FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Mobilní scanner dokumentů
Student:	Adam Gelatka
Vedoucí:	Ing. Dominik Veselý
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Cílem práce je návrh a prototypová implementace mobilní aplikace pro scanovaní dokumentů pomocí fotoaparátu mobilního telefonu.

1. Prozkoumejte možnosti scanování dokumentů pomocí algoritmů počítačového vidění a fotoaparátu mobilního telefonu.
2. Z nalezených postupů zvolte jeden, který implementujete. Klíčové vlastnosti jsou detekce dokumentu, jeho ořez a následná transformace dokumentu, která zajistí lepší čitelnost (změna perspektivy a zvýraznění kontur).
3. Po dohodě s vedoucím práce zvolte platformu (Android, iOS,...) a vhodné implementační prostředí a v něm navrhněte a implementujte jednoduchou mobilní aplikaci, která umožní scanovat dokumenty vybraným postupem a dále je umožní nahrávat na vzdálený server. Server dodá vedoucí práce.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 19. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Mobilní scanner dokumentů

Adam Gelatka

Katedra softwarového inženýrství
Vedoucí práce: Ing. Dominik Veselý

4. června 2020

Poděkování

Rád bych poděkoval Ing. Dominikovi Veselému za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 4. června 2020 . . . . . .

Odkaz na tuto práci

Gelatka, Adam. *Mobilní scanner dokumentů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Práce zkoumá problematiku skenování dokumentů pomocí fotoaparátu mobilního telefonu a navrhuje prototyp aplikace na skenování dokumentů. Zároveň využívá různorodé metody zpracování obrazu v kontextu detekce prvků v obraze a jejich následné postprodukce. U nalezených metod je zhodnocena vhodnost pro dané použití a dopad jednotlivých parametrů na efektivitu skenování. Vhodné řešení problematiky je demonstrováno na iOS aplikaci s podporou vzdáleného serveru.

Klíčová slova skener dokumentů, mobilní aplikace, Cannyho detektor hran, Gaussovo rozostření, OpenCV, iOS, konvoluce.

Abstract

This bachelor thesis is focused on the problematics of scanning documents using a mobile phone camera and suggests a prototype of document scanning application. It also uses various methods of image processing for object detection and postproduction. The functionality and the impact of the individual parameters of the viable methods are compared to determine the effectivity of scanning. A viable solution of the problematics is demonstrated by an iOS application with the support of a distant server.

Keywords document scanner, mobile application, Canny edge detector, Gaussian blur, iOS, convolution

Obsah

Seznam obrázků

Seznam tabulek

Úvod

S nástupem moderních technologií se zvyšuje poptávka po digitalizaci obsahu všeho druhu, ať už jde o fotky, filmy, knihy, ale i o dokumenty a doklady. Díky digitalizaci jsou důležitá data dostupná z jakéhokoliv místa, relativně v bezpečí a přehledně zorganizovaná. Programů a aplikací soustředících se na tuto problematiku je na trhu celá řada, to ovšem nemění nic na faktu, že jsou podobné technologie čím dál tím více žádané.

Na digitalizaci obsahu plynule navazuje tato bakalářská práce, která se zaobírá skenováním veškerých druhů dokumentů pomocí integrované kamery v mobilním telefonu. V rámci celé práce bude za dokument považován veškerý obsah, který je na obdélníkovém, popřípadě čtvercovém podkladu. Kapitola rešerše se zabývá průzkumem metod a postupů, které slouží k rozpoznávání obsahu v obraze, v tomto případě dokumentů s již zmíněnými rysy. Knihoven na práci s obrazem je dostupná široká škála, avšak ne všechny jsou ideální pro zadefinované použití. Veškeré objevené technologie projdou zhodnocením jak z funkční stránky, tj. s jakou chybností a kvalitou celý proces funguje, tak i ze stránky efektivity, kde velkou roli hraje rychlost a náročnost algoritmů. Z výstupu testování je nutné odvodit, které kombinace nastavení jednotlivých operací jsou nejvhodnější. Pro zachování plynulosti rozpoznávání obsahu v obraze a zobrazení obrazových dat z kamery v reálném čase je zapotřebí najít rovnováhu v přesnosti a náročnosti zvolených algoritmů. Nejrozšířenější knihovna pro práci s obrazem se nazývá OpenCV. Právě na ní se bude práce ve velké míře soustředit.

V úvodu práce je nutné prozkoumat veškeré technologie, které budou využity při tvorbě aplikace, zejména však při vývoji skenovacího algoritmu. Kapitola návrhu se soustředí na výběr návrhového vzoru a na návrh GUI. Realizační část této práce pojednává o implementaci skenovacího algoritmu a jeho integraci do iOS aplikace. Veškeré části algoritmu jsou v následující kapitole otestovány, zejména z rychlostní stránky. Na závěr je celá práce zhodnocena.

Kapitola 1

Cíl práce

Cílem práce je návrh a prototypová implementace mobilní aplikace pro skenování dokumentů pomocí fotoaparátu mobilního telefonu. Aplikace bude určena pro operační systém iOS vyvíjený firmou Apple inc., konkrétně budou podporovány mobilní telefony řady iPhone. V následujících odstavcích jsou uvedeny veškeré cíle práce.

Prvním krokem práce je prozkoumání a otestování možností počítačového vidění, které mohou být přínosem pro řešení mobilního skenování. Zohlednit se musí několik aspektů, zejména časová složitost algoritmů, náročnost implementace a celková funkčnost.

Z nalezených postupů bude zvolen a implementován ten, jehož klíčové vlastnosti jsou detekce dokumentu, jeho ořez, následná transformace dokumentu a postprodukce, která zajistí lepší čitelnost. U algoritmů na úpravu dokumentu není nutné dbát na efektivitu, neboť nejsou součástí kontinuální analýzy obrazu a nejsou využívány tak často.

V rámci třetího cíle vznikne mobilní aplikace pro operační systém iOS. Po dohodě s vedoucím práce byla zvolena platforma iOS a vývojové prostředí Apple Xcode. Použité budou programovací jazyky Swift, Objective-C a C++. Aplikace bude tedy dostupná výhradně pro zmíněnou platformu. Kromě funkcionality skenování je součástí zadání podpora vzdáleného serveru.

Kapitola 2

Rešerše

V této kapitole se práce zaobírá analýzou dostupných technologií pro práci s obrazem. Za myšlenkou skenování pomocí fotoaparátu se skrývá mnoho grafických, potažmo matematických operací.

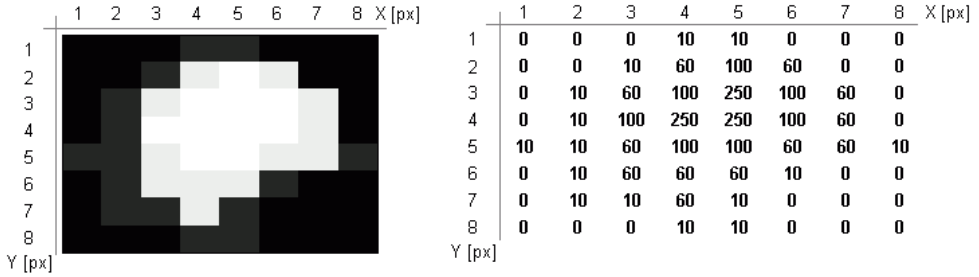
V průběhu analýzy obrazu, v tomto případě detekce dokumentu, se na každý výstupní snímek z kamery musí aplikovat určitá sekvence operací, která má za výsledek detekci uživatelem hledaného objektu. Pod touto řadou úprav se skrývají dva typy operací. První půlkou jsou funkce, které obraz připraví na následující detekci, ať už pomocí úprav barev, rozmazání, změn kontrastu apod. Do druhé části se řádí samotná detekce, která může být založena na více faktorech, např. na ostrosti hran, barevném rozdílu, ...

Za zváženou stojí fakt, že většina grafických operací (např. rozmazání, změna kontrastu) stojí na matematické operaci násobení matic. Z toho plyne, že mohou mít poměrně vysokou složitost. Vzhledem k tomu, že je chtěné obraz analyzovat v reálném čase, tak je klíčové dbát na dostatečnou rychlost operací tak, aby nedošlo ke snížení snímkovací frekvence při zobrazení dat z kamery. Proto je nutné všechny nalezené operace řádně otestovat a zvolit ty, které vyhovují nejen funkcionalitou, ale i rychlostí. Následující část práce zkoumá teoretický podklad, který je důležitý pro jednotlivé grafické operace, které bude využívat zejména skenovacího algoritmus.

2.1 Reprezentace obrazových dat

Rastrová obrazová data jsou v počítačových systémech reprezentována jako trojrozměrné matice s rozměry x, y, k, kde souřadnice x, y udávají pozici obrazového bodu v matici a hodnota k určuje barevný kanál. Jednotlivé hodnoty matice určují jasovou úroveň daného obrazového bodu. Hodnota je nejčastěji osmibitové číslo, tj. 256 různých odstínů. Alternativně lze říct, že se jedná o osmibitovou barevnou hloubku.

Příkladem může být všem známý RGB prostor, kde jsou obrazová data složena ze tří barevných složek: červené, zelené a modré. Každá z těchto složek bude mít v počítačové reprezentaci přiřazenou jednu matici a následným aditivním složením všech tří matic se dosáhne výsledku, který oko vnímá jako barevný obraz. [1]



Obrázek 2.1: Reprezentace jednokanálového obrázku maticí [1]

2.2 Konvoluce

Konvoluce je matematický operátor, který ze dvou funkcí na vstupu vytvoří funkci novou. Používá se k úpravě různých druhů signálu, kde ze dvou vstupních signálů vyprodukuje jeden výstupní. Mimo jiné je konvoluce velmi často využívána v počítačové grafice.

Definice 2.2.1. Jednorozměrná konvoluce [2] je definována mezi diskrétními funkcemi $f(x)$ a $h(x)$ v bodě x vztahem:

$$(f * h)(x) = \sum_{i=-k}^{k} f(x-i)h(i) \ ,$$

kde v případě zpracování obrazu nazýváme funkci f vstupním obrazem, funkci h jádrem, neboli konvoluční maskou a hodnota k se stanoví jako polovina velikosti jádra. Výstupem operátoru v bodě x je součet pronásobených se překrývajících prvků jádra a vstupního obrazu.

Konvoluci lze rozšířit na libovolný počet dimenzí, avšak pro použití na 2D obrazová data stačí definice pro dva rozměry. Analogicky lze pokračovat až do n-té dimenze.

Definice 2.2.2. Odvozením z lineární konvoluce lze získat vzorec pro dvourozměrnou diskrétní konvoluci mezi funkcemi $f(x)$ a $h(x)$ v bodech x, y ve tvaru:

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-l}^{l} f(x - i, y - j)h(i, j) \ ,$$

kde k je polovina velikosti jádra na ose x a l polovina velikosti jádra na ose y. V případě, že je jádro čtvercové, platí $k = l$.

Kdykoliv bude v následujícím textu této bakalářské práce zmíněn pojem konvoluce, je tím automaticky myšlena diskrétní dvourozměrná konvoluce. Jednorozměrná definice nebude nadále relevantní.

2.2.1 Konvoluční maska (jádro)

Konvoluční maska pro dvoudimenzionální konvoluci, taktéž zvaná jádro či kernel, je dvourozměrná matice malých rozměrů, nejčastěji velikosti 3×3 a 5×5. Volba hodnot matice, alternativně nazývaných vahami, zásadně ovlivňuje výsledek konvoluce. Dále bude v textu pojem dvourozměrné jádro zkracován pouze na jádro.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Obrázek 2.2: Příklad konvoluční masky

2.2.2 Filtry

Filtr je trojrozměrná struktura složená z jednoho či více jader. Většina obrazových dat se skládá z více kanálu. Při procesu konvoluce je zapotřebí provést operaci na každý kanál zvlášť. Právě proto zavádíme pojem filtr, který se skládá z více kernelů a kde každý z nich je přiřazen určitému kanálu obrazového vstupu. Filtry obecně slouží k úpravě obrázků, ať už jde o rozmazávání, doostřování, zvýrazňování hran, ...

Častou chybou je záměna pojmů filtr a jádro. Filtry mají dva typy, lineární a nelineární. V rámci této práce se budeme však věnovat pouze lineárním a proto není nutné rozebírat jejich rozdíly.

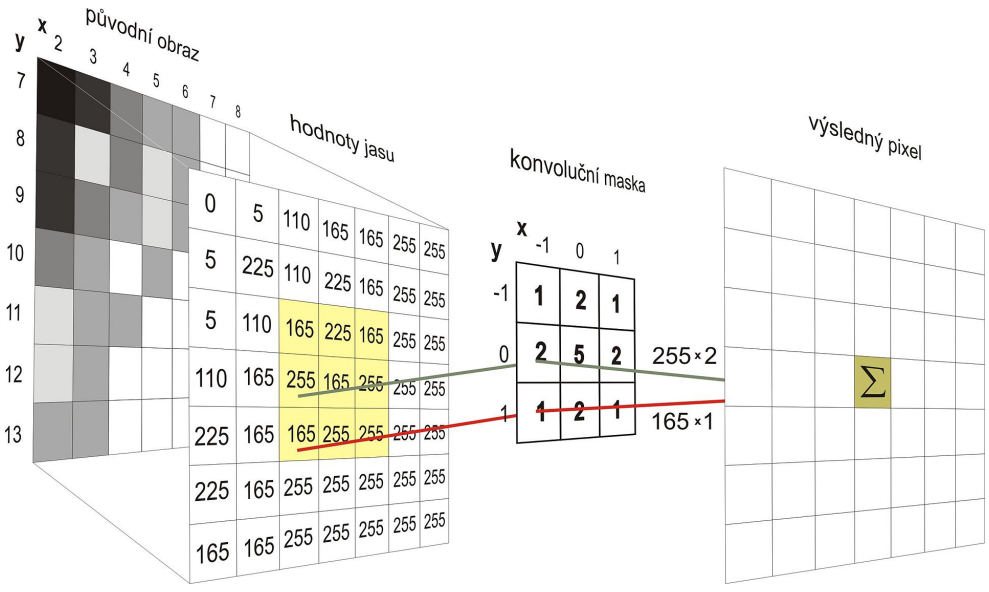
2.2.3 Průběh operace konvoluce

Princip konvoluce je patrný již z definice 2.2.2. Pro snazší představu funguje celý proces na bázi „přikládání“ jádra na vstupní matici. Všechny překrývající hodnoty se vzájemně vynásobí a sečtou do jednoho údaje, který je uložen do nové matice na pozici shodnou se středem přiloženého jádra. Tento proces se iterativně opakuje na všech bodech vstupní matice. Obrázek 2.3 poukazuje na zmíněný proces.

2.2.4 Okrajové podmínky

Při konvoluční operaci je nutné zohlednit tzv. okrajové podmínky [3]. Tento jev nastává tehdy, kdy jádro zasahuje mimo oblast vstupní matice a není možné provádět výpočty s hodnotami, které nejsou definované.

Tento problém lze řešit několika způsoby. Jedním z nich je, že se jádro umísťuje tak, aby nezasahovalo mimo hranice vstupní matice. To má ovšem za následek ořez obrazu. Další metodou je tzv. padding, kdy je vstupní matice vhodně rozšířena tak, aby jádro na okrajích původního vstupu již nezasahovalo mimo. Nově vzniklému místu se musejí přiřadit hodnoty. Toho lze snadno docílit zrcadlením hodnot původní matice přes okraje. Druhým přístupem je vyplnění nulami, což má za následek mírnou nepřesnost konvoluce.



Obrázek 2.3: Grafické znázornění diskrétní konvoluce na obrazových datech [4]

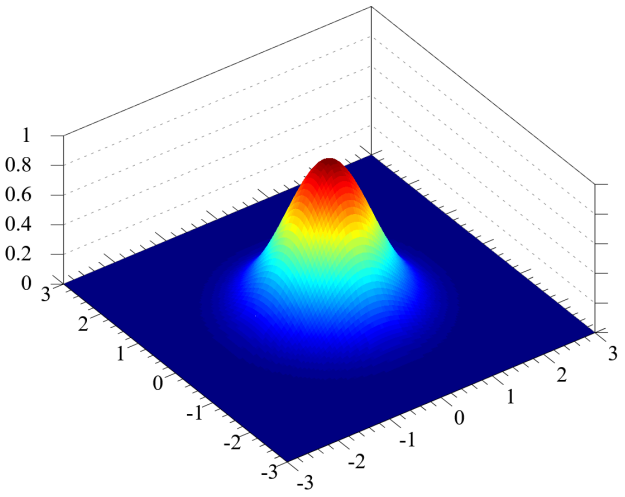
2.3 Gaussovo rozostření

Gaussovo rozostření [5] je rozostřovací/rozmazávací filtr, který využívá Gaussovu funkci pro výpočet hodnot jádra. Při konvoluci obrázku s tímto jádrem dochází k úpravě, která je vnímána jako rozmazání. Gaussův filtr nachází uplatnění při snížení šumu a mimo jiné také při redukci detailů. Dopad Gaussova rozostření je klíčový při procesu hledání dokumentů v obraze.

Definice 2.3.1. Zobecněná gaussova funkce $g(x, y)$ je ve dvourozměrném prostoru definována vztahem:

$$g(x, y) = \frac{1}{2\pi\sigma^2} \exp(-\frac{x^2 + y^2}{2\sigma^2}) \ ,$$

kde σ značí směrodatnou odchylku Gaussova rozdělení a hodnoty x, y jsou vzdálenostmi od středu konvoluční matice na příslušných osách. Čím vyšší je konstanta σ zvolena, tím k většímu rozmazání dojde [6].



Obrázek 2.4: Gaussova funkce ve 2D [7]

2.3.1 Jádro Gaussova rozostření

Jednotlivé hodnoty jádra Gaussova rozostření vycházejí z Gaussovy funkce. Při dosazení vzdáleností na osách x,y od středu jádra do Gaussovy funkce 2.3.1 je vypočtena váha pro příslušející pozici v kernelu. Velikost kernelu závisí na volbě parametru σ z definice 2.3.1. Závislost lze zvolit různými způsoby, je dobré však zohlednit „milník“, který říká, že se hodnoty vzdálenější než $3*\sigma$ od středu matice blíží nule a tím pádem nejsou relevantní. Kvůli tomu nemá velký smysl volit vyšší hodnoty než $3*\sigma$. Nejen pro jádra Gaussova rozostření obecně

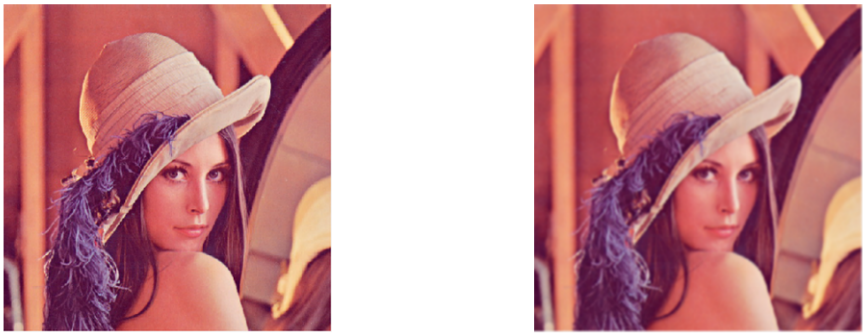
platí, že jsou nejčastěji čtvercového tvaru o lichých velikostech[1]. Pro potřeby této práce je jádro zadefinováno následovně:

Definice 2.3.2. Mějme kernel Gaussova rozostření o velikosti $kS \times kS$ a metodu *Round*, která zaokrouhluje k nejbližšímu celému číslu, kde platí [8]:

$$kS = \text{Round}(2 * (3 * \sigma) + 1) \ ,$$

v opačném případě lze získat hodnotu σ z velikost kernelu kS pomocí:

$$\sigma = 0.3 * ((kS - 1) * 0.5 - 1) + 0.8 \ .$$



(1) (2)

Obrázek 2.5: Ukázka Gaussova rozostření: 1) Originální snímek [9]; 2) Rozostřený snímek

Součet všech aproximovaných vah konvoluční matice Gaussova rozostření není roven přesně jedné. Je vhodné váhy normalizovat celkovým součtem všech prvků matice. Bez normalizace může docházet ke změně jasu vstupních obrazových dat. Příkladem konvoluční masky pro Gaussův filtr je matice na obrázku 2.6.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Obrázek 2.6: Příklad jádra Gaussova rozostření

[1] Tato skutečnost plyne z techniky konvoluce a symetričnosti Gaussovy funkce

2.4 Převod barevného obrazu do šedotónového

Mezi nejzákladnější úpravy obrazových dat patří převod z barevného RGB prostoru na stupně šedi [10], při kterém dochází k redukci třech kanálu na jeden. Způsobu, jak převodu docílit, je několik. Je však dostačující zmínit pouze ten nejpoužívanější, který najde uplatnění i v rámci této práce. Základem je vypočítat hodnotu intenzity nově vzniklého obrazového bodu. S ohledem na citlivost lidského oka na rozdílné vlnové délky vznikl vzorec, který je daný vztahem:

$$I = 0,2125R + 0,7154G + 0,0721B \ ,$$

kde R značí intenzitu červené, G zelené a B modré barvy. Výsledkem výpočtu je hodnota I, která udává světelnou intenzitu nově vzniklého bodu na stupnici šedi v rozsahu 0-255.

2.5 Cannyho detektor hran

Cannyho detektor hran [11], dále jen Canny, je vícekrokový algoritmus sloužící, jak už z názvu plyne, k detekci hran v obrazových datech. Více fází umožňuje kvalitní rozpoznávání různých druhů hran a díky tomu je Canny jedním z nejpopulárnějších a nejpoužívanějších algoritmů na detekci hran. Ukázkový výstup Cannyho detektoru hran lze vidět na obrázku 2.9. Průběh se dělí na následující fáze:

1. **Snížení šumu:** Algoritmy na detekci hran jsou obecně náchylné na šum v obraze a Canny není výjimkou. Šum by mohl být chybně vyhodnocován jako hrana a je tedy žádoucího se ho zbavit. Nejčastěji se na odstranění šumu v případě Cannyho používá Gaussovo rozostření s jádrem o velikost 5×5 a směrodatnou odchylkou $\sigma = 1.4$.

2. **Kalkulace gradientu:** Při kalkulaci gradientu dochází k detekci intenzity hran a jejich směru, který lze určit pomocí výpočtu gradientu. Hrany se nacházejí tam, kde dochází ke změně intenzity obrazových bodů. K detekci lze využít Sobelův filtr zvlášť v horizontálním a vertikálním směru, který je definován jádry:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix},$$

   kde K_x značí první derivaci v horizontálním a K_y ve vertikálním směru. Intenzitu hrany a její směr lze snadno spočítat pomocí:

$$G = \sqrt{I_x^2 + I_y^2}, \quad \Theta = \arctan\left(\frac{I_y}{I_x}\right) \ ,$$

   kde G značí intenzitu a Θ úhel (směr) gradientu hrany.

3. **Non-maximum suppression:** Předchozí krok zapříčiní izolaci hran, které jsou však rozdílných šířek a intenzit. Konceptem detektorů hran je nalezení pouze nejnutnějších bodů, které tvoří obrysy. Proto je žádoucí použít techniku non-maximum suppression, která všechny hrany zúží na šířku jednoho obrazového bodu. Algoritmus non-maximum suppression využívá vypočtených hodnot z předchozího kroku, které vypovídají o intenzitě (G) jednotlivých obrazových bodů a směr gradientu hran (Θ), ve kterých se dané body nacházejí. Proces běží v následujících krocích:

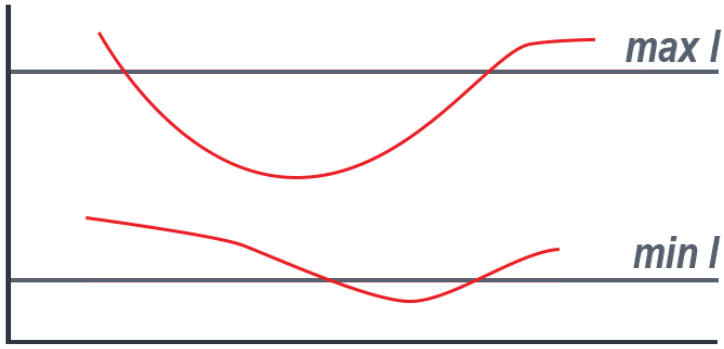
   a) Porovná intenzitu každého bodu s okolím v obou směrech gradientu.

   b) Pokud je intenzita daného bodu v rámci okolí nejvyšší, je zachován. V opačném případě je zahozen.

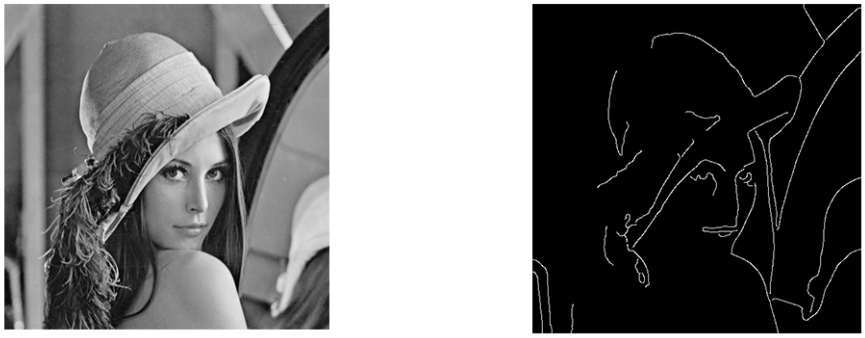
   Postupnou aplikací těchto dvou kroků vznikne obraz, ve kterém jsou pouze hrany o šířce jednoho obrazového bodu. Na obrázku 2.7 reprezentují červené čtverce body s největší intenzitou a šedé gradient kolem nich. Algoritmus zachová pouze červené body.

Obrázek 2.7: Směry gradientů hran v obraze

4. **Prahování s hysterezí:** Posledním krokem Cannyho detektoru hran je prahování s hysterezí, které zajistí, aby byly zachovány pouze části obrazu, které jsou skutečně hranami. Je nutné zvolit dva prahy intenzity obrazových bodů, jmenovitě *minI* a *maxI*. Body, které mají vyšší intenzitu než *maxI* jsou považovány za hrany. Naopak body s nižší intenzitou než *minI* jsou vyřazeny. O zbytku bodů, které leží mezi prahy se rozhodne tak, že pokud daný bod sousedí s potvrzenou hranou, je zachován. V opačném případě je zahozen. Grafické znázornění prahování s hysterezí lze vidět na obrázku 2.8.



Obrázek 2.8: Prahy intenzity při selekci hran



(1) (2)

Obrázek 2.9: Cannyho detekce hran: 1) Originální snímek [9]; 2) Detekované hrany

2.6 Detekce kontur

Na výstupu Cannyho detekce hran jsou obrazová binární data, kde jedna barva zastupuje hrany a druhá zbylou plochu. Pro jakékoliv operace s hranami je zapotřebí lepší reprezentace a proto je nutné zavedení pojmu kontura.

2.6.1 Kontura

Konturu lze definovat jednoduše jako křivku spojující souvislé body podél obvodu mající stejnou barvu nebo intenzitu. Kontury představují nástroj pro analýzu tvarů, detekci objektů a jejich rozpoznávání. V knihovně OpenCV

jsou kontury reprezentovány jako vektory bodů, kde každý bod je definován souřadnicemi na osách x a y [12].

2.6.2 Algoritmus Suzuki85

K převodu hran z obrazových dat na kontury se nejčastěji využívá Suzukiho algoritmu [13] a například v knihovně OpenCV je zastoupen knihovní funkcí `findContours(...)`.

2.7 Transformace

Grafické transformace slouží k úpravě obrazových dat do různých podob, ať už jde o posunutí, otáčení, zkosení, nebo v rámci této práce důležité perspektivní transformování. Celá operace je postavena na násobení obrazových dat vhodně zvolenou transformační maticí.

2.7.1 Transformační matice

Základem dvourozměrné transformační matice je matice jednotková, o které je všeobecně známo, že při násobení s libovolnou jinou maticí nedojde k žádné změně. Jednotlivé transformace nahrazují hodnoty v jednotkové matici tak, aby při vynásobení takto vytvořené matice s libovolnou jinou maticí došlo k požadované operaci. [14]. Příkladem může být transformace posunutí, kde je míra posunu určena vektorem $\vec{p} = (t_x, t_y)$, přičemž transformační matice vypadá následovně:

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} .$$

2.7.2 Získání transformační matice perspektivy

Transformační matici perspektivy lze vypočítat podle vzorce, který v případě definice 2.7.1 vychází z implementace knihovny OpenCV.

Definice 2.7.1. Transformační matici perspektivy o rozměru 3×3 lze získat ze zdrojové čtveřice bodu *src* a cílové čtveřice *dst* následovně:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = trans\_matice \times \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} ,$$

kde $dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$.

2.7.3 Perspektivní transformace

Perspektivní transformaci lze provést za předpokladu, že je k dispozici vstupní a transformační matice. Jedná se tedy o proces, při k kterém dochází k aplikaci transformační matice na libovolnou vstupní matici, potažmo obrazová data.

Definice 2.7.2. Mějme vstupní matici *src*, výstupní matici *dst* a transformační matici M typu 3×3, pak lze provést perspektivní transformaci pomocí:

$$dst(x, y) = src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right) \ .$$

2.8 Laplaceův operátor

Laplaceův operátor, zkráceně Laplacián, je v oblasti grafiky dvourozměrná izotropická míra druhé prostorové derivace obrazových dat. Laplacián zvýrazňuje oblasti, ve kterých dochází k prudkým rozdílům v intenzitě obrazových bodů. Díky této vlastnosti se často používá na detekci hran. [15].

Definice 2.8.1. Laplaceův operátor $L(x, y)$ obrazu s intenzitou obrazových bodů $I(x, y)$ je definován vztahem:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \ .$$

V oblasti zpracování obrazu je Laplacián aproximován diskrétní konvolucí. Nejčastěji používaná jádra pro 4-sousedství, resp. 8-sousedství o velikost 3×3, kde údaj sousedství vypovídá a počtu sousedících pixelů se středem jádra. Zmíněná jádra vypadájí následovně:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, resp. \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \ .$$

2.8.1 Výpočet míry ostrosti

Laplacián 2.8 zle využít mimo jiné i při detekci míry rozostření obrazu. Metoda se nazývá variací Laplaciána [16] a je založena na jednoduchém principu, při kterém dochází ke konvoluci Laplaciánova filtru s požadovaným obrazem, ze kterého je pak následně spočítána variance. Hodnota vzešlá z výpočtu vypovídá o relativní míře rozostření zpracovaného obrazu.

Základní předpoklad je, že pokud se v obraze nachází velká variace, tak se v něm nalézá široké spektrum hodnot – indikující ostré i neostré hrany

(přechody), které se nalézají v normálním zaostřeném obraze. Pokud se ale v obraze nalézá velmi malá variace, tak je v něm i malé spektrum hodnot, indikující malý počet hran. Čím více je obraz rozostřený, tím méně ostrých hran se v něm nalézá [17]

Kapitola 3

Použité technologie

Před začátkem samotného vývoje je potřeba zvolit vhodné technologie pro vývoj. Aplikace cílí na platformu iOS, u které má vývojář možnost volby z několika programovacích jazyků. Dále je nutné vybrat vývojové prostředí a frameworky, které budou ulehčovat práci s PDF soubory, obrazem, vzdáleným přístupem, grafickým rozhraním, ...

3.1 Knihovna pro práci s obrazem

Veškeré získané vědomosti týkající se nejen práce s obrazem je nutné převést z teoretické hladiny do implementační. Ačkoliv veškeré zmíněné techniky a nástroje lze implementovat takříkajíc „from the scratch“[2], je lepší využít již existující ověřené knihovny, které danou funkcionalitu nabízejí. Možností je mnoho, od OpenCV, Dlib, BoofCV až po vlastní implementaci od firmy Apple inc. zvanou VisionKit [18]. Pro potřeby této práce byla však vybrána kniha OpenCV z několika důvodů. Nejen, že se jedná o jeden z nejznámějších frameworků na poli zpracování obrazu, ale vyniká i efektivností, variabilitou a širokou podporou, neboť právě tato knihovna nativně podporuje vývoj pro námi zvolené cílový operační systém, iOS.

3.1.1 OpenCV

OpenCV [19] je otevřená multiplatformní knihovna zaměřená na počítačové vidění a zpracování obrazu. Obsahuje více jak 2500 optimalizovaných algoritmů, které jsou vnitřně implementovány v relativně efektivních jazycích, jako je například C++. V rámci celé práce bude na všechny grafické operace používána výhradně knihovna OpenCV.

[2] Český překlad zní: „Od základu“

3.1.2 VisionKit

VisionKit [18] je framework, který umožňuje využít systémový iOS skener dokumentů. Vychází z knihovny na zpracování obrazu zvané Vision [18] a soustředí se výhradně na skenování dokumentů. Framework vyšel s příchodem iOS verze 13 a to ho činní zcela novým a nezažitým. Zároveň nedisponuje tak velkou variabilitou, jako například algoritmus navržený pomocí prvků z knihovny OpenCV. Právě kvůli tomuto důvodu nebyl vybrán v rámci této práce, ačkoliv do budoucna se může jednat a velmi atraktivní řešení.

3.2 Použité programovací jazyky

3.2.1 Swift

Swift je univerzální, multi-paradigmatický, open-source programovací jazyk postavený na moderním přístupu k bezpečnosti, výkonu a softwarovým návrhovým vzorům. Swift má představovat náhradu za C-based jazyky (C, C++ a Objective-c) a tím pádem musí být těmto jazykům schopný rychlostně konkurovat. Swift je mimo jiné objektově orientovaný a podporuje také funkcionální i imperativní programování. V rámci této práce bude použit při tvorbě iOS aplikace. [20]

3.2.2 Objective-C++

Objective-C++ není sám o sobě programovacím jazykem, nýbrž spíše možností kombinovat Objective-C a C++ kód v jednom zdrojovém souboru. Objective-C je jazyk, který se používal pro vývoj iOS aplikací až do té doby, dokud jej nenahradil modernější Swift. Spojení Objective-C a C++ kódu v jednom souboru je klíčové při využívání knihovny OpenCV v rámci vývoje iOS aplikací.

3.3 Podpora vzdáleného serveru

Jedním z cílů práce je podpora vzdáleného serveru, na který bude možné ukládat veškeré naskenované dokumenty. V průběhu práce byl po dohodě s vedoucím práce vybrán framework CloudKit.

3.3.1 CloudKit

CloudKit [18] je framework vyvíjený firmou Apple Inc., který poskytuje API pro přesun dat mezi zařízeními s iOS a Apple iCloud serverem. CloudKit není náhradou existujících datových objektů v aplikaci, ale pouze poskytuje doplňkové služby pro správu přenosu dat. Pokud je zapotřebí ukládat data do soukromého prostoru, musí mít uživatel validní iCloud účet. V případě, že jsou

data ukládána do veřejného prostoru, není účet potřeba. Záznamy v CloudKitu jsou páry typu klíč-hodnota uložené ve slovníku. Framework CloudKit byl vybrán z toho důvodu, že umožňuje poměrně snadnou integraci vzdáleného serveru. Jediným omezením, které ovlivní i tuto práci, je nutnost vlastnění placeného vývojářské účtu. Bez něj není možné propojení s CloudKitem plně zprovoznit. Ovšem pro vydání jakékoliv aplikace na AppStore je taktéž zapotřebí placeného účtu a to znamená, že by byl automaticky zpřístupněn i CloudKit.

3.4 Vývojové prostředí

Jako vývojové prostředí byl zvolen program Apple Xcode, který obsahuje vývojářské nástroje pro vývoj iOS a Mac OS aplikací. Nejnovější verze, nesoucí pořadové číslo 11, je dostupná zdarma ke stažení na Mac App Store a je kompatibilní pouze s Mac OS. Součástí IDE je například iOS Simulator, který slouží k testování aplikací bez nutnosti připojení fyzického zařízení a také nepříliš proslulý Interface Builder, jehož údělem má být snadný a intuitivní návrh uživatelského rozhraní.

3.5 Možnosti tvorby GUI

S využitím Xcode IDE lze navrhovat grafické rozhraní pro iOS aplikace čtyřmi rozdílnými způsoby:

- **iOS Storyboards:** Vizuální nástroj, které je postaven na konceptu storyboardů, se nazývá Interface Builder a je určen pro tvorbu GUI. Vyznačuje se poměrně snadným ovládáním a intuitivním prostředím. Nevýhodou je však obtížné verzování a omezení při tvorbě vlastních grafických prvků.
- **NIBs (XIBs):** Každý NIB soubor obsahuje data jednoho obrazového prvku a lze jej tvořit ve Storyboardu. Jedná se tedy o nástroj pro tvorbu GUI a předchůdce Storyboardů.
- **Vlastní kód:** Tvorba GUI pomocí kódu nevyužívá grafické nástroje a veškeré prvky, animace a přechody jsou tvořeny čistě pomocí programování. To přináší téměř neomezené možnosti při tvorbě GUI, ovšem samotný proces návrhu je o poznání složitější než u předchozích dvou možností. Velkou výhodou je také bezproblémová správa verzí.
- **SwiftUI:** SwiftUI je určitým kompromisem mezi storyboardy a vlastním kódem. Umožňuje návrh GUI deklarativní Swift syntaxí a jedná se o nejnovější způsob návrhu. SwiftUI nebude v této práci použit, a to z toho důvodu, že je zatím na počátku svého vývoje je výhodnější se držet odzkoušených metod.

Nelze říci, který z nástrojů je nejlepší, a který naopak nejhorší. Na každé specifické použití se hodí nástroj jiný. V případě této práce je GUI tvořeno vlastním kódem a to z několika důvodů. Tím hlavním je fakt, že většina grafických prvků skenovací aplikace musí být vytvořena na míru. Dalším důvodem je bezproblémové verzování a přehlednost, neboť veškerá deklarace prvků GUI je na jednom místě, a to v kódu.

3.5.1 UIKit

Framework UIKit poskytuje potřebnou GUI infrastuktruru pro iOS i tvOS aplikace. Nabízí architekturu zvanou *View and Window Architecture*, která slouží k implementování uživatelského rozhraní. Dále poskytuje zpracování událostí (např. Multi-Touch), podporu animací, zobrazení dokumentů, kreslení, ...

3.5.2 SnapKit

SnapKit [21] je framework třetí strany a umožňuje snadno definovat omezení jednotlivých grafických prvků, včetně pozice a velikosti. Jedná se o intuitivnější interpretaci frameworku Auto Layout od společnosti Apple.

3.6 Ostatní frameworky

3.6.1 ReactiveSwift

ReactiveSwift je framework třetí strany, který nabízí primitiva postavená na konceptu proudů hodnot v čase. Tento koncept je často nazýván reaktivním programováním a jedná se o moderní a populární paradigma. Mezi nejpoužívanější základní datové typy ReactiveSwiftu patří `Signal`, `SignalProducer` a `MutableProperty`. Při použití reaktivní přístupu u prvků grafického rozhraní je zapotřebí frameworku ReactiveCocoa, který obaluje různé aspekty Cocoa frameworků deklarativními ReactiveSwift primitivy. [22]

3.6.2 PDFKit

PDFKit je framework pro iOS a MacOS spravovaný firmou Apple a slouží pro snadnou práci se soubory formátu PDF, které lze díky němu snadno vytvářet, upravovat a zobrazovat. [18]

3.6.3 Realm

Realm je cross-platform mobilní databáze, která běží přímo uvnitř zařízení a je podporována i při vývoji iOS aplikací. Nejedná se pouze o wrapper okolo nativního Core Data frameworku. V rámci této práce je Realm využíván při ukládání a správě dokumentů. [23]

Kapitola 4

Analýza

4.1 Popis aplikace

Cílem této práce je vytvořit aplikaci pro mobilní operační systém iOS, která umožní snadné skenování dokumentů pomocí integrovaného fotoaparátu v mobilním zařízení. Uživatel bude schopen v krátkém sledu operací převést libovolný dokument do elektronické podoby. Jednoduchost této aplikace podporuje fakt, že bude mít automatický režim, při kterém je dokument ve scéně samočinně detekován a bez nutnosti dalších zásahu uživatele připraven k uložení. Samozřejmě je uživateli k dispozici sada nástrojů, se kterou je možné u daného skenu měnit ořez, barevný prostor a rotaci. Součástí aplikace je galerie naskenovaných dokumentů, včetně možnosti zobrazení, mazání a exportu do PDF. Veškerá data mohou být ukládána na aplikací podporovaný iCloud Drive.

4.2 Konkurenční aplikace

Tato práce se soustředí zejména na analýzu metod zpracování obrazu a návrh algoritmu, který bude sloužit ke skenování dokumentů pomocí kamery mobilního fotoaparátu. Výstupem práce je tedy aplikace, která spíše demonstruje navržený algoritmus. To ovšem neznamená, že se nebude jednat o plnohodnotný nástroj určený pro snadné skenování dokumentů. I přes absenci nutnosti originality je vhodné provést průzkum několika konkurenčních aplikací, analyzovat jejich spolehlivost a uživatelské rozhraní včetně ovládání. Vzhledem k tomu, že se tato práce zaměřuje na platformu iOS, je více než dostačující provést průzkum pouze v kontextu této platformy.

4.2.1 Scanner Mini

Scanner Mini je jednou z nejzajímavějších skenovacích aplikací na App Store. Zejména díky tomu, že nabízí širokou škálu funkcí, mezi které patří automa-

tický režim skenování, inteligentní úprava dokumentů a dokonce v placené verzi i možnost detekce textu. Scanner mini byl velkou inspirací při tvorbě této práce.

- **Vývojář:** Readdle Inc.
- **Hodnocení na App Store:** 4,8/5
- **Zdarma/Placená:** Obojí

4.2.2 ScanPro

Aplikace ScanPro je taktéž dobrým nástrojem při skenování dokumentů a zpřístupňuje uživateli velké množství různorodých nastavení. Uživatelské rozhraní je velmi dobře zpracované, avšak částečně se vymyká vzhledu základních iOS aplikací.

- **Vývojář:** doo GmbH
- **Hodnocení na App Store:** 4,8/5
- **Zdarma/Placená:** Obojí

4.2.3 Scanner App

Scanner App je velmi podobný aplikaci ScanPro, s tím rozdílem, že využívá jednodušší flat-design. Sada funkcí je přirovnatelná k ostatním, již zmíněným aplikacím. Nevýhodou je velké množství reklam v bezplatné verzi.

- **Vývojář:** BP Mobile LLC
- **Hodnocení na App Store:** 4,6/5
- **Zdarma/Placená:** Obojí

4.2.4 Porovnání konkurenčních aplikací

U výše zmíněných aplikací je vhodné provést porovnání klíčových funkcí, a to jak v bezplatné, tak i v placené verzi. Při návrhu skenovací aplikace by bylo vhodné, aby nabízela alespoň podobnou funkcionalitu, jako bezplatné verze konkurenčních aplikací. Mezi porovnávané parametry patří automatický režim, možnost skenování více stránek, skenování textu, podpora vzdáleného serveru, export do PDF a přítomnost reklam.

Tabulka 4.1: Porovnání funkcionalit bezplatných verzí konkurenčních aplikací

	Auto	Multi-Page	OCR	Cloud	PDF	Reklamy
Scanner Mini	Ano	Ano	Ne	Ano	Ne	Ne
ScanPro	Ano	Ano	Ne	Ne	Ano	Ne
Scanner App	Ano	Ne	Ne	Ne	Ano	Ano

Tabulka 4.2: Porovnání funkcionalit placených verzí konkurenčních aplikací

	Auto	Multi-Page	OCR	Cloud	PDF	Reklamy
Scanner Mini	Ano	Ano	Ano	Ano	Ano	Ne
ScanPro	Ano	Ano	Ano	Ano	Ano	Ne
Scanner App	Ano	Ano	Ano	Ne	Ano	Ne

4.3 Podporovaná zařízení

Podporovaná zařízení budou mobilní telefony z řady Apple iPhone s operačním systémem iOS 13.0 a novějším. Podpora pouze nejnovějšího operačního systému se může zdát poměrně radikální, ovšem podle statistiky [24] ze dne 22.05.2020 je iOS verze 13.0 a vyšší nainstalována na 77,88% zařízení. Z toho plyne fakt, že je zvolená podpora naprosto dostačující a v případě potřeby by bylo možné aplikaci s malými zásahy upravit tak, aby byla kompatibilní i se staršími verzemi.

4.4 Podporované formáty dokumentů

Aplikace bude v základu podporovat standardní formáty dokumentů a je navržená tak, aby byla nabídka formátů snadno rozšiřitelná. Formáty, které byly vybrány, jsou následující:

- DIN A5
- DIN A4
- DIN A3
- US business card (US vizitka)
- EU business card (EU vizitka)

4.5 Funkční a nefunkční požadavky

4.5.1 Funkční požadavky

- **Automatický režim skenování:** Aplikace nabízí režim, při kterém dochází k automatické detekci okrajů dokumentu ve scéně, jeho následné transformaci, oříznutí a další postprodukci.
- **Manuální režim skenování:** V určitých případech může nastat situace, kdy nelze použít automatický režim a je nutné označit dokument ve scéně ručně. K tomu slouží manuální režim.
- **Podpora více stránkových dokumentů:** Aplikace podporuje skenování více stránek dokumentů, které jsou po dokončení procesu skenování složeny do jednoho celku.
- **Volba formátu dokumentu:** Uživatel si před procesem skenování může u každé stránky dokumentu vybrat její formát.
- **Úprava oblasti dokumentu:** Uživatel má možnost upravit hrany dokumentu, které mohou být nevhodně detekovány automatickým režimem, nebo špatně zvoleny v manuálním módu samotným uživatelem.
- **Změna barevného prostoru:** U jednotlivých stránek naskenovaného dokumentů je možné přecházet mezi barevným a šedotónovým prostorem.
- **Rotace dokumentu:** Aplikace umožňuje provádět rotace s každou stránkou dokumentu zvlášť. Rotování probíhá ve skocích o 90 stupňů.
- **Uložení dokumentu do galerie:** Jednotlivé naskenované dokumenty lze pojmenovat a uložit do lokální galerie.
- **Zobrazení dokumentu** Veškeré naskenované dokumenty je možné v lokální galerii zobrazovat.
- **Export a správa dokumentů:** V galerii lze veškeré dokumenty odstraňovat a exportovat, přičemž při exportu je využit základní iOS sharesheet.

4.5.2 Nefunkční požadavky

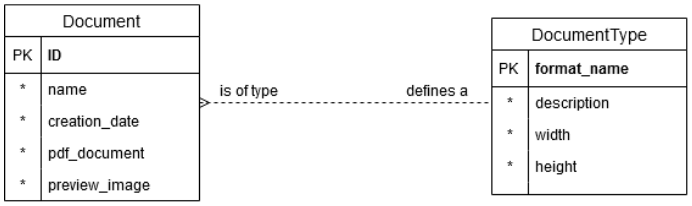
- **Aplikace pro iOS:** Aplikace je navržena pro operační systém iOS s podporou verze iOS 13.
- **Podpora vzdáleného serveru:** Veškeré naskenované dokumentu je možné ukládat na vzdálený server, resp. iCloud Drive.
- **Intuitivní uživatelské rozhraní:** GUI aplikace je jednoduché a intuitivní, inspiruje se v oficiálních iOS guidelines [25].

4.5.3 Business procesy

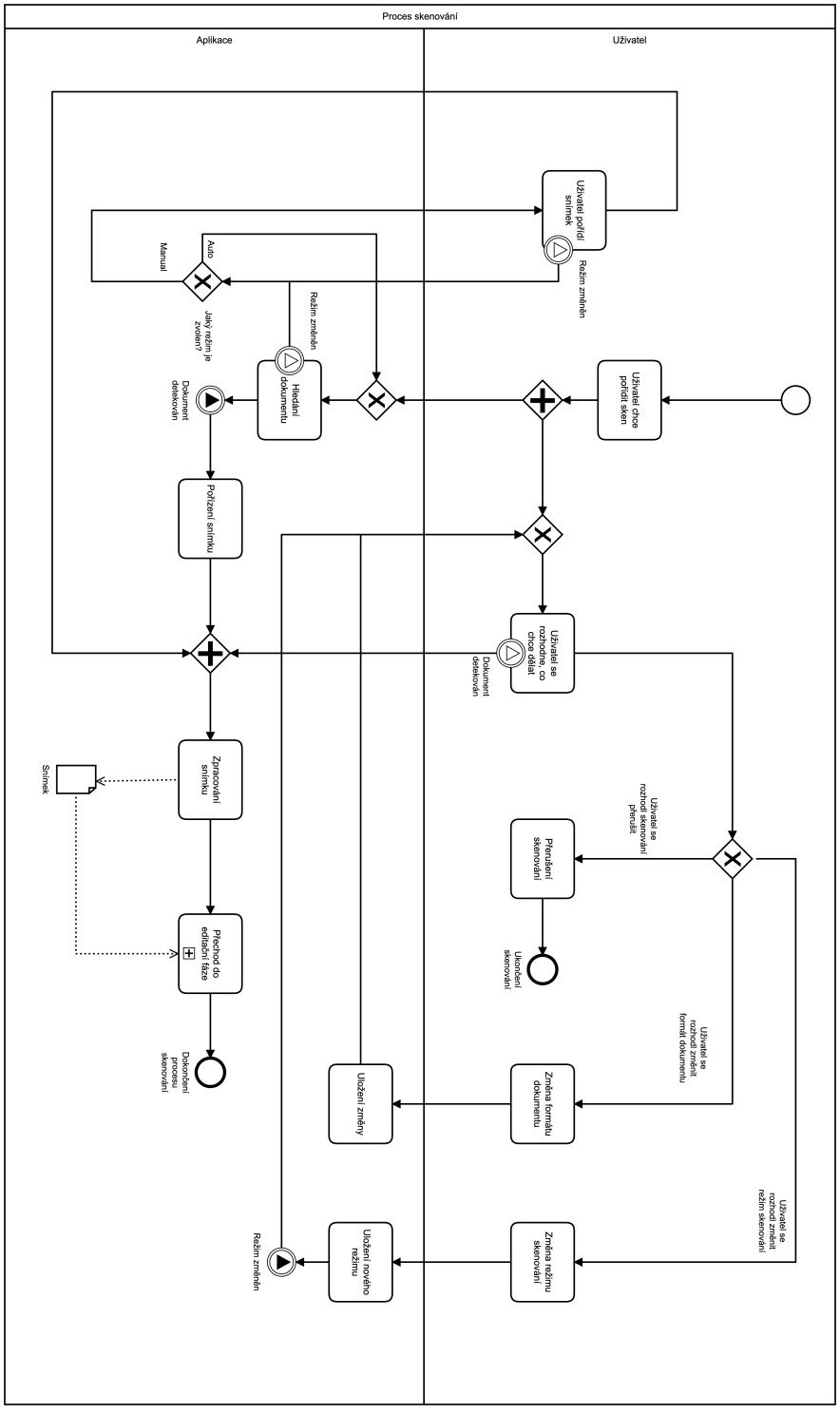
Při návrhu aplikace je dbáno na zachování jednoduchosti ovládání ze strany uživatele, který v co nejmenším počtu kroků dokáže naskenovat libovolný dokument. Z toho důvodu se v aplikace nevyskytuje mnoho procesů, avšak ty nejdůležitější jsou zachyceny na BPMN diagramech 4.2 a 4.3. Verze v plném rozlišení je dostupná na přiloženém CD.

4.5.4 Databáze dokumentů

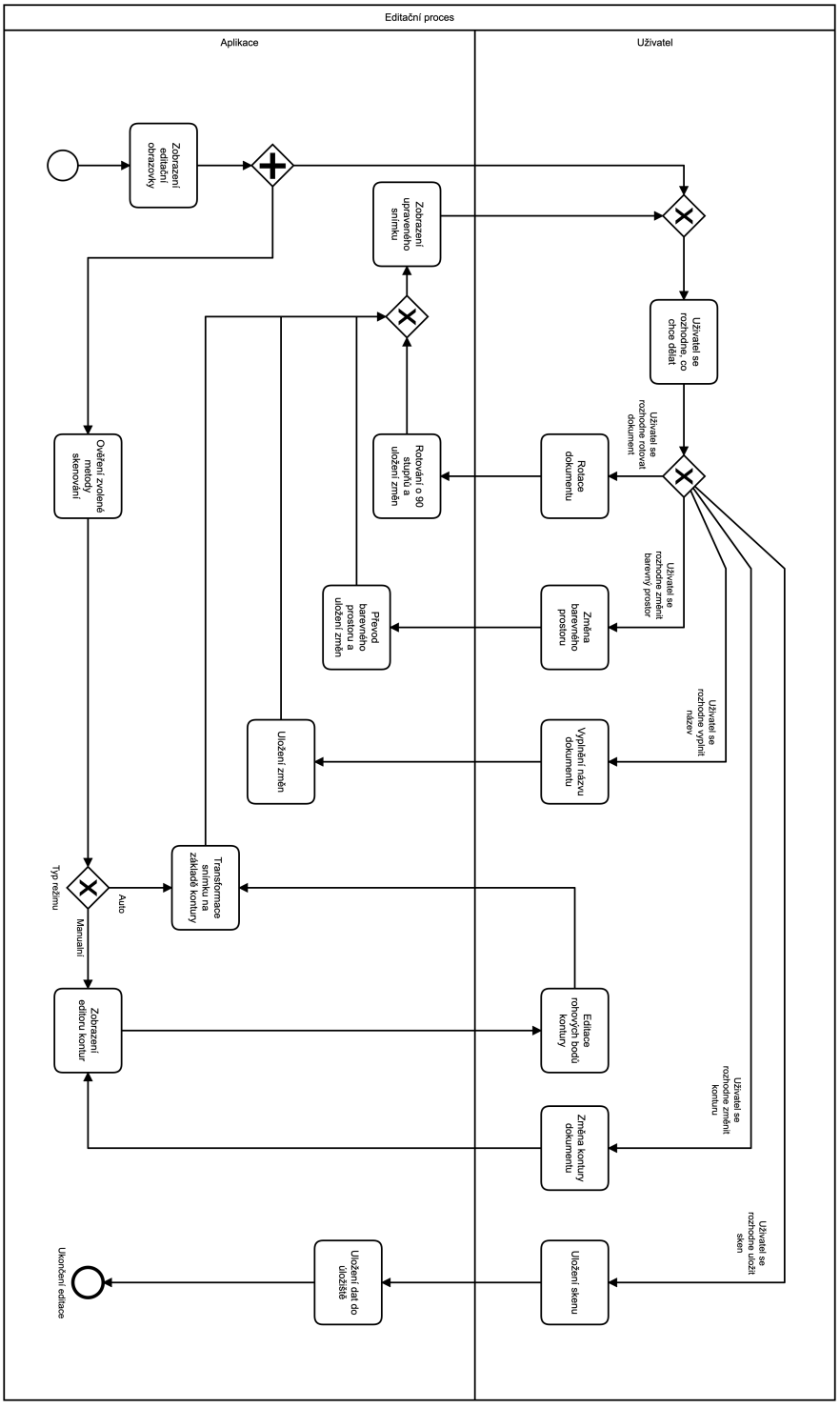
Veškeré pořízené dokumenty jsou v rámci aplikace ukládány do Realm databáze 3.6.3 a díky tomu jsou trvale uloženy i po restartu aplikace. Všechny informace o databázovém modelu plynou z konceptuálního diagramu na obrázku 4.1.



Obrázek 4.1: Konceptuální model databáze dokumentů



Obrázek 4.2: BPMN Diagram procesu skenování



Obrázek 4.3: BPMN Diagram procesu editování skenu

Kapitola 5

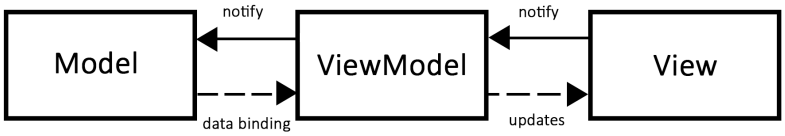
Návrh

V této kapitole se práce zaobírá návrhem grafického rozhraní výsledné aplikace a výběrem návrhového vzoru, podle kterého se postupuje při samotném vývoji. Nedílnou součástí aplikace je skenovací algoritmus, jehož logika byla však z velké míry popsána v kapitole 2. Z toho plyne, že není nutné provádět další návrh a celkový rozbor skenovacího algoritmu včetně implementace je popsán v kapitole 6.

5.1 Architektura

Model-View-ViewModel [26], zkráceně MVVM je strukturální návrhový vzor, který rozděluje objekty do tří kategorií. Diagram MVVM lze vidět na obrázku 5.1.

- **Model** uchovává data aplikace, nejčastěji se jedná o structy nebo jednoduché třídy.
- **View** slouží k zobrazení grafické části aplikace včetně ovládacích prvků.
- **ViewModel** obsahuje veškerou aplikační logiku, transformuje informace z modelu v hodnoty, které může vrstva view zobrazit.



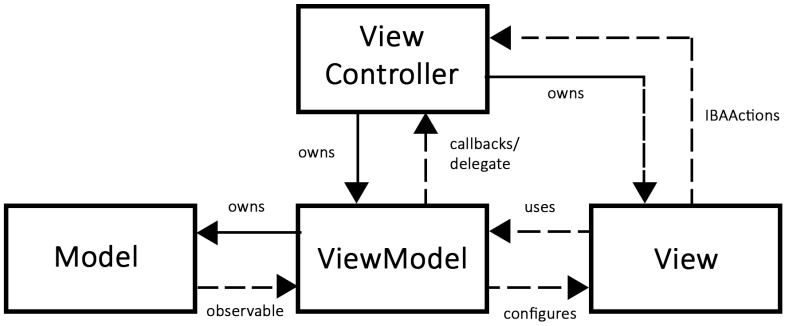
Obrázek 5.1: Model-View-ViewModel diagