TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut
Informaatika eriala

Juhan Hion

Make It White - valge tasakaalu rakendus iOSile

Bakalaureusetöö (6 EAP)

Juhendaja: Margus Niitsoo

Autor: ..... "....." mai 2013
Juhendaja: ..... "....." mai 2013
Lubatud kaitsmisele
Professor: ..... "....." mai 2013

TARTU
2013

Sisukord

Sissejuhatus

Värvitasakaalu [1] saab defineerida globaalse värviintensiivsusena, mille üheks alaliigiks on valge tasakaal. Digitaalkaamerad üritavad küll tarkvaraliselt värvitasakaalu korrigeerida, pakkudes nii automaatseid kui ka konkreetsetele valgusallikatele mõeldud seadeid, kuid kehvas valguses jäävad tulemused ilma käsitsi valge tasakaalu seadistamata ikkagi ebarahuldavateks ning pildid jäävad valgusallikast olenevalt tonaalsuselt liiga kollasteks või liiga sinisteks. Sellest puudusest ei ole prii ka Apple'i iPhone'i kaamerad.

Käesoleva töö käigus loodi iPhone'ile fotode valge tasakaalu muutmise rakendus Make It White, millega saab pildilt valge koha leida ja selle järgi värvikorrektuuri teostada. Tõuke iPhone'ile rakenduse loomiseks andis autori soov peale nelja-aastast erinevate iPhone'ide kasutamist ja erinevate rakenduse-ideede peas keerutamist viimaks üks nendest ideedest ka realiseerida.

Mobiilsete seadmetega pildistamisest on saanud igapäevaelu osa – inimestele meeldib pildistada ennast, oma õhtusööke ja kasse. Enamasti aga toimub see halbades valgustingimustest, mis, hoolimata pildistaja osavusest või esteetilistest taotlustest, ei võimalda kvaliteetse tulemuseni jõuda. Nii on Make It White'i laadse rakenduse jaoks vajadus olemas ning seetõttu ka võimalik, et see võib leida endale kasutajaid Apple'i rakenduste poe klientide seast, kes on selle eest nõus raha maksma.

Käesolev töö on jaotatud kolme peatükki ning kahte lisasse:

- Esimene peatükk kirjeldab valge tasakaalu probleemi ning seletab, miks loodi just selline rakendus.
- Teine peatükk annab ülevaate valge tasakaaluga seonduvast värviteooriast ning seletab lahti selle töö käigus kasutatud mõisted.
- Kolmas peatükk tutvustab põgusalt Objective-C programmeerimiskeelt ning keskendub seejärel loodud rakenduse tehnilise lahenduse keerulisemate aspektide kirjeldamisele.
- Lisas 1 on loodud rakenduse ekraanipildid ning kasutusjuhend.
- Lisas 2 on digitaalselt toodud rakenduse lähtekood.

1. Valge tasakaalu probleem ja loodud rakenduse põhjendus

1.1. Valge tasakaalu probleem

Valge tasakaalu problemaatikat on kõige lihtsam näitlikustada fotodega. Joonisel 1 on kujutatud esmalt ebakorrektse valge tasakaaluga pilt ning seejärel korrigeeritud valge tasakaaluga pilt.

Joonis 1 - Valge tasakaalu näide – algne ja korrigeeritud foto

Antud näitepildi algne versioon on tehtud iPhone 5 kaameraga ning korrigeeritud Make It White'iga. Joonise vasakul poole on näha, mis juhtub värvidega, kui pilt on tehtud nõrgas hõõglambi valguses ja kasutatud on automaaset valge tasakaalu algoritmi. Kui pilt oleks tehtud luminofoorlambi valguses oleks see liialt sinine.

Ebakorrektne valge tasakaal ei ole ainult digitaalsete kaamerate probleem, see on olnud olemas juba esimestest värvifotodest alates. Analoogkaamerate puhul lahendati valge (ja üldse värvi-) tasakaalu probleeme füüsiliste vahenditega – objektiivide eest kasutati valgussituatsiooni korrigeerimiseks filtreid ning eriti filmikaamerate puhul ka spetsiaalseid filme. Samuti ei oodatud värvifotodelt pikka aega täiuslikku elu peegeldust, vaid nihkunud värvitasakaal lisas neile pigem väärtust.

Digitaalsete kaamerate plahvatusliku levikuga muutus valge tasakaalu probleem aga ühest hetkest aktuaalseks, sest laiatarbe digikaamerate pildikvaliteet saavutas ideaalsetes valgustingimustes tavainimese silma jaoks praktiliselt täiusliku kvaliteedi.

Sestap on tänapäeval kõikides müüdavates digitaalsetes fotokaamerates olemas tarkvaraline valge tasakaalu kompenseerimise algoritm, kuid autori kogemused amatöör-fotograafina on näidanud, et nõrgas tehisvalguses tehtud fotodel jääb värvitasakaal tihti ebakorrektseks.

Enamikes laiatarbe digikaamerates on valge tasakaalu automaatrežiimile lisaks tavaliselt erinevate keeruliste valgussituatsioonide jaoks mõeldud spetsiaalrežiimid, kuid mobiilsete seadme, mis oma funktsioonilt on ennekõike telefonid ja pihuarvutid ning alles seejärel kaamerad, tehaseseadetes sellised võimalus puuduvad. Apple'i iPhone ei ole selles suhtes erand, vaid pigem reegel. Apple'i Camera rakendus iPhone'ile on spartaliku kasutajaliidese musternäide ning sellel puudub igasugune peenem häälestamisvõimalus.

iPhone'ile on saadaval paljusid kolmandate poolte poolt arendatud kaamerarakendusi, milledest paljudel on olemas laiatarbe digikaameratele sarnanevad valge tasakaalu režiimid (Näiteks: KingCamera) või omavad võimalust juba pildistamise hetkel valge punkt valida (Näiteks: ProCamera).

Esimesel juhul toodavad aga ka need rakendused kehvades valgussituatsioonides vale valge tasakaaluga pilte. Teisel juhul tähendab valge punkti otsimine pildistamise hetkel seda, et pildi tegemine võtab rohkem aega, sest valge punkt tuleb kadreerimisel paika sättida. Selline ajakulu läheb aga käesoleva töö autori silmis vastuollu kompaktkaamerate, mida mobiilne seade asendada püüab, suuna-ja-pildista (*point-and-shoot*) eetosega.

1.2. Loodud rakenduse põhjendus

Enamik tänapäeva kompaktkaameraid ei ole veel võrguühenduse võimalusega ja eeldavad piltide jagamiseks või mugavaks vaatamiseks arvuti kasutamist. Arvutit kasutades on piltide järeltöötlemine juba aastakümneid arenenud protsess ning seal on olemas nii automaatsed (Näiteks: Apple'i iPhoto) kui ka manuaalsed (Näiteks: Apple'i Aperture või Adobe Photoshop) vahendid valge tasakaalu seadmiseks.

Mobiilsete ja võrku ühendatud seadmete levik on muutnud arvuti vahelülina kasutamise ebavajalikuks, sest pilte on võimalik jagada ning esitada otse seadmest. See

omakorda tekitab potentsiaalse vajaduse (ja turunišši) pilditöötlemisrakenduste järele mobiilsetes seadmetes.

Käesoleva töö autori jaoks ongi erinevad iPhone'i mudelid viimastel aastatel edukalt asendanud digitaalset fotokaamerat, sest nende pildikvaliteet on laiatarbe digitaalkaameratele päris lähedale jõudnud. Puudused pildikvaliteedis teeb tasa arvuti kasutamise vajaduse puudumine. Kuid arvuti ahelast eemaldamisega kaasneb vajadus lisada teatav hulk fototöötlusvahendeid mobiilseadmesse.

Valge tasakaalu muutmise jaoks ei ole aga autor ühtegi Make It White'i ideega samasugust funktsionaalsust pakkuvat rakendust Apple'i AppStore'ist [2] leidnud. Värvitasakaalu muutmisfunktsionaalsus on olemas paljudes iPhone'i komplekssetes pilditöötlusrakendustes (Näiteks: Apple Photos, Adobe Photoshop Express), kuid seal on pakutud funktsionaalsus kas automaatne, või siis liuguritega iga värvikanali jaoks.

Kuid automaatne värvitasakaal põhineb pildianalüüsil ning värvuste statistilisel jaotusel, mis ei garanteeri tugevalt moondunud värvitasakaalu korral korrektset tulemust, sest värve üritatakse "keskele" nihutada. Liuguritega värvitasakaalu muutmine on aga käesoleva töö autori arvates liialt aeganõudev ja keeruline protsess, mida on mugavam teha juba suure ekraanil ja seega arvuti abil.

