja, tako da ne potrebuje baterije. Svetlomer preko kanala za mikrofon sporoča meritve iz svetlobnega senzorja, katere potem telefon prebere in ustrezno interpretira. iOS naprava preko enega od stereo kanalov stalno pošilja določen signal, ki ga svetlomer uporabi za pretvorbo v energijo za napajanje.

2.3 Sorodne rešitve

Zaenkrat je naš izdelek še edini take vrste, torej zunanji svetlomer, ki se priklopi na iOS napravo. Glede na vrsto naše naprave pa bi lahko govorili o dveh sorodnih vrstah naprav, ki že obstajata. Prva so sorodni izdelki za merjenje svetlobe za iOS naprave. Druga pa naprave, ki uporabljajo podoben

način komunikacije.

2.3.1 Izdelki za merjenje svetlobe na iOS napravah

Sem spadajo predvsem aplikacije za iOS, ki izkoriščajo kamero telefona, da pridobijo približek trenutne osvetlitve. V intervalih zajemajo sliko, nato pa povprečijo osvetlitev vseh posameznih točk, da dobijo približen rezultat. Takih aplikacij je več, morda najboljša/najlepša med njimi je Photometer [8]. Glavna hiba teh aplikacij je, da s tem načinom preko iOS kamere ni mogoče pridobiti dovolj natančnega približka osvetlitve, da bi lahko tako aplikacijo resno uporabljali. Poleg tega kamera bere odbito svetlobo, kar smo že ugotovili, da ni vedno zanesljivo.

Obstaja pa tudi pripomoček v obliki plastične kupole, ki kamero iOS naprave spremeni v merilec vpadne svetlobe. Imenuje se Luxi [9] in deluje v kombinaciji s prej omenjenimi aplikacijami. Ta nekoliko izboljša rezultate teh aplikacij, vendar ne dovolj. Poleg tega mora biti oblikovan za ohišje vsake iOS naprave posebej, zato ni povsem univerzalen.

2.3.2 Naprave, ki komunicirajo preko audio vhoda

Obstaja že kar nekaj naprav, ki komunicirajo na podoben način, kot smo ga izbrali mi. Naj omenimo slovenski izdelek za merjenje sladkorja v krvi 2in1.SMART [10]. Še en zanimiv projekt pa je HiJack [11], ki so ga razvili na Univerzi Michigan v ZDA, ki se prav tako napaja preko audio vhoda. Sem spadajo tudi že prej omenjeni Square [1] in Thermodo [2].

Vse te naprave na izviren način izkoriščajo audio vhod telefona za komunikacijo z nekakšnim senzorjem, ki se nahaja na zunanji napravi in ga ni moč najti v iOS napravi.

2.4 Razvoj za iOS

Večkrat smo že omenili iOS (predhodno iPhone OS) operacijski sistem, ki ga je razvil in ga uporablja Apple Inc. Teče na Apple napravah, pri katerih večino interakcije predstavlja zaslon na dotik: iPhone, iPod Touch, iPad. Na vseh teh napravah je uporabniški vmesnik zelo podoben. Trenutno je aktualna verzija iOS7. iOS predstavlja več kot samo operacijski sistem. Sem spada tudi možnost razvijanja aplikacij in podpora razvijalcem ter možnost trženja teh aplikacij na njihovem trgu, App Store [12]. Za razvoj na tej platformi smo se odločili zaradi več razlogov.

Velika prednost iOS je, da je narejen za točno določeno strojno opremo, zato lahko deluje dobro, predvidljivo in brez izjem. Tako so določene velikosti ekrana, hitrost naprave, velikost pomnilnika in ostale podrobnosti s čimer lahko omogočijo boljšo podporo razvijalcem aplikacij. Njihovo orodje za razvoj programe opreme oziroma SDK (angl. Software Development Kit) je zelo dobro v primerjavi z ostalimi in razvijalcu prihrani kar nekaj nevšečnosti.

Fragmentacija iOS naprav je mnogo manjša kot na primer pri Android napravah. To v našem primeru pomeni veliko, saj smo lahko programsko opremo razvijal tako, kot da jo razvijamo samo za eno napravo in ni bilo prilagajanja in testiranja za vsako napravo posebej, kar se lahko izkaže za zelo težko in zamudno. Včasih prilagoditev za določeno napravo celo ni možna in bi jo bilo potrebno izključiti iz podpore.

Platforma iOS sicer ne zaseda največji delež trga, a tudi z mnogo manjšim deležem prinaša največji dobiček med mobilnimi platformami [13]. To nam pove tudi nekaj o uporabnikih te platforme – ti so pripravljeni kupiti še nekaj dodatnega za svoj mobilni telefon, naj bo to aplikacija ali pa zunanja oprema. Po raziskavah sodeč so iOS uporabniki tudi bolj pripravljeni kupiti novo, še nepreizkušeno, neuveljavljeno napravo. Prav tako ti uporabniki veljajo za bolj ustvarjalne, med njimi je več fotografov kot drugje [14].

Glavni programski jezik je Objective-C, ki predstavlja objektno nadgradnjo jezika C. Med pisanjem programa lahko uporabljamo tudi klasični C in C++.

2.5 Cilj naloge

Iz programskega vidika je cilj naloge izdelati aplikacijo za iOS napravo, ki omogoča dobro uporabo zunanjega svetlomera. Svetlomer mora delovati zanesljivo, kar pomeni, da mora biti komunikacija med iOS napravo in svetlomerov kar najbolj zanesljiva. Potrebno je zagotoviti komunikacijo s čim manj napakami, če pa se te že pojavijo pa jih je potrebno ustrezno zaznati in zavreči, da ne pride do uporabe napačnega podatka in prikaza napačne meritve. V primerjavi merjenja s samostojnim profesionalnim svetlomerom mora delovati natančno, z nič ali s čim manj odstopanji.

Čim bolje moramo izkoristiti tudi dobro mobilno platformo, ki lahko z dobro aplikacijo zagotovo ponudi boljšo uporabniško izkušnjo od starejših naprav za merjenje svetlobe. Prikazati moramo tudi napredek v tehnologiji; aplikacija bo po opravljeni meritvi omogočala shranjevanje beležk. Ena beležka bo vsebovala podatke o trenutni meritvi, čas in datum, pripeti bo mogoče sliko in trenutno GPS lokacijo. Te beležke bo možno shraniti pod različne skupine, za vse pa bo mogoče hraniti tudi varnostno kopijo v oblaku.

Poglavje 3

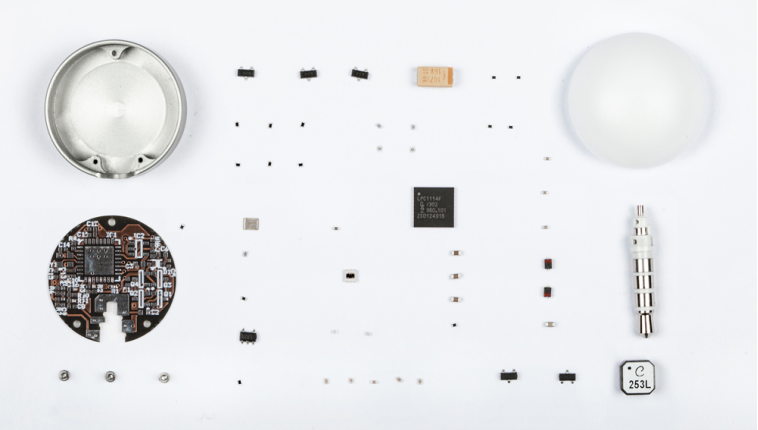
Strojna oprema

3.1 Opis elementov

Za delovanje svetlomera je potrebnih okoli 50 elementov, ki jih lahko vidimo na sliki 3.1. Izmed teh je bistvenega pomena svetlobni senzor. Uporabili smo trenutno najboljši digitalni senzor, ki ga je moč dobiti. Senzor lahko zaznava svetlobo od 0,015 do 64000 luksov. Našemu svetlomeru omogoča zelo natančno merjenje svetlobe in je še bolj natančen od analognih, ki so v uporabi v nekaterih ostalih profesionalnih svetlomerih.

Drugi pomemben element je procesor. Uporabili smo ARM Cortex-M0 procesor. Procesor je 32-biten in lahko deluje s frekvencami do 50 MHz. Med drugim vsebuje 64 kB flash spomina, 8 kB delovnega spomina in eno I^2C vodilo [15].

Pomemben del vezja predstavljajo tudi elementi, ki omogočajo pretvorbo zvočnega signala iz audio vhoda v električno energijo, ki je potrebna za napajanje elementov na svetlomeru.



Slika 3.1: Vsi elementi strojne opreme svetlomera.

3.2 Način delovanja

Za napajanje svetlomera nismo hoteli uporabiti baterije, ker bi to povzročilo dodatne komplikacije tako nam kot tudi uporabniku. Če bi uporabili polnilno baterijo, bi moral svetlomer dobiti dodaten priključek za polnjenje. Če pa bi uporabili navadno baterijo, bi morali predvidevati, da bo uporabnik lahko zamenjal baterijo, ko bi se le-ta izpraznila. Mi pa smo želeli vse skupaj kar se da poenostaviti in narediti napravo, ki deluje zelo trdno in se ne razstavi zlahka.

Tako smo odkrili, da lahko iz audio izhoda iOS naprave pridobimo dovolj električne energije za napajanje elektronike z nizko porabo. Da pa lahko ta električni tok uporabimo, ga je potrebno popraviti, električno napetost je potrebno ojačati in izhod je potrebno filtrirati.

Na vezju se zato nahaja skupek elementov, ki se imenuje "žetvenik" (angl. harvester), ki skrbi za pretvorbo sinusnega signala iz levega kanala audio izhoda v električno energijo. Električna energija se še okrepi z ustrezno tuljavo, nato pa se uporabi za napajanje glavnih porabnikov, svetlobnega senzorja ter mikroprocesorja.

Pri tem načinu pridobivanja električne energije smo odvisni od kvalitete elementov na pretvorniku digitalnega signala v analognega (angl. DAC –

Digital to Audio Converter) na audio izhodu. Od moči tega pretvornika je potem odvisen tudi naš izkoristek električne energije [16]. Zaenkrat vse iOS naprave uporabljajo dovolj močne pretvornike, da zagotovijo dovolj energije.

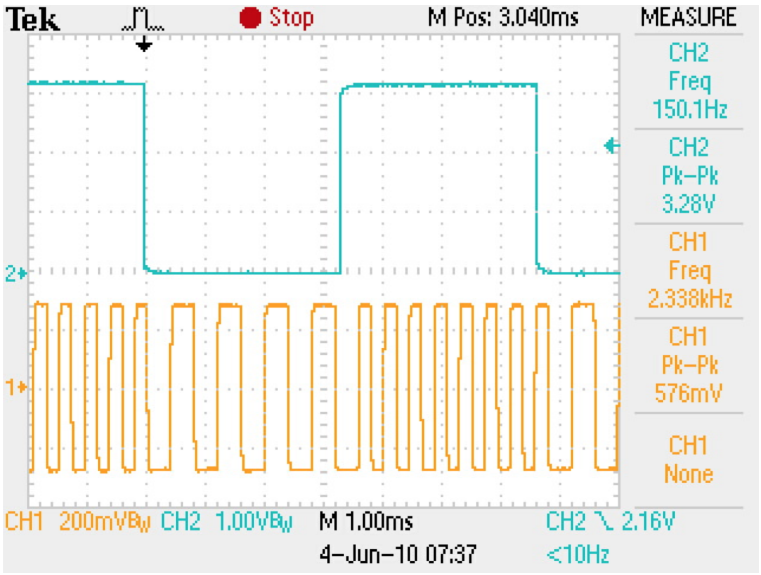
3.3 Pošiljanje podatkov

Med delovanjem svetlomera, ko so vsi elementi oskrbljeni z električno energijo, ta vedno ponavlja zanko za branje in pošiljanje meritve. Komunikacija med svetlomerom in iOS napravo temelji na FSK modulaciji (angl. Frequency-Shift Keying). FSK je shema za frekvenčno modulacijo, s katero prenesemo digitalno informacijo preko diskretnih frekvenčnih sprememb prenosnega signala [17].

