VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POŘIZOVÁNÍ VYSOCE KVALITNÍCH SNÍMKŮ ROVINNÝCH POVRCHŮ CHYTRÝM TELEFONEM

CAPTURING VERY HIGH QUALITY IMAGES OF PLANAR SURFACES BY A SMARTPHONE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VÍT ŠEBELA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Šebela Vít**
Program: Informační technologie
Název: **Pořizování vysoce kvalitních snímků rovinných povrchů chytrým telefonem**
Capturing Very High Quality Images of Planar Surfaces by a Smartphone
Kategorie: Zpracování obrazu

Zadání:

1. Nastudujte základy tvorby mobilních aplikací, zaměřte se na pořizování obrazu vestavěnou kamerou.
2. Nastudujte problematiku zpracování obrazu, zaměřte se na registraci/zarovnání rovinných obrázků a fúzi obrázků do obrázku s vysokým rozlišením (superresolution).
3. Vytvořte jednoduchou aplikaci pro smartphone, která umožní pořizovat obrázky do datové sady pro vývoj algoritmů.
4. Pořiďte datovou sadu pro vývoj algoritmů; průběžně ji rozšiřujte.
5. Experimentujte s algoritmy zpracování obrazu nad pořízenými daty, vyviňte použitelné (optimální) řešení.
6. Integrujte vyvinuté algoritmy do mobilní aplikace pro pořizování vysoce kvalitních obrázků.
7. Iterativně vylepšujte vytvořenou aplikaci pro dosažení dobré uživatelské použitelnosti a pro maximální kvalitu výstupů.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Android Developers: https://developer.android.com/index.html
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této práce je navrhnout a vytvořit mobilní aplikaci pro platformu iOS, která umožní pořízení vysoce kvalitní fotografie rovinné plochy. Aplikace se tak může hodit do nejrůznějších případů užití, kde je za potřebí vysoká kvalita fotografie, ať už je to vyfotografování smlouvy, vzoru dřeva nebo obalu knihy. Hlavní stavební kámen aplikace je sejmutí sady snímků plochy, které se použijí k výpočtu výsledné fotografie pomocí vyvinutého algoritmu. Vlastní výpočet probíhá za pomocí knihovny OpenCV. K zarovnání pořízených snímků se používá homografie, kvalita výsledného obrázku je pak dosažena kombinací technik průměrování snímků, zvýšení rozlišení a rekonstrukcí detailů ve fotografii pomocí Lanczosovy interpolace a zaostření pomocí techniky Unsharp mask.

Abstract

The main goal of this bachelor's thesis is to design and create mobile app for iOS platform, which will allow the capture of high quality image of planar surface. Application can be used in lots of use cases where high quality of captured image is needed, like capturing photo of contract, wood pattern or book cover. Main building block of the application is to capture set of photos of planar surface, which is then used for calculation of the resulting high quality image. Calculation is performed using OpenCV library. Planar homography is used for image alignment, the quality of the resulting image is then achieved by combination of image averaging techniques, increase of resolution and reconstruction of details using Lanczos interpolation and image sharpening using Unsharp mask technique.

Klíčová slova

Planární Homografie, Klíčové body, Zpracování obrazu, interpolace Nejbližší soused, Lanczosova interpolace, mobilní aplikace, iOS, iPhone, Zaostření obrazu

Keywords

Planar Homography, Keypoints, Image Processing, Nearest Neighbor interpolation, Lanczos Interpolation, mobile app, iOS, iPhone, Image Sharpening

Citace

ŠEBELA, Vít. *Pořizování vysoce kvalitních snímků rovinných povrchů chytrým telefonem.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Pořizování vysoce kvalitních snímků rovinných povrchů chytrým telefonem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci na téma Pořizování vysoce kvalitních snímků rovinných povrchů chytrým telefonem vypracoval samostatně pod vedením pana profesora Herouta. Všechna odborná literatura, ze které jsem čerpal, je uvedena v seznamu literatury.

.....
Vít Šebela
10. května 2021

Poděkování

Chtěl bych poděkovat panu prof. Ing. Adamu Heroutovi, Ph.D. za jeho aktivní pomoc při řešení problémů, za pravidelné konzultace, poskytnuté rady díky jeho dlouholetým zkušenostem a motivující přístup.

Obsah

1 Úvod **2**

2 Mobilní aplikace na iOS **3**
2.1 Existující řešení . . . 3
2.2 Vývoj na iOS . . . 4
2.3 Model-View-Controller ve vývoji pro iOS . . . 6
2.4 UIKit pro tvorbu aplikací . . . 7
2.5 MVVM a SwiftUI . . . 8
2.6 Návrh a tvorba uživatelského rozhraní na iOS . . . 8

3 Zvýšení kvality obrazu **10**
3.1 Knihovna OpenCV . . . 10
3.2 Neuronové sítě pro super-rozlišení a jejich nevýhody . . . 10
3.3 Rekonstrukce scény z více obrázků . . . 12
3.4 Zarovnání snímků . . . 12
3.5 Sloučení několika snímků do jednoho . . . 16
3.6 Snížení šumu v obraze . . . 18
3.7 Techniky zvýšení rozlišení obrazu . . . 21
3.8 Zaostření obrazu . . . 23

4 Návrh a tvorba aplikace pro snímání povrchů **25**
4.1 Jednoduchá mobilní aplikace pro sejmutí sady fotek pro vývoj algoritmu . . 25
4.2 Vývoj a testování algoritmu pro zvýšení kvality obrázku . . . 26
4.3 Návrh uživatelského rozhraní . . . 29
4.4 Implementace aplikace pro zvýšení kvality snímku . . . 30

5 Uživatelské testování a zhodnocení výsledků aplikace **35**
5.1 Uživatelské testování . . . 35
5.2 Zhodnocení výsledků aplikace . . . 36

6 Závěr **37**

Literatura **38**

Kapitola 1

Úvod

Tento text slouží jako technická zpráva k tvorbě aplikace HighQualityPhoto pro mobilní zařízení s operačním systémem iOS od firmy Apple. Zaměřením této mobilní aplikace je jejím uživatelům poskytnout možnost získat vysoce kvalitní fotografii rovinné plochy, kterou by nezískali výchozí aplikací pro fotografování integrované do systému iOS. Aplikace na základě více pořízených snímků, se kterými provádí potřebné výpočty, poskytne fotografii ve vyšším rozlišení, s potlačeným šumem, rekonstruovanými detaily, odstraněnými odlesky a zaostřenými hranami. Výsledná fotografie je potom vysoce kvalitní obdobou fotografie pořízené pomocí výchozí aplikace systému iOS pro pořizování snímků.

Práce obsahuje úvod do problematiky vývoje mobilních aplikací pro operační systém iOS od firmy Apple, popis potřebné teorie pro zpracování obrazu a jeho úpravu k získání co nejkvalitnější fotografie, popis a implementaci výsledné aplikace a výstupy uživatelského testování.

V aplikaci byly za účelem zarovnání fotografií použity techniky registrace a navigace v obrazu, dále pro odstranění nebo potlačení šumu ve fotografii byly použity techniky vyhlazování (rozmazání) obrazu. Pro sloučení více pořízených snímků rovinné plochy byly použity techniky sloučení snímků za účelem rekonstrukce detailů a jejich zvýraznění. A pro zviditelnění hran a detailů byly použity techniky zaostření obrazu.

Hlavním cílem bylo poskytnout uživatelům jednoduchou aplikaci, kterou by mohli používat na denní bázi v nejrůznějších případech užití, kdy potřebují kvalitní fotografii snímané plochy a nepotřebují k tomu zrcadlový fotoaparát nebo mobilní telefon z vyšší třídy s dvěma a více kamerami. Ohled byl při vývoji brán také na rychlost zpracování při zachování co nejvyšší kvality výsledného snímku po zpracování.

Cíle práce se podařily dosáhnout a výsledkem je aplikace, která obsahuje funkční implementaci algoritmu pro zvýšení kvality snímků, obsahuje přívětivé a jednoduché uživatelské rozhraní a poskytuje dobré výsledky.

Kapitola 2

Mobilní aplikace na iOS

Volba mobilní platformy, na které je aplikace vyvíjena, se odvíjela podle nejpoužívanějších platforem. Nedávalo by smysl vytvářet aplikaci pro mobilní platformu, kterou nikdo nepoužívá. Z dostupných *statistik*[1], které se zabývají používáním operačních systémů na mobilní platformě, se dá v dnešní době mluvit jen o platformách Android a iOS, kde Android má téměř 72% zastoupení a iOS je zbylých 28 %.

V dnešní době existuje spoustu frameworků umožňující vývoj mobilních aplikací pro obě platformy. Zástupci jsou např. *React Native*[2] nebo *Flutter*[3]. Tyto frameworky se označují jako *hybridní*, protože běží nad nativním kódem mobilních platforem. Kód, který se v těchto frameworcích píše, se z nich překládá do nativního kódu platformy pomocí integrovaného překladače. Tento způsob vývoje se zdá jako efektivní a ekonomický, protože je třeba psát aplikaci jen v jednom kódu a aplikace bude dostupná pro obě platformy. Pro základní funkcionalitu a dostupnost nejpoužívanějších API je tento přístup vývoje dostačující, ale pro získání přístupu k hardwarovým modulům mobilní platformy (kamera, mikrofon, apod.) je třeba vyvíjet aplikaci nativně. Rozhodl jsem se pro vývoj aplikace pro platformu iOS, protože jsem letitý uživatel a vždy mě zajímal vývoj pro iOS a tato práce byla ideální příležitost pro rozšíření znalostí o vývoji pro tento systém. Aplikaci pro platformu Android mám v plánu v budoucnu vytvořit a rozšířit si tak znalosti o vývoji pro jinou platformu než jen iOS.

2.1 Existující řešení

Za existující řešení v podobě mobilních aplikací se zaměřením na vyšší kvalitu snímku se považují mobilní aplikace specializované na úpravu fotografií a aplikace pro prohlížení fotografií integrované přímo do systému iOS, které poskytují základní možnosti úpravy fotek, ale nemají implementované složitější operace a výsledek tak není tak kvalitní jako v případě specializovaných aplikací.

Mobilní aplikace poskytující velice široké možnosti pro úpravu fotek je *Lightroom Photo Editor*[4] od firmy Adobe. Tato aplikace uživatelům poskytuje možnosti úprav obrazu na základě letitých zkušeností v oboru firmy Adobe, která se specializuje v oblasti zpracování obrazu od 90. let minulého století. Nevýhodou aplikace je, že základní úpravy fotografií jsou obsaženy v bezplatné verzi aplikace, ale pokročilejší techniky jsou pouze v placené verzi

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] https://reactnative.dev/
[3] https://flutter.dev/
[4] https://apps.apple.com/cz/app/adobe-lightroom-photo-editor/id878783582?l=cs

aplikace. Další nevýhodou může být náročnější uživatelské prostředí, kdy se předpokládá, že uživatel aplikace již zná názvy technik zpracování obrazu a jejich efekty ve zpracování fotografií a není tedy pro úplné začátečníky.

Další mobilní aplikací pro úpravu fotografií, která poskytuje velice široké možnosti pro úpravu fotografií, je *Snapseed*[5] od firmy Google. Aplikace poskytuje 29 nástrojů pro úpravu fotografií, poskytuje práci se soubory typu JPG a RAW, poskytuje filtry pro zabarvení fotografie, úpravu do vzhledu retro, apod. Aplikace ale nepodporuje práci s fotkami proprietárního typu HEIC firmy Apple. Aplikace také po testování pro účely této práce vykazuje chyby např. při ukládání upraveného snímku, kdy je uložen původní neupravený, aplikace se opakovaně bez upozornění sama ukončila, někdy se zasekla a potřebovala restart, apod. Tato práce by měla nedostatky aplikace Snapseed eliminovat a uživatelům poskytnou univerzální, rychlé řešení pro snímání velice kvalitních fotek bez dlouhých úprav v prostředí aplikace.

2.2 Vývoj na iOS

Firma Apple je hlavním vývojářem všech prostředků potřebných pro vývoj aplikací na platformu iOS od vlastního programovacího jazyku po vývojové prostředí používané k psaní aplikací. K vývoji se v dnešní době používá v největším zastoupení programovací jazyk *Swift*[6], který firma Apple vyvíjí. Dříve se používal programovací jazyk *Objective-C*[7], ale v roce 2014 jej pomalu začal nahrazovat Swift díky jednoduchosti syntaxe, vysoké vyjadřovací schopnosti jazyka a rychlosti běhu. Objective-C se v dnešní době používá pro vývoj pořád, ale většina v něm napsaných frameworků se pomalu migruje a přepisuje na Swift. Je tedy vhodné jazyku alespoň rozumět a umět ho číst pro dobrou orientaci při vývoji.

Pro vývoj aplikací nativně na platformu iOS je třeba mít k dispozici počítač s operačním systémem OS X a naistalovaným vývojovým prostředím Xcode od firmy Apple. To je považováno částí vývojářů jiných platforem za velkou nevýhodu, protože počítače od firmy Apple jsou podstatně dražší.

2.2.1 Vývojové prostředí Xcode

Xcode[8] je vývojové prostředí vyvíjené firmou Apple. Obsahuje balíčky a integrované moduly pro vývoj pro operačních systémů iOS, ipadOS, macOS, watchOS a tvOS. Xcode obsahuje interaktivní rozsáhlou dokumentaci programovacích jazyků Swift a Objective-C, ve které jsou popsány části jazyků, doporučení pro psaní v jazycích, nejpoužívanější konstrukce apod. Je tedy na denní pořádku její využívání.

Xcode podporuje nejen jazyky Swift a Objective-C, ale obsahuje podporu i pro známé jazyky C, C++, Objective-C++, Java, Python, Ruby, AppleScript a Rez. Je tedy univerzálním vývojovým prostředím určeným pro vývoj nejrůznějších projektů v různých programovacích jazycích.

Pro testování funkčnosti aplikace je k dispozici funkce simulátoru iOS, která virtualizuje systém iOS běžící na virtuálním stroji. Vývoj je tedy možný i bez fyzického zařízení, pokud není potřeba využívat hardwarové moduly fyzického mobilního telefonu nebo jiného zařízení.

[5] https://apps.apple.com/us/app/snapseed/id439438619

[6] https://developer.apple.com/swift/

[7] https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[8] https://developer.apple.com/xcode/

Obrázek 2.1: Ukázka vývojového prostředí Xcode s puštěným simulátorem iOS.