Apple'i seadmetes on laialt levinud nõndanimetatud "ühe funktsiooni rakendused", mis on loodud tegema ainult ühte asja. Võib arvata, et selline praktika sai alguse vajadusest teha ilmselgelt vajalikke asju, mida Apple'i enda standardtarkvara ei võimaldanud. iPhone'i ja seal töötava operatsioonisüsteemi iOSi [3] algusaegadel oli Apple'i enda poolt pakutavate rakenduste funktsionaalsus tihti äärmiselt piiratud. Markantse näitena võib tuua veel paari aasta eest laialt levinud fotode pööramise rakendused, sest Apple Photos rakendus lihtsalt ei pakkunud seda võimalust. Samuti on selletüübiliste rakenduste levikule kaasa aidanud asjaarmastajatest pildistajate hulga suur kasv, kes ootavad keeruliste mitmefunktsiooniliste programmide asemele lihtssaid, konkreetseid ja kiireid lahendusi baasprobleemidele.

AppStore'is on sestap sadu sellest filosoofiast lähtuvaid rakendusi. Näiteks on töö autori telefonis hetkel järgmised ainult ühte fototöötlusfunktsiooni pakkuvad rakendused:

- AutoStitch – panoraamfotode loomiseks
- ColorSplash – fotode ilmestamiseks neid osaliselt halltoonidesse konverteerides
- Diptic – kollaažide loomiseks
- Tilt-Shift Generator – piltide fookusplaani muutmiseks

- Flickr – piltide Flickr.com-i laadimiseks

Idee ainult valge tasakaalu muutmist teostavast rakendusest sündis sarnase rakenduse puudumise ning ühe funktsiooni rakenduste laia leviku koosmõjul. Samuti mängisid ainult ühte funktsiooni realiseeriva rakenduse kasuks otsustamisel rolli praktilised kaalutlused: see, et tegemist on autori esimese iOSi rakendusega, selle teostamise ajalised piirangud ja planeeritud realisatsiooni tehniliste aspektide keerukus. Rohkemate funktsioonide realiseerimine oleks hinnanguliselt toonud kaasa eksponentsiaalse ajavajaduse kasvu ning oleks muutnud loodava rakenduse järjekordseks (arvatavasti keskpäraseks) pilditöötlejaks. Esimese iOSi rakendusena olid planeerimisfaasis suurematekst riskideks ettenägematud tehnilised probleemid ning rakenduse arendamise käigus oligi autoril ühel hetkel tunne, et see rakendus jääbki tehniliste probleemide tõttu sündimata. Laiema funktsionaalsuse planeerimine oleks rakenduse valmimise eeldatavasti nurjanud. Samuti tuli rakenduse kasutajakogemuse sujuvuseks ning kasutusmugavuseks kasutada mitmeid keerulisi tehnoloogilisi lähenemisi, nagu näiteks lõimede kasutamine või keerulised pildioperatsioonid.

2. Värviteooria

2.1. Valge tasakaal

Tuntud kaameravalmistaja Nikoni eestikeelne kasutajatoe artikkel [4] seletab valge tasakaalu olemuse lahti järgmiselt: “Valge tasakaalu kasutatakse värvide reguleerimiseks, et tagada valgete objektide ilmumine valgena. Objekte võib valgustada palju erisuguseid valgusallikaid, sh päikesevalgus, hõõglambid ja luminofoorvalgustus, millel on erinev värvitemperatuur. Ehkki palja silmaga vaadatuna tunduvad need erinevad valgusallikad andvat sama värvi valgust, eraldavad need tegelikult eri varju või värviga valgust. /…/ Digitaalkaamera pildiandur esitab neid värvierinevusi, nagu need on. Selle tulemusena näib foto, värvi täiendava töötlemiseta, muutuvat valgusallika kohaselt.”

Kõik digitaalsete kaamerate (ja kaameraid sisaldavate seadmete) tootjad pakuvad oma kaamerates vähemalt automaatset valge tasakaalu parandamist, kallimatel mudelitel on võimalik valida erinevate režiimide vahel ning hea õnne korral on võimalik valge tasakaal hallkaarti [5] kasutades käsitsi paika seada.

Apple ei ole avalikustanud iPhone’i (3G/4/5) kaameraga seonduvat valge tasakaalu informatsiooni, kuid isiklikule kogemusele põhinedes (mida jagavad ka erinevat postitused Internetis) kasutatakse ka nendes seadmetes automaatset valge tasakaalu parandamist.

Nikon [4]:”Automaatne valge tasakaal töötleb automaatselt pilti soovimatute pildinihete eemaldamiseks, näiteks muudab hõõglampide valguses tehtud pildid sinisemaks, et korrigeerida seda tüüpi valgustuse punakat nihet. Harilikult tekitab automaatne valge tasakaal soovitud tulemusi, ilma et fotograaf peaks muret tundma valgustuse tüübi pärast.”

Autori kogemused erinevate iPhone mudelitega ning erinevate digitaalsete kompakt- ja peegelkaameratega on näidanud, et iga kaamera jaoks leidub

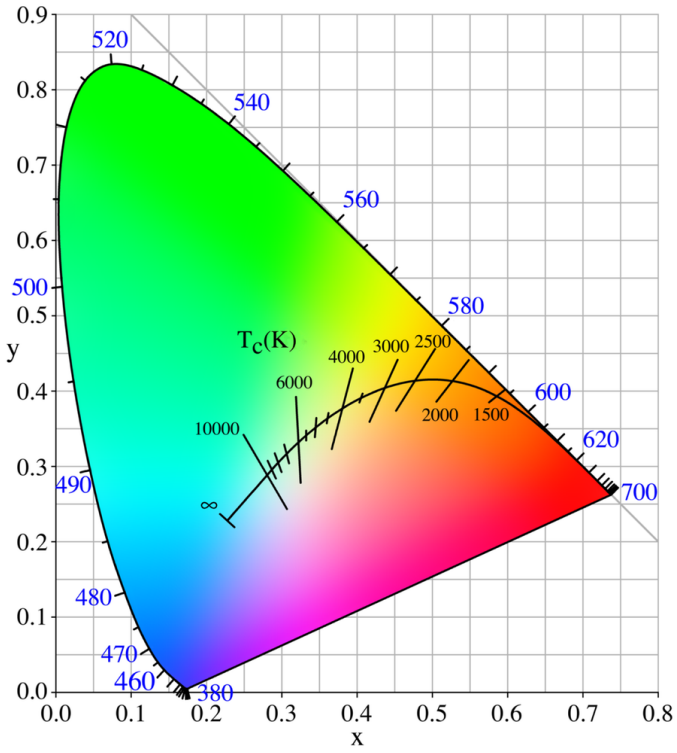
valgussituatsioon, kus automaatse valge tasakaalu algoritm ei anna enam soovitud tulemusi.

2.2. Värvitemperatuur

Toodud valge tasakaalu seletuses on kasutusel mõiste värvitemperatuur, mis on valge tasakaalu seadistamise juures olulisim mõiste.

Joe Smith defineerib oma artiklis *Calculating Color Temperature and Illuminance using the TAOS TCS3414C Digital Color Sensor* [7] värvitemperatuuri järgmiselt: "Värvitemperatuur, mõõdetud Kelvinites (K), viitab temperatuurile, milleni tuleks soendada absoluutselt must keha [8] (või Plancki radiaator), tootmaks mingit kindlat värvi valgust. [...] Absoluutselt musta keha soojendamise käigus kiirgab see esmalt energiat infrapunaspektrumis ning seejärel nähtavas spektrumis punase, oranži, valge ja viimaks sinakas-valge valgusena."

Oluline on märkida, et värvitemperatuuriga ei saa kirjeldada kõiki värve, sest absoluutselt must keha ei omanda soojendamisel kunagi kõiki värvitoone (näiteks rohelist või lillat). Värvitemperatuur omab seega tähendust ainult toonide puhul, mida absoluutselt must keha omandada suudab. Absoluutselt musta keha võimalikud värvitoonid ja neile vastav värvitemperatuur on välja toodud joonisel 2 oleva Plancki lookusega. Inimsilmaga valgena tunnetatud toonid omandavad kehvas valgussituatsioonis tehtud fotodel aga just neid toone, olles hõõglambi valgusel tehtud fotodel punasemad ning luminofoorlambi valgusel tehtud fotodel sinisemad.



Joonis 2 – Kromaatilisusdiagramm xyY värviruumis Plancki lookusega [9]

Valgusallikate jaoks, mis ei tooda valgust kuumuse läbi, kasutatakse nende värvitemperatuuri iseloomustamiseks korreleeritud värvitemperatuuri (*Correlated Color Temperature* edaspidi CCT). Smith [7]: "Korreleeritud värvitemperatuur juurutati tegelemaks lairibavalgusallikatega, mis ei ole absoluutselt musta keha poolt modelleeritavad. CCT on defineeritud absoluutselt musta keha temperatuurina, mille kromaatilisuspunkt on lähim mitte-planckilise valgusallika kromaatilisuspunktile. CCT on põhimõtteliselt kirjeldus, kas valgus on sinakas-valge, neutraalne, või punakas-valge."