V našem primeru gre za binarni FSK, torej uporabljamo dva tona. Na sliki 3.2 lahko vidimo primer FSK modulacije prikazane na osciloskopu, kjer zgornji del predstavlja podatkovni tok, spodnji del pa FSK modulacijo zgornjih podatkov. V našem primeru je digitalna ničla predstavljena s 4900 Hz tonom, digitalna enka pa s 7350 Hz tonom. Za določen čas se logična ničla pretvori v 4900 Hz sinusni signal, logična ena pa v 7350 Hz sinusni signal. S tako modulacijo lahko dosežemo hitrost do 1225 bitov/sekundo [18]. Za bolj optimalno porabo naš svetlomer uporablja nižjo hitrost, ker nam to povsem zadošča. Zato tudi procesor ne deluje z maksimalno frekvenco, ampak z nižjo, ker se tako zniža tudi njegova poraba.

Mikroprocesor preko I^2C vodila prebere trenutno vrednost iz svetlobnega senzorja. To vrednost ustrezno interpretira in jo razbije na štiri dele, vsakega dolgega osem bitov. Te bajte potem modulira s FSK modulacijo in dobljen signal pošlje na kanal, ki se sicer uporablja za mikrofon na audio vhodu. Preko tega kanala potem iOS naprava prebere to informacijo.

Ta postopek se ponovi približno petkrat na sekundo. Ta hitrost povsem zadošča za našo uporabo. Če bi želeli, bi lahko z maksimalno hitrostjo na iOS napravo dobili 12 meritev iz senzorja svetlobe na sekundo, vendar bi bilo to absolutno odveč in nepotrebno.



Slika 3.2: Primer FSK modulacije podatkovnega toka [16].

Poglavje 4

Načrtovanje programske opreme

Veliko različnih aspektov zastavljene končne aplikacije zahteva dobro načrtovanje. Če se nekdo prvič sreča z Objective-C jezikom, mora biti še posebej pazljiv, ker ima ta jezik nekoliko specifično sintakso in posebnosti kot so protokoli, kategorije, bloki kode, napake med izvajanjem so predstavljene kot objekti razreda `NSError` itd. [19].

Za pravilno delovanje aplikacije in svetlomera je potrebno razrešiti kar nekaj stvari. Najprej je treba določiti, kako in s kakšnim signalom najbolj optimalno napajati svetlomer. Potrebno je definirati FSK demodulacijo sprejetih moduliranih bitov. Uskladiti je potrebno komunikacijo in tip paketov, da prihaja do čim manj napak pri sprejemanju podatkov. Dobljeni podatek oziroma meritev je potrebno pravilno interpretirati in ga s fotografskimi enačbami ustrezno uporabiti za prikaz informacije. Zajetna funkcionalnost aplikacije je tudi shranjevanje podatkov v oblak, tu se je potrebno odločiti za način shranjevanja – ali uporabiti obstoječo rešitev ali izdelati svojo.

Pri načrtovanju in implementaciji aplikacije si bomo veliko pomagali z Cocoa Touch ogrodji. Ta zbirka vsebuje ključna ogrodja za izdelavo iOS aplikacij. Pomagajo nam definirati izgled aplikacije in omogočajo postavitev osnovne strukture aplikacije in podporo osnovnih funkcionalnosti, kot je vnos

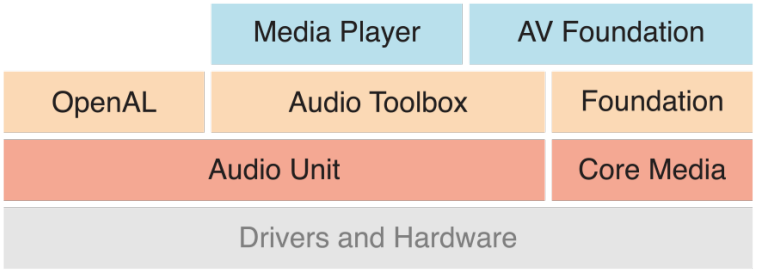
z dotikom [12].

4.1 Napajanje svetlomera

Za učinkovito pretvorbo energije preko zvočnega signala je potrebno določiti frekvenco sinusnega signala, ki bo poslan na svetlomer. Po teoretičnih izračunih bi morala biti ta frekvenca 22.9 kHz [16]. Vedeti moramo, da je ta frekvenca povsem na robu zmožnosti iOS naprav. Apple namreč trdi, da ima na primer iPhone frekvenčni razpon od 20 Hz do 20 kHz [20]. Na srečo imamo popoln nadzor znotraj frekvenčnega razpona, tako da lahko generiramo signal okoli 20 kHz, ki bo dosegel skoraj optimalen izkoristek moči pri pretvarjanju energije [16].

Optimalno frekvenco bomo določili empirično z različnimi frekvencami sinusnega signala okrog 20 kHz. Poizkusili bomo tudi z ne čistim sinusnim signalom, ampak z nekoliko popačenim, delno stopničastim in popolnoma stopničastim. Zagotoviti je potrebno tudi čim bolj konstanten signal na izhodu, da na svetlomeru ne bo prihajalo do prekinitev.

Za generiranje zvočnega signala se je potrebno odločiti med Cocoa Touch ogrodjema `MediaPlayer` in `AudioUnit`. Ker `AudioUnit` dostopa do najnižjih programskih nivojev na iOS zvočnem skladu, kot je razvidno na sliki 4.1, je za uporabo potrebno večje predznanje kot za na primer `MediaPlayer`, pri katerem z vsem rokujemo na višjem nivoju. Ima pa `AudioUnit` določene



Slika 4.1: iOS programski audio sklad.

prednosti, ki so v našem primeru zelo pomembne: predvajanje v realnem

času, vhod in izhod z nizko latenco [21]. Zaradi teh razlogov smo se odločili uporabiti `AudioUnit`.

4.2 Sprejemanje bitov

Poleg napajanja svetlomera je seveda pomembna tudi komunikacija v smeri proti napravi. Potrebno je narediti mehanizem, ki bo ves čas bral podatke iz mikrofona, te podatke shranil v začasen seznam in jih ustrezno obdelal. Cilj je iz sprejetih surovih vrednosti mikrofona razbrati bite, ki so bili poslani.

Algoritem za pridobivanje podatkov iz mikrofona je v grobem sledeč:

1. Preberi vrednosti iz mikrofona v seznam.
2. Na teh vrednostih opravi FSK demodulacijo, da dobimo bite.
3. Bite razdeli na skupine po 8 bitov oziroma bajte.
4. Bajte pošlji naprej v razred, ki bo odgovoren za interpretacijo teh bajtov.
5. Neuporabljeni biti bodo pripeti pri naslednjem branju iz mikrofona.

4.2.1 Branje

Za zajem vrednosti mikrofona bomo uporabili storitve `AudioQueue` ogrodja, ki podobno kot `AudioUnit` deluje na nekoliko nižjem programskem nivoju, saj je več uporabe klasičnega jezika C. Vendar še vedno ni potrebno predznanje delovanja strojne opreme za zajem vrednosti mikrofona. `AudioQueue` omogoča samo zajem surovih vrednosti mikrofona, brez dodatnih in za nas nepotrebnih dodatkov, zaradi česar bo tudi izvajanje najhitrejše.

Potrebno bo definirati lasten razred, ki bo upravljal z informacijami o stanju, formatom in potjo o zajetih podatkih. Napisati bo treba funkcijo, ki se bo poklicala ob dejanskem zajemu podatkov iz mikrofona. Pred vsakim začetkom zajema je potrebno poskrbeti, da ima `AudioQueue` na voljo pomnilnik za zajem vrednosti [22].

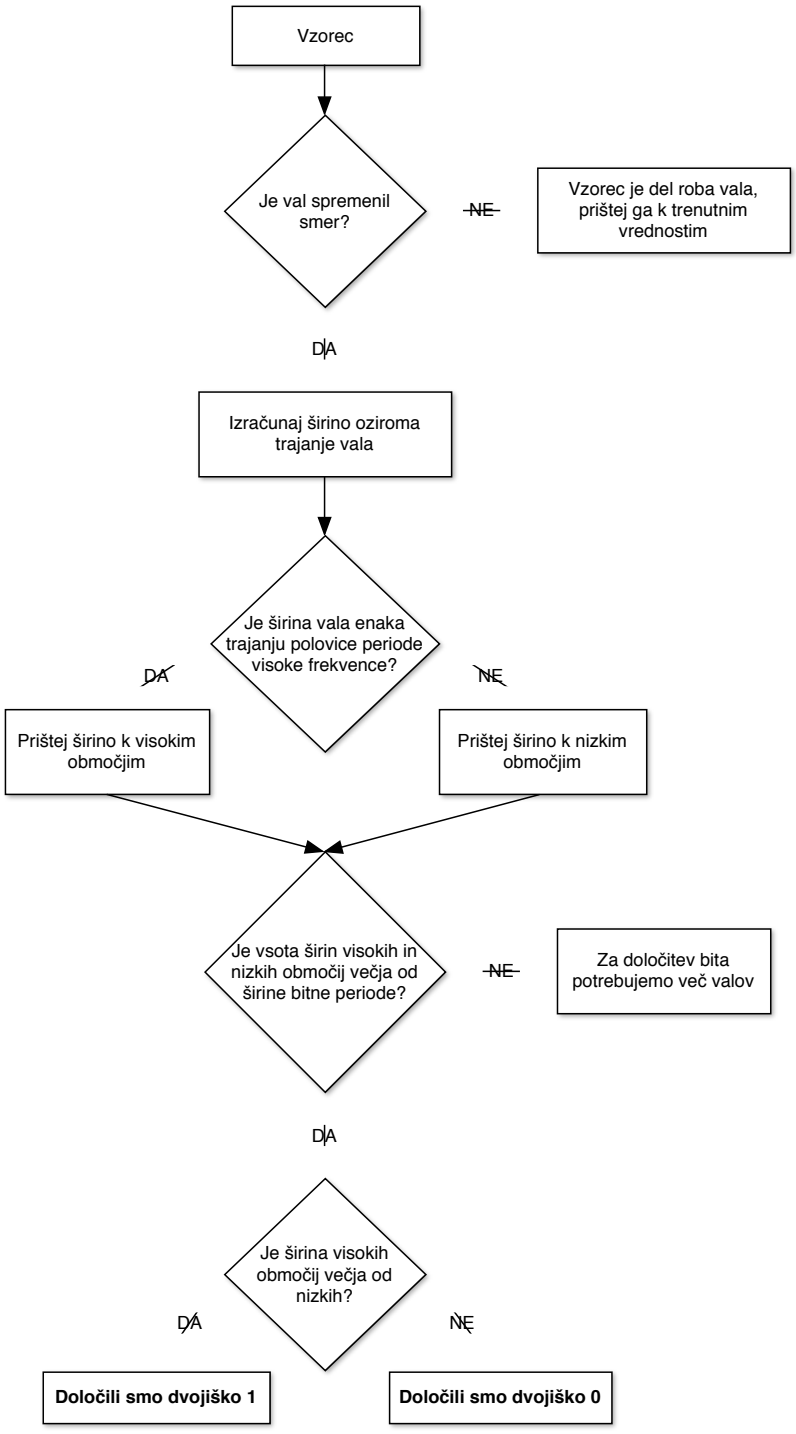
4.2.2 FSK demodulacija

FSK demodulacija je obraten proces FSK modulacije, pri katerem se iz zakodiranega signala razberejo poslani podatki. V našem primeru je potrebno iz surovih vrednosti mikrofona ugotoviti ali je bil v danem časovnem oknu bitne periode poslan podatek za dvojiško nič, ena ali pa nič od tega.

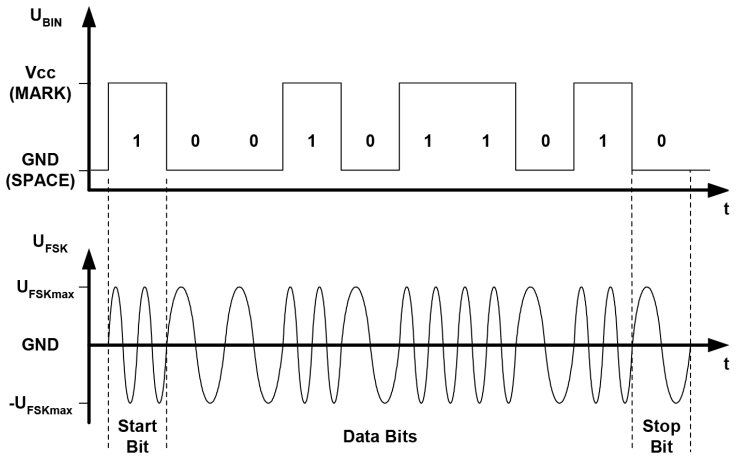
Kombinacija našega svetlomera in iOS naprave deluje kot modem – modulator in demodulator. Bolj natančno bi mu lahko rekli "softmodem" (angl. software modem) [23].

Mikrofon vzorči signal s frekvenco 44,1 kHz, kar pomeni, da mora naš algoritem za FSK demodulacijo obdelati 44100 vzorcev na sekundo. Vzorce iz `AudioQueue` strukture bo pridobil v več kosih, večkrat na sekundo in bo tako sproti ugotavljal trenutne bite. Osnovna struktura algoritma je vidna na sliki 4.2.

Algoritem bo vsak nov vzorec najprej primerjal s prejšnjim. Če pri tem vzorcu signal ne spremeni smeri, še naprej pada oziroma raste, potem je ta vzorec del enega roba signala, ki ga bomo upoštevali kasneje, tako da si njegovo vrednost samo zabeležimo. Če pa pri tem vzorcu signal spremeni smer, potem izračunamo širino oziroma trajanje od prejšnje sprememba smeri do tega vzorca. Tako dobimo širino pol-vala. Če je ta širina manjša oziroma enaka trajanju polovice periode visoke frekvence, to širino prištejemo k visokim območjem, sicer pa k nizkim. Zatem preverimo, če je vsota visokih in nizkih območjih že večja od širine ene bitne periode. Če še ni, to pomeni, da za določitev bita potrebujemo več pol-valov. Če pa je že, potem smo dobili dvojiško ena, ko je širina visokih območij večja od širine nizkih, sicer pa dvojiško nič [24, 18].



Slika 4.2: Struktura algoritma za FSK demodulacijo.



Slika 4.3: Poenostavljen prikaz FSK demodulacije signala [24].

Cilj tega algoritma je lepo razviden na sliki 4.3 (ki ni točen prikaz naše komunikacije, ampak prikazuje veliko nižje frekvence), kjer je na spodnjem delu prikazan sprejet signal, na zgornjem pa demodulirani podatki. Vidimo, da iz signala z nizko frekvenco demoduliramo dvojiško nič in iz signala z visoko frekvenco dvojiško ena.

4.2.3 Uporaba bitov

Iz algoritma za FSK demodulacijo bomo pridobili bite. Te bite bo potrebno začasno shranjevati, dokler ne bomo dobili zaustavitvenega bita. Takrat bomo imeli shranjen en bajt. Ti bajti bodo shranjeni v vrsti, tako da bodo na voljo za branje iz drugih delov aplikacije.

Za dobro delovanje aplikacije je potrebno več-nitno izvajanje. Če bi vso zaznavanje, računanje in shranjevanje izvajali na niti, ki skrbi za grafično izrisovanje, potem bi naša aplikacija delovala počasno in neodzivno. Tu nam pomaga že iOS, ki deloma že sam poskrbi za to. Branje vrednosti mikrofona poteka na niti v ozadju, tako da bo naša funkcija za branje in FSK demodulacijo tudi tekla v ozadju, saj ji ne bomo ročno spreminjali niti za izvajanje. Iz več bajtov bomo sestavili en podatek. Za izmenjavo podatkov

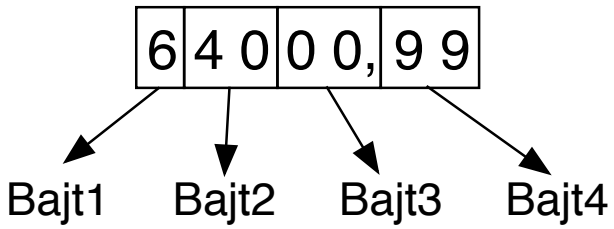
med različnimi nitmi bo potrebno te podatke shranjevati v vrsti, v katero bo funkcija na eni niti dodajala podatke, funkcija iz druge niti pa brala oziroma odstranjevala podatke iz te vrste.

4.3 Dešifriranje bajtov v podatke

Vsak pridobljen bajt bo potrebno najprej preveriti, če se je pravilno prenesel. Uporabljali bomo liho pariteto, tako da mora biti zadnji paritetni bit ena, če je v bajtu sodo število enic, oziroma nič, če je v bajtu liho število enic.

Svetlomer bo pošiljal bite združene v bajte, en podatek pa bo možno sestaviti iz štirih zaporednih bajtov. Prvi bajt paketa bo predstavljal glavo, ki bo povedala za kakšen tip podatka gre - ali je to meritev iz senzorja ali pa gre za druge metapodatke. Iz bitov se bo po odstranitvi paritetnega bita sestavil poslan podatek.

Štirje bajti so potrebni, da lahko predstavimo vrednosti senzorja od 0 do 64000,99.



Slika 4.4: Prikaz delitve podatka po bajtih.

Na sliki 4.4 vidimo, kako se bo maksimalna vrednost razdelila na štiri bajte, tako da so deset-tisočice, tisočice in stotice, desetice in enice in decimalke predstavljene vsaka v svojem bajtu.

4.4 Interpretacija podatkov

Ko bomo iz bajtov sestavili dejanski podatek, nam sam po sebi ne bo veliko pomenil, saj predstavlja samo vrednost iz senzorja. Da bomo dobili pravo uporabno vrednost, lukse v prostoru, bo potrebno dobljeno vrednost pomnožiti s konstanto, ki bo predstavljala prepustnost zbiralne sfere. Ta podatek bo že uporaben in bo tudi predstavljen v aplikaciji. Luksi so standardna mera, ki predstavlja količino lumnov na kvadratni meter. Povedo nam, kakšna je osvetljenost prostora. S tem podatkom lahko na primer ugotovimo, če je pisarna dovolj osvetljena.

Iz luksov lahko z uporabo foto enačbe 4.1 dobimo EV (angl. Exposure Value) - stopnjo osvetljenosti, ki jo poznajo fotografi. Spremenljivka E_v predstavlja znano vrednost luksov v prostoru, C pa je kalibracijska konstanta za osvetljenost, ki mora biti prilagojena za naš svetlomer. ISO, ki predstavlja občutljivost kamere na svetlobo, je pravtako znana vrednost, ki jo določi uporabnik. Z znanim EV pa bomo lahko z uporabo enačbe 4.2 končno izračunali tudi želen foto parameter - čas t ali zaslonko f [5].

$$EV = log_2 \frac{ISO \cdot E_v}{C} \tag{4.1}$$

$$EV = log_2 \frac{f^2}{t} \tag{4.2}$$

Izračunan foto parameter je za fotografa cilj merjenja svetlobe. Sedaj moramo določiti le še ustrezen uporabniški vmesnik, ki bo uporabniku smiselno omogočal izvedbo meritve in prikaz izračunanega parametra.

4.5 Uporabniški vmesnik

Načrtovanje uporabniškega vmesnika je zelo pomemben del pri izdelavi mobilne aplikacije. Zaradi manjšega zaslona oziroma delovne površine moramo poskrbeti, da optimalno uporabimo celoten zaslon, da so vsi interaktivni elementi dostopni brez težav, da so vsi napisi berljivi in tudi, da je uporabniški

vmesnik ustrezen za prikaz želenih informacij.

S prihodom iOS7 so se smernice za obliko uporabniškega vmesnika bistveno spremenile. Večji poudarek je na vsebini, ki jo skušamo sporočati, dodatni nepotrebni lepotni elementi niso več zaželeni. Izkoristimo lahko visoko resolucijo zaslona za prikaz različnih vrst in velikosti pisav, tudi najtanjših. Aplikacija naj bi bila zgrajena iz virtualnih nivojev, ki uporabniku pomagajo pri uporabi in razumevanju aplikacije [25].

4.5.1 Načrtovanje

V našem primeru je načrtovanje uporabnega uporabniškega vmesnika ključnega pomena, saj moramo uporabniku ponuditi veliko izbir in prikazati povratno informacijo svetlomera, za katero ni točno določene oblike. Glavna funkcionalnost aplikacije je prikaz izračunanega foto parametra glede na izvedeno meritev svetlobe. Uporabniku morajo biti vedno vidni vsi trije foto parametri (ISO, čas, zaslonka). Dva od teh treh parametrov sta zaklenjena, tretji pa je odklenjen, kar pomeni, da se pri izvedeni meritvi izračuna njegova pravilna vrednost. Uporabnik na kameri nastavi ISO (občutljivost filma oziroma tipala) in še en parameter (čas ali zaslonko), nato pomeri svetlobo na želenem mestu, da dobi tretji parameter, ki ga lahko nato nastavi na kameri. Aplikacija mora tako uporabniku nuditi možnost odklepa želenega parametra, izbiro zaklenjenih parametrov, izvedbo meritve ter prikaz izračunanega parametra.

Poleg glavne fotografske funkcionalnosti mora aplikacija omogočati tudi prikaz osnovne enote osvetljenosti prostora (luksov), shranjevanje beležk, pregled beležk in nastavitve. Zaradi velikega števila teh funkcionalnosti bo aplikacija razdeljena na več pogledov: foto, osvetljenost, beležke, nastavitve.

4.5.2 Navigacija

V primeru več pogledov se moramo odločiti tudi za tip navigacije med temi pogledi. Tipi navigacije se v mobilnih aplikacijah delijo na tri skupine:

Hierarhična: uporabnik se odloča ob vsakem pogledu, dokler ne pride do želene vsebine. Za novo izbiro mora najprej obratno ponoviti korake.

Eno-nivojska: uporabnik lahko v vsakem trenutku direktno zamenja primarno kategorijo aplikacije, ker so vse na voljo iz glavnega pogleda.

Vsebinska: navigacija je lahko določena z vsebino aplikacije, na primer premikanje strani v knjigi [25].

Odločili smo se za eno-nivojsko aplikacijo v obliki skritega stranskega menija, ker menimo, da je najbolj primeren. Je enostaven za uporabo, že kar popularen med uporabniki, intuitiven, prihrani veliko dragocenega prostora in zelo prilagodljiv.

Do stranskega menija bo mogoče dostopati s pritiskom na namenski gumb ali pa z gesto, tako da v desno podrsamo po zaslonu. Na stranskem meniju bodo za izbiro na voljo vse glavne kategorije oziroma pogledi aplikacije.

4.5.3 Opis glavnih pogledov

Kot smo že omenili, bo glavni del aplikacije foto pogled. Foto parametri bodo predstavljeni s tremi navideznimi drsniki. Uporabnik bo lahko podrsal levo ali desno po zaslonu za izbiro manjšega ali večjega parametra. Na dnu bo gumb oziroma območje, ki bo ob dotiku sprožil meritev. Po izvedbi meritve, se bo na ustrezno vrednost nastavil drsnik odklenjenega parametra. Na vrhu se bo poleg gumba za meni nahajal tudi gumb za shranitev trenutnih parametrov v beležko.

Drugi pogled bo prikazoval trenutno osvetljenost v luksih ter EV vrednost. Navidezni drsnik bo omogočal preklapljanje med dvema enotama: luksi in fc (angl. foot-candle), kar je ameriška enota za osvetljenost (količina lumnov na kvadratni čevelj) in je samo za konstanto različna od luksov.