2.2.2 Programovací jazyk Swift

Programovací jazyk Swift je multi-paradigmatický jazyk vyvíjený firmou Apple a uvedený v roce 2014 jako open-source alternativa k Objective-C. Hlavním cílem při vývoji jazyka Swift bylo omezení chyb programátora a režie na udržení paměti bez úniků oproti jazyku Objective-C. Programovací jazyk Swift je kompilován souborem nástrojů *LLVM*[9].

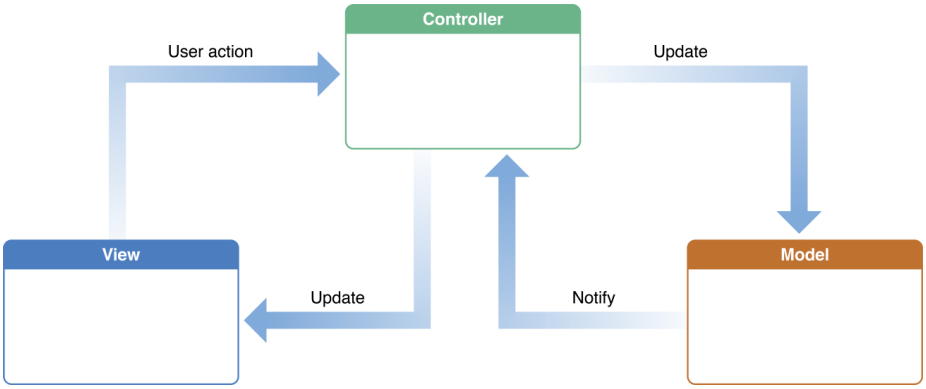
Oproti jazyku Objective-C má Swift vyšší vyjadřovací sílu a menší prostor pro chyby programátora, následující kód ukazuje sémanticky stejný kód s definicí a konkatenací proměnné typu String s jiným řetězcem.

```
// Kod v Objective-C
NSString *str = @"Hello,";
str = [str stringByAppendingString:@" world!"];

// Kod se stejnou semantikou zapsany v programovacim jazyku Swift
var str = "Hello,"
str += " world!"
```

Programovací jazyk Swift využívá stejný runtime pro Objective-C na systémech macOS, iOS a jeden kód napsaný v jazyku Swift může být spuštěn na více zařízeních bez větších problémů. Swift se stává hlavním programovacím jazykem v dnešní komunitě vývojářů pro platformy od firmy Apple a postupně plní cíl nahradit jazyk Objective-C pro vývoj mobilních aplikací. V této práci byl využit programovací jazyk Swift v kombinaci s Objective-C, Objective-C++ a C++. Použití více jazyků bude popsáno v dalších kapitolách této práce.

[9] https://llvm.org/



Obrázek 2.2: Blokové schéma návrhového vzoru Model-View-Controller. Bloky představují části aplikace, které mají podle návrhového vzoru rozdělené funkce. Převzato z [3].

2.3 Model-View-Controller ve vývoji pro iOS

Model-View-Controller (MVC) [3] je návrhový vzor používaný při vývoji počítačových aplikací. Firma Apple návrhový vzor Model-View-Controller využívá v sadě frameworků zvaných *Cocoa* [6], které zajišťují prostředí pro chod aplikací vyvíjených pro platformy iOS a OS X. Návrhový vzor definuje tři role objektů aplikace: Model, View a Controller. Návrhový vzor zároveň definuje způsob, jakým spolu jednotlivé objekty aplikace s přiřazenými rolemi komunikují. Jednotlivé role jsou od sebe odděleny abstraktními hranicemi danými povahou role a přes tyto hranice spolu komunikují bez znalosti jejich implementace. Využití návrhového vzoru MVC při vývoji přináší benefity v podobě lepší znovupoužitelnosti objektů aplikace, lépe definovaných rozhraní těchto objektů a lepší rozšířitelnost aplikace.

Model

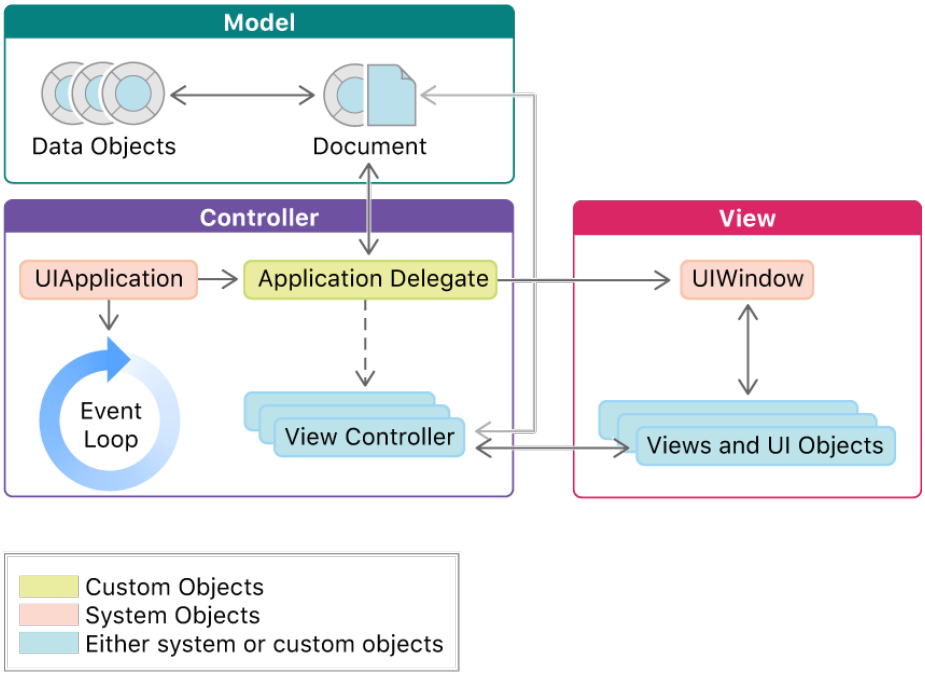
Model je část návrhového vzoru, která má za úkol správu dat aplikace. Model data aplikace zapouzdřuje a poskytuje logiku, která zajišťuje bezpečnou správu pro zapisování, čtení a změny v datech jiným částem aplikace. Data uložená a zapouzdřená v modelu určují ve velké části stav aplikace kroky, které aplikace je schopna ze stavu provést. V rámci této práce Model může představovat databázi obrázků vyfocených z kamery a poskytovat ostatním částem aplikace manipulaci s uloženými obrázky.

View

Část View návrhového vzoru MVC je část, se kterou uživatelé interagují a kterou vidí na obrazovce svého zařízení. Objekty typu View v sobě mají zapouzdřenou logiku pro vykreslení na obrazovku a pro interakci s částmi uživatelského rozhraní uživatelem. Hlavní zodpovědnost objektů typu View je zobrazovat data z objektu typu Model a informovat uživatele o jeho stavu, který přes View mohou měnit.

Controller

Objekty typu Controller spojují objekty View a Model logikou, která zajišťuje synchronizaci dat mezi nimi. Skrz objekt typu Controller se objekt View dozví o změně v objektu Model a překreslí tak obrazovku, aby odpovídala stavu objektu Model. Tento proces fun-



Obrázek 2.3: Běžná struktura aplikace psané ve frameworku UKit využívající návrhových vzorů Model-View-Controller a delegace [5]

guje i obráceným směrem, kdy objekt View od uživatele získá data, která mají přepsat data v objektu Model a objekt Controller tento proces zprostředkuje. Objekty typu Controller mohou také kontrolovat, začínat a ukončovat životní cyklus všech ostatních částí aplikace. Objekt Controller tedy interpretuje uživatelovi akce v objektech View a komunikuje změny s objekty typu Model a obráceně.

2.4 UIKit pro tvorbu aplikací

UIKit [5] je nejrozšířenější framework pro tvorbu aplikací pro platformy od firmy Apple. Poskytuje vývojářům potřebnou infrastrukturu a základ pro tvorbu uživatelských rozhraní, zpracování a správu událostí aplikace pro multi-touch události z displeje nebo zpracování interakcí s jádrem systému. Framework UIKit byl vyvíjen podle návrhového vzoru Model-View-Controller, popsaného v sekci 2.3. Framework UIKit je součástí vývojového prostředí aplikací *Cocoa (Touch)*[10] pro tvorbu aplikací pro platformy iOS a OS X.

UIKit se stará automaticky o životní cyklus objektů typu View, správu animací jednotlivých objektů v uživatelském rozhraní, společně s frameworkem *Foundation* [2] poskytuje rozhraní pro přístup k hardwarovým modulům zařízení, bezpečnost a zapouzdření uživatelských dat a spravuje a přiděluje zdroje zařízení podle potřeby. UIKit byl v této práci použit z důvodu stability frameworku, podporovaných nízko úrovňových akcí s hardwarovými moduly zařízení. Struktura frameworku UIKit se řídí podle návrhového vzoru Model-View-Controller (MVC), podle kterého jsou rozděleny funkční části aplikace a je jim přiřazen účel. Na obrázku 2.3 je ukázána klasická struktura aplikace psaná ve frameworku UIKit řídící se podle návrhového vzoru Model-View-Controller.

[10] https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html

Framework UIKit využívá ve svojí struktuře návrhového vzoru *delegace* [1]. Návrhový vzor definuje strukturu, kdy si objekty drží referenci na jiný objekt (delegáta) a na základě této reference si navzájem posílají zprávy ve vhodný čas. Delegát může na zprávu reagovat překreslením vzhledu, změnou svého stavu nebo jiných objektů aplikace, případně vrací hodnotu ovlivňující způsob zpracování přicházející události. Hlavní přínos návrhového vzoru delegace je snadné přizpůsobení chování několika objektů v jednom centrálním objektu.

Framework UIKit byl použit pro vývoj této aplikace jako stabilní a dlouho používaný základ většiny mobilních aplikací vyvíjených pro platformu iOS. Další důvod, proč byl zvolen framework UIKit, byl jeho naučení v rámci mého budoucího zaměření na vývoj mobilních aplikací.

2.5 MVVM a SwiftUI

SwiftUI [4] je nejnovější balík nástrojů pro tvorbu aplikací pro platformy od firmy Apple představený v roce 2019 podporující deklarativní způsob tvorby uživatelských rozhraní. SwiftUI využívá návrhového vzoru Model-View-ViewModel [16] narozdíl od frameworku UIKit využívající Model-View-Controller. SwiftUI je nativní pro všechny platformy a to iOS, MacOS, ipadOS, watchOS a tvOS. Framework SwiftUI by měl tedy nahradit framework UIKit pro svůj deklarativní způsob tvorby uživatelského rozhraní, jednodušší přenositelnost mezi platformami a rychlejší, intuitivnější tvorbu aplikací. V této práci ale nebyl framework i přes své výhody použit, protože obsahuje spoustu chyb a nedávalo mi smysl se učit novější framework pro vývoj aplikací než UIKit, který je považovaný za dlouhodobý základ a jeho znalost je klíčová.

Framework SwiftUI tedy využívá návrhový vzor Model-View-ViewModel vytvořený firmou Microsoft v roce 2005 s cílem oddělit objekty typu View od jakékoliv závislosti na podobě objektu Model. Na obrázku 2.4 je ukázáno blokové schéma architektury návrhového vzoru MVVM ilustrující vztahy mezi objekty a posílání zpráv při událostech mezi nimi. Objekty typu View by měly být tedy plně nezávislé na specifických implementacích objektů typu Model oproti návrhovému vzoru MVC, ve kterém jsou objekty typu View a Model závislé na konkrétní implementaci. Část návrhového vzoru MVVM View je zodpovědná za definici struktury, rozložení a vzhledu uživatelského rozhraní. Část ViewModel implementuje vazby mezi daty a uživatelským rozhraní a je zodpovědná za správnou synchronizaci zobrazení správných dat, změny stavu aplikace prostřednictvím událostí upozorňujících na změnu dat, apod. Část Model zapoudřuje data aplikace společně s logikou pro jejich správu.

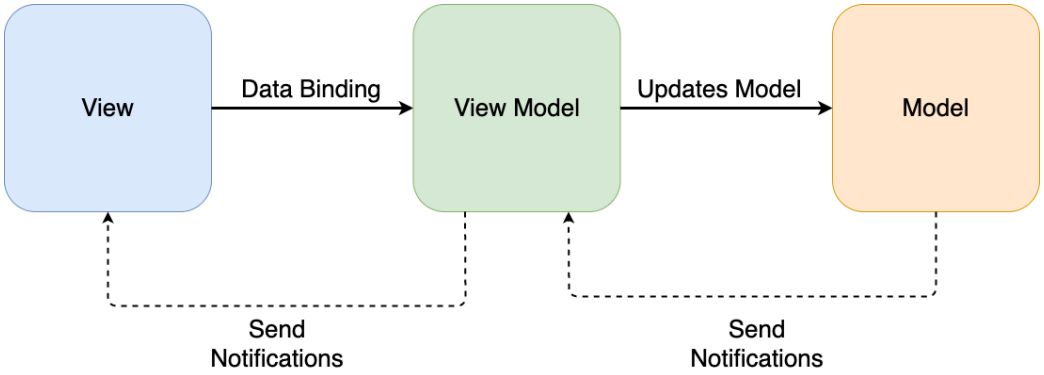
2.6 Návrh a tvorba uživatelského rozhraní na iOS

Tvorba uživatelských rozhraní pro mobilní aplikace na platformu iOS pomocí frameworku UIKit lze provádět dvěma hlavními způsoby:

- Pomocí nástroje *Interface Builder*[11] integrovaného do IDE Xcode,
- Zápisem přímo v kódu aplikace.

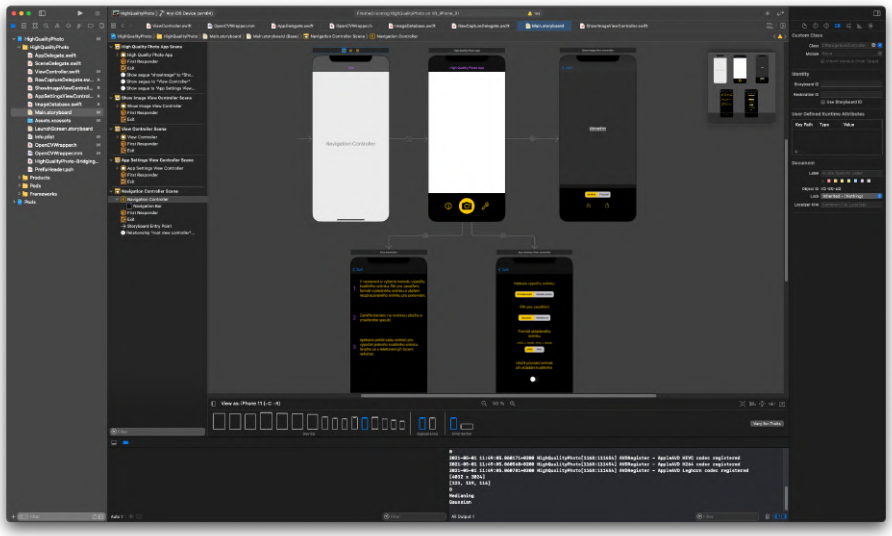
Výše uvedené přístupy k tvorbě uživatelského rozhraní mohou být libovolně kombinovány, např. logika uživatelského rozhraní se může psát v kódu aplikace a vzhled definovat pouze v nástroji Interface Builder.