2.3. Värviruumid

Loodud on erinevaid värviruumide mudeleid, mis lähtuvad erinevatest kaalutlustest. Järgnevas antakse ülevaade värviruumidest, mis olid aluseks selles töös loodud rakendusele.

2.3.1. RGB

Seadmetes kuvatavad pildid opereerivad RGB [10] värviruumis, sest nende iga füüsilise piksli kuvamisene taandub mis iganes ekraanil alati punase, rohelise ja sinise värviallika kombineerimisele. RGB värviruum on aga seadmespetsiifiline – iga seade reprodutseerib antud RGB väärtust erinevalt vastavalt oma füüsilistele karakteristikutele. Seega eeldab RGBs esitatud värvi samamoodi kuvamine alati mingisugust värvihaldust [11]. Värvihalduse pikem käsitlemine jääb aga selle töö raamidest välja.

2.3.2. XYZ ja LMS

Wikipedial [12] põhinedes on XYZ ehk CIE [13] 1931 värviruum üks esimestest matemaatiliselt defineeritud värviruumidest. XYZ värviruum baseerub inimese silma kolmikstiimuli (*tristimulus*) väärtustel.

Inimese silmas on keskmise ja kõrge heledusega värvinägemise fotoretseptoriteks kolme eri tüüpi kolvikesi [14], mille tundlikkus on suurim vastavalt lühikestel (*Short*, 420 – 440 nm), keskmistel (*Middle*, 530 – 540 nm) ja pikkadel (*Long*, 560 – 580 nm) lainepikkustel. Seega saab kolme erineva stiimuli tasemete väärtusega kirjeldada põhimõtteliselt ükskõik millist inimsilmale eristuvat värvi. Neid väärtuseid kajastab LMS [15] värviruum.

Inimsilm kogeb heas valguses erinevate värvide suhtelist heledust hinnates rohelist valgust heledamana kui sama tugevusega punast või sinist valgust. Heledusfunktsioon [16], mis kirjeldab erinevate lainepikkuste tunnetatud heledust, on laias laastus analoogne keskmiseid lainepikkuseid kogevate kolvikeste sagedus-vastuvõtlikkusega.

XYZ värviruum kasutab seda fakti, defineerides Y-i väärtuse heledusena, Z-i väärtuse enamvähem võrdsena lühikesi lainepikkuseid tajuva kolvikese vastuvõtlikkusega ja X-i väärtuse lineaarse kombinatsioonina kõikide kolvikeste vastuvõtlikkuskõveratest, mis on valitud olema mittenegatiivsed. Seega on XYZ kolmikstiimuli väärtused analoogsed, kuid mitte võrdsed LMS väärtustega.

2.3.3. xyY

Nagu eelnevalt sai osutatud, näeb inimsilm värve kolmikstiimuli väärtustena, seega iga värv asub kolmemõõtmelises ruumis. Kontseptuaalsel tasandil saab värvi olemuse aga jagada kaheks: heledus ning kromaatilisus. Näiteks valge ja hall värv on põhimõtteliselt samad värvid, kuid nad erinevad oma heleduse poolest. xyY [17] värviruum on kasutusel kolmikstiimuli väärtuste kahemõõtmelisse ruumi teisendamiseks, millele antakse lisaks veel heleduse väärtus.

Kuna XYZ värviruumis on Y-i väärtus heledus, on võimalik see teisendada xyY värviruumi, kus x-i ja y-i väärtused on funktsioonid kõigist kolmest kolmikstiimuli väärtusest. xyY värviruumi kromaatilisuse diagramm on kujutatud joonisel 2.

2.4. McCamry valem

xyY värviruumis asuvast värvist on võimalik McCamry valemit kasutades arvutada CCT ehk korreleeritud värvitemperatuur. Smith [7] ütleb, et: "McCamry väidab, et valem suudab pakkuda antud x ja y kromaatilisuskoordinaatide korral maksimaalset absoluutset viga, mis on väiksem kui 2 kraadi Kelvinit värvitemperatuuridel vahemikus 2856 K kuni 6500 K (vastates CIE standardsetele valgustajatele [18] (*illuminant*) A kuni D65)." Smith [7] toob välja ka McCamry valemi:

$$CCT = 449n^3 + 3535n^2 + 6823{,}3n + 5520.33$$

$$\text{Milles } n = (x - 0{,}3320) / (0{,}1858 - y)$$

2.5. Valge tasakaalu korrigeerimine

Tehniliselt toimub valge tasakaalu korrigeerimine iga piksli värvi korrigeerimise kaudu. Võimalusi, kuidas algsest ebasobiva valge tasakaaluga pikslist arvutada sobiva valge tasakaaluga piksel on mitmeid.

2.5.1. RGB nihutamine

Kõige naivistlikum lähenemine, mille käesoleva töö autor suutis ilma värviteooriaga tutvumata omal käel välja mõelda, on lihtsalt iga RGB värviruumis asuva piksli väärtuse nihutamine. Eeldusel, et valge värvi korral on R = 255, G = 255 ja B = 255 ning mingi subjektiivne valge omab muid komponentide väärtuseid (R'_w, G'_w, B'_w), siis korrigeeritud valge tasakaaluga piksel saadakse joonisel 3 kujutatud diagonaalmaatriksiga.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 255/R'_w & 0 & 0 \\ 0 & 255/G'_w & 0 \\ 0 & 0 & 255/B'_w \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Joonis 3 - RGB skaleerimise diagonaalmaatriks [19]

2.5.2. Tasakaalustamine läbi mõne alternatiivse värviruumi

Paremaid tulemusi andev lähenemine on konverteerida RGB väärtused mõnda teise värviruumi, sest teisendus RGBst ükskõik millisesse eespool käsitletud värviruumi kätkeb endas mõne CIE standardse valgustaja kasutamist.

Teisendust saaks sellel juhul läbi viia kahel erineval viisil. Esimesel juhul teisendataks RGB mõnda teise värviruumi (näiteks XYZ või LMS) mõne standardse valgustaja juures ja seejärel sellest värviruumist tagasi RGBsse mingi subjektiivse valgepunkti juures. Teisel juhul tehtaks teisendus RGB värviruumist LMS värviruumi ja tagasi mõne standardse valgustaja juures ning valge tasakaalu korrigeerimine viidaks läbi LMS värviruumis, kasutades von Kriesi transformatsiooni [20], mis on kujutatud joonisel 4.

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 1/L'_w & 0 & 0 \\ 0 & 1/M'_w & 0 \\ 0 & 0 & 1/S'_w \end{bmatrix} \begin{bmatrix} L' \\ M' \\ S' \end{bmatrix}$$

Joonis 4 - von Kriesi transformatsioonimaatriks

2.5.3. Tasakaalustamine läbi värvitemperatuuri muutmise

Käesoleva töö praktilises osas kasutati tehnilistel põhjustel[1] valge tasakaalu muutmiseks värvitemperatuuri muudatust, mis autori hinnangul põhineb samuti von Kriesi transformatsioonil.

Arvutamaks antud protseduuri sisendparameetriks olevat subjektiivselt valge koha[2] värvitemperatuuri RGB värviruumis olevast pikslist, tuleb läbi viia kolm teisendust:

- RGB -> XYZ
- XYZ -> xyY
- xyY -> CCT

Teisendamine RGB värviruumist XYZ värviruumi viiakse läbi kahe sammuga:

- komponentide lineariseerimine
- lineaarsete komponentide teisendamine

Edasised teisendused on mõlemad ühesammulised ja ka matemaatiliselt lihtsad, kuid siiski allpool välja toodud.

2.5.3.1. RGB Komponentide lineariseerimine

RGB komponentide lineariseerimiseks kasutatakse ümberpööratud sRGB [21] ahendamist-laiendamist [22] (*companding* [23]), mille valem on toodud joonisel 5, kus V denoteerib lineariseerimata komponendi väärtust, v lineariseeritud komponendi väärtust ning komponendi skaala on vahemikus [0; 1].

$$v = \begin{cases} V/12.92 & V \le 0.04045 \\ \left((V + 0.055)/1.055\right)^{2.4} & V > 0.04045 \end{cases}$$

Joonis 5 - Ümberpööratud sRGB ahendamine-laiendamine

2.3.5.2. RGB Lineaarsete komponentide teisendamine

Lineariseeritud komponendid teisendatakse seejärel XYZ värviruumi, kasutades transformatsioonimaatriksit, mille väärtused sõltuvad valitud valgustajast ja RGB

[1] Põhjused on lahti seletatud loodud tarkvara kirjelduses.
[2] Mõiste "subjektiivselt valge koht" on kasutusel tähistamaks kohta fotol, mis kasutaja kognitiivse kogemuse järgi on valge (taldrik, silmavalge, hammas etc.), kuid mis ei ole seda fotol automaatse valge tasakaalu algoritmi mittekorrektse töö tõttu.

värviruumist. Käesolevas töös eeldati (sest Apple ei ole vastavaid andmeid avaldanud), et kasutusel on sRGB värviruum D65 valgustajaga. Sellele vastav lineaar-RGB -> XYZ transformatsiooni maatriks on toodud joonisel 6.

```
0.4497288   0.3162486   0.1844926
0.2446525   0.6720283   0.0833192
0.0251848   0.1411824   0.9224628
```

Joonis 6 - AppleRGB ja D65 valgustajaga transformatsioonimaatriksi RGB -> XYZ teisenduseks

2.5.3.3. Teisendus XYZ värviruumist xyY värviruumi ja CCT arvutamine

Teisendus XYZ värviruumist xyY värviruumi on triviaalne, nagu nähtub jooniselt 7.

$$x = \frac{X}{X+Y+Z} \; y = \frac{Y}{X+Y+Z}$$

Joonis 7 - Teisendus xyY värviruumi

xyY värviruumist CCT arvutamiseks kasutatakse McCamry valemit, sest subjektiivselt valge koht jääb olemuslikult oma temperatuurilt selle valemi poolt kaetud temperatuurivahemikku.

3. Loodud rakendus Make It White

Käesoleva töö aluseks on iOS versioonile 6 loodud rakendus Make It White, mille ekraanipildid ning kasutusjuhend on toodud lisas 1. Rakenduse eesmärgiks on korrigeerida ühe või mitme samas valgussituatsioonis tehtud foto valge tasakaalu.

Make It White on taotluslikult väga lihtsa lahendusega: rakendusel on ainult üks kasutuslugu, puuduvad seadistusvõimalused ning kasutajalt ei oodata muud sisendit kui nuppude puudutamine ning luubi ja liuguri lohistamine.

Rakenduse ainsas kasutusloos on järgmised sammud:

1. Kasutaja valib mõnest seadmes olevast pildialbumist ühe või mitu sarnases valgussituatsioonis (valgussituatsiooni sarnasus on kasutaja otsus) tehtud fotot.
2. Kasutaja valib ühe nendest fotodest malliks ning otsib sellelt koha, mis on subjektiivselt valge, kasutades selleks virtuaalset luupi.
3. Luubi sobivale kohale seisma jätmise järel loob rakendus valitud koha värvitemperatuuri arvestades korrigeeritud värvidega eelvaate ning kuvab kasutajale liuguri värvide peenhäälestuseks.
4. Kasutaja peenhäälestab vajadusel värve ning laseb seejärel valitud subjektiivselt valge koha ja peenhäälestatud tonaalsuse järgi töödelda ka kõik muud esimesel sammul välja valitud fotod.
5. Rakendus töötleb fotod ning salvestab need albumisse, kust nad algselt võeti.

3.1. Objective-C

Apple'i seadmetele tarkvara loomine erineb märgatavalt näiteks veebirakenduste või personaalarvutil kasutava tarkvara loomisest. Üheks kõige märgatavaks erinevuseks on kogu ökosüsteemi kinnisus – selleks, et legaalselt iOSi seadmetele rakendusi luua ja neid AppStore’is levitada, on vajalik Apple arvuti olemasolu, tasuline liitumine Apple

iOS Developer [24] programmiga ning rangelt soovituslik on mõne füüsilise seadme olemasolu, olgu selleks iPhone, iPod Touch, iPad või iPad Mini.

Apple'i arendusvahendite keskmeks on Xcode [25], mis on täiesti tänapäevane arendusvahend ning ei jää oma funktsionaalsuses mitte millegagi alla näiteks Microsofti Visual Studio 2012le ja mõnes kohas isegi ületab seda. Xcode sisaldab ka tarkvaralist iOS simulaatorit [26], kuid mitmed sensoritega seonduvad funktsioonid ja teegid on kasutatavad ja proovitavad ainult füüsilise seadmega.

iOS platvormile on võimalik rakendusi luua kasutades ainult Objective-C programmeerimiskeelt. Objective-C on objektorienteeritud kõrgkeel, mis lisab Smalltalki [27] stiilis sõnumid C-keelele.

Objective-C teeb muudest keeltest erinevaks kasutusel olevad kaks paralleelsüntaksit:

- kogu mitte-objektorienteeritud süntaks on identne klassikalise C süntaksiga
- kogu objektorienteeritud süntaks on implementeeritud Smalltalki stiilis sõnumitega

Samuti puuduvad Objective-C keeles nimeruumid, nende asenduseks on kasutusel klasside prefikseerimine – enamik süsteemseid klasse on prefikseeritud NS (NeXTSTEP) prefiksiga, erinevatesse raamistikesse kuuluvad klassid on prefikseeritud raamistu initsiaalidega etc.

3.1.1. Mäluhaldus

Erinevalt kõikidest skriptimiskeeltest ning Javast ja C#-ist tuleb Objective-Cd kasutades teadlikult mäluhaldusele mõelda.

Objective-C mäluhaldus baseerub *retain* loenduril, mida iga instantsi kohta hoitakse. Programmeerija ülesanne on seda loendurit suurendada ja vähendada vastavalt objektide elueale. Iga instants, mis vajab viidet mõnele teisele instantsile, peab tema poolt viidatava instantsi *retain*-loendurit suurendama hetkel, kui viidatav instants viitava instantsi kätte jõuab. Kui viitav instants enam viidatavat instantsi ei kasuta, siis tuleb loendurit vähendada. Loenduriga opereerimiseks kasutatakse *retain, copy* ja *release* sõnumeid. Kui loendur jõuab nullini, vabastatakse mälu.

Protsessi lihtsustamiseks saab iOSi jaoks rakendusi programmeerides kasutada *Automatic Reference Counting*ut [28] (lühidalt ARC), mis lisab *retain, copy* ja *release*

sõnumid enne kompileerimist toimuva staatilise koodianalüüsi käigus automaatselt. ARC vabastab programmeerija objektorienteeritud maailmas toimuvast mäluhaldusest välja arvatud *property*tega seonduvad mäluhaldusdirektiivid.

Allan kirjeldab raamatus *Learning iOS Programming* [29] mäluhaldusdirektiivide semantikat järgmiselt: "*assign-, retain-, copy-, weak-* ja *strong*-atribuudid juhivad väärtuse seadmise aksessormeetodit ja on teineteist välistavad. *Assign*-atribuut on vaikeväärtus ja tähendab, et genereeritud seadmismeetod kasutab lihtsalt omistamist. *Retain*-atribuut spetsifitseerib, et omistamisel tuleks objektile saata *retain*-sõnum ning eelmisele väärtusele *release*-sõnum. [...] *Copy*-atribuut tähendab seda, et omistamisel tuleb teha omistatavast objektist koopia lihtsa omistamise asemel. [...] *Weak*-atribuut spetsifitseerib, et objektiga on nõrk (mitteomav) seos. Kui objekt deallokeeritakse siis seatakse *property* väärtus automaatselt *nil*iks. [...] Ümberpööratult indikeerib *strong*-atribuut omandavat seost – klassiinstants võtab viidatava objekti enda omandusse ning seda ei deallokeerita enne, kui omanik pole deallokeeritud."

Oluline on märkida, et ARC ei toimi C-maailmas ning C ja Objective-C maailmade vaheline liikumine eeldab hoolikat mäluhalduse planeerimist. Mäluhaldus on väga mahukas teema ning selle edasine põhjalikum käsitlemine ei kuulu antud töö raamesse.

3.2. Kasutatud kolmandate poolte teegid

Rakenduse loomisel katsuti kasutada võimalikult palju standardseid Apple'i poolt pakutavaid komponente. Apple'i komponentide ebasobivuse või nende puudumisel kasutati võimalusel vabavaralisi alternatiive.

Kuna Apple'i poliitika [30] ei luba iOSi rakendustesse binaarselt linkida kolmandate poolte teeke, siis osutus vabavaraliste lahenduste leidmine väga lihtsaks, sest kommertsteekidel puudub Apple'i poliitika tõttu turg, aga arendajad soovivad oma tehtud tööd ikka jagada.

Rakenduses on kasutusel järgmised vabavaralised teegid:

- ELCImagePickerController[3]

[3] https://github.com/elc/ELCImagePickerController (19.04.2013)

- MBProgressHUD[4]
- UIImageView+GeometryConversion kategooria[5]

3.2.1. ELCImagePickerController

ELCImagePickerControllerit kasutatakse seadmes olevate pildialbumite loetelu kuvamiseks, albumi valimiseks, albumis olevate piltide kuvamiseks ning ühe või mitme pildi välja valimiseks. ELCImagePicker on kujutatud lisas 1 joonisel 2.