Pogled za pregled beležk bo v obliki tabele, ki je osnovni iOS gradnik, zato ima dobro podporo in je enostaven za uporabo. Na vrhu se bodo nahajali gumbi za dodajanje skupine in urejanje. S pritiskom na beležko se bo prikazal bolj podroben pogled beležke.

Pogled za nastavitve bo, podobno kot za beležke, v obliki tabele, kjer bodo naštete vse možne nastavitve.

4.5.4 Shranjevanje beležk v oblaku

Povezljivost aplikacije z internetom in shranjevanje beležk v oblaku je tudi pomembna funkcionalnost, ki zahteva dobro načrtovanje. Shranjevanju v oblaku je dandanes že močno razširjeno, tako da je na voljo že več rešitev. Zato smo se odločili, da ne bomo razvijali svoje, ampak bomo uporabili že obstoječo storitev. Uporabili bomo Dropbox, ker je najbolj razširjen, ima najboljšo podporo za razvijalce, dober SDK z obširno dokumentacijo.

Shranjevanje v oblak mora delovati transparentno. Uporabnik mora samo vključiti to funkcijo, vse ostalo se mora dogajati skrito. Dropbox to omogoča, saj vedno v ozadju sinhronizira datoteke, ki se nahajajo v Dropbox mapi. Zaradi tega je tudi uporaben za sinhronizacijo med več napravami, saj so datoteke vedno sinhronizirane med vsemi napravami, ki so povezane z istim Dropbox računom.

Potrebno je tudi določiti, v kakšni obliki bodo beležke shranjene na napravi. Če uporabnik ne vključi Dropbox možnosti, morajo biti lokalno shranjene na napravi, ko pa ima vključeno to možnost, morajo biti na voljo v Dropbox mapi. Odločili smo se, da bodo beležke shranjene v JSON formatu. JSON predstavlja kompaktnejšo alternativo XML formatu in je format datoteke za izmenjavo podatkov. Vanj lahko shranimo slovarje, sezname, tekst in vrednosti [26]. Če bo z beležko shranjena tudi slika, bo ta slika v JPG formatu shranjena v isti mapi kot JSON datoteka.

Vse te datoteke bodo shranjene v lokalni mapi, ki nam je na voljo za shranjevanje na iOS napravi. Če bo uporabnik vklopil Dropbox sinhronizacijo pa bo potrebno vse te datoteke prestaviti v Dropbox datotečni prostor, da ne bomo hranili dvojnih kopij datotek. Podobno, če uporabnik izklopi sinhronizacijo, je potrebno vse iz Dropbox datotečnega prostora skopirati v lokalni prostor, da v nobenem primeru ne pride do izgube shranjenih beležk.

Poglavje 5

Razvoj programske opreme

Po postavljenem načrtu je sledila implementacija aplikacije. V tem poglavju so opisani ključni deli aplikacije ter težave, ki so pojavile pri implementaciji.

5.1 Projekt v Xcode

Za razvoj aplikacije za iOS so potrebni:

- Mac računalnik
- Xcode
- iOS SDK

Xcode je Apple-ovo razvijalsko okolje, ki vključuje urejevalnik izvorne kode, urejevalnik grafičnega vmesnika in še veliko drugih funkcionalnosti.

iOS SDK razširja Xcode z orodji, prevajalniki in ogrodji, ki so potrebni za razvijanje za iOS. SDK vključuje tudi simulator iOS naprav, ki je uporaben za testiranje grafičnega vmesnika, za testiranje komunikacije pa smo uporabljali iPhone.

Xcode že vključuje predloge za najbolj standardne aplikacije. Za naš primer je bil izbran predlog "prazne aplikacije" (angl. empty application), ki nam predpripravi vso začetno strukturo elementov in datotek, brez dodanega standardnega grafičnega vmesnika [27].

Najprej smo razvili komunikacijski del, nato grafični vmesnik ter na koncu še shranjevanje v oblak. Sočasno je potekal tudi razvoj ogrodja za komunikacijo (angl. framework), ki ga opisuje zadnji del tega poglavja.

5.2 Razvoj temeljev komunikacije

Razvoj komunikacije lahko razdelimo na dva dela glede na smer komunikacije: iz iOS naprave proti svetlomeru in obratno. V smeri iz iOS naprave proti svetlomeru govorimo o napajanju svetlomera, obratno pa o sprejemu podatkov. Sprejem podatkov lahko razdelimo na tri glavne dele: sprejem vrednosti iz mikrofona, obdelavo teh vrednosti ter pridobivanje informacije iz sprejetih podatkov.

5.2.1 Napajanje svetlomera

Svetlomer z iOS naprave napajamo tako, da po enem od stereo kanalov predvajamo ustrezen signal, kar svetlomeru omogoči dovolj energije za delovanje.

Za predvajanje tona smo uporabili funkcije `AudioUnit` razreda, ki predstavlja najbolj osnoven razred za dostop do audio funkcionalnosti, kar nam omogoča hitro odzivnost in predvajanje poljubnega tona oziroma signala.

Proces predvajanja je sledeč:

1. Med izvajanjem pridobimo referenco knjižnice, ki definira audio enoto, ki jo bomo uporabili za predvajanje.

2. Instanciramo audio enoto.

3. Konfiguriramo audio enoto s potrebnimi nastavitvami in pripravimo signal za predvajanje.

4. Inicializiramo audio enoto, da se pripravi na upravljanje z zvokom.

5. Pričnemo s predvajanjem [21].

Del 3. točke in sicer del funkcije za pripravo signala za predvajanje je viden v izvorni kodi 5.1. Koda je del funkcije `RenderTone`, ki skrbi za predvajanje tona. Funkcija najprej določi amplitudo in vzorčni korak za frekvenco 20050 Hz ter pridobi referenco do audio predpomnilnika, nato pa v zanki računa vrednosti sinusnega signala ter vrednosti zapiše v audio predpomnilnik, katerega `AudioUnit` razred uporabi za predvajanje [28].

```
1  double amplitude = generator->amplitude;
2  double theta = generator->theta;
3  double theta_increment = 2.0 * M_PI * generator->frequency / generator->
       sampleRate;
4
5  // This is a mono tone generator so we only need the first buffer
6  int channel = 0;
7
8  Float32 *buffer = (Float32 *)ioData->mBuffers[channel].mData;
9
10 // Generate the samples
11 for (UInt32 frame = 0; frame < inNumberFrames; frame++)
12 {
13     buffer[frame] = sin(theta) * amplitude;
14     theta += theta_increment;
15     if (theta > 2.0 * M_PI)
16         theta -= 2.0 * M_PI;
17 }
```

Izvorna koda 5.1: Izračun vrednosti signala za predvajanje.

Do najbolj optimalnega signala za napajanje svetlomera smo prišli empirično. Poizkušali smo z različnimi frekvencami sinusnega signala, različnimi oblikami (čisti sinusni, delno stopničasti, popolnoma stopničasti) in z različnimi amplitudami. Za najboljšega se je izkazal čisti sinusni signal s frekvenco 20050 Hz, ki je ravno na meji zmogljivosti iOS naprav.

5.2.2 Sprejem vrednosti iz mikrofona

Za zajem vrednosti iz mikrofona smo uporabili funkcije storitve `AudioQueue`. Izdelati je bilo potrebno naslednje stvari:

1. Definirati posebno strukturo, ki ima vpogled v stanje in format sne-

manja.

2. Implementirati povratno funkcijo, ki se kliče ob snemanju in izvaja dejansko zajemanje vrednosti.

3. Nastaviti ustrezne vrednosti strukture iz 1. točke, torej definirati frekvenco vzorčenja in način shranjevanja vrednosti.

4. Ustvariti nov vnos v `AudioQueue` z ustreznimi nastavitvami, povratno funkcijo ter nastaviti način za neprekinjeno snemanje.

5. Zagnati `AudioQueue` snemanje.

```
1 // If there is audio data, analyze it
2 if (numberOfPackets > 0)
3     analyzeSignal((SAMPLE*)audioQueueBuffer->mAudioData, audioQueueBuffer->
          mAudioDataByteSize / BYTES_PER_FRAME, signalAnalyzer);
4
5 // If not stopping, re-enqueue the buffer so that it can be filled again
6 if ([signalAnalyzer isRunning])
7     AudioQueueEnqueueBuffer (audioQueue, audioQueueBuffer, 0, NULL);
```

Izvorna koda 5.2: Sprejem vrednosti iz mikrofona.

V izvorni kodi 5.2 vidimo del povratne funkcije, ki se pokliče ob vsaki napolnitvi audio predpomnilnika. Funkcija najprej preveri, če so bile zajete kakšne vrednosti, nato pa te vrednosti ne shranjuje v datoteko, ampak referenco na vrednosti ter število teh vrednosti poda naprej v funkcijo, ki izvaja prepoznavanje bitov. Po vsakem zajemu, če se snemanje ne ustavi, je potrebno sprostiti audio predpomnilnik oziroma ga ponovno nastaviti [29].

5.2.3 Obdelava vrednosti

Po zajemu vrednosti iz mikrofona je potrebno dobljen signal analizirati. S FSK demodulacijo iz signala pridobimo bite, te bite sestavimo v bajte, bajte pa zapisujemo v vrsto.

Funkcija za analizo signala se premika vzorec po vzorec po zajetih vrednostih in zaznava, kdaj signal obrne smer. Ko je razlika med dvema zaporednima vzorcema dovolj velika, smo zaznali polovico periode sinusnega signala (pol-val), ki se nahaja med tem vzorcem in prejšnjim vzorcem, pri katerem je signal spremenil smer. Širino oziroma trajanje tega pol-vala sporoči funkciji za FSK demodulacijo.

Funkcija za demodulacijo iz pol-vala ugotovi ali je signal visoke frekvence ali nizke frekvence. Trajanja nizkih in visokih pol-valov hrani v ločenih spremenljivkah in ko je vsota teh dveh večja od trajanja bitne periode, vemo, da smo zaznali bit. Bit je dvojiška ena, če je vsota visokih frekvenc večja od nizkih in obratno za dvojiško nič. Dobljeni bit posredujemo naprej v funkcijo, ki bite sestavlja v bajte. Če se nahajamo v stanju, ko je bajt prazen, potem funkcija za demodulacijo čaka na nizek začetni bit in ignorira vse visoke bite.

Funkcija za sestavljanje bitov v bajte dobi kot vhodni parameter podatek ali gre za visok ali nizek bit. Del funkcije, ki se izvaja v primeru, da shranjujemo bite, ki so del bajta (ni začetni bit), lahko vidimo v izvorni kodi 5.3.

```
1  case FSKBits:
2      if(bitPosition <= 7){   // Data Bits
3          newState = FSKBits;
4          [self saveBit:isHighBit]; // Save the current bit
5      } else if(bitPosition == 8){    // Stop Bit
6          newState = FSKStart;
7          ReceivedData *receivedData = [receivedBytesManager receivedByte:bits
               ];
8
9          if (receivedData) // We have a new data package
10             [self newReceivedData:receivedData];
11
12         bits = 0;
13         bitPosition = 0;
14     }
15     break;
```

Izvorna koda 5.3: Shranjevanje bitov.

Trenutne bite hrani v 8-bitni znakovni spremenljivki, v katero z logičnimi operacijami (premik v levo oziroma množenje z 2) zapisujemo nove bite. Ko

zapišemo 8 bitov, to spremenljivko kot bajt predamo funkciji za sestavljanje podatkov [18].

5.2.4 Razumevanje meritve iz bajtov

Za pretvorbo bajtov v podatek o meritvi svetlomera skrbi ločen razred, ki vsebuje metode za preverjanje paritete bajta, tipa bajta in za sestavljanje več bajtov v eno meritev. Objekt tega razreda hrani zadnje štiri sprejete bajte. Če ima na voljo štiri še neizkoriščene bajte, potem poskusi sestaviti meritev.

```
1  case DATA:
2      // Check all the parities
3      if ([self isOddParityForByte:byte0] && [self isOddParityForByte:byte1]
           && [self isOddParityForByte:byte2] && [self isOddParityForByte:byte3
           ]) {
4          // Get rid of the parity bit
5          int param0 = ((byte0 >> 1) & 0x07); // Also, get rid of the header
6          int param1 = (byte1 >> 1);
7          int param2 = (byte2 >> 1);
8          int param3 = (byte3 >> 1);
9
10         // Check if the values are within the limits
11         if ((param0 < 6 && param1 <= 99 && param2 <= 99 && param3 <= 99) ||
               (param0 == 6 && param1 < 40 && param2 <= 99 && param3 <= 99)){
12             double paramDataByte = param0 * 10000 + param1 * 100 + param2 +
                   ((double)(param3)/100);
13             // Everything is ok, create DATA
14             receivedData = [[ReceivedData alloc] initWithType:DATA andData:
                   paramDataByte];
15         } else { // Data is corrupted
16             receivedData = [[ReceivedData alloc] initWithType:WRONG_DATA
                   andData:0];
17         }
18         byteCounter = 0;
19     } else { // Parity was wrong
20         receivedData = [[ReceivedData alloc] initWithType:WRONG_PARITY
               andData:0];
21     }
22     break;
```

Izvorna koda 5.4: Pridobivanje podatka iz bajtov.

Prvi bajt hrani glavo podatka, ki pove ali gre za meritev ali kakšen drug tip podatka (metapodatki o podrobnostih strojne opreme svetlomera). V izvorni kodi 5.4 lahko vidimo, da v primeru podatka o meritvi najprej za vse štiri bajte preverimo, ali je pariteta ustrezna. Nato vse bajte zamaknemo za eno mesto v desno oziroma delimo z 2, da izgubimo paritetni bit, ki ne nosi informacije. Za nove vrednosti preverimo, če so znotraj dovoljenih meja ter jih ustrezno sestavimo v eno vrednost. Dobljena vrednost je meritev, ki je bila poslana iz svetlomera.

Objekt s to vrednostjo se v niti v ozadju zapiše v vrsto. Na glavni niti se objekti berejo iz te vrste in uporabljajo za nadaljnje računanje in prikazovanje uporabniku.

Implementirali smo protokol, ki objektu omogoča, da sprejema podatke iz svetlomera. Ko se objekt prijavi kot delegat protokola in implementira določene funkcije, ki mu poleg drugega omogočajo tudi sprejem podatkov, lahko sprejete meritve smiselno uporabi in prikaže uporabniku. Takrat, ko je delegat prisoten, komunikacija deluje in se sprejemajo novi podatki, sicer pa se napajanje in sprejemanje ustavi, da po nepotrebnem ne porabljamo energije iOS naprave. Poleg sprejema podatkov, protokol delegatu sporoča tudi informacija o začetku in koncu komunikacije (priklop in odklop svetlomera iz audio vhoda iOS naprave).

V povprečju sprejmemo 5 podatkov o meritvi na sekundo. Ti podatki se ne kombinirajo pri prikazu uporabniku, ampak se vedno uporabi le najnovejša. V primeru, da ima uporabnik vključeno neprekinjeno merjenje, potem se mu prikazujejo vse meritve zaporedoma.

5.3 Težave pri komunikaciji

Zaradi več-nitnega branja in pisanja v vrsto, kjer so shranjeni podatki o meritvah, so se pojavljali problemi. Želeli smo implementirati vrsto, ki ne bi uporabljala zaklepanja vrste, da ena nit ne bi rabila čakati na drugo, da konča z izvajanjem. S tem bi pridobili na hitrosti izvajanja. Zaradi

narave komunikacije in velikega hkratnega spreminjana vrste, je prihajalo do ABA problema. Do ABA problema prihaja med sinhronizacijo: pomnilniška lokacija ima med dvema branjema isto vrednost (A), ampak njeno vrednost je vmes spremenila druga nit (iz A v B in nazaj v A) in tako zavedla prvo nit. To privede do nedefinirane vrste [30].

Namesto reševanja ABA problema, smo se odločili za implementacijo vrste z zaklepanjem. Tudi hitrost komunikacije je dosti počasnejša od zmožnosti iOS naprave, tako da zaklepanje vrste ne vpliva bistveno na hitrost izvajanja. Uporabili smo `NSCondition` razred, ki je del Apple Cocoa ogrodja. Ob vsakem branju in pisanju se vrsta zaklene in ostane zaklenjena do konca operacije. Vmes morajo ostale niti, ki hočejo dostopati do vrste, počakati oziroma spijo. Na koncu operacije se vrsta odklene in jo lahko uporabi naslednja nit. Ta rešitev uporablja princip ključavnice, ki je podoben semaforju, le da lahko ključavnico odklene samo tista nit, ki jo je zaklenila, semafor pa lahko odklene poljubna nit.

5.4 Uporaba grafičnih elementov

Pomemben del pri izdelavi aplikacije je tudi uporaba grafičnih elementov. Uporabili oziroma implementirali smo tri glavne elemente: stranski meni, drsnike za izbiro parametrov ter tabele.

5.4.1 Stranski meni

Za implementacijo stranskega menija smo uporabili odprto-kodno knjižnico `ViewDeck` [31], ki je zelo prilagodljiva in enostavna za uporabo. Ob zagonu aplikacije navedemo, da za glavni pogled skrbi objekt tipa `ViewDeck`, nato pa ob inicializaciji tega objekta sporočimo kateri objekt naj uporabi za glavni in levi pogled, kot je vidno v izvorni kodi 5.5. Enako smo nastavili tudi desni stranski meni, ki je v uporabi na glavnem foto pogledu aplikacije.

```
1 LeftMenuViewController *leftMenuVC = [[LeftMenuViewController alloc]
      initWithNibName:@"LeftMenuViewController" bundle:nil];
2 RightMenuViewController *rightMenuVC = [[RightMenuViewController alloc]
      initWithNibName:@"RightMenuViewController" bundle:nil];
3 MainViewController *mainViewController = [[MainViewController alloc]
      initWithNibName:@"MainView" bundle:nil];
4
5 self.viewDeckController = [[IIViewDeckController alloc]
      initWithCenterViewController:mainViewController leftViewController:
      leftMenuVC rightViewController:rightMenuVC topViewController:nil
      bottomViewController:nil];
```

Izvorna koda 5.5: Inicializacija ViewDeck objekta.

5.4.2 Drsniki za izbiro parametrov

Na glavnem foto pogledu se nahajajo trije drsniki, vsak za enega od treh foto parametrov. Z drsanjem v levo in desno lahko spreminjamo vrednost želenega parametra.

Drsnike smo implementirali na podlagi Cocoa Touch razreda `UIScrollView`, ki omogoča uporabo drsnega območja. Prilagodili smo ga tako, da se premika samo horizontalno ter se ustavi vedno na sredini oznake, ki predstavlja vrednost enega parametra, zato da so vsi trije parametri vedno poravnani. Poleg tega se morata vedno premikati po dva parametra skupaj, da se ohranjajo razmerja med foto parametri. Kot delegat objekta tipa `UIScrollView` lahko dobimo podatek o vsakem premiku tega objekta, kar smo izkoristili za hkratno premikanje še drugega drsnika. V izvorni kodi 5.6 lahko vidimo funkcijo `scrollViewDidScroll`, ki prepozna, kateri drsnik premika uporabnik ter zraven premakne še drugi ustrezen drsnik (glede na to, kateri parameter je odklenjen). Parameter lahko odklenemo tako, da pritisnemo na njegov drsnik, s čimer tudi zaklenemo parameter, ki je bil prej odklenjen, saj hkrati ne moreta biti odklenjena dva parametra.

```
1  -(void)scrollViewDidScroll:(UIScrollView *)scrollView
2  {
3      if (scrollView == _isoPicker.valuesScrollView && _isoPicker.
           isMovingFromUser) {
4      // Iso scroller is the initial moving scroller
5          if (photoParams.unlockedParameter == TimeUnlocked) {
6              [self parallelScroll:_isoPicker andSecondPicker:_fPicker
                   inOppositeDirection:NO];
7          } else if (photoParams.unlockedParameter == ApertureUnlocked) {
8              [self parallelScroll:_isoPicker andSecondPicker:_tPicker
                   inOppositeDirection:YES];
9          } else if (photoParams.unlockedParameter == IsoUnlocked) {
10             [self parallelScroll:_isoPicker andSecondPicker:_tPicker
                   inOppositeDirection:YES];
11         }
12     } else if (scrollView == _tPicker.valuesScrollView && _tPicker.
           isMovingFromUser) {
13     // Time scroller is the initial moving scroller
14         [self parallelScroll:_tPicker andSecondPicker:_fPicker
               inOppositeDirection:NO];
15     } else if (scrollView == _fPicker.valuesScrollView && _fPicker.
           isMovingFromUser) {
16     // Aperture scroller is the initial moving scroller
17         [self parallelScroll:_fPicker andSecondPicker:_tPicker
               inOppositeDirection:NO];
18     }
19 }
```

Izvorna koda 5.6: Hkratno premikanje dveh drsnikov.

5.4.3 Tabele

Pomembni grafični elementi, ki so pogosto uporabljeni, so tabele. Te se uporabljajo v nastavitvah, v pregledu beležk ter v podrobnem pogledu beležke.

Za prikaz tabel smo uporabili Cocoa Touch razred `UITableView`, ki omogoča visoko prilagodljivost in že sam poskrbi za upravljanje vsebine, ki je višja od ene višine zaslona. Objektu tipa `UITableView` je potrebno samo sporočiti, koliko vrstic bo v tabeli, kateri objekt bo priskrbel podatke in kateri objekt bo delegat za določene funkcije. Nato je potrebno implementirati potrebne funkcije, ki priskrbijo podatke. Za izris in ostalo poskrbi razred `UITableView`.

5.5 Uporaba GPS lokacije

Pri shranjevanju beležk imamo možnost uporabe GPS lokacije. Kasneje, na pregledu beležke, se nam lokacija in naslov prikažeta na mapi.

Za pridobivanje lokacije na iOS napravi smo uporabili razred `CLLocation-Manager`. Objektu tega razreda nastavimo želeno natančnost, nato pa izvedemo funkcijo za pridobivanje lokacije. Ko pridobi podatek o lokaciji z dovolj veliko natančnostjo, nam to sporoči v metodi delegata, ki jo lahko vidimo v izvorni kodi 5.7. Ko sprejmemo ta podatek, lahko ustavimo iskanje lokacije, ali pa počakamo na še bolj natančen podatek. Podatek vsebuje geografsko širino in dolžino ter nadmorsko višino.

```
1  - (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(
       NSArray *)locations {
2      CLLocation *newLocation = [locations lastObject];
3      double timeInterval = [newLocation.timestamp timeIntervalSinceNow]*-1;
4
5      if (timeInterval > 60) // Location is too old
6          return;
7
8      location = newLocation;
9      [self updateGeocoder]; // Tell geocoder we have a new location
10     [self.gpsButton setBackgroundColor:[UIColor orangeColor]];
11
12     if (newLocation.verticalAccuracy < 0 || newLocation.horizontalAccuracy <
           0) // invalid accuracy
13         return;
14
15     if (newLocation.horizontalAccuracy > 100 || newLocation.verticalAccuracy
           > 50) { // Accuracy radius is too large
16     return;
17
18     // Location is good, stop updating it
19     [locationManager stopUpdatingLocation];
20     [self.gpsButton setBackgroundColor:[UIColor emerlandColor]];
21 }
```

Izvorna koda 5.7: Pridobivanje GPS lokacije.

Z uporabo razreda `CLGeocoder` lahko na podlagi geografske širine in dolžine pridobimo naslov te lokacije: ulico, mesto, pošto, državo. Ta naslov

se prikaže skupaj s točko na zemljevidu. Za prikaz zemljevida smo uporabili Cocoa Touch ogrodje `MapKit`, ki nam na enostaven način omogoča prikaz pogleda zemljevida, izbire območja ter prikaz ene točke na zemljevidu [32].

5.6 Uporaba Dropbox API

V aplikaciji je možno vklopiti sinhronizacijo beležk z Dropboxom. Dropbox ima za razvijalce na voljo tri različne vmesnike - API-je (angl. Application Programming Interface): Core, Sync in Datastore. Izbrali smo Sync API, ker je najbolj ustrezal zahtevam aplikacije: omogoča enostavno sinhronizacijo datotek (bolj enostavno kot Core) in dovoljuje shranjevanje poljubnih vrst datotek (Datastore ne omogoča shranjevanja slik).

```
- (BOOL)saveData: (NSData *)data toPath: (NSString *)filePath {
    DBError *error = nil;
    DBPath *path = [[DBPath root] childPath:filePath];
    DBFile *file;
    // Check if path exists
    if (![_filesystem fileInfoForPath:path error:&error]) {
        // Create file if it doesn't exist
        file = [_filesystem createFile:path error:&error];
    } else {
        // Open file if it exists
        file = [_filesystem openFile:path error:&error];
    }
    if (!file) // Error opening file
        return NO;
    if (![file writeData:data error:&error]) // Error writing to file
        return NO;

    // Write was successful, close the file
    [file close];
    return YES;
}
```

Izvorna koda 5.8: Shranjevanje podatkov v Dropbox datotečni prostor.

Sync API uporablja nekaj ključnih razredov za komunikacijo z Dropboxom. Postopek vzpostavitve sinhronizacije začnemo z razredom `DBAccount-`

`Manager`, ki nam omogoča povezavo uporabniškega računa z aplikacijo. Brez tega uporaba Dropbox-a seveda ni mogoča. Ko imamo povezan uporabniški račun, lahko z razredom `DBFileSystem` dostopamo do Dropbox lokalnega datotečnega sistema, ki nam omogoča standardne funkcije datotečnega sistema: izpis seznama datotek in map ter ustvarjanje, urejanje, premikanje in izbris le-teh. Z datotekami delamo preko razreda `DBFile`. Ta nam omogoča pisanje v in branje iz datoteke. V izvorni kodi 5.8 vidimo, kako zapišemo podatke na določeno pot. Najprej pogledamo, če datoteka že obstaja. Če ne obstaja, naredimo novo, sicer samo odpremo obstoječo in v njo zapišemo podatke. Na koncu datoteko še zapremo. Vsakič, ko datoteko spremenimo, jo Sync API v ozadju avtomatsko sinhronizira z Dropbox strežniki [33].

5.7 Izdelava ogrodja za uporabo komunikacije

Poleg aplikacije smo izdelali tudi ogrodje, ki omogoča komunikacijo s svetlomerom. Xcode ima na voljo samo predlogo za statično knjižnico, ki doda nekaj dela končnemu razvijalcu, saj mora ta poleg knjižnice uvoziti še vse javne zaglavne datoteke (angl. public headers). Zato smo se odločili za izdelavo ogrodja [34], ki vse potrebne datoteke združi v eno in tako zmanjša obseg dela razvijalcu.

Ogrodje je bilo razvito sočasno v ločenem projektu. Razvijanje ogrodja se ne razlikuje bistveno od razvijanja aplikacije. Ogrodje vsebuje vse razrede potrebne za komunikacijo ter še en dodaten razred, v katerem je definiran protokol. Ta razred je edina javna zaglavna datoteka, ki jo lahko vidi končni razvijalec. Iz nje lahko razbere protokol, katere metode lahko implementira in kako bo uporabljal komunikacijo za pridobivanje podatkov iz svetlomera.

V izvorni kodi 5.9 lahko vidimo deklaracijo protokola. Delegat tega protokola ima za implementacijo na voljo sedem neobveznih metod, ki mu med drugim sporočajo začetek in konec komunikacije, ali je bil svetlomer prepoznan in sprejem podatkov.

```
1  @protocol CommunicationManagerDelegate <NSObject>
2  @optional
3  -(void)communicationManagerDidNotGetRecordPermission; // iOS7 permission
4  -(void)communicationManagerDidRecognizeLumu;
5  -(void)communicationManagerDidNotRecognizeLumu;
6  -(void)communicationManagerDidReceiveData: (double)value;
7  -(void)communicationManagerDidStartLumu;
8  -(void)communicationManagerDidStopLumu;
9  -(void)communicationManagerDidRecognizeVolumeButtonPressed;
10 @end
```

Izvorna koda 5.9: Protokol ogrodja za komunikacijo.

Poglavje 6

Uporaba

6.1 Končni izdelek

Končni izdelek je aplikacija za iOS naprave, prilagojena za iPhone, ki uporabniku omogoča komunikacijo z zunanjim svetlomerom, izbor želenih foto parametrov, merjenje svetlobe in izračun ustreznega foto parametra. Aplikacija je objavljena v Apple-ovi trgovini z aplikacijami App Store in je brezplačna [35]. Svetlomer je mogoče kupiti na spletni strani projekta [36]. Na sliki 6.1 lahko vidimo naš svetlomer in aplikacijo v uporabi na iPhone-u.

Aplikacija je razdeljena na štiri glavne poglede: fotografski, osnovno merjenje svetlobe, pregled beležk in nastavitve. Med različnimi pogledi lahko izbiramo na levem stranskem meniju, do katerega dostopamo s pritiskom na gumb zgoraj levo, ali pa tako, da po zaslonu podrsamo v desno.

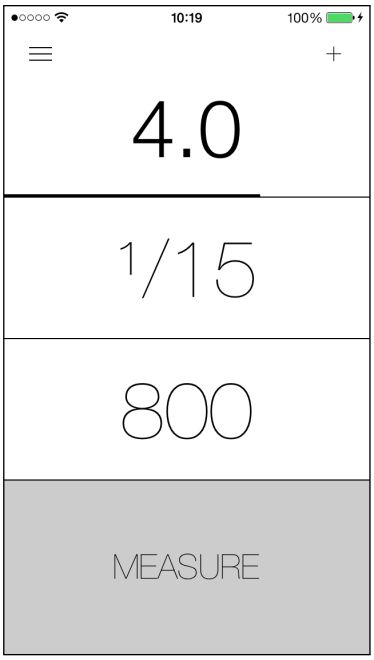
Fotografski pogled in pogled za osnovno merjenje svetlobe uporabljata komunikacijo svetlomerom, ko pa se nahajamo na ostalih dveh pogledih, se komunikacija avtomatsko izklopi, da po nepotrebnem ne porabljamo energije. Komunikacija se tudi avtomatsko vklopi in izklopi ob priklopu oziroma ob odklopu svetlomera. Prav tako se samodejno ob začetku komunikacije glasnost predvajanja na iOS napravi nastavi na najvišjo vrednost, da lahko pridobimo kar največ energije. Ob koncu komunikacije se glasnost ponastavi na prejšnjo vrednost. Uporaba svetlomera je zanesljiva. Aplikacija točno prepozna kdaj

Slika 6.1: Svetlomer v uporabi na iPhone-u.

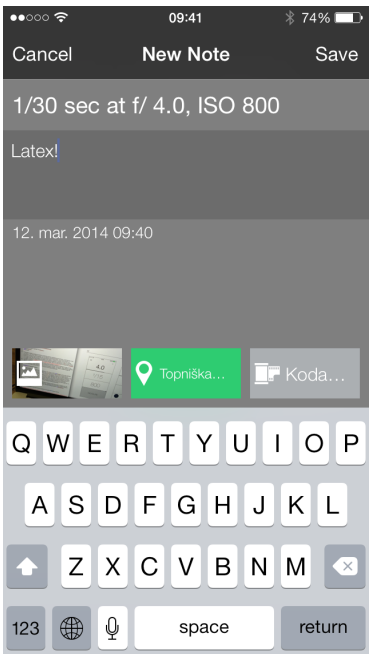
je priključen svetlomer in kdaj ne. Tudi sama komunikacija deluje dobro, v povprečju sprejmemo do 5 meritev na sekundo. Delež napačno sprejetih podatkov je nizek in nima negativnega vpliva na uporabniško izkušnjo. Vsi napačno sprejeti podatki se ustrezno prestrežejo, tako da se nepravilni podatki ne prikažejo uporabniku.

6.1.1 Fotografski pogled

Ta pogled vsebuje glavne funkcionalnosti aplikacije. Kot vidimo na sliki 6.2, se na njem nahajajo tri vrednosti treh foto parametrov. Če po vrednostih podrsamo levo ali desno, nam navidezni drsniki omogočajo izbiranje vrednosti parametrov. Te vrednosti niso poljubne, ampak so že vnaprej določene in jih fotografi poznajo. Parameter za meritev izberemo oziroma ga odklenemo



Slika 6.2: Fotografski pogled.



Slika 6.3: Dodajanje beležke.

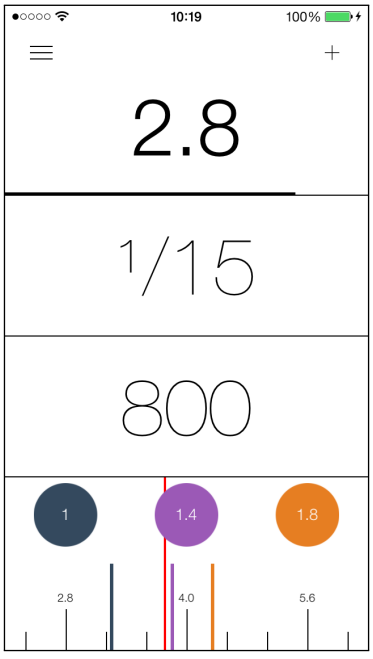
tako, da enostavno pritisnemo nanj. Pod drsniki se nahaja gumb, s katerim izvedemo meritev. Ob meritvi se izbran parameter za meritev prestavi na izračunano vrednost, pod njim pa se prikaže delno polna črta, ki prikazuje

odmik meritve od naslednje vrednosti. Izračunan parameter se vedno nastavi na najvišjo vrednost, ki jo še preseže v seznamu vrednosti, relativna razlika do naslednje vrednosti pa je prikazana na spodnji delno polni črti. Ta črta nam prikazuje ali je meritev bližje prejšnji ali pa naslednji vrednost parametra.

Slika 6.2 prikazuje stanje po izvedbi meritve, ko je bila izračunana vrednost zaslonke 5.15. Drsnik zaslonke se zato nastavi na vrednost 4.0, ker naslednje vrednosti 5.6 ne preseže. Vrednost relativne razlike izračunane vrednosti in naslednje vrednosti prikazuje enačba 6.1. Ta razlika je prikazana z odebeljeno črto pod drsnikom zaslonke, ki zasede 72% širine zaslona.

$$\frac{5.15 - 4.0}{5.6 - 4.0} = 0.72 \tag{6.1}$$

Nad drsniki se nahaja gumb za novo beležko, ki odpre nov pogled, ki ga vidimo na sliki 6.3. Beležki lahko poleg podatkov o foto parametrih dodamo še besedilo, sliko in lokacijo.

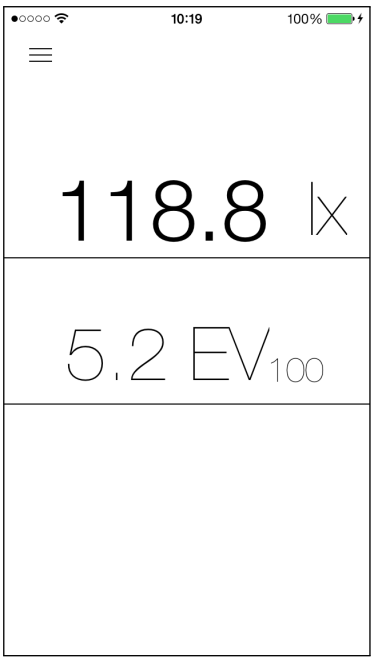


Slika 6.4: Pogled za povprečenje.

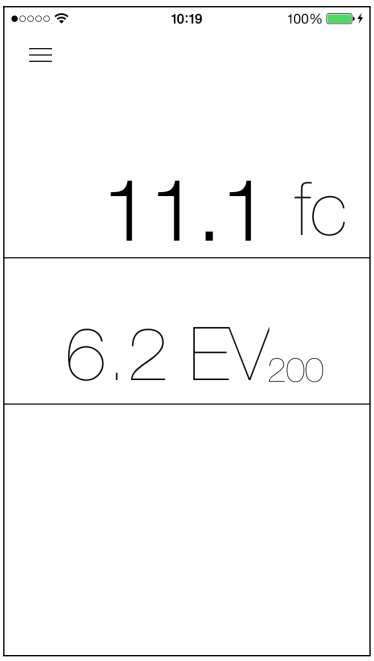
Če po gumbu za meritev podrsamo v levo, se nam odpre še desni stranski

meni, na katerem lahko izbiramo med navadnim načinom in načinom za povprečenje več meritev. Način za povprečenje več meritev nam omogoča izvedbo do 9 meritev, ki so hkrati prikazane na zaslonu v obliki krogov, kot vidimo na sliki 6.4. Na skali je prikazana vrednosti vsake posamezne meritve, rdeča vertikalna črta pa prikazuje povprečje vseh meritev. Zgornji trije parametri so prav tako nastavljeni na povprečno vrednost. S pritiskom na krog, lahko izberemo posamezno meritev, s čimer se tudi zgornji trije parametri nastavijo na vrednost te meritve. V krogih je zapisano razmerje med EV vrednostmi meritev, ki fotografu predstavlja kontrastno razmerje. Z daljšim pritiskom na krog, lahko le-tega odstranimo iz seznama.

6.1.2 Pogled za osnovno merjenje svetlobe



Slika 6.5: Merjenje v luksih.



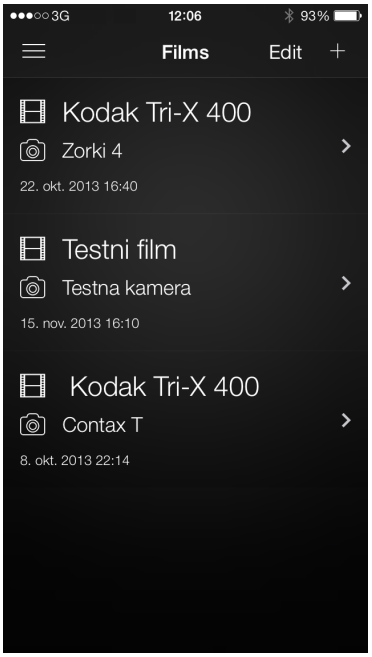
Slika 6.6: Merjenje v enoti fc.

Kot vidimo na sliki 6.5, se na pogledu za osnovno merjenje svetlobe nahaja le podatek o osvetljenosti prostora v luksih ter stopnja osvetljenosti EV (angl. Exposure Value). Če podrsamo po podatku o luksih, lahko zamenjamo enoto

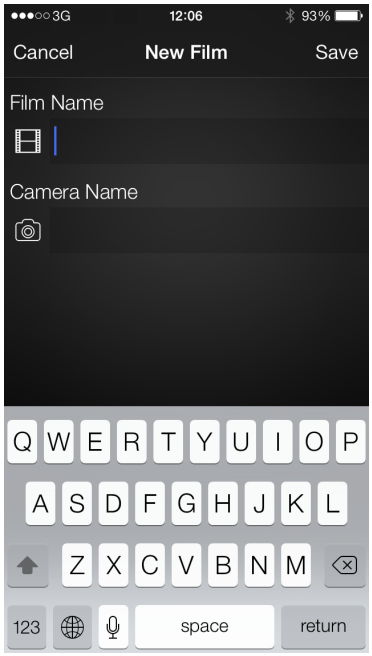
za osvetljenost prostora v fc, ki je vidna na sliki 6.6. Podobno lahko podrsamo po EV podatku, da zamenjamo ISO vrednost, ki vpliva na EV vrednost.

6.1.3 Pregled beležk

Beležke je možno shranjevati v ločene skupine, ki jih naredimo sami. Skupine lahko predstavljajo različne fotoaparate, ki jih uporablja uporabnik, ali pa različne analogne filme, ki jih je uporabnik uporabljal za fotografiranje. Na pregledu beležk so najprej naštete vse skupine, ki jih je uporabnik dodal, kar prikazuje slika 6.7. Novo je mogoče dodati s pritiskom na gumb zgoraj desno. Dodajanje nove skupine oziroma filma vidimo na sliki 6.8.

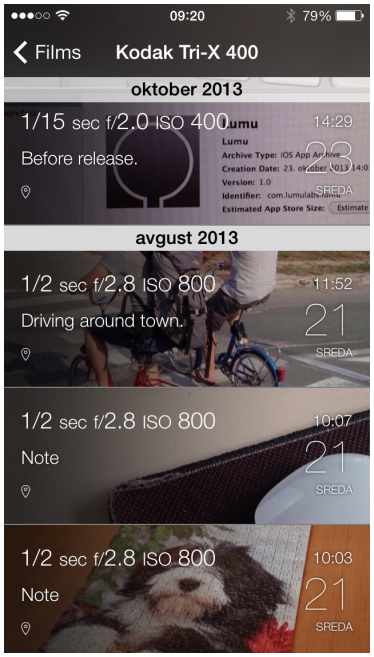


Slika 6.7: Pregled skupin beležk z uporabo tabele.

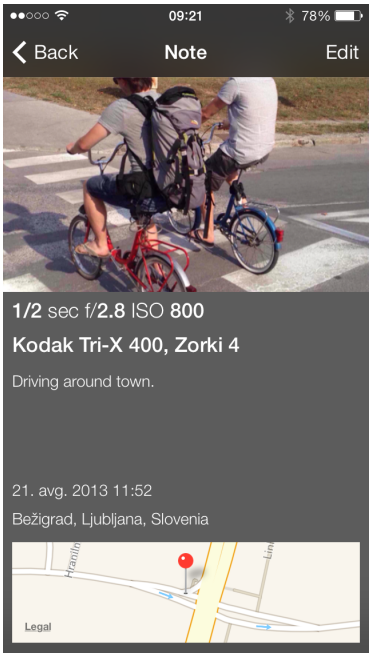


Slika 6.8: Dodajanje skupine beležk.

Po izbiri skupine se nam prikažejo vse beležke, ki so shranjene v tej skupini, kar prikazuje slika 6.9. Po izbiri beležke se prikažejo vse podrobnosti te beležke, vključno s sliko, lokacijo in datumom, kot je vidno na sliki 6.10. S pritiskom na sliko ali lokacijo se odprejo ločeni pogledi s podrobnostmi.



Slika 6.9: Pregled beležk z uporabo tabele.



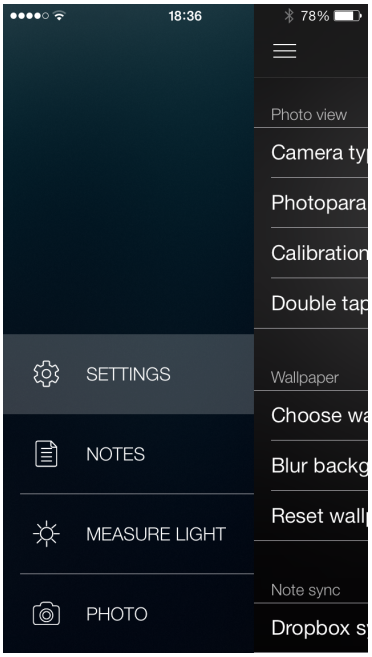
Slika 6.10: Podrobnosti beležke.

6.1.4 Nastavitve

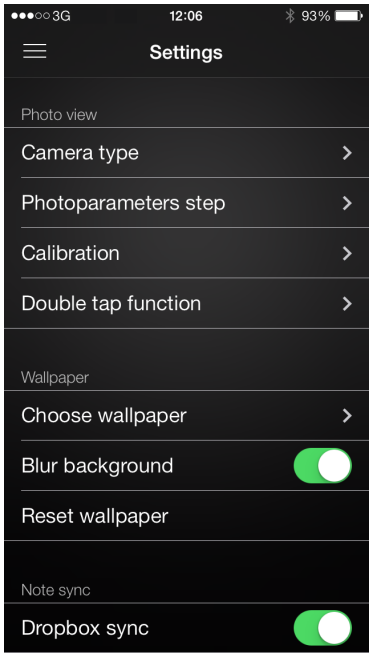
Z uporabo stranskega menija, ki ga prikazuje slika 6.11, lahko pridemo do nastavitev aplikacije. V nastavitvah lahko spremenimo izgled in delovanje aplikacije. Kot lahko vidimo na sliki 6.12 lahko nastavimo:

- Tip kamere: analogna ali digitalna.
- Korak foto parametrov: 1, 1/2 ali 1/3.
- Kalibracija meritev (za prilagoditev meritev določeni kameri).
- Izbira funkcije ob dvojnem pritisku na gumb za meritev:
  - neprekinjeno merjenje
  - optimalna nastavitev vseh treh foto parametrov glede na trenutno osvetljenost

- Izbira, zameglitev in ponastavitev ozadja, ki se uporablja v aplikaciji.
- Vklop ali izklop Dropbox sinhronizacije.
- Pošiljanje mnenj, pohval in vprašanj v obliki elektronske pošte.



Slika 6.11: Stranski meni.



Slika 6.12: Nastavitve aplikacije.

6.2 Primerjava z merilcem svetlobe

Merilci svetlobe se delijo na dve glavni veji: natančne oziroma certificirane in manj natančne prenosne. Slednje je mogoče kupiti že v malo bolj založenih trgovinah z elektronsko opremo, njihova cena (nekaj deset evrov) pa je mnogo nižja od certificiranih (100 - 150 evrov). Certificirani merilci se uporabljajo predvsem v laboratorijih ter v primerih, kjer je potrebna velika ponovljivost meritev. Zaradi redne kalibracije in testov je tudi njihovo vzdrževanje dražje.

Pri merilcih svetlobe je svetlobni senzor prekrit z ravnim difuzorjem. Največ odziva pokaže senzor, ko pada svetloba pravokotno nanj in najmanj, ko pada pod majhnim kotom. Naš svetlomer uporablja hemisferični difuzor, zato pri meritvi osvetljenosti praviloma vedno dobimo nekoliko višje vrednosti, še posebej v enakomerno osvetljenih prostorih. Poenostavljeno rečeno, naš svetlomer sešteva svetlobo, ki pada iz veliko širših kotov kot navaden merilec svetlobe z ravnim difuzorjem. Tako dobljeni vrednosti vse pogosteje rečejo tudi "efektivna osvetljenost", medtem ko merilec svetlobe izmeri "pravilno osvetljenost", to je količina svetlobnega toka, ki pade na enoto površine [6].

Zaradi nekoliko drugačnega odziva difuzorja naš svetlomer ne bo konkurenčen certificiranim merilcem svetlobe. Zaradi možnosti natačne kalibracije pa je z vidika uporabnosti lahko konkurenčen večini navadnih merilcev svetlobe.

6.3 Primerjava s fotografskim svetlomerom

Že obstoječe fotografske svetlomere (ki so še na trgu) ne delimo na amaterske ter profesionalne, ampak jih ločimo po funkcijah. Poznamo take, ki merijo le ambientalno svetlobo ter ostale, ki zmorejo poleg tega meriti še svetlobne bliske iz studijskih bliskavic (angl. flash) [5]. Naš svetlomer ne podpira funkcije merjenja bliskavic. V največji meri zato, ker so zaradi hitrega razvoja in pojavljanja novih tehnologij taki merilci vse manj v uporabi. Še posebej v zadnjem desetletju je uporaba svetlomerov upadla ter se obdržala zgolj pri videosnemalcih, analognih in profesionalnih fotografih.

Cilj naloge je bil izdelava aplikacije za svetlomer, ki lahko nadomesti samostojne svetlomere. Če naš svetlomer postavimo ob bok fotografskemu svetlomeru in izvajamo meritve različnih svetlobnih pogojev, bosta kazala enake rezultate, kot lahko vidimo na sliki 6.13. Zato lahko rečemo, da je glede natančnosti popolnoma primerljiv z mnogo dražjimi fotografskimi svetlomeri. Cena našega svetlomera je okoli 100 evrov, kar je dosti ceneje od fotografskih

Slika 6.13: Primerjava obeh svetlomerov.

svetlomerov, ki stanejo 200 evrov in več. Za potrebe fotografov, videosnemalcev in ostalih uporabnikov svetlomerov bi lahko rekli, da je naš svetlomer še preveč natančen. Poleg tega ga lahko tudi kalibriramo za različne fotoaparate oziroma okuse. Bistvena prednost naše rešitve leži v aplikaciji ter najrazličnejših možnostih uporabe, pri katerih so obstoječi svetlomeri omejeni. Z uporabo iOS naprave lahko ponudimo boljšo uporabniško izkušnjo, ki je svetlomeri do sedaj še niso imeli. Menimo, da že sedaj nudimo boljši uporabniški vmesnik, z upoštevanjem mnenj uporabnikov in fotografov pa ga bomo lahko še nadgradili in izboljšali.

6.4 Uporaba ogrodja

Uporaba našega razvitega ogrodja za komunikacijo s svetlomerom je relativno preprosta in standardna za razvijalca iOS aplikacij. Razvijalec v svoj projekt najprej uvozi datoteko ogrodja. V želenem razredu, ki bo upravljal s komunikacijo, mora uvoziti javno zaglavno datoteko, kot je to vidno v izvorni kodi 6.1.

```
1 #import <CommunicationManager/CommunicationManager.h>
```

Izvorna koda 6.1: Uvoz ogrodja

Zatem je potrebno objekt prijaviti za delegata in implementirati vse potrebne metode, ki so deklarirane v protokolu. Te metode ogrodje ob ustreznem času izvede. Na primer, ob vsakem sprejetem podatku pošlje delegatu sporočilo o podatku, tako da izvede metodo, ki je temu namenjena.

```
1 - (void)communicationManagerDidReceiveData:(double)value
2 {
3     self.valuLabel.text = [NSString stringWithFormat:@"%.1f", value];
4 }
```

Izvorna koda 6.2: Sprejem vrednosti preko protokola

Primer metode za sprejem meritve je viden v izvorni kodi 6.2, ki kot vhodni parameter dobi decimalno število, ki predstavlja meritev. To število enostavno prikaže na zaslonu naprave.

Poglavje 7

Zaključek

V diplomskem delu smo opisali potek razvoja iOS aplikacije za svetlomer in razvoj ogrodja za uporabo komunikacije s svetlomerom. Dosegli smo večino zastavljenih ciljev. Aplikacija deluje dobro, zanesljivo in omogoča intuitivno uporabo svetlomera. Hitrost komunikacije je dovolj visoka in redko prihaja do slabo sprejetih podatkov. Ogrodje je enostavno za uporabo in deluje enako dobro kot komunikacija v aplikaciji.

Tudi prejeti odzivi prvih uporabnikov so večinoma pozitivni. Všeč jim je izgled in delovanje aplikacije, uporaba svetlomera in izvajanje meritev je enostavna. Fotografije, ki so bile posnete s pomočjo našega svetlomera so pravilno osvetljene. Vse to kaže na odlično uporabniško izkušnjo, ki je za nas tudi najbolj pomembna.

Seveda vedno obstaja prostor za napredek in izboljšanje. Tudi našo aplikacijo je še možno izboljšati. V nadaljevanju so opisane možne optimizacije in nadaljnje delo.

7.1 Odprava napak in optimizacija

Pri izdelavi programske opreme se zelo redko izognemo vsem napakam. V našem primeru se včasih pojavi napaka pri generiranju zvoka zaradi preklapljanja audio seje med različnimi aplikacijami, ki uporabljajo audio izhod.

To je izjema, ki se redko zgodi in je težko ponovljiva. Potrebno bi bilo izboljšati začetek audio seje in preverjanje, ali je audio izhod popolnoma na voljo oziroma je v uporabi v drugi aplikaciji.

Kot smo že omenili, aplikacija sama nastavi maksimalno možno glasnost, da pošljemo čim več energije svetlomeru. Nekaj te energije je odvečne, ker je svetlomer ne potrebuje toliko za delovanje. Za optimalen izkoristek in minimalno porabo energije na iOS napravi, bi lahko določili optimalno glasnost in tako nekoliko zmanjšali obremenjenost baterije na iOS napravi.

Z iOS7 so se pojavili novi problemi. Opustili so podporo za direktno nastavljanja glasnosti iOS naprave. To pomeni, da v prihodnosti mogoče ta funkcija ne bo več na voljo, kar bi pomenilo, da bi aplikacija morala uporabniku sporočiti, da mora sam nastaviti maksimalno glasnost, kar bi zmanjšalo enostavnost uporabe aplikacije.

V iOS7 prihaja do še ene težave. Včasih nam operacijski sistem ne sporoči dovolj hitro, da je prišlo do spremembe audio kanala - v našem primeru izklopa svetlomera - tako da se pred zaključkom audio seje lahko sliši signal za napajanje svetlomera visoke frekvence, ki ni najbolj prijeten za uho. Potrebno bi bilo poskusiti na kakšen drugačen način rešiti to težavo, oziroma jo rešiti, ko bodo na voljo popravki za iOS7.

Za maksimalen izkoristek bi lahko izboljšali tudi komunikacijo. Pri izmenjavi podatkov bi lahko vrsto za zaklepanje nadomestili z vrsto brez zaklepanja. Potrebno bi bilo rešiti ABA problem. Z vrsto brez zaklepanja ne bi nobena od niti več morala spati in čakati na izvajanje.

7.2 Prihodnost

Glede na popularnost našega svetlomera in pozitivne odzive uporabnikov menimo, da je nadaljnji razvoj smiseln. Potrebno bi bilo analizirati vsa mnenja in predloge uporabnikov, da bi vedeli, katere dele aplikacije je potrebno popraviti in kaj je potrebno dodati. Zagotovo je to aplikacijo možno še izboljšati. Veliko uporabnikov si želi video način, saj video snemalci uporab-

ljajo nekoliko drugačne parametre ob izvajanju meritev. Aplikacija bi morala vsebovati ločen pogled za video snemalce, za merjenje pa bi se lahko uporabljal isti svetlomer.

Poleg te aplikacije, menimo, da se splača izboljšati tudi ogrodje, dodati več funkcionalnosti, da bi privabili še več drugih razvijalcev, ki bi lahko implementirali svoje ideje za uporabo svetlomera. Na ta način bi lahko razširili uporabnost svetlomera in privabili še več uporabnikov, ki ne bi bili nujno fotografi, ampak tudi drugi, ki uporabljajo svetlomere in merilce svetlobe na drugih področjih.

Literatura

[1] (2014) Square. Dostopno na:
https://squareup.com/features

[2] (2014) Thermodo. Dostopno na:
http://thermodo.com

[3] (2014) Chipolo. Dostopno na:
http://chipolo.net

[4] (2014) Photography 101.8 – the light meter. Dostopno na:
http://digital-photography-school.com/photography-1018-meter

[5] R. E. Jacobson, G. G. Attridge, S. F. Ray, N. R. Axford, *The manual of Photography: photographic and digital imaging – 9th ed.*, Focal Press, 2000

[6] (2014) Light Meter — Wikipedia, the free encyclopedia. Dostopno na:
http://en.wikipedia.org/wiki/Light_meter

[7] (2014) Apple MFI program. Dostopno na:
https://developer.apple.com/programs/mfi/

[8] (2014) Photometer app for iOS. Dostopno na:
https://itunes.apple.com/us/app/photometer/id429873747?mt=8

[9] (2014) Luxi adapter for iPhone. Dostopno na:
http://www.kickstarter.com/projects/jamesflynn/luxi-incident-light-meter-adapter-for-iphone

[10] (2014) Merilec krvnega sladkorja 2in1.SMART. Dostopno na: http://www.vpd.si/sl/Merilci_krvnega_sladkorja/2in1_smart

[11] (2014) HiJack project. Dostopno na: http://web.eecs.umich.edu/∼prabal/projects/hijack

[12] (2014) Apple Inc., iOS Technology Overview, sep. 2013. Dostopno na: https://developer.apple.com/library/ios/documentation/Miscellaneous /Conceptual/iPhoneOSTechOverview/iOSTechOverview.pdf

[13] (2014) Android dominates market share, but Apple makes all the money. Dostopno na: http://www.forbes.com/sites/tonybradley/2013/11/15/android-dominates-market-share-but-apple-makes-all-the-money/

[14] (2014) What does the average android, iOS user look like? Dostopno na: http://www.pcmag.com/article2/0,2817,2391339,00.asp

[15] (2014) ARM LPC111X product data sheet. Dostopno na: http://www.nxp.com/documents/data_sheet/LPC111X.pdf

[16] Y. Kuo, S. Verma, T. Schmid, P. Dutta, *Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface technical report*, Electrical Engineering and Computer Science Department University of Michigan, Michigan 2010

[17] (2014) Frequency-shift keying. Dostopno na: http://www.princeton.edu/∼achaney/tmve/wiki100k/docs/Frequency-shift_keying.html

[18] (2014) SoftModemTerminal project. Dostopno na: https://code.google.com/p/arms22/downloads/detail?name=Soft-ModemTerminal-004.zip

[19] (2014) Apple Inc., Programming with Objective-C, dec. 2012. Dostopno na:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ProgrammingWithObjectiveC.pdf

[20] (2014) iPhone audio specifications. Dostopno na: http://www.iosrecording.com/tech-notes-iphone-audio-specifications

[21] (2014) Apple Inc., Audio Unit hosting guide for iOS, sep. 2010. Dostopno na: https://developer.apple.com/library/ios/documentation/MusicAudio/Conceptual/AudioUnitHostingGuide_iOS/AudioUnitHostingGuideForiOS.pdf

[22] (2014) Apple Inc., Audio Queue Services Programming Guide, dec. 2013. Dostopno na: https://developer.apple.com/library/ios/documentation/MusicAudio/Conceptual/AudioQueueProgrammingGuide/AudioQueueProgrammingGuide.pdf

[23] (2014) Modem — Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Modem

[24] Texas Instruments, MSP430 embedded soft-modem demo, jul. 2004. Dostopno na: http://www.ti.com/lit/an/slaa204/slaa204.pdf

[25] (2014) Apple Inc., iOS Human Interface Guidelines, okt. 2013. Dostopno na: https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf

[26] (2014) Introducing JSON. Dostopno na: http://www.json.org

[27] (2014) Apple Inc., Start Developing iOS Apps Today, okt. 2013. Dostopno na: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/RoadMapiOS.pdf

[28] (2014) Apple aurioTouch2 project. Dostopno na: https://developer.apple.com/library/ios/samplecode/aurioTouch2/Introduction/Intro.html

[29] (2014) Apple SpeakHere project. Dostopno na: https://developer.apple.com/library/ios/samplecode/SpeakHere/Introduction/Intro.html

[30] D. Dechev, P. Pirkelbauer, B. Stroustrup, Understanding and Effectively Preventing the ABA Problem in Descriptor-based Lock-free Designs, *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium*, Carmona, Seville, maj 2010. Dostopno na: http://www.stroustrup.com/isorc2010.pdf

[31] (2014) ViewDeck project. Dostopno na: https://github.com/Inferis/ViewDeck

[32] D. Mark , J. Nutting , J. LaMarche , F. Olsson, *Beginning iOS 6 Development: Exploring the iOS SDK*, Apress, str. 619 - 635, 2013.

[33] (2014) Dropbox Sync API for iOS documentation. Dostopno na: https://www.dropbox.com/developers/sync/docs/ios

[34] (2014) iOS framework. Dostopno na: https://github.com/jverkoey/iOS-Framework

[35] (2014) iOS application for our light meter. Dostopno na: https://itunes.apple.com/us/app/lumu/id730969737?mt=8

[36] (2014) Our light meter project web page. Dostopno na: http://lu.mu