[11] https://developer.apple.com/xcode/interface-builder/



Obrázek 2.4: Blokové schéma návrhového vzoru Model-View-ViewModel [16].

Nástroj *Interface Builder* poskytuje jednoduchý a rychlý způsob interaktivní tvorby uživatelského rozhraní. Uživatelské rozhraní aplikace lze tvořit pomocí „drag and drop“ interaktivního způsobu, kdy se z knihovny prvků uživatelského rozhraní vybere objekt, který se stane součástí aplikace viz obrázek 2.5. Aplikace Interface Builder poskytuje možnost tvorby rozhraní nejen pro platformu iOS, ale i macOS. Uživatelská rozhraní jsou ukládána do souborů zvaných *storyboards*, ve kterých jsou mapovány vztahy mezi jednotlivými prvky a více pohledy. Interface Builder využívá návrhu MVC a jeho použití v Cocoa Touch prostředí pro tvorbu aplikací. Umožňuje tak jednoduše oddělit uživatelské rozhraní od mechanismů a dat aplikace. Pro vývoj aplikací pro různé modely mobilních telefonů iPhone používá aplikace Interface Builder vestavěný systém rozložení prvků *Auto Layout*, pomocí kterého se definují omezení kladená na uživatelská rozhraní aplikací pro kompatibilitu mezi velikostmi obrazovek zařízení. Vyvíjená aplikace tak může být jednoduše použitelná na více zařízeních právě díky těmto omezením.



Obrázek 2.5: Ukázka aplikace Interface Builder integrované do prostředí Xcode.

Kapitola 3

Zvýšení kvality obrazu

Základním předpokladem pro digitální zpracování obrazu a jeho úprava k vyšší kvalitě je jeho snímání. V případě mobilních telefonů, které v dnešní době disponují ve většině případů kvalitní kamerou, není problém získat kvalitní digitální obraz. Digitální obraz je tedy snímek pořízený kamerou mobilního telefonu a je definován jako obrazová funkce $f(x, y)$, která má hodnoty podle odpovídající veličiny. Tato funkce je poté představována jako matice obsahující data o snímaném obrazu. Prvky této matice jsou nazývány pixely, což jsou nedělitelné, nejmenší jednotky popisu obrazu, tzv. obrazové elementy. Prostor skládající se z pixelů je označován jako rastr, který může být čtvercový, hexagonální apod. Tento fakt je důležitý zejména pro výpočet vzdáleností v obrazu.

3.1 Knihovna OpenCV

Knihovna OpenCV (Open Source Computer Vision Library) [17] je open-source knihovna obsahující implementaci stovek algoritmů zpracování obrazu. Její zaměření je primárně na počítačové vidění v reálném čase. Knihovna je psána v programovacích jazycích C/C+, ale existují i rozhraní pro jazyky Java a Python využívající původní implementace knihovny. Knihovna je tak snadno přenositelná mezi platformami. Mezi nejrozšířenější podporované platformy patří Windows, Linux, macOS, ale i např. FreeBSD, NetBSD nebo OpenBSD. Knihovnu je také možné použít na mobilních platformách Android a iOS, což vedlo na její výběr pro vývoj této aplikace.

3.2 Neuronové sítě pro super-rozlišení a jejich nevýhody

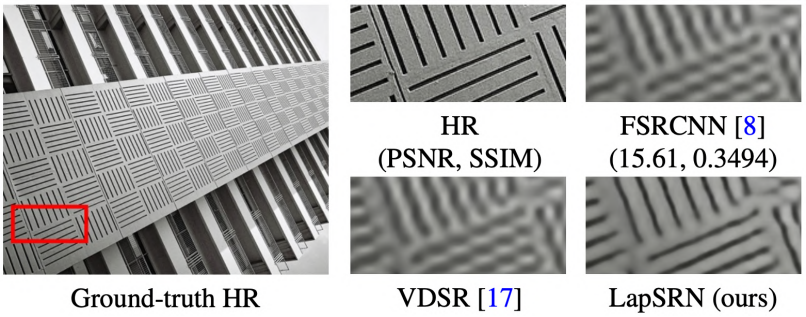
Umělé neuronové sítě nebo také hluboké neuronové sítě jsou na základě nedávného výzkumu na vzestupu co se týče super-rozlišení obrazu. Podle výzkumu, který probíhá v Soulu v Jižní Koreji, mají neuronové sítě EDSR+ a MDSR+ [15] velice dobré výsledky co se týče změn rozlišení obrazu a rekonstrukci detailů v něm.

Metody super-rozlišení pomocí neuronových sítí používají přístupy se vstupem několika snímků stejné scény s různými expozicemi nebo se vstupem jen jednoho snímku scény. Přístup se vstupem několika snímků vykazuje lepší výsledky, protože neuronové sítě skládají snímky za účelem rekonstrukce nebo zvýraznění vysokých frekvencí a zachování nízkých frekvencí ve výsledném obraze za pomocí dat z několika vstupních snímků obsahující více informací. Naopak sítě používající přístup se vstupem jednoho snímku musí snímek rekonstruovat na základě jejich učící sady, která určuje výsledek zpracování snímku.

Obrázek 3.1: Ukázka funkce hlubokých neuronových sítí EDSR+ a MDSR+ při porovnání s bikubickou interpolační metodou a originální snímkem ve vysokém rozlišení (obrázek označený HR). Obrázek převzatý z [15].

Detaily v obraze, které by byly klasickými metodami používanými pro zvýšení rozlišení, např. bikubickou interpolací, ztraceny, jsou pomocí hlubokých neuronových sítí zachovány a někdy upraveny tak, že odpovídají původnímu snímku ve vysokém rozlišení viz obrázek 3.1. Interpolační metody využívají k výpočtu již známé hodnoty pro odhad nových hodnot, které nejsou v průběhu výpočtu známé. Na rozdíl od interpolačních metod v oblasti super-rozlišení obrazu hluboké neuronové sítě rekonstruují výsledný obraz na základě jejich učení na datové sadě. Hluboká neuronová síť tedy může být učena na datové sadě, která obsahuje vzorky vyhovující pro použití neuronové sítě pro rekonstrukci detailů fotek přírody. Ovšem výsledky neuronových sítí nemusí být již tak vhodné pro zpracování obrazu rovinných ploch. Použití neuronových sítí může vést na artefakty v obraze, které jsou vytvořeny na základě učení. Neuronová síť LapSRN [14] pojmenována na základě Laplaceových pyramid, podle kterých je navržena, má nedostatky v některých případech ve spojování malých mezer mezi detaily v obrázku nebo mazání tenkých čar a tím vytváří nové obrazce, které v původním snímku vůbec nebyly viz obrázek 3.2.

Chybné rekonstrukce detailů v obraze neuronovými sítěmi se staly problémem v komerční sféře, když v roce 2013 inženýr z německa D. Kriesel objevil substituci čísel při skenování dokumentu fotokopírkou od firmy Xerox. Podle výsledků testování a informací od vedení vývoje skenování a digitálního zpracování dokumentů se došlo k závěru, že problémy, kdy skener vymění čísla, způsobuje softwarová komponenta *Pattern matching engine* obsažená v modulu zpracování naskenovaného dokumentu. Komponenta ukládá zpracovávané segmenty dokumentu jen jednou, ale používá je při rozložení stránky několikrát. Při rozložení se děje chybné porovnávání segmentů, kdy se segmenty vymění za úplně jiné, které komponenta vyhodnotila jako identické. Více informací k tématu je v článku inženýra Kriesela [10].



Obrázek 3.2: Ukázka funkce hluboké neuronové sítě LapSRN pro super-rozlišení. Obrázek převzatý z [15].

Další aspekt, proč v této práci nebyly použity hluboké neuronové sítě, je náročnost na výpočet. Mobilní aplikaci by čas výpočtu neměl trvat dlouho a měl by být snížen na minimum, aby potencionální uživatele neodradil čas zpracování. To v případě výpočtů nutných pro zpracování obrazu neuronovými sítěmi není možné garantovat. Jejich modely mohou být různě náročné na výpočet a výkon procesoru mobilního telefonu je limitující faktor.

Po konzultaci s vedoucím této práce jsem se tedy rozhodl pro konvenčnější přístup k technikám super-rozlišení obrázků a hluboké neuronové sítě nepoužít.

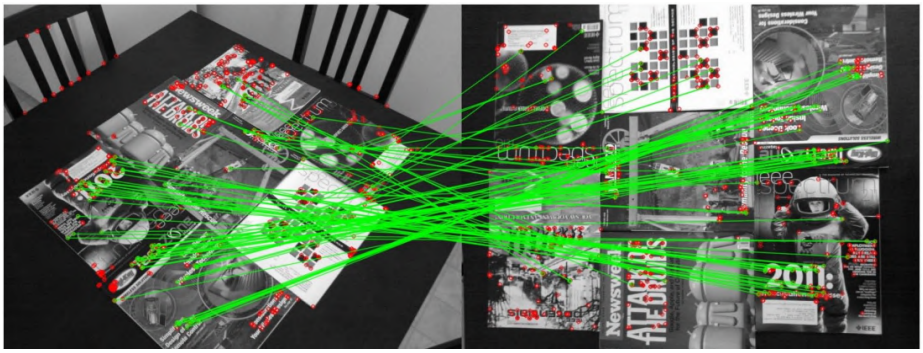
3.3 Rekonstrukce scény z více obrázků

Na základě rozhodnutí nepoužívat v práci neuronové sítě bylo třeba najít způsob, jakým lze z vyfotografovaného snímku více informací pro rekonstrukci scény. Nejjednodušší způsob, jak získat více dat pro zpracování obrazu, je pořídit více snímků stejné scény. Teoreticky čím více snímků datová sada obsahuje, tím více dat by mělo být k dispozici pro výpočet kvalitnějšího snímku. V kapitole 4 je popsáno, kolik snímků by datová sada měla obsahovat pro výpočet dostatečně kvalitního snímku. Hlavním cílem rekonstrukce scény je tedy skládání obrázků tak, aby byly rekonstruovány detaily, které nejsou v jednotlivých snímcích datové sady zachyceny dostatečně kvalitně (málo ostré, rozmazané, málo vidět). Výsledkem by měl být snímek obsahující ostré a dobře viditelné detaily. Několik přístupů, jakými lze tohoto výsledku dosáhnout, je popsáno v dalších sekcích této práce.

3.4 Zarovnání snímků

Při pořízení sady snímků mobilní telefonem mohou být snímky zachyceny v různých pozicích kamery a tím pádem sada obsahuje snímky s různou perspektivou. Tento stav datové sady pro slučování snímků není ideální, protože při použití některých technik popsaných v dalších kapitolách může při zpracování obrazu docházet ke vzniku artefaktů právě kvůli různým perspektivám snímků v sadě. Tento stav datové sady lze vyřešit nebo alespoň v některých případech vylepšit za pomoci zarovnání snímků.

Zarovnání datové sady probíhá podle prvního pořízeného snímku v ní. Tento přístup se osvědčil na základě testování zarovnávání snímků. V případě, kdy se zarovnávalo podle jiného snímku v pořadí pořízení, nebyly výsledky zarovnání vhodné pro další zpracování – na okrajích zarovnaných snímků vznikaly artefakty, snímky se nezarovnaly dostatečně dobře na základě příliš jiné pozice kamery v průběhu snímání, apod.



Obrázek 3.3: Ukázka spojení klíčových bodů v obrázcích pomocí detektoru ORB, zelené linky jsou validní spojení, červené body jsou nespojené klíčové body. Převzato z [20].

Registrace a zarovnání snímků je obvyklý proces při zpracování obrazu a existují již velice dobře odladěné algoritmy usnadňující práci při zarovnávání obrazu. Jedním z těchto algoritmů je algoritmus *ORB* popsaný v další podkapitole.

3.4.1 Detektor klíčových bodů ORB

Základním stavebním kamenem pro zarovnání snímků je nalézt určitou podobnost v zarovnávaných snímcích a podle ní snímky upravit. Pro tento účel byl použit detektor klíčových bodů ORB (Oriented FAST and Rotated Briefly) [20]. Detektor ORB má velký význam v oblasti zpracování obrazu, ve kterém se používá pro navigaci ve snímku, rozpoznávání a klasifikaci cílů, pro spojování obrázků, registraci klíčových bodů, zarovnávání apod. Na obrázku 3.3 je ukázáno nalezení a spojení klíčových bodů v dvou obrázcích. Detektor ORB je rozdělen do tří kroků:

- extrakce klíčových bodů z obrazu,
- vygenerování jejich deskriptorů,
- a porovnávání klíčových bodů.

Extrakce klíčových bodů

Detektor ORB používá k detekci klíčových bodů vylepšený algoritmus FAST (Features from Accelerated Segment Test) [19]. Hlavní myšlenka detekce klíčových bodů říká, že pokud je pixel markantně odlišný od pixelů v jeho okolí, je patrné, že by mohl být bodem na hraně.

Proces detekce probíhá následovně:

- Detekce klíčových bodů v obrázku. Prvně se zvolí pixel p v obrázku a předpokládá se, že jeho jas je I_p. Poté se nastaví prahová hranice T. Potom je pixel p zvolen jako centrální v okolí 16 pixelů a porovnají se stupně šedi mezi pixelem p a ostatních z jeho okolí. Pokud je jas po sobě jdoucích N pixelů v okolí vyšší než $I_p + T$ nebo menší než $I_p - T$, pak je pixel p považován za klíčový bod.
- Detekce rohů v obraze. Detektor ORB používá vylepšení původního algoritmu FAST s Harrisovým detektorem rohů, který vykazuje lepší výsledky než původní detektor rohů implementovaný v algoritmu FAST. Více informací je v literatuře [19]. Vylepšení spočívá v nalezení rohových bodů původním algoritmem FAST a jejich porovnání

výstupem z Harrisova detektoru, seřazení podle stupňů šedi a ponechání N rohových bodů. Výpočet rohových bodů Harrisovým detektorem je popsán v rovnici

$$M = \sum w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{3.1}$$

kde M je matice 2×2, $w(x,y)$ je okénková funkce, I_x je hodnota změny klíčového bodu v horizontálním směru, I_y je hodnota změny klíčového bodu ve vertikálním směru.