ELCImagePickerControlleri kasutamise tingis Apple'i sarnast funktsionaalsust pakkuvast komponendist (UIImagePickerController [31]) puuduv mitme pildi valimise võimalus.

Käesolevas rakenduses on ELCImagePickerController kasutusel modifitseeritud kujul, sest erinevalt originaalimplementatsioonist on Make It White'i kontekstis olulised viited valitud piltidele (ALAsset [32]), mitte nende piltide mingisugused konkreetsed representatsioonid, mida ELCImagePickerController algselt tagastas.

Samuti on ELCImagePickerControllerist välja toodud viide ALAssetLibraryle [33], sest ALAssetiga tegutsemiseks on vaja taustal alati initsialiseeritud ALAssetLibraryt. ALAssetLibraryst tehti staatiline *singleton* [34] *property*.

Pikemas perspektiivis on plaanis ELCImagePickerControlleri praegune visuaalne pool, mis on arendatud käsitööna, asendada UICollectionView'l [35] põhineva implementatsiooniga. Selle muudatuse motivaatoriks on praeguses implementatsioonis puuduv tugi ekraanigeomeetria korrektseks ringiarvutamiseks seadme pööramisel [36] ning Apple'i "Photos" rakendusest erinev käitumine piltide joondamisel.

3.2.2. MBProgressHUD

Kuigi seadmes on olemas teek UIProgressHUD, millega näidata modaalseid progressiindikaatoreid, kuulub see nn "dokumenteerimata APIde" alla, mille kasutamine ei ole lubatud [29]. Sarnast ning isegi paremat funktsionaalsust pakub MBProgressHUD, mis on näha lisas 1 joonisel 6.

[4] https://github.com/jdg/MBProgressHUD (19.04.2013)
[5] https://github.com/nubbel/UIImageView-GeometryConversion (19.04.2013)

MBProgressHUD on kasutusel muutmata kujul. Ainsa etteheitena võib välja tuua arhitektuurilise otsuse, et MBProgressHUD käivitab ise taustaprotsessi ning seega otsustab, kuidas seda käivitada. Make It White'i kontekstis see aga reaalset probleemi ei tekitanud.

3.2.3. UIImageView+GeometryConversion kategooria

Nimetatud kategooria on muutmata kujul kasutuses ekraani koordinaatsüsteemis olevate punktide teisendamiseks sellel kuvatava pildivaate koordinaatsüsteemi, mis võib erineda ekraani omast. Komponendil puudub visuaalne representatsioon.

3.3. Loodud lahendus

Kõik Apple'i seadmetele loodavad rakendused peaksid Apple'i soovi kohaselt järgima *Model-View-Controller* [37] (edasi MVC) mustrit. MVC mustrist on küll võimalik mööda hiilida, kuid Make It White on samuti just sellele mustrile vastavalt loodud. Lisaks erinevatele MVC mustri poolt oodatavatele objektidele on rakenduses vaid tehnoloogiliselt nõutav AppDelegate [38] ning erinevaid staatilisi abilisi hoidev klass.

Vastavalt Objective-C tavale on kõik klassid prefikseeritud paketi prefiksiga "MIW".

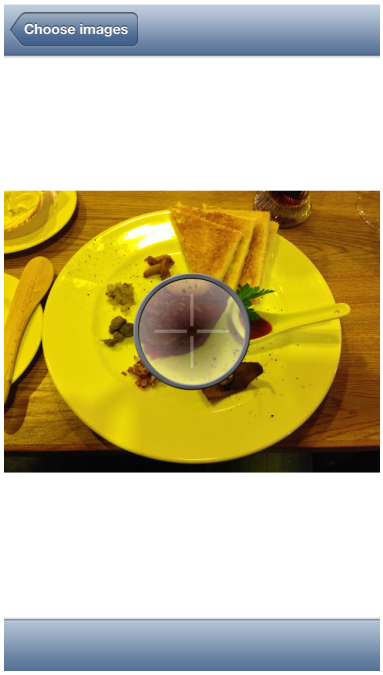
Rakenduse tehnilisest lahendusest käsitletakse kahte selle keerulisemat osa:

- MIWLoupeView - vaade subjektiivselt valge koha märkimiseks
- Valge tasakaalu muutmise tehniline lahendus

3.3.1. Luubi vaade

Luubi vaade MIWLoupeView on Make It White'i kõige keerulisem vaade, mille eesmärgiks on kuvada "läbi luubi" (st suurendatult) selle alla jäävat pildi osa. Seejuures käsitletakse MIWLopeView keskkohta subjektiivselt valge kohana. Samuti teeb luubi

vaade kättesaadavaks meetodi, mis tagastab luubi keskkohas oleva piksli värvuse MIWColor objektina. Luubi vaade on kujutatud joonisel 8.



Joonis 8 - Luubi vaade

Enamik kasutajaliidese programmeerimisele kulunud ajast kulust just selle vaate loomisele. Keeruliseks tegid selle vaate järgmised aspektid:

- erinevad koordinaatsüsteemid
- erinevad piltidega opereerimise raamistikud
- subjektiivselt valge koha värvuse määramine

3.3.1.1. Erinevad koordinaatsüsteemid

Luubi vaatesse kuvatakse suurendus piirkonnast, mille kohal luup pildil viibib. Selleks, et suurendus ei oleks pikseleeritud ning oleks originaalvärvides ka peale eelvaate loomist, ei tehta seda mitte ekraanitõmmisest (mis oleks olnud tehniliselt kõige lihtsam lahendus), vaid selle aluseks on koopia samast pildist, mida kasutajale esmalt MIWImageView's kuvatakse.

Luubi vaadet kuvatakse UIScrollView's oleva UIImageView kohale, mis tähendab, et luubi vaate koordinaadid on UIScrollView koordinaatsüsteemis. Lõikamaks välja luubi vaate aluseks olevast pildist sobiv piirkond, tuleb need esiteks teisendada UIImageView koordinaatsüsteemi, mis suumi kasutamise korral ei pruugi olla täielikult nähtav ja võib omada täiteääri, ning seejärel seal kuvatava pildi koordinaatsüsteemi, mis kattub luubi vaate aluseks oleva pildi koordinaatsüsteemiga.

Koordinaatsüsteemide vaheline konverteerimine osutus ootamatult töömahukaks tehniliseks ülesandeks ning probleemi aitas lõplikult lahendada alles UIImageView+GeometryConversion kategooria kasutuselevõtt. Konverteerimine tehakse kahes osas UIScrollView'st UIImageView'sse, võtte arvesse UIScrollView suumiskaalat ja nähtavat piirkonda, ning UIImageView'st selle all olevasse pilti, kasutades UIImageView+GeometryConversion kategooriat.

3.3.1.2. Erinevad piltidega opereerimise raamistikud

iOSi seadmetel on kasutusel kaks piltidega opereerimise raamistikku: kõrgtasemel olev objektorienteeritud UIKit [39] (millel baseerub kogu iOS UI) ning selle all asuv mitte-objektorienteeritud C raamistik Core Graphics [40]. Esialgsed katsetused Make It White'i ainult UIKitil baseeruvana luua jooksid liiva jõudlusprobleemide tõttu – luubi liigutamine, mille käigus genereeritakse jooksvalt suurendatud pilti, hakkas reaalsel riistvaral kasutatuna hakkima. Seetõttu tuli pildid luubi kuva joonistamise juures ühest raamistikust teise ja tagasi konverteerida. UIKitist väljumine tõi kaasa vajaduse luubi vaate programmaatiliselt joonistamise ja mälu halduse järele.

Konverteerimise tegi probleemseks aga see, et UIKit ainsa iOSi-spetsiifilise tehnoloogiana ja OS X-ist pärinev Core Graphics toimivad erinevates koordinaatsüsteemides – Core Graphicsi pildid on UIKiti koordinaatteljestikus vaikimisi horisontaaltelje suhtes peegelduses (st tagurpidi). Samuti tuli Core Graphicsit kasutades ümmarguse pildi saamiseks (sest luup on ümmargune) terve graafikakontekst ümmarguseks muuta. UIKitis oleks selle saanud ära teha pildimaskiga [41]. Nende sammude tegemise vajadusest aru saamine neelas arvestava hulga aega.

3.3.1.3. Subjektiivselt valge koha värvuse määramine

Luubi keskkohas asuva piksli värvus on sisendiks kogu järgnevale pilditöötlusele. Värvuse saamiseks joonistati luubi all olev piksel 1x1 suurusele graafikakontekstile, mille väljundiks oli ekraani asemel baidijada.

3.3.2. Valge tasakaalu muutmise tehniline lahendus

Käesolevat tööd planeerides oli töö idee baseeritud naiivsel RGB nihutamise tehnilisel lahendusel. Töö käigus aga selgus, et selle probleemi lahendamiseks on

tegelikult olemas ka Apple'i enda poolt pakutud lahendused, sest iOS 5-est alates on olemas tugi Core Image [42] filtritele. Nii nihkus töö fookus Core Image'i kasutamisele.

Põhjuseks fookuse nihutamisele oli soov liigsete tehnoloogiliste riskide vältimiseks kasutada võimalikult palju olemasolevaid tehnilisi lahendusi. Kuigi subjektiivselt valge piksli värvi hankimise algoritm oleks väga lihtsalt laiendatav naiivsele RGB nihutamisele, rääkisid selle kahjuks potentsiaalsed jõudlusprobleemid. Ühelt poolt oleks selles kohas olnud väga lihte mõnda mäluhalduslikku viga teha, teisalt on iOSi seadmetega võimalik pildistada panoraamfotosid, mille suurusel ei ole sisuliselt piire ning seega oleks tulnud väga hoolikalt mõelda efektiivsele mälukasutusele teisendusprotsessis, mis, olles küll väga huvitav probleemistik, oleks töö olemust liialt muutnud.

Tagasi vaadates oleks ilmselt eeldatavate jõudlusprobleemide lahendamise ajakulu olnud samas suurusjärgus Core Image'ga eksperimenteerimisele kulunud ajale. Peamine probleemide põhjustaja oli Apple'i väga napp dokumentatsioon [43], mis juhtis eksperimenteerimise käigus mitmetele tupikteedele. Teisalt oli see sama napp dokumentatsioon koos varem sõnastatud otsusega võimalikult palju olemasolevaid teeke kasutada selleks tõukejõuks, mis tingis vajaduse lähemalt tutvuda valge tasakaalu olemuse ning sellega seonduva teooriaga.

Eksperimenteerimise käigus prooviti järgmiseid filtreid:

- CIWhitePointAdjust [43]
- CIColorMatrix [43]
- CITemperatureAndTint [43]

Peale mitmeid katseid ei suutnud ei CIWhitePointAdjust, mis dokumentatsiooni järgi peaks olem valge punkti nihutamiseks, ega CIColorMatrix, mis dokumentatsiooni järgi peaks olema kõikide värvioperatsioonide südameks, soovitud tulemust anda.

Lõplik lahendus baseerub seega CITemperatureAndTint filtril. Filtri üheks sisendparameetriks olev värvitemperatuur tingis realiseeritud teisendused erinevate värviruumide vahel ning CCT arvutamise. Kuna esmased, ainult temperatuuril põhinenud katsed ei andnud rahuldavaid tulemusi, lisandus rakenduse kasutajaliidesesse liugur värvitooni muutmiseks, mis osutuski viimaseks piltmõistatuse puuduvaks tükiks.

Nii subjektiivselt valge koha värvitemperatuur kui ka kasutaja poolt valitud värvitonaalsus on kapseldatud MIWColor objekti ning edastatakse meetodile, mis genereerib nii eelvaate kui ka lõplikud transformeeritud pildid.

3.4. Võimalikud edasiarendused

Make It White'i loomise tingis võimalus ühendada n-ö meeldiv kasulikuga ehk luua AppStore'i sobiv rakendus, millest töö autor on ise puudust tundnud, ning kirjutada selle põhjal ka käesolev töö. Kui teine eesmärk on praeguseks täidetud, siis esimese eesmärgi soovitaval tasemel[6] saavutamiseks on vaja teha veel mõned sammud:

- Asendada standardsete Apple'i komponentide kujundus rakendusespetsiifilise kujundusega. Kujunduse kontseptsioon ning implementeerimisplaan on juba olemas.
- Animeerida erinevaid kasutajaliidese osi:
    - Luubi ekraanile ilmumine
    - Iga pildi töötlemise lõppedes selle eelvaate uuendamine
- Rakenduse taustale saatmise korral töötlemise lõppemise teate postitamine Notification Centerisse [44]
- Erinevate orientatsioonide tugi. Rakendus ise on võimeline tööks kõikides orientatsioonides, kuid ELCImagePicker mitte.
- Erinevate keelte tugi.

[6] Rakendus peaks olema piisavalt ahvatleva välimusega ja piisavalt võimalusterohke, et see leiaks võimalikult laialdase kasutajaskonna.

Kokkuvõte

Käesoleva töö aluseks loodi iOS 6-el baseeruv iPhone'i rakendus Make It White, mis laseb lihtsa vaevaga sarnases valgussituatsioonis tehtud fotode valge tasakaalu korrigeerida. Loodud rakendus koosneb 38 koodifailist ning 2522 koodireast ning selle loomisele kulus ligikaudu 100 tundi arendusaega.

Selgitamaks, miks on valge tasakaal digifotograafias problemaatiline, tutvustas töö selle olemust ning kirjeldas kaalutlusi, millel põhinedes sündis otsus Make It White just sellisena luua nagu ta loodud sai.

Seejärel andis töö põhjalikuma teoreetilise ülevaate valge tasakaaluga seonduvast värviteooria mõisteaparatuurist ja tutvustas töös kasutatud valemeid valge tasakaalu muutmiseks.

Töö praktilise osa ülevaates kirjeldati loodud tehnilist lahendust ning anti ülevaade selle keerukamatest ja enim aega nõudnud aspektidest.

Töö käigus valminud rakendus Make It White ei ole pelgalt selle töö jaoks loodud käeharjutus, vaid töö selle kallal jätkub ka peale antud kirjatöö valmimist, sest töö autor leiab rakendusel potentsiaali olevat.

Töö kirjutamisele eelselt püstitatud eesmärgid saavutati umbes 80% ulatuses, sest loodud rakendus on küll oma põhifunktsionaalsuselt valmis, kuid vajab veel lisaviimistlust enne AppStore'is publitseerimist.

Töö valmimise käigus avastas autor, et iOS platvormile rakenduste loomine on väga huvitav tegevus ning peale Make It White'i valmimist on plaanis juba järgmised rakendust ning nende arendamiseks on loodud startup firma.

Make It White – a white balance adjustment app for iPhone

Bachelor's thesis

Juhan Hion

Abstract

The goal of this thesis is to describe how and why a white balance adjustment app was created for iPhone.

To facilitate understanding why this kind of app is necessary, an overview of the white balance problem domain and a tour of currently existing solutions on iPhone are given. Based on these descriptions the philosophical choices made about what kind of app to develop are described and justified.

Since changing a photograph's white balance is not a trivial thing, just enough theory is introduced to provide the necessary mathematical background to white balance adjustments. Built upon this background, the chosen colour correction solution is described in step by step in theoretical terms.

Finally, a brief technical overview is provided to point out the more complex implementation problems solved during the development of the app.

The author of this thesis hopes that when Make It White is completed and published it will find buyers from Apple's AppStore.

Kasutatud kirjandus

[1] Wikipedia. *Color balance*
http://en.wikipedia.org/wiki/Color_balance (20.04.2013)

[2] Wikipedia. *App Store (iOS)*
http://en.wikipedia.org/wiki/App_Store_(iOS) (20.04.2013)

[3] Apple. *iOS 6*
http://www.apple.com/ios/ (28.04.2013)

[4] Nikon. *Mis on valge tasakaal?*
https://nikoneurope-et.custhelp.com/app/answers/detail/a_id/46300/~/mis-on-valge-tasakaal%3F (17.04.2013)

[5] Wikipedia. *Grey card*
http://en.wikipedia.org/wiki/Grey_card (17.04.2013)

[7] Smith, Joe 2009. *Calculating Color Temperature and Illuminance using the TAOS TCS3414C Digital Color Sensor*. Intelligent Opto Sensor Designer's Notebook 25:1-7
http://www.ams.com/eng/content/download/251586/993227/version/2 (17.04.2013)

[8] Wikipedia. *Absoluutselt must keha*
http://et.wikipedia.org/wiki/Absoluutselt_must_keha (17.04.2013)

[9] Wikipedia. *Planckian locus*
http://en.wikipedia.org/wiki/Planckian_locus (17.04.2013)

[10] Wikipedia. *RGB*
http://en.wikipedia.org/wiki/Rgb (17.04.2013)

[11] Wikipedia. *Color management*
http://en.wikipedia.org/wiki/Color_management (17.04.2013)

[12] Wikipedia. *CIE 1939 color space*
CIE 1939 color space http://en.wikipedia.org/wiki/CIE_1931_color_space (17.04.2013)

[13] Wikipedia. *International Commission on Illumination*
http://en.wikipedia.org/wiki/International_Commission_on_Illumination (17.04.2013)

[14] Wikipedia. *Kolvikesed*
http://et.wikipedia.org/wiki/Kolvikesed (17.04.2013)

[15] Wikipedia. *LMS color space*

http://en.wikipedia.org/wiki/LMS_color_space (17.04.2013)

[16] Wikipedia. *Luminosity function*

http://en.wikipedia.org/wiki/Luminosity_function (17.04.2013)

[17] Wikipedia. *CIE xy chromacity diagram and the CIE xyY color space*

http://en.wikipedia.org/wiki/XyY#CIE_xy_chromaticity_diagram_and_the_CIE_xyY_color_space (17.04.2013)

[18] Wikipedia. *Standard illuminant*

http://en.wikipedia.org/wiki/Standard_illuminant (17.04.2013)

[19] Wikipedia. *White balance*

http://en.wikipedia.org/wiki/White_balance (17.04.2013)

[20] Wikipedia. *Chromatic adaption*

http://en.wikipedia.org/wiki/Von_Kries_transform#Von_Kries_transform (17.04.2013)

[21] Wikipedia. *sRGB*

http://en.wikipedia.org/wiki/SRGB (17.04.2013)

[22] Lindbloom, Bruce 2012. *RGB to XYZ*

http://brucelindbloom.com/index.html?Eqn_RGB_to_XYZ.html (17.04.2013)

[23] Wikipedia. *Companding*

http://en.wikipedia.org/wiki/Companding (17.04.2013)

[24] Apple. *iOS Dev Center*

https://developer.apple.com/devcenter/ios/index.action (15.04.2013)

[25] Wikipedia. *Xcode*

http://en.wikipedia.org/wiki/Xcode (15.04.2013)

[26] Apple. *Testing and Debugging in iOS Simulator*

http://developer.apple.com/library/ios/#documentation/IDEs/Conceptual/iOS_Simulator_Guide/TestingontheiOSSimulator/TestingontheiOSSimulator.html (15.04.2013)

[27] Wikipedia. *Smalltalk*

http://en.wikipedia.org/wiki/Smalltalk (15.04.2013)

[28] Wikipedia. *Automatic Reference Counting*

http://en.wikipedia.org/wiki/Automatic_Reference_Counting (16.04.2013)

[29] Allan, Alasdair 2012. *Learning iOS Programming.* Sebastopol: O'Reilly

[30] Apple. *App Store Review Guidelines*

https://developer.apple.com/appstore/resources/approval/guidelines.html (19.04.2013, kättesaadav ainult registreeritud arendajatele)

[31] Apple. *UIImagePickerController Class Reference*

http://developer.apple.com/library/ios/#documentation/uikit/reference/UIImagePickerController_Class/UIImagePickerController/UIImagePickerController.html (19.04.2013)
[32] Apple *ALAsset Class Reference*
http://developer.apple.com/library/ios/#documentation/AssetsLibrary/Reference/ALAsset_Class/Reference/Reference.html (19.04.2013)
[33] Apple. *ALAssetLibrary Class Reference*
http://developer.apple.com/library/ios/#documentation/AssetsLibrary/Reference/ALAssetsLibrary_Class/Reference/Reference.html (19.04.2013)
[34] Wikipedia. *Singleton pattern*
http://en.wikipedia.org/wiki/Singleton_pattern (19.04.2013)
[35] Apple. *UICollectionView Class Reference*
http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UICollectionView_class/Reference/Reference.html (19.04.2013)
[36] Apple. *Supporting Multiple Device Orientations*
http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/RespondingtoDeviceOrientationChanges/RespondingtoDeviceOrientationChanges.html (19.04.2013)
[37] Apple. *Model-View-Controller*
http://developer.apple.com/library/ios/#documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html (19.04.2013)
[38] Apple. *Core App Objects*
http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/AppArchitecture/AppArchitecture.html (19.04.2013)
[39] Apple. *UIKit Framework Reference*
http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit_Framework/_index.html (19.04.2013)
[40] Apple. *Core Graphics Framework Reference*
http://developer.apple.com/library/ios/#documentation/CoreGraphics/Reference/CoreGraphics_Framework/_index.html (19.04.2013)
[41] Wikipedia. *Mask (computing)*
http://en.wikipedia.org/wiki/Image_mask#Image_masks (19.04.2013)
[42] Apple. *About Core Image*
https://developer.apple.com/library/mac/#documentation/graphicsimaging/Conceptual/CoreImaging/ci_intro/ci_intro.html (20.04.2013)

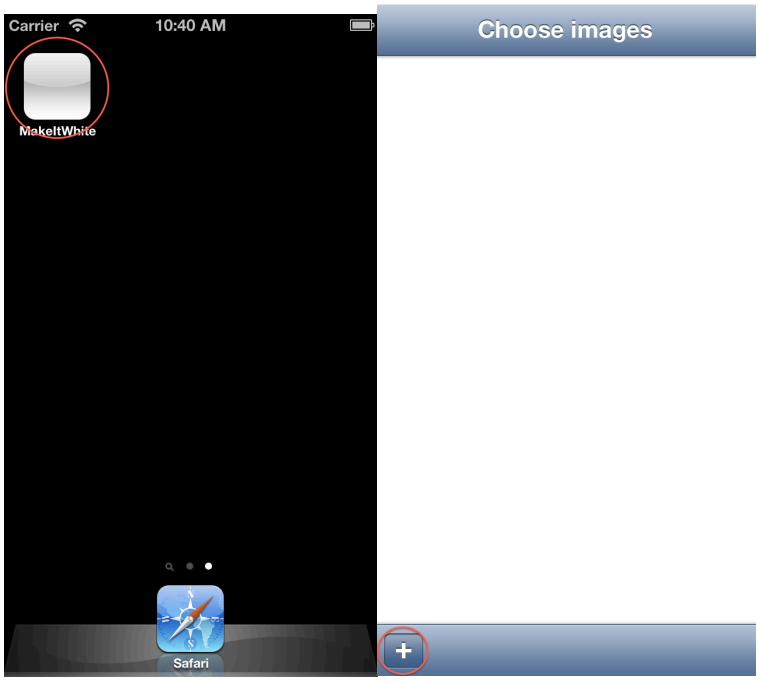
[43] Apple. *Core Image Filter Reference*
https://developer.apple.com/library/mac/#documentation/graphicsimaging/reference/CoreImageFilterReference/Reference/reference.html (20.04.2013)
[44] Wikipedia. *Notification Center*
http://en.wikipedia.org/wiki/Notification_Center (20.04.2013)

Lisa 1 – Make It White ekraanipildid ning kasutusjuhend

Make It White on kasutatav iOS6-te jooksutavatel Apple'i seadmetel. Seda on testitud järgmiste seadmetega:

- iPhone 5
- iPhone 4

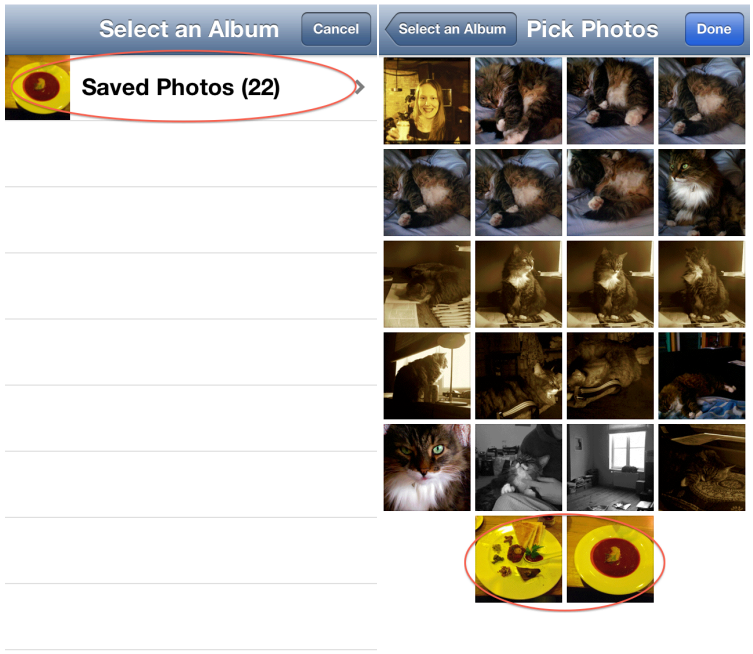
1. Rakenduse käivitamiseks tuleb puudutada seadme ekraanil asuvat MakeItWhite'i ikooni (joonis 1).
2. See järel kuvatakse tühi töödeldavate piltide loend. Töödeldavate piltide lisamiseks tuleb puudutada "+"-nuppu (joonis 1).



Joonis 1 - Rakenduse käivitamine ja töödeldavate piltide valimise alustamine

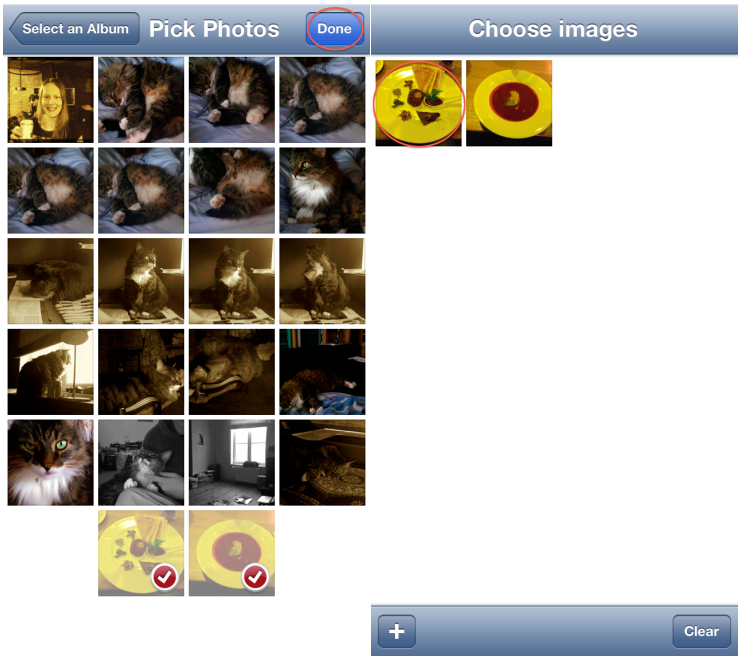
3. Piltide lisamiseks tuleb esmalt valida soovitav album ning seda puudutada (joonis 2).
    - Vajadusel saab toimingu "Cancel"-nuppu puudutades katkestada.

4. Rakendus kuvab albumis olevate piltide eelvaated nii, et esmalt kuvatakse kõige värskemad pildid (joonis 2). Valimaks pilte, mida töödelda, tuleb nende eelvaateid puudutada.
    - “Select an Album”-nuppu puudutades on võimalik minna tagasi albumi valiku kuvale.
    - “Done”-nuppu puudutades tegevus lõpetatakse ning kasutaja viiakse tagasi sammule 2 (joonis 3).



Joonis 2 - Albumi valik ja selle piltide loend

5. Olles soovitud pildid välja valinud, viib “Done”-nupu puudutamine kasutaja töödeldavate piltide kuvale (joonis 3).
6. Töödeldavate piltide kuval saab töötlemiseks valitud piltidest hulgast eelvaadet puudutades valida ühe piltidest subjektiivselt valge koha määramiseks (joonis 3).
    - “+”-nupu puudutamine tühjendab tehtud valiku ning viib kasutaja tagasi sammule 3 (joonis 2).
    - “Clear”-nupu puudutamine tühjendab tehtud valiku ning kasutaja on tagasi sammule 2 (joonis 1).

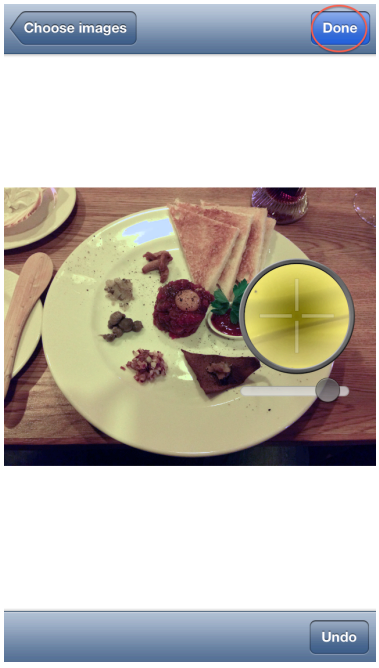


Joonis 3 - Pildivaliku lõpetamine ja valge koha otsimise alustamine

7. Subjektiivselt valget kohta saab otsida pildil olevat luupi nihutades. Valgena käsitletakse punkti, mis on täpselt luubi keskel (joonis 4).
8. Lohistamise lõppedes genereerib rakendus pildilt valitud koha järgi eelvaate ning kuvab luubi alla liuguri täppiskorrigeerimiseks (joonis 4).
    - "Done"-nuppu puudutades viiakse kasutaja sammule 10 ning alustatakse piltide töötlemist ja salvestamist.
    - "Undo"-nuppu puudutades kuvatakse pilt uuesti originaalvärvides.

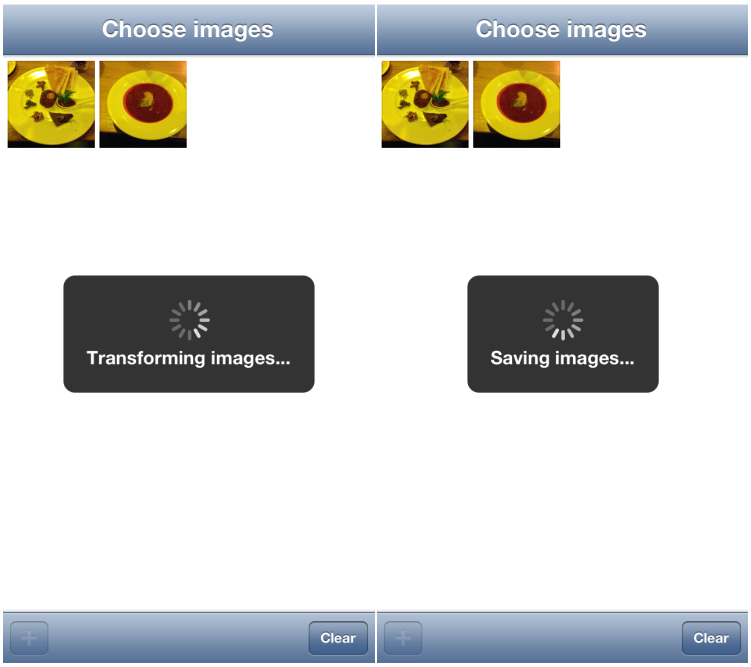
Joonis 4 - Subjektiivselt valge koha otsimine

9. Liuguri liigutamise lõppedes genereerib rakendus uue eelvaate (joonis 5). “Done”-nuppu puudutades alustatakse piltide töötlemist ja salvestamist.



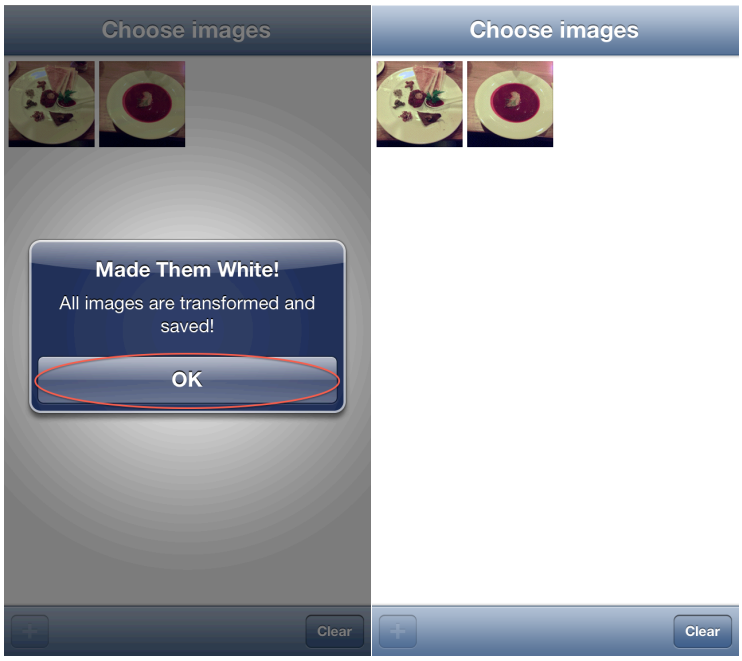
Joonis 5 - Töötlusprotsessi käivitamine

10. Piltide töötlemise käigus ei ole rakenduse kasutajaliides puudutustele reageeriv ning kuvab lisaks protsessi toimumise indikaatorile ka staatus-sõnumeid (joonis 6).



Joonis 6 - Protsessiindikaator ja staatussõnumid

11. Salvestamise lõppemisest teavitatakse kasutajat hüpikaknaga (joonis 7).
12. Hüpikakna sulgemise järel uuendatakse töötlemiseks valitud piltide eelvaateid väljendamaks nende muutunud värve. Sellega on rakenduse töö selle pildikomplektiga lõppenud (joonis 7).
    - "Clear"-nuppu puudutades saab kasutaja alustada järgmise pildikomplekti töötlemist ning ta suunatakse tagasi sammule 2 (joonis 1).



Joonis 7 - Töötlusprotsessi lõpp

Lisa 2 – Loodud rakenduse lähtekood

<lisatud digitaalselt>

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Juhan Hion**
(sünnikuupäev: 4.02.1978)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
MakeItWhite – valgetasakaalu rakendus iOSile
mille juhendaja on **Margus Niitsoo**

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **07.05.2013**