- Výpočet invariance klíčového bodu proti změně měřítka pomocí algoritmu FAST.
- Stanovení směru klíčového bodu. Je třeba zajistit invarianci klíčových bodů při rotaci snímku. To se zajistí pomocí intezity centroidu [18]. V malé části obrázku označované jako obrazový blok B, je moment této části definován jako

$$m_{pq} = \Sigma_{x,y \in B} x^p y^q I(x,y), p, q = 0, 1, \tag{3.2}$$

kde x, y jsou souřadnice pixelu, $I(x,y)$ je hodnota stupňů šedi pixelu na souřadnicích (x,y). Potom je nalezen centroid části obrázku pomocí rovnice

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right), \tag{3.3}$$

kde nultý moment m_{00} je váha obrazového bloku a první moment m_{01}, m_{10} je centroid bloku. Poté jsou geometrický střed bloku O a centroid spojeny k získání směrového vektoru $\overrightarrow{OC}$. Směr klíčového bodu je tedy definován jako

$$\theta = \arctan \left(\frac{m_{01}}{m_{10}} \right). \tag{3.4}$$

Extrakce klíčových bodů vede na získání klíčových bodů v obrázku, které jsou invariantní proti rotaci a změně měřítka obrázku.

Generování deskriptorů

Po extrakci klíčových bodů se vypočítají deskriptory. Deskriptor je reprezentován vektorem určité délky. Deskriptory pro každý klíčový bod se vypočítají pomocí algoritmu *BRIEF* [8]. BRIEF je deskriptor vektoru obsahující hodnoty 0 a 1

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & p(x) \geq p(y) \end{cases}, \tag{3.5}$$

kde $p(x)$, resp. $p(y)$, jsou hodnoty stupně šedi v okolí x, resp. y, okolo klíčového bodu. K redukci šumu v obrazu se použije Gaussův filtr. na základě klíčového bodu p, který je centrální v jeho okolí, se z tohoto okolí o velikosti $S \times S$ náhodně vybere N párů pixelů. Hodnota jasu se poté v těchto párech bodů porovná. Z porovnání vznikne N-dimenzionální vektor obsahující N binárních hodnot

$$f_N(p) = \sum_{1 \leq i \leq N} 2^{i-1} \tau(p; x_i, y_i), \tag{3.6}$$

Detektor ORB používá vylepšený algoritmus Steered BRIEF, který zajišťuje invarianci klíčových bodů při rotaci snímku. Rotační matice R_θ je tedy

$$R_\theta = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \tag{3.7}$$

Matice R_θ a N párů pixelů tvoří matici Q

$$Q = \begin{bmatrix} x_1, x_2, \ldots, x_N \\ y_1, y_2, \ldots, y_N \end{bmatrix}. \tag{3.8}$$

Potom je provedena korekce rotace k obdržení Q_θ

$$Q_\theta = R_\theta Q. \tag{3.9}$$

Po všech předchozích dílčích krocích je obdržen směrový deskriptor

$$g_N(p, \theta) = f_N(p) | (x_i, y_i) \in Q_\theta. \tag{3.10}$$

Porovnávání klíčových bodů

Po obdržení klíčových bodů z obrázků a jejich deskriptorů se přechází k porovnání deskriptorů na základě jejich podobnosti. Pokud klíčový bod $x_t^m, m = 1, 2, ..., M$ z obrázku I_t a klíčový bod $y_{t+1}^n, n = 1, 2, ..., N$ z obrázku I_{t+1} jsou k dispozici, porovnají se pomocí algoritmu Brute-Force Matcher. Tento algoritmus měří vzdálenosti mezi každým klíčovým bodem x_t^m a x_{t+1}^n a vybere nejbližší klíčové body jako shodu. V případě binárního BRIEF vektoru se vzdálenost porovnává pomocí Hammingovy vzdálenosti v dvou stejně dlouhých binárních vektorech.

3.4.2 Homografie

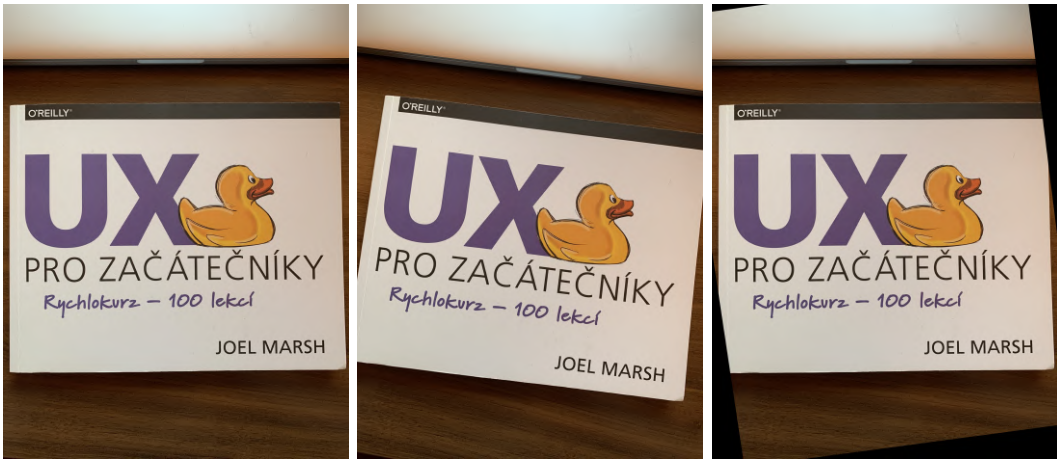
Po nalezení a zjištění korespondence klíčových bodů a deskriptorů mezi dvěma snímky, se snímky zarovnají pomocí *homografie* [7]. Homografie je v počítačovém vidění definována jako projektivní mapování bodů mezi dvěma planárními plochami, které odkazují na stejnou lokaci v obrázcích. Například obraz, který je zobrazen na displeji mobilního telefonu jako ukázka při focení 2D planární plochy, je příklad homografie. Homografie je ve 2D prostoru matice H s rozměry 3×3 mapující body p prvního obrazu s body p' druhého obrazu

$$wp = Hp', \tag{3.11}$$

kde w je měřítko. Homografie je tak užitečný nástroj pro zarovnání dvou obrázků, protože pomocí ní lze vypočítat potřebnou rotaci a translaci obrazu pro jeho zarovnání. Na obrázku 3.4 je ukázáno zarovnání snímků pomocí homografie. Zarovnává se pootočený snímek podle referenčního snímku určující základ pro nalezení homografie mezi snímky.

3.4.3 RANSAC

Pro nalezení homografie se často v knihovnách pro počítačové vidění používá metoda *RANdom SAmple Consensus (RANSAC)* [11]. RANSAC je algoritmus pro odhad parametrů ze souboru pozorovaných dat, který obsahuje odlehlé hodnoty, tzv. *outliers*. Opak outliers jsou tzv. *inliers*, což jsou vzorky odpovídající řešení. Odlehlé hodnoty potom nemají vliv na odhady parametrů. RANSAC je technika převzorkování, která generuje kandidátní řešení za



Obrázek 3.4: Ukázka zarovnání snímku pomocí homografie. Vlevo referenční snímek pro zarovnání, uprostřed pootočený, vpravo pootočený snímek upravený a zarovnaný pomocí homografie.

pomoci minimálního počtu dat potřebných k odhadu parametrů vstupního souboru dat. Jak uvedli Fischler a Bolles [11], konvenční vzorkovací techniky používají k získání vhodného kandidátního řešení co nejvíce vstupních dat a až po výpočtu řešení se zabývá problémem odlehlých hodnot ve vstupních datech. RANSAC k odhadu kandidátního řešení používá nejmenší možný počet dat a v průběhu výpočtu tato data průběžně rozšiřuje na základě získaných odhadů řešení. Více informací k algoritmu RANSAC je uvedeno v literatuře [11].

3.5 Sloučení několika snímků do jednoho

Po zarovnání snímků obsažených v datové sadě pro výpočet je třeba vytvořit výsledný obraz. Existuje několik způsobů, jakými lze sloučit snímky do sebe, ale jen některé vykazují dobré výsledky. Některé metody způsobují vytváření artefaktů v obraze slučováním snímků, jiné mohou způsobit viditelnost okrajů jednotlivých překrytých fotografií a ve výsledku žádným způsobem nevytvoří lepší obrázek ze všech v datové sadě. Metody zvolené pro tuto práci ale ve většině případů vykazují výsledky velice dobré, kdy úspěšně rekonstruují detaily, odstraní odlesky, vylepší barvy, apod. Důležitým požadavkem na metody sloučení obrázků také je její rychlost výpočtu. Jak už bylo zmíněno, aplikace by měla provést co nejrychlejší výpočet kvalitní fotky, aby uživatele neodradila dlouhým zpracováním. Právě proto jsem se rozhodl pro jednoduché metody, které nevyužívají náročného vážení pixelů, rozhodovacích struktur, apod.

Jednou z těchto jednoduchých metod je *průměrování snímků*, které v případě dobrého zarovnání snímků v datové sadě ze snímků vytvoří snímek s lepšími barvami a s výraznějšími detaily. Tím, že metoda využívá prostého průměrování pixelů, dosahuje dobrých výsledků v případech, kdy je detail ve snímcích zaměřen i z nepatrně jiných úhlů kamery, což má dobrý výsledek při rekonstrukci tohoto detailu z více snímků. Metoda má ale i své nevýhody (vytváření artefaktů, apod.), které jsou popsány v další podkapitole 3.5.1.

Další jednoduchá metoda pro slučování snímků, která byla použita v rámci této práce, byla *medián z pixelů snímků*, kterou jsem po konzultaci s vedoucím práce navrhl a implementoval. Vstup metody předpokládá s datovou sadou snímků, které jsou zarovnané. Metoda vykazuje stabilnější výsledky (méně umělých artefaktů) než metoda průměrování

Obrázek 3.5: Ukázka rozmazání obrázku vlivem průměrování ze snímků zachycených v jiný čas, pod jiným úhlem kamery, apod.

snímků, ale v některých případech neposkytne výslednému obrazu takovou kvalitu. Detaily metody budou popsány v následující podkapitole 3.5.2.

3.5.1 Průměrování snímků

Průměrování snímků je jedna z metod slučování více snímků do jednoho s cílem získání více detailů z jednotlivých obrázků. Metoda pracuje jednoduchým způsobem, kdy se vypočítá průměr hodnot pixelů na souřadnicích (x, y) ve snímcích. Pro co nejlepší výsledky metody je požadováno, aby sada vstupních obrázků byla zarovnána. Bez zarovnání může vznikat nechtěný efekt, kdy se ve výsledném obrázku začnou objevovat rozmazaná místa. V případě rozmazání hran v obrázku tento jev začíná být problém viz obrázek 3.5. Pro doplnění informací k metodě průměrování je vhodné zmínit, že pokud se bude průměrovat ze snímků i jen lehce odlišných, detaily ve snímcích se začnou prolínat a vzniknou nechtěné artefakty vedoucí k destrukci původních detailů ve všech snímcích. Proto je vhodné používat pro průměrování datovou sadu snímků buď zachycenou ze stativu nebo zarovnanou pomocí digitálního zpracování a zarovnání obrazu.

V případě, že snímky jsou správně zarovnány a mají v sobě jen nepatrné odlišnosti, metoda poskytuje velice slibné výsledky, kdy výsledný obrázek obsahuje kvalitní rekonstruované detaily, méně odlesků, plynulejší přechody mezi barvami apod. Další aspekt, proč používat průměrování, je snížení šumu v obrázku výpočtem. Šum může vznikat v obrázku např. nepříznivými podmínkami pořízení fotografie (špatné světelné podmínky, nekvalitní kamera, apod.) a metoda průměrování ho může potlačit nebo snížit vcelku úspěšně viz obrázek 3.6. Pro rovinné plochy je efekt vylepšení a rekonstrukce detailů stejný, obrázek krajiny byl použit pro pěknou demonstraci vylepšení finálního obrázku pomocí metody průměrování snímků.

3.5.2 Medián z pixelů snímků

Tato metoda využívá k výpočtu medián z hodnot pixelů v jednom bodě datové sady snímků na jejím vstupu. Metoda na rozdíl od průměrování vykazuje lepší výsledky v ohledu na vytváření artefaktů z odlišností v některých snímcích. Lépe kombinuje více snímků dohromady

Obrázek 3.6: Ukázka funkce průměrování snímků a její výstup z 33 snímků. Vlevo jeden snímek ze sady, vpravo hotový průměrovaný snímek. Obrázky převzaty z [9].

a lépe schovává viditelné hrany jednotlivých snímků použitých pro výpočet. Na druhou stranu ale neposkytuje tak kvalitní výsledky jako metoda průměrování. Metoda může způsobovat rozmazání hran a detailů z důvodu nezarovnaných snímků na vstupu stejně jako průměrování, ale má stabilnější výstupy právě oproti průměrování – má větší toleranci při lehce pootočených snímcích, lépe vyhlazuje přechody mezi jednotlivými snímky a má stabilnější barvy v různých světelných podmínkách. Bohužel kvůli výběru hodnot mediánu z pixelů různých snímků vznikají metodou vybledlejší barvy než metodou průměrování viz obrázek 3.7 za cenu vyšší stability slučování snímků touto metodou. Jediná nevýhoda této metody oproti průměrování je tedy vyblednutí barev, všechny ostatní pozitivní efekty na rekonstrukci detailů, odstranění šumu nebo redukci odlesků jsou zachovány.

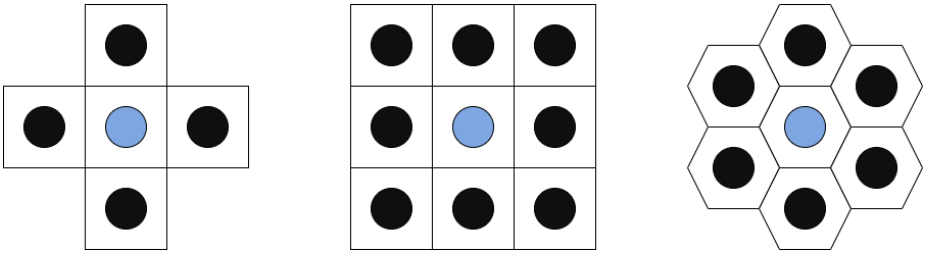
3.6 Snížení šumu v obraze

Při zpracování obrazu se hojně používají metody pro snížení šumu v obraze nebo také odstranění vysokých frekvencí, které se také nazývají metody vyhlazování obraze. Nejčastější technika používaná v oblasti zpracování obrazu pro odstranění šumu je *vyhlazování*, někdy také nazývána rozmazávání obrazu. Vyhlazování obrazu je používáno pro různé případy úprav fotografií, ale právě jedno z nejčastějších je snížení šumu při pořízení obrazu kamerou.

3.6.1 Lineární a nelineární filtry

Vyhlazování se provádí aplikací filtru na obraz, který způsobí změnu hodnot pixelů. Lineární filtry se od nelineárních liší procesem, kterým se získá vyfiltrovaný obraz. Lineární filtry vypočítávají výsledný obraz jako kombinaci lineární části obrazové matice s filtrem, který funguje jako dolní nebo horní propust ve stanoveném okolí. Okolí mohou mít různé rozměry,

Obrázek 3.7: Vliv metody mediánu pixelů ze snímků na změnu barev vypočítaného snímku (nahoře) oproti původnímu (dole).



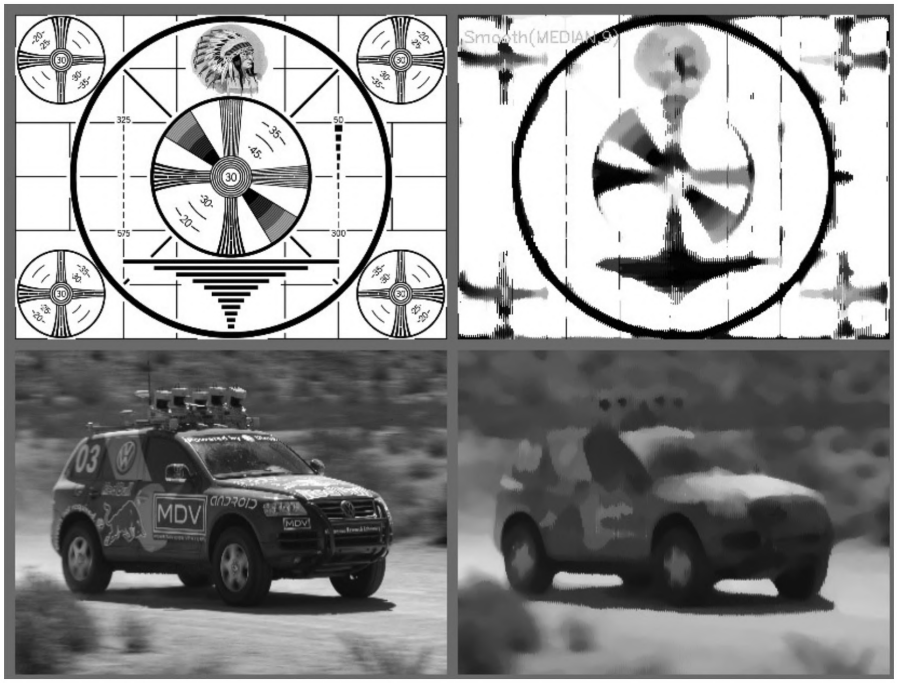
Obrázek 3.8: Ukázka typů okolí pixelu. Vlevo čtyřokolí, uprostřed osmiokolí, vpravo šestiokolí.

nejčastější jsou tzv. čtyřokolí a osmiokolí, ale v jiných případech a s jiným typem rastru se používají i jiná složitější okolí, např. šestiokolí viz obrázek 3.8.

Pro potlačení vysokých frekvencí, tedy redukci šumu v obraze, se používají lineární filtry typu dolní propust. Jednou z nejjednodušších metod pro potlačení šumu je *průměrování*, kdy se z daného okolí pixelu vypočítá průměr hodnot právě tohoto pixelu. To má za následek odstranění hodnot pixelů, které se podstatně liší od svého okolí a tedy potlačení šumu ve stanoveném okolí, které je většinou osm pixelů okolo středového pixelu. Hlavní nevýhodou průměrování je, že rozmazává hrany detailů za cenu odstranění šumu. Konvoluční maska průměrování pro okolí pixelu 3×3 může vypadat následovně:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.12}$$

Nelineární filtry využívají k výpočtu výsledného obrazu algoritmy výběru barvy z okolí na rozdíl od lineárních filtrů, které vyberou lineární kombinaci vstupních hodnot obrazu. Základními nelineárními filtry jsou výběr minima a výběr maxima ze stanoveného okolí.



Obrázek 3.9: Ukázka funkce mediánového filtru pro vyhlazení obrazu. Převzato z [7].

Mediánový filtr je příklad nelineárního filtru, kdy se z okolí pixelu vybere hodnota mediánu ze všech pixelů v okolí. Mediánový filtr zanechává hrany o poznávání lépe než výše zmíněné průměrování, ale může se stát, že filtr poruší tenké čáry v obrazu. V dalších kapitolách jsou popsány filtry, které byly použity v této práci.

3.6.2 Mediánový filtr

Mediánový filtr [7] je nelineární filtr, který je velice efektivní v odstraňování ostrého šumu, např. šum „sůl a pepř“ obsažený ve zpracovávaném obrázku. Princip mediánového filtru spočívá ve stanovení hodnoty pixelu na základě výpočtu mediánu hodnot z jeho okolí. Velikost a typ okolí je specifikováno před začátkem výpočtu. Nejčastější velikost okolí je 3×3, ale mohou být i větší a asymetrické. Čím větší je okolí ve výpočtu mediánového filtru, tím více bude obraz rozmazaný a tím pádem tedy bude ztrácet úroveň detailů. Vybrané hodnoty pixelů z okolí se seřadí a poté se z nich vybere medián, který se uloží do středu tohoto okolí (právě zpracovávaného pixelu). Oproti klasickému průměrování má mediánový filtr lepší výsledky v případě použití na zašumělý obraz, kdy průměrování může k výpočtu hodnoty pixelu použít i diametrálně odlišné hodnoty považované za tzv. *outliers* a vznikne další šum. Mediánový filtr je schopen tyto hodnoty vynechat a snížit tak úroveň šumu v obrazu. Ukázka funkce mediánového filtru je na obrázku 3.9.

3.6.3 Gaussův filtr

Gaussův filtr [12] rozšiřuje klasické průměrování o průměrování s Gaussovským jádrem. Gaussův filtr se používá k vyhlazování a rozmazání obrazu, tedy snižování šumu a odstranění detailů. Oproti průměrování používá tato metoda Gaussovo jádro, které v okolí pixelů zvýší váhu prostředního pixelu společně s jeho 4-okolím. Gaussův filtr může používat například

Obrázek 3.10: Ukázka funkce Gaussova filtru pro vyhlazení obrazu. Převzato z [7].

konvoluční masku tohoto tvaru:

$$m = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \tag{3.13}$$

Odezva Gaussova filtru ve 2D prostoru je:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{3.14}$$

kde x je vzdálenost od původního bodu v horizontálním směru, y je vzdálenost ve vertikálním směru, σ je standardní odchylka Gaussova rozdělení.

Gaussův filtr podává lepší výsledky než Mediánový filtr při vyhlazování obrazu viz obrázek 3.10, ale je náročnější na výpočet.

3.7 Techniky zvýšení rozlišení obrazu

Zvýšení rozlišení obrazu je nejčastěji prováděno za pomocí *interpolace obrazu*. Interpolace za pomocí matematického výpočtu zvýší počet pixelů v obraze na základě známých hodnot. Techniky interpolace se liší složitostí a náročností výpočtu a také podávaných výsledků, ale některé metody poskytují velice dobré výsledky, které mohou být užitečné pro další zpracování obrazu. Metody interpolace na základě jejich charakteristiky mohou některé detaily v obraze při výpočtu vypustit a některé zase zesílit, ale většinou je rekonstrukce detailů uspokojující a odpovídající původnímu snímku s nepatrným rozdílem v obsahu. Metody interpolace tedy určitým způsobem také vyhlazují obraz, ale vyhlazování je spíše nevyžádaný efekt než požadovaná vlastnost. V ideálním případě by interpolace při zvýšení

Obrázek 3.11: Porovnání funkce interpolací pro 4× zvýšení původního rozlišení obrázku. Vlevo původní snímek, uprostřed metoda Nejbližšího souseda, vpravo metoda Lanczosova.

rozlišení obrazu měla zachovat a rekonstruovat všechny detaily v původním obraze. Na obrázku 3.11 jsou ukázány obrázky, které byly převzorkovány na 4× vyšší rozlišení pomocí interpolace Nejbližšího souseda a Lanczosova převzorkování.

3.7.1 Interpolace Nebližší soused

Interpolace Nejbližší soused [13] je jedna z jednodušších interpolací obrazu. Metoda je interpolace nultého řádu – interpolace hodnotou, která je nejblíže k požadované. Vypočítává nové hodnoty pixelů duplikací hodnot okolních pixelů toho právě zpracovávaného. Hodnota nejbližšího pixelu od právě zpracovávaného se použije pro dosazení do právě vypočítávaného pixelu. Vzdálenost pixelů se vypočítává různými způsoby, ale nejčastější je Euklidovská vzdálenost, ve 2D může být zapsána jako:

$$v = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}. \tag{3.15}$$

Způsob výpočtu nových hodnot pixelů metodou Nejbližší soused nevede na zásadní vylepšení a metoda není příliš vhodná pro použití, kdy je důležitá kvalita výsledného obrazu viz obrázek 3.11. Metoda je výpočetně nenáročná a v porovnání s jinými metodami poskytuje horší výsledky za stejný čas zpracování. Proto se často používají jiné metody a tato metoda je vhodná pro specifické případy použití, např. v případě levného a rychlého zvýšení rozlišení obrazu bez priority zvýšení nebo zachování kvality původního snímku.

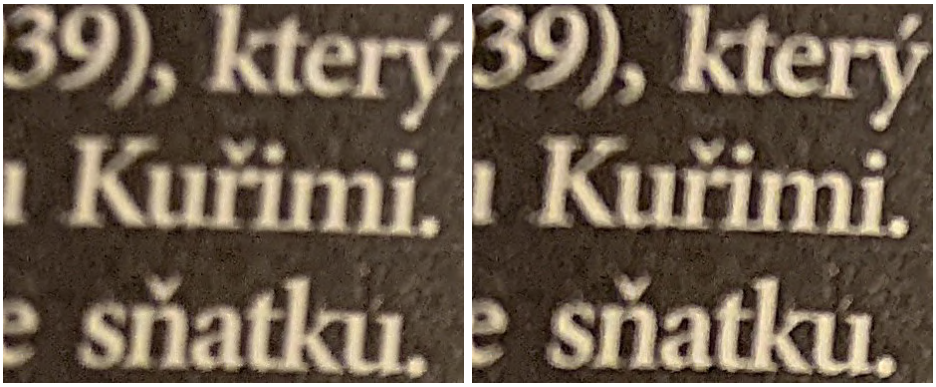
3.7.2 Lanczosova interpolace

Lanczosova interpolace [13] nebo také Lanczosovo převzorkování je metoda převzorkování vstupního signálu využívající Lanczosova jádra a funkce $sinc(x)$. Funkce $sinc(x)$ je neomezená a proto je potřeba využít všechny hodnoty ze vstupní diskrétní funkce – pixely obrazu. V praxi to znamená, že metoda je vysoce výpočetně náročná. Na druhou stranu poskytuje nejvěrnější výsledky oproti ostatním metodám využívajícím interpolaci pomocí polynomu N-tého řádu. Funkce $sinc(x)$ je definována jako

$$sinc(x) = \frac{\sin(\pi x)}{\pi x}. \tag{3.16}$$

Jádrem metody je normalizovaná funkce $sinc(x)$, která je násobena Lanczosovým jádrem nebo někdy také nazývaným *sinc* okénkem

$$L(x) = \begin{cases} sinc(x)sinc(x/a) & \text{pro } -a < x < a, \\ 0 & \text{jinak.} \end{cases}, \tag{3.17}$$



Obrázek 3.12: Ukázka zaostření snímku při zachování stejného rozlišení s použitím techniky Unsharp Masking.

kde a je pozitivní integer určující velikost jádra. Lanczosova metoda je metoda poskytující jedny z nejlepších výsledků při zvýšení rozlišení vstupního obrazu, a proto byla v této práci použita právě pro tento účel. Čas zpracování metody je vyšší než u jiných metod o zhruba stovky milisekund, ale není to zásadní rozdíl při reálném použití, aby se metoda nedala v této práci použít. Více informací o Lanczosově interpolaci je v literatuře [13]. V obrázku 3.11 je pěkně vidět, jak metoda zrekonstruovala detaily a některé i zvýraznila bez vzniku artefaktů při výpočtu.

3.8 Zaostření obrazu

Zaostření obrazu je využíváno ve spoustě oborů, ve kterých se užívá zpracování obrazu. Při fotografování se ostření snímku může hodit, pokud je potřeba ve snímku zvýraznit některé detaily, které v nezaostřeném snímku nejsou zřetelně vidět. Zaostření snímku by se tedy dalo pochopit jako zvýraznění hran detailů. Toto zvýrazňování je v nejobecnějším pojetí prováděno na základě podmínky, zda hrana má dostatečný kontrast oproti okolí.

Ostrost obrazu je dána dvěma faktory: *rozlišením obrazu* a *akutancí*. Rozlišení určuje počet pixelů ve snímku a tím pádem jeho velikost. Není pravidlo, které by říkalo, že vyšší rozlišení značí vyšší ostrost snímku. Ovšem pokud byl snímek pořízen kvalitní kamerou ve vyšším rozlišení, je ve snímku více informací a detailů než ve snímku pořízeném v nižším rozlišení. Akutance je subjektivní vnímání ostrosti, které souvisí s kontrastem hran ve snímku. Vnímání dvou snímků ve stejném rozlišení může být jiné právě na základě akutance. Snímek s vyšší akutancí bude na pohled ostřejší, i když nebude mít vyšší rozlišení. Na obrázku 3.12 jsou dva obrázky ve stejném rozlišení a jeden je zaostřen pomocí techniky Unsharp masking – upravený snímek je ostřejší bez zvýšení rozlišení.

3.8.1 Unsharp masking

V aplikaci byla využita technika zaostření snímku *Unsharp masking*. Tato technika využívá přístupu, kdy se ze snímku, který chceme zaostřit, odečte jeho rozostřená verze. Rozostřená verze je v anglickém překladu *Unsharp mask* a je tedy základem pro název techniky. Rozostřený snímek původního je tedy rozmazaný snímek, který je získán pomocí filtrů rozostření. Technika Unsharp masking využívá Gaussova filtru k získání rozmazané

masky snímku. Postup výpočtu ostrého snímku je popsán rovnicí

$$F(x, y) = (1 + c) \times I(x, y) + (-c) \times U(x, y), \tag{3.18}$$

kde $F(x, y)$ reprezentuje jas pixelu na souřadnicích (x, y) v zaostřeném snímku, $I(x, y)$ a $U(x, y)$ reprezentují jas korespondujících pixelů v originálním a rozostřeném snímku. Konstanta c stanovuje relativní váhy originálního a rozostřeného snímku.

Rovnice 3.18 popisuje funkci filtru Unsharp masking, kdy se odečtou vážené části originálního a vyhlazeného snímku. Toto odečtení má za následek zvýraznění vysokých frekvencích ve snímku na úkor snížení některých nízkých frekvencí a je tak možné, že zaostření povede na mírnou změnu barev v obraze. Kontrast hran v obraze je potom zvýšen na základě tohoto odečtení a tento jev zvýraznění hran vede na zaostření snímku viz obrázek 3.12.

Kapitola 4

Návrh a tvorba aplikace pro snímání povrchů

V této kapitole je popsán přístup k návrhu aplikace pro zvýšení kvality snímků rovinných ploch. Vývoj aplikace jsem si na začátku rozčlenil podle zadání na několik bodů:

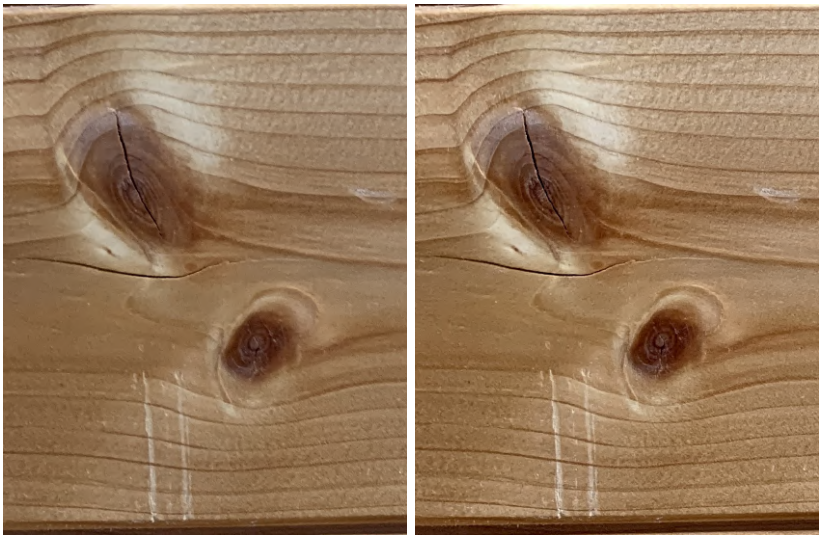
- Nastudovat možné způsoby zpracování obrazu a jeho vylepšení,
- vytvořit jednoduchou aplikaci pro získání datových sad snímků pro vývoj algoritmu,
- vytvořit algoritmus pro vylepšení kvality obrázku s fúzí několika snímků do sebe,
- experimentovat s algoritmem a vylepšit jej pro nejlepší výsledky ve většině případů,
- vytvořit mobilní aplikaci a implementovat do ní vyvinutý algoritmus,
- aplikaci ladit a testovat její výstupy.

Téma zpracování obrazu jsem nastudoval a popsal v kapitole 3, ve kterém jsou popsány techniky použité v této aplikaci. V dalších sekcích této kapitoly je popsán přístup k návrhu aplikace využívající právě tyto techniky.

4.1 Jednoduchá mobilní aplikace pro sejmutí sady fotek pro vývoj algoritmu

Mobilní telefony od firmy Apple v nativní aplikaci *Fotoaparát* poskytují možnost sejmutí velkého počtu snímků za krátký časový úsek tzv. sekvenci fotek. Tato funkce se zdála být vhodná pro počáteční sejmutí datové sady snímků, ale po zjištění, že jen prvních pár snímků v závislosti na počtu celkově sejmutých snímků tímto režimem je zachováno v plné kvalitě a ostatní snímky jsou komprimovány, jsem toto řešení zavrhnul. Snímky lze zpět dekomprimovat do původní kvality, ale pouze přes nativní aplikace systému iOS. Tento způsob je nevhodný z pohledu časové náročnosti a počtu jednotlivých kroků nutných pro získání datové sady snímků z aplikací.

V první fázi návrhu aplikace bylo tedy třeba získat více datových sad snímků pro vývoj a testování algoritmů zpracování obrazu. Po konzultaci s vedoucím práce jsem se rozhodl vytvořit aplikaci pro mobilní telefon iPhone s operačním systémem iOS, která by umožňovala snímat velký počet snímků v nejvyšší možné kvalitě pro získání dostatečného množství testovacích snímků.



Obrázek 4.1: Ukázka výsledku průměrování ze zarovnané sady snímků. Vlevo první ze sady snímků, vpravo průměrovaný z 10 snímků.

Aplikace má sloužit jako pomocný nástroj pro vývoj algoritmu. Proto není třeba dbát na hezké uživatelské prostředí, množství funkcí aplikace, apod. Aplikace má pouze sejmout daný počet snímků a uložit je patřičným způsobem. Pro maximální jednoduchost jsem zvolil ukládání snímků ihned po jejich sejmutí do nativní aplikace systému iOS *Fotky*, ze které nebyl problém snímky exportovat.

4.2 Vývoj a testování algoritmu pro zvýšení kvality obrázku

Po získání dostatečně velké datové sady snímků jsem začal vyvíjet algoritmus pro fúzi těchto snímků a zvýšení kvality výsledného snímku. Algoritmus jsem vyvíjel na počátku v programovacím jazyce *Python*[1], pro který poskytuje knihovna OpenCV [17] rozhraní. Jazyk Python poskytuje vysokou vyjadřovací sílu a byl tak snadnou volbou pro jednoduché a rychlé testování algoritmu.

V ideálním případě by měly být všechny snímky ve formátu bezeztrátového typu, např. PNG. Problém nastal v jejich velikosti a času zpracování, protože snímky typu PNG ve 4K rozlišení přímo z kamery bez jakéhokoliv zpracování zabírají okolo 100 MB paměti a při této velikosti by jakýkoliv výpočet trval nepřiměřeně dlouho. Po testování formátů snímků TIFF, DNG a HEIC jsem se rozhodl pro použití formátu HEIC, který poskytuje dobrý kompromis mezi kvalitou a velikostí snímku pro co nejmenší čas zpracování. Řešení to není ideální, protože formát HEIC není bezeztrátový, ale jiné řešení kvůli času zpracování a přidělování zdrojů systému iOS nebylo optimální a výsledek zpracování snímků formátu DNG a HEIC byl téměř k nerozeznání, ale čas zpracování byl v jednotkách sekund oproti minutám.

Při testování fúze obrázků pomocí průměrování jsem došel k závěru, že je třeba obrázky před jejich fúzí zarovnat. Jinak vzniká přesný opak kvalitnějšího snímku viz obrázek 3.5 vlivem pohybu mobilního zařízení v ruce při pořízení snímků. Pro zarovnání snímků jsem použil algoritmus ORB 3.4.1 a homografii 3.4.2. Po zarovnání snímků a průměrování vznikl

[1] https://www.python.org/

Obrázek 4.2: Ukázka výsledku metody mediánu z pixelů snímků ze zarovnané sady snímků. Vlevo první ze sady snímků, vpravo vypočítaný z 10 snímků.

lépe vypadající obrázek než při nezarovnání viz obrázek 4.1. Počet snímků potřebných pro zvýšení kvality sloučením se podle mého testování pohyboval od 4 do 10 snímků. Při deseti a více snímcích nějaké rovinné plochy použitých pro sloučení do jednoho snímku přestal být rozdíl v kvalitě znatelný.

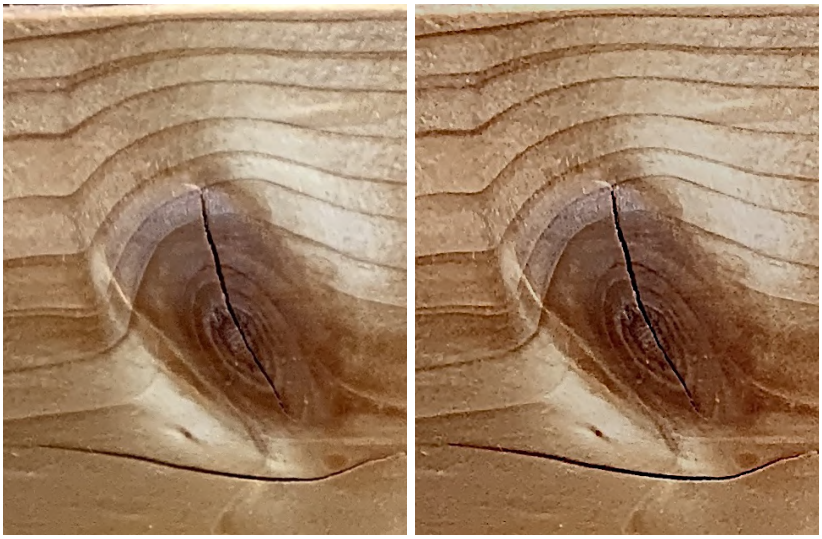
Po konzultaci s vedoucím práce jsem se rozhodl vyzkoušet techniku slučování obrázků pomocí výběru mediánu pixelů v jednom bodě zarovnaných snímků. Technika funguje způsobem, kdy se z obrázků na souřadnicích (x, y) uloží hodnota pixelu, která se poté seřadí s ostatními hodnotami pixelů z ostatních obrázků ze sady a ze seřazených hodnot se vybere medián, který slouží jako nová hodnota pixelu ve vypočítaném snímku. Metoda má podobné výsledky jako metoda průměrování a nevytváří tolik rozmazané hrany v případě špatně zarovnaných snímků, více je popsáno v sekci 3.5.2. Ukázka funkce metody je zobrazena na obrázku 4.2. Po testování metod jsem se rozhodl, že do výsledné aplikace implementuji výběr metody, pokud si uživatel bude chtít vyzkoušet jejich funkce.

Po testování slučování snímků výše popsanými metodami jsem se rozhodl otestovat zvýšení rozlišení obrázků pomocí metod interpolací obrazu. Metody, které jsem použil, jsou detailněji popsány v sekci 3.7. Testováním při zvyšování rozlišení snímků jsem došel k závěru, že jediná technika, která dokáže uspokojivě zvýšit rozlišení snímku bez ztráty informací a s kvalitní rekonstrukcí, je metoda Lanczosova 3.7.2. Ostatní metody (Bikubická [13], Bilineární [13], Nejbližší soused [13]) neměly tak dobré výsledky a jejich výpočet trval podobnou dobu. Proto jsem se rozhodl použít pro zvýšení rozlišení obrazu Lanczosovu interpolaci.

Zvýšením rozlišení snímků v datové sadě se získá více informací, které mohou sloužit ke kvalitnější fúzi snímků a rekonstrukci detailů. Proto jsem se rozhodl tento postup do výsledného algoritmu přidat. Výsledky fúze obrázků se zvýšeným rozlišením před výpočtem oproti fúzi obrázků v původním rozlišení jsou ukázány na obrázku 4.3.

Z obrázku 4.3 je vidět, že zvýšení rozlišení snímků v datové sadě a následné průměrování vede na zvýraznění a zaostření některých detailů a celkově tak na kvalitnější snímek.

Po konzultaci s vedoucím jsem se rozhodl do algoritmu zařadit zaostření obrazu, které zvýrazňuje hrany a detaily v obraze. Po průzkumu metod ostření snímků jsem se rozhodl



Obrázek 4.3: Fúze obrázků průměrováním se zvýšením rozlišení snímků datové sady o 200% Lanczosovou interpolací. Vlevo průměrovaný v původním rozlišení, vpravo průměrovaný ze snímků se zvýšeným rozlišením.

použít metodu *Unsharp Masking* (kapitola 3.18) pro její dobré výsledky a rychlost výpočtu ostrého snímku. Metoda pracuje způsobem, kdy se z původního obrázku, který je třeba zaostřit, vypočítá a uloží jeho rozmazaná verze. Tato rozmazaná verze je získána některým z filtrů obrazu pro potlačení šumu. V této práci jsem použil Gaussův a Mediánový filtr popsané v sekci 3.6. Rozmazaná maska se poté odečte od původního snímku čímž vznikne maska nazývaná *Unsharp mask*. Maska se poté přičte k původnímu snímku a dojde ke zvýraznění hran a tedy zaostření snímku. Po testování jednotlivých rozmazaných masek a výsledků ostření jejich pomocí jsem se rozhodl do výsledné aplikace zařadit výběr filtru.

Ostření jsem provedl na jednotlivých snímcích sady před jejich sloučením, ale tento způsob vede na přeostřený snímek se spoustou šumu kvůli zvýraznění hran jejich zesvětlením po fúzi. Po aplikaci Gaussova filtru pro redukci šumu má obrázek pořád nepřirozeně světlé hrany a barvy celkově neodpovídají originálu viz obrázek 4.4.

Ostření snímků před jejich sloučením tedy nevede k uspokojivým výsledkům. Vyzkoušel jsem výsledný snímek zaostřit až po průměrování ze sady a tento postup vedl k výsledku, který zachoval barvy a ostření zajistilo dobrou viditelnost rekonstruovaných detailů. Právě tento přístup jsem použil ve výsledném algoritmu.

Po testování výše uvedených technik pro zvýšení kvality snímku rovinné plochy jsem navrhl sekvenci kroků, která zaručí kvalitní snímek:

1. Nahraj sadu snímků.
2. Pomocí Lanczosovy interpolace zvyš rozlišení snímků na danou hodnotu.
3. Proveď registraci snímků a jejich zarovnání.
4. Proveď fúzi snímků zvolenou metodou.
5. Výsledný snímek zaostři.
6. Ulož výsledný snímek.

Obrázek 4.4: Snímek průměrovaný ze zaostřené sady snímků metodou *Unsharp masking*. Vlevo průměrovaný snímek z nezaostřené sady, uprostřed průměrovaný snímek ze zaostřené sady, vpravo průměrovaný snímek ze zaostřené sady s redukcí šumu pomocí Gaussova filtru.

Výstup sekvence kroků by měl tedy ideálně zvýšit ostrost snímku, zrekonstruovat jemné detaily v obrázku a celkově snímek vylepšit.

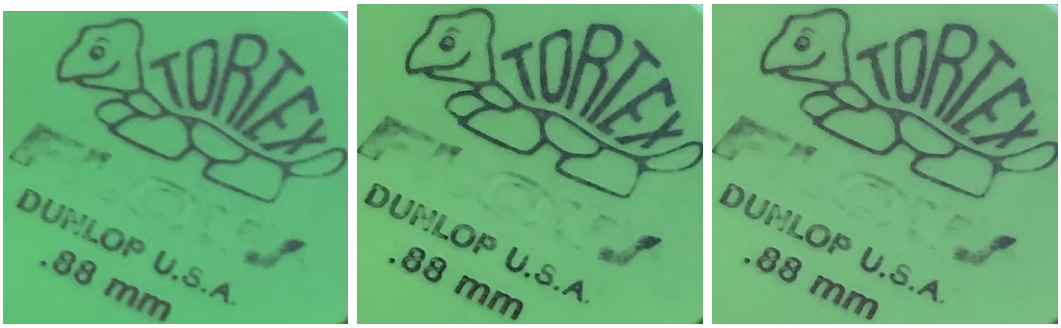
Po implementaci algoritmu do mobilní aplikace jsem narazil na problém, kdy systém iOS nepřidělil dostatečnou velikost operační paměti a aplikaci ukončil. S tímto problémem jsem se na počítači s operačním systémem macOS nesetkal a i po refaktorizaci a zefektivnění kódu nebyl systém iOS schopný aplikaci udržet v chodu. Výkon CPU zařízení problémem nebyl, ale velikost operační paměti je limitujícím faktorem. Proto jsem musel sekvenci kroků upravit s cílem maximálního zachování kvality výsledného snímku z původní sekvence. Po úpravě jednotlivých kroků zpracování a parametrů, se kterými se snímky zpracovávají, má použitá sekvence v aplikaci tuto podobu:

1. Nahraj sadu snímků.
2. Proveď registraci snímků a jejich zarovnání.
3. Proveď fúzi snímků zvolenou metodou.
4. Pomocí Lanczosovy interpolace zvyš rozlišení snímků na danou hodnotu.
5. Výsledný snímek zaostři.
6. Ulož výsledný snímek.

V porovnání těchto dvou sekvencí jsou rozdíly ve výsledném obrázku malé a proto jsem se rozhodl upravenou sekvenci implementovat do mobilní aplikace. Porovnání výsledných obrázků je možné vidět na obrázku 4.5.

4.3 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní jsem měl za cíl, aby bylo minimalistické a aby se uživatel, který nikdy aplikaci nepoužíval, zorientoval v aplikaci ihned. V případě foto-aplikace je vhodné a důležité, aby uživatel viděl to, co fotí. Je tak vhodné zahrnout zobrazení výstupu kamery do uživatelského rozhraní společně s tlačítkem spouště. Uživatel by měl mít možnost



Obrázek 4.5: Porovnání výstupu (uprostřed) sekvence pro zvýšení kvality snímku rovinné plochy s upravenou sekvencí pro mobilní aplikaci (vpravo) s jedním ze snímků z datové sady (vlevo). Vypočítaný snímek má zvýšené rozlišení o 200%, sloučení se provedlo průměrováním.

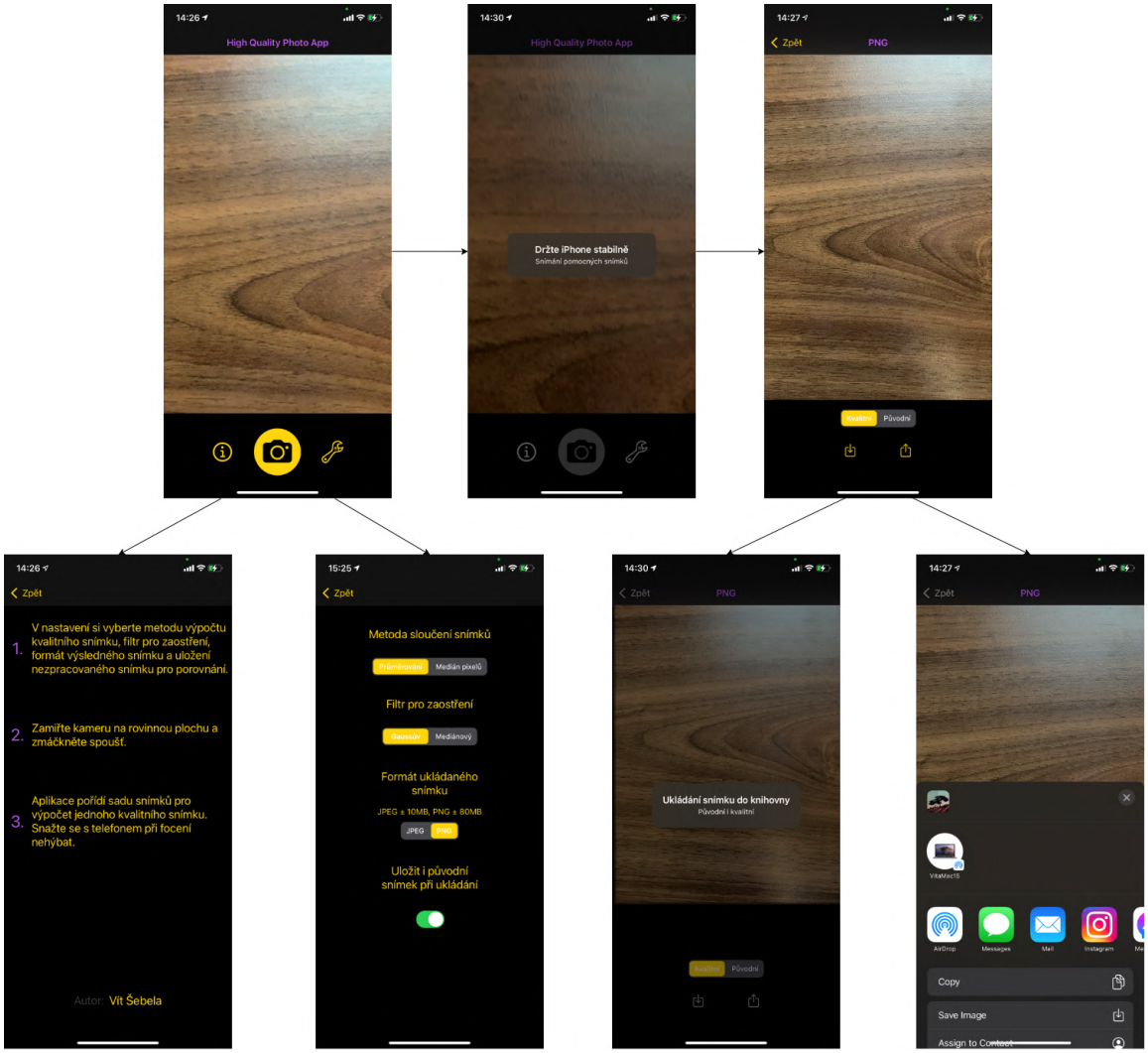
si přizpůsobit některá nastavení aplikace, např. metodu sloučení snímků (kapitola 3.5). Záložku nastavení s příslušnými přepínači jsem navrhl jako další obrazovku aplikace, na kterou se uživatel dostane jedním kliknutí. Další obrazovka dostupná jedním kliknutím z hlavní obrazovky je informační záložka, která uživateli představí jednoduchý způsob použití aplikace. Po vyfotografování rovinné plochy je uživatel informován o stavu zpracování vyskakovacím oknem s jednoduchým doporučení. Po zpracování je řízení předáno na jinou obrazovku, ve které uživatel může porovnat vypočítaný kvalitní snímek s jedním ze snímků datové sady. Na obrazovce s ukázkou snímků má uživatel možnost snímky přiblížit gestem prstů. Uživatel má potom možnost si snímek uložit nebo jej sdílet z aplikace pomocí sdílecího listu systému iOS. Na záhlaví obrazovky je zobrazen formát snímku, který si uživatel může vybrat v záložce nastavení.

Na obrázku 4.6 je zachycena posloupnost jednotlivých obrazovek a rozmístění prvků uživatelského rozhraní. Na počáteční obrazovce, po zapnutí aplikace, jsou 3 tlačítka spouštějící tyto akce: zobrazení informací, spoušť a zobrazení nastavení aplikace. Informace a Nastavení uživatele přesměruje na obrazovku s příslušným obsahem. Záložka Nastavení obsahuje následující prvky: volba sloučení snímků, volba filtru pro zaostření snímku, formát ukládaného snímku a přepínač nastavení souběžného uložení kvalitního i původního snímku při ukládání tlačítkem na prezentační obrazovce se snímky.

4.4 Implementace aplikace pro zvýšení kvality snímku

Aplikace je psána podle návrhového vzoru MVC (kapitola 2.3), který odděluje uživatelské rozhraní, logiku a data. Aplikace je tak rozdělena do souborů, které budou v dalších sekcích popsány:

- `Main.storyboard`,
- `ViewController.swift`,
- `RawCaptureDelegate.swift`,
- `ShowImageViewController.swift`,
- `AppSettingsViewController.swift`,



Obrázek 4.6: Ukázka navrženého uživatelského rozhraní aplikace

- `ImageDatabase.swift`,
- `OpenCVWrapper.mm`.

4.4.1 Získání snímků pomocí vestavěné kamery

První krok v návrhu bylo vytvoření funkce aplikace pro snímání datové sady snímků. V dřívějších kapitolách této práce již bylo zmíněno, že jsem zvolil systém iOS od firmy Apple z toho důvodu, že jsem dlouholetý uživatel produktů firmy Apple a vývoj aplikací pro systém iOS mě zajímal a chtěl jsem se jej v rámci této práce naučit. Pro vývoj uživatelského rozhraní aplikace jsem zvolil framework UIKit [5], který poskytuje potřebné nástroje pro jednoduchou tvorbu prvků uživatelského prostředí. Po prozkoumání možností získávání dat z hardwarových modulů zařízení (mikrofon, kamera, apod.) jsem se rozhodl pro použití knihovny firmy Apple *AVFoundation*[2] poskytující rozhraní pro práci audio-vizuálními médii, např. výstup z kamer zařízení.

[2] https://developer.apple.com/av-foundation/

Třída `ViewController` je podtřídou `UIViewController`, která zajišťuje aktualizaci zobrazovaného obsahu, provádí zpětnou vazbu na uživatelovi akce, stará se o rozmístění prvků UI, apod. a je v ní definována logika pro správu snímání fotek z kamery, kontrolu procesu zpracování obrázku a řízení toku dat. Pro správu dat ze vstupních a výstupních zařízení zařízení jsem použil třídu patřící do knihovny AVFoundation a to `AVCaptureSession`, která umožňuje založení relace pro získání výstupu kamery zařízení v reálném čase. Všechny dále zmíněné třídy a atributy s prefixem „AV“ jsou součástí knihovny AVFoundation. Vytvořený objekt typu `AVCaptureSession` je konfigurovatelný pomocí instančních atributů určujících vstupy a výstupy. Výstup z kamery je získán pomocí třídy `AVCaptureDevice` reprezentující vstupní zařízení. Kamera zařízení, reprezentovaná objektem typu `AVCaptureDevice`, je poté připojena do relace pomocí instanční metody `addInput()` jako vstupní zařízení. Objekt reprezentující kameru je možné konfigurovat a nastavit tak např. zoom pomocí atributu objektu `zoomFactor`, automatické zaostření pomocí atributu `focusMode`, apod. Pro získání dat z kamery je třeba do relace připojit objekt typu `AVCapturePhotoOutput` reprezentující rozhraní pro správu výstupních dat z kamery pomocí metody `addOutput()`. Pro zobrazení výstupních dat z kamery na obrazovku zařízení v reálném čase je použit objekt typu `AVCaptureVideoPreviewLayer`, který slouží jako zobrazovací vrstva výstupu kamery. Akce zmáčknutí spouště kamery je namapováno na tlačítko typu `UIButton`, které má k sobě definovanou akci v podobě metody `takePicturesButton()`, která se provede po každém zmáčknutí tlačítka. V metodě se vytvoří prázdné pole pro uložení snímků z kamery `capturedImages` a vymaže se proměnná pro výsledný snímek `capturedImages`. Poté je na obrazovku zobrazen indikace snímání sady fotek upozorněním pomocí třídy `UIAlertController`.

Po úspěšně provedených předchozích krocích se zavolá metoda `takePicture()`, ve které je opakovaně volána metoda `captureRawFormat()` na daný počet snímků, které se mají pořídit. Nastavení formátu snímaného obrázku je uloženo v proměnné odkazující objekt typu `AVCapturePhotoSettings`. Uvnitř metody `captureRawFormat()` je volána metoda `capturePhoto()` objektu typu `AVCapturePhotoOutput`, které se přes parametry předá nastavení formátu snímané fotografie pomocí proměnné `photoSettings` a odkaz na delegátní objekt. Delegátní objekt, zajišťující uložení pořízeného snímku, je instancí třídy `RawCaptureDelegate`, ve které je po zachycení snímku volána metoda `photoOutput()`, ve které je v atributu `photo` uložena fotka pokud bylo zachycení úspěšné. Pro každou pořízenou fotku se vytvoří jeden delegátní objekt, který vždy uloží fotku do statické třídy `ImageDatabase` pomocí statické metody `ImageDatabase.writeImage()`. Po úspěšném uložení všech fotek se v metodě `takePicture()` asynchronně spustí kus kódu, který zavolá metodu `processImages()`, která řídí zpracování obrázků.

4.4.2 Uživatelské nastavení aplikace

Uživatel si v obrazovce Nastavení, na kterou se dostane přes tlačítko na hlavní obrazovce, může nastavit některé parametry výpočtu snímku. Uživatel si může vybrat typ metody pro sloučení snímků (sekce 3.5), který se použije při výpočtu. Další nastavení je možnost výběru Gaussova nebo Mediánového filtru (sekce 3.6) pro výpočet rozmazané masky při použití techniky Unsharp masking (sekce 3.18). Dále si uživatel může vybrat formát výsledného snímku pro uložení a to buď PNG nebo JPEG. Volbu jsem zařadil, protože testování na uživatelích ukázalo, že někteří uživatelé požadovali volbu formátu JPEG kvůli výsledné velikosti snímku, která se pohybuje v případě JPEG okolo 10 MB a v případě bezeztrátového

formátu PNG okolo 80 MB. Poslední uživatelské nastavení je volba, zda se při ukládání snímků má uložit i původní snímek, ze kterého se kvalitní složil.

4.4.3 Zpracování snímků a zobrazení výsledku na obrazovku

V hlavní řídící třídě aplikace `ViewController` je po úspěšném zachycení a uložení zavolána metoda `processImages()`, která řídí zpracování obrázků. V metodě se vytvoří instance třídy `OpenCVWrapper`, díky které je možné použít knihovnu OpenCV přímo v kódu psaném v programovacím jazyce Swift. Třída `OpenCVWrapper` je wrapper kódu psaném v Objective-C++, který přímo umožňuje vložení čistého kódu v C++ a který je použitý pro implementaci zpracování obrázku pomocí knihovny OpenCV. V metodě `processImages()` se prvně zkontrolují uživatelská nastavení přístupem ke statickým atributům třídy `ImageDatabase` a poté je zavolána metoda třídy `OpenCVWrapper.process4()` s datovou sadou snímků a nastaveními v argumentech, která vrací hotový zpracovaný obrázek. Testováním jsem zjistil, že nejlepší poměr času zpracování a kvality výsledného obrázku je při použití datové sady o velikosti čtyř snímků. S více snímky byla aplikace nestabilní a v některých případech, pokud systém iOS neměl k dispozici většinu operační paměti zařízení, aplikace narazila na omezení démona systému řídící přidělování paměti a byla nečekaně ukončena. Po obdržení zpracovaného snímku se řízení předá třídě `ShowImageViewController`, která řídí zobrazení zpracovaného a jednoho snímku z datové sady pro porovnání. Pro předání dat mezi podtřídami třídy `UIViewController`, je použita metoda `performSegue()`, která zajistí přechod mezi zobrazovacími plochami. Po zavolání metody `performSegue()` je systémem zavolána metoda `prepare()`, ve které se zkontroluje objekt typu `UIViewController`, kterému se předává řízení a poté se jí předá zpracovaný a původní snímek přes její statické atributy `image` a `originalImage`.

Po předání řízení do `ShowImageViewController` je na obrazovku zařízení zobrazen snímek pomocí třídy `UIImageView`. Zobrazovaný snímek je buď zpracovaný nebo první z datové sady a přepínání snímků se provádí pomocí prvku `UISegmentedControl`. Snímek je možné přiblížit gestem prstů až 10× a přiblížení má na starost třída `UIScrollView`. Indikován je formát kvalitního obrázku v horní části displeje, který se uloží do nativní aplikace Fotky pomocí metody `UIImageWriteToSavedPhotosAlbum()` v případě zmáčknutí tlačítka v levé spodní části displeje. Pro jiné akce s obrázkem, např. sdílení obrázku do jiných aplikací, jsem do pravé spodní části přidal tlačítko spouštějící prvek `UIActivityViewController`, který zajišťuje zobrazení sdílecího listu systému iOS.

4.4.4 Implementace zpracování snímků pomocí OpenCV

Soubor `OpenCVWrapper.mm`, který obsahuje implementaci třídy `OpenCVWrapper`, obsahuje implementovanou sekvenci kroků pro zvýšení kvality snímku. Knihovna OpenCV je do projektu importována pomocí nástroje `CocoaBeans` a je možné ji používat v projektu, který je převážně psán v programovacím jazyce Swift, díky jazyku Objective-C++, do kterého je možné zapisovat kód v čistém C++ s rozhraním metod a funkcí psaných v Objective-C.

Metoda `process4()` na vstupu přijímá nastavení metody slučování snímků, typ filtru pro zaostření a sadu snímků. Snímky z kódu psaném v jazyce Swift jsou typu `UIImage` a je třeba je převést na typ `Mat`, kterým knihovna OpenCV reprezentuje obrázek. To je provedeno pomocí metody `UIImageToMat()` vracející přetypovaný obrázek. Po převodu všech snímků z datové sady na typ `Mat` se snímky zarovnají pomocí metody `alignImages()`, která na vstupu přijímá pole snímků k zarovnání, referenční snímek, oproti kterému se bude zarovnávat a prázdné pole pro uložení zarovnaných snímků. Po zarovnání snímků jsou snímky

uloženy v poli `alignedImages` a pole je předáno podle výběru metody sloučení snímku jako argument metodám `averageOfImages()` a `calculateMedian()`, které sloučí snímky danou metodou. Metody vrací sloučený snímek a uloží jej do proměnné `processedImage`, která je předána metodě knihovny OpenCV `resize()`, která snímek převzorkuje na rozlišení vyšší o 200 % Lanczosovou interpolací. Převzorkovaný snímek je předán metodě `unsharpIm()`, která v argumentu očekává snímek k zaostření, parametry zaostření a typ filtru pro rozmazanou masku. Zaostřený snímek je poté uložen do proměnné `sharpenedImage`, která je převedena z typu `Mat` na `UIImage` a poté vrácena z metody `process4()`.

Kapitola 5

Uživatelské testování a zhodnocení výsledků aplikace

5.1 Uživatelské testování

Pro testování funkčnosti aplikace jsem zvolil techniku uživatelského testování, která do procesu testování aplikace zahrnuje samotné uživatele a poskytuje tak okamžitou zpětnou vazbu. Zahrnutí uživatelů do testování má také podstatný význam v odhalování chyb, které mohly být vývojářům skryté.

Testování jsem prováděl za pomoci kamarádů a rodinných příslušníků. Celkově jsem aplikaci testoval na 15 uživatelích. Testování jsem prováděl za účelem odhalení funkčních chyb aplikace, ale také jsem používal uživatelskou zpětnou vazbu ke vzhledu rozhraní, jednoduchost a použitelnost. Uživatelské testování se skládalo ze 3 testů:

- Zapnutí aplikace, navigace na obrazovku s informacemi a nastavení příslušných hodnot v obrazovce s nastavením,
- Pořízení snímku vazby knihy a jeho uložení s uložením původního snímku zároveň,
- Pořízení snímku vazby knihy a jeho porovnání s originálem a následné sdílení přes aplikaci Zprávy.

Při testování mobilní aplikace jsem po uživatelích chtěl zpětnou vazbu ke vzhledu uživatelského rozhraní, k rozmístění tlačítek, funkci jednotlivých prvků rozhraní a na celkovou intuitivnost aplikace. Na základě této zpětné vazby jsem upravoval uživatelské rozhraní až do stavu, kdy všech 15 uživatelů nemělo výtky nebo poznatky ke vzhledu a funkčnosti. Po testování vhledu a funkčnosti uživatelského rozhraní aplikace jsem uživatele vyzval k porovnání výsledné fotky z aplikace oproti originálu. V průběhu testování bylo v aplikaci odhaleno celkově 5 chyb:

- Nefunkčnost tlačítka pro ukládání fotek,
- Zaseknutí přibližování výsledného snímku po otočení telefonu do vodorovné polohy,
- Pád aplikace po opětovném zmáčknutí spouště při pořizování fotek,
- Nefunkční zobrazení výstupu kamery po opuštění aplikace na domovskou obrazovku a opětovném spuštění,

- Zaseknutí aplikace při vracení z obrazovky s nastavením zpět na hlavní obrazovku.

První 2 chyby našel hned první uživatelský test, protože jsem před ním sám aplikaci dostatečně netestoval a testoval jsem ji pouze pomocí simulace v prostředí Xcode. První chyba byla opravena jednoduše – chyběla jen hodnota v argumentu metody pro ukládání fotky. Druhá chyba byla způsobena povolením režimu *landscape* aplikace, který zapříčinil po otočení zařízení o 90 stupňů jiné rozmístění prvků uživatelského rozhraní a hodnoty rozhraní pro zvětšení fotky mimo hranice. Třetí chybu jsem objevil až při testování osmým uživatelem, kdy uživatel omylem zmáčkl spoušť při snímání fotek datové sady a aplikace spustila proces znovu a to s únikem paměti v podobě nezpracovaných snímků, který vedl k ukončení aplikace systémem iOS. Chybu jsem vyřešil deaktivací tlačítka v průběhu snímání fotek a jeho opětovnou aktivací po úspěšném dokončení procesu. Čtvrtá chyba byla odhalena desátým uživatelem, kdy uživateli přišla v průběhu testování aplikace zpráva a přešel do aplikace Zprávy, aby na ni odepsal, a po vrácení do testované aplikace nefungovala ukázka výstupu kamery v reálném čase, ale aplikace fungovala dál. Problém nastal v opětovné inicializaci relace výstupu kamery, kdy nebyl ošetřen jeden případ pro znovuobnovení. Poslední chybu odhalil čtrnáctý uživatel, ale chyba se po vypnutí aplikace přes manažera běžících aplikací a po opětovném spuštění již neprojevila. Je to tak jediná chyba, která v průběhu uživatelského testování nastala a nebyl jsem schopen ji plně nalézt vyřešit.

5.2 Zhodnocení výsledků aplikace

Cílem celé této práce bylo navrhnout aplikaci, která vyfotografuje snímek rovinné plochy a vytvoří z něj vysoce kvalitní fotografii, která odhalí jemné detaily v obraze, které mohou být potlačeny při pořizování snímků výchozí aplikací systému iOS.

Na základě zaměření aplikace jsem požádal 3 kamarády, jejichž povoláním je fotografování, aby se přidali k testování výstupů aplikace a zhodnocení jejich kvality. Jejich subjektivní hodnocení je, že výsledný snímek obsahuje více detailů než snímek vyfotografovaný kamerou zařízení bez zpracování, má přirozenější barvy a kvalitní snímek by použili pro své potřeby radši než snímek pořízený kamerou bez zpracování. Tito uživatelé testovali snímky z výsledné mobilní aplikace a tak byly kvalitní snímky získány upravenou sekvencí při zpracování obrázku pro mobilní aplikaci (sekce 4.2), která neposkytuje kvalitnější výstup než původní sekvence zpracování obrázku. Cíl získání vysoce kvalitního snímku rovinné plochy za pomocí mobilní aplikace jsem tedy považoval za splněný.

Kapitola 6

Závěr

V této technické zprávě se pojednává o návrhu, tvorbě a testování mobilní aplikace pro pořizování vysoce kvalitních snímků rovinných povrchů. Před začátkem návrhu a vývoje aplikace jsem nastudoval potřebnou literaturu, která se zabývá tvorbou mobilních aplikací pro systém iOS a zpracování obrazu pomocí knihovny pro počítačové vidění OpenCV. Pro tvorbu mobilní aplikace jsem vyzkoušel programovací jazyk Swift v kombinaci s Objective-C++, naučil se pracovat s balíkem nástrojů UIKit a vyzkoušel si základy tvorby mobilních aplikací pro systém iOS. Vytvořil jsem aplikaci, která umí vyfotografovat sadu snímků a zpracovat je příslušným způsobem pro získání jedné vysoce kvalitní fotografie. Nastudoval jsem techniky registrace a zarovnávání obrazu pomocí detektoru ORB a homografie. Tyto techniky jsem poté úspěšně implementoval do aplikace. Nastudoval jsem možnosti sloučení (fúze) snímků a jejich přednosti a nevýhody a implementoval dvě z nich: průměrování a výběr mediánu z pixelů snímků v jednom bodě. Výsledkem průměrování je kvalitnější snímek než druhou zmíněnou metodou, ale výpočet je náchylnější k rozmazání obrázků, které se dobře nezarovnaly. Pro zvýšení rozlišení obrázků jsem použil metody interpolace, konkrétně jsem testoval metodu Nejbližšího souseda a Lanczosovu interpolaci. Interpolace Nejbližší soused je rychlejší a méně náročná na výpočet, ale Lanczosova interpolace poskytuje mnohem lepší rekonstrukci detailů při zachování podobné doby výpočtu. Pro zaostření výsledného obrázku jsem použil metodu Unsharp masking, která poskytuje dobré výsledky ve většině případů zaostření obrázků. Otestoval jsem náročnost celého výpočtu a zjistil, že výpočet nejkvalitnějšího možného snímku je možný na počítači a ne v aplikaci běžící na mobilních telefonech iPhone, ale upravením výpočtu lze získat pořád velice kvalitní snímek rovinné plochy. Uživatelským testování jsem získával od uživatelů zpětnou vazbu ke vzhledu a funkci aplikaci a iterativně jsem aplikaci vylepšoval podle připomínek těchto uživatelů.

Literatura

[1] Apple. *Delegation* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html.

[2] Apple. *Foundation* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/documentation/foundation.

[3] Apple. *Model-View-Controller* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[4] Apple. *SwiftUI* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/xcode/swiftui/.

[5] Apple. *UIKit* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit.

[6] Apple. *What is Cocoa?* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html.

[7] Bradski, G. a Kaehler, A. *Learning OpenCV*. O'Reilly Media, Inc., September 2008. ISBN: 978-0-596-51613-0.

[8] Calonder, M., Lepetit, V., Strecha, C. a Fua, P. BRIEF: Binary Robust Independent Elementary Features. In:. Září 2010, sv. 6314, s. 778–792. DOI: 10.1007/978-3-642-15561-1_56. ISBN 978-3-642-15560-4.

[9] Cox, S. *The Overlooked Technique of Image Averaging* [online]. Březen 2021 [cit. 2021-04-26]. Dostupné z: https://photographylife.com/image-averaging-technique.

[10] D., K. *Xerox scanners/photocopiers randomly alter numbers in scanned documents* [online]. 2014 [cit. 2021-04-26]. Dostupné z: https://www.dkriesel.com/en/blog/2013/0802_xerox-workcentres_are_switching_written_numbers_when_scanning.

[11] Fischler, M. A. a Bolles, R. C. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. červen 1981, sv. 24, č. 6, s. 381–395. DOI: 10.1145/358669.358692. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/358669.358692.

[12] Fisher, R., Perkins, S., Walker, A. a Wolfart, E. *Gaussian Smoothing* [online]. 2003 [cit. 2021-04-28]. Dostupné z: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.

[13] Getreuer, P. Linear Methods for Image Interpolation. *Image Processing On Line*. Září 2011, sv. 1. DOI: 10.5201/ipol.2011.g_lmii.

[14] Lai, W.-S., Huang, J.-B., Ahuja, N. a Yang, M.-H. Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, s. 5835–5843. DOI: 10.1109/CVPR.2017.618.

[15] Lim, B., Son, S., Kim, H., Nah, S. a Lee, K. M. Enhanced Deep Residual Networks for Single Image Super-Resolution. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, s. 1132–1140. DOI: 10.1109/CVPRW.2017.151.

[16] Microsoft. *Model-View-ViewModel* [online]. 2021 [cit. 2021-04-21]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[17] OpenCV. *OpenCV* [online]. 2021 [cit. 2021-04-29]. Dostupné z: https://opencv.org/.

[18] Rosin, P. Measuring Corner Properties. *Computer Vision and Image Understanding*. Únor 1999, sv. 73, s. 291–307. DOI: 10.1006/cviu.1998.0719.

[19] Rosten, E., Porter, R. a Drummond, T. FASTER and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*. 2010, sv. 32, s. 105–119. DOI: 10.1109/TPAMI.2008.275. Dostupné z: http://lanl.arXiv.org/pdf/0810.2434.

[20] Rublee, E., Rabaud, V., Konolige, K. a Bradski, G. ORB: An efficient alternative to SIFT or SURF. In: *2011 International Conference on Computer Vision*. 2011, s. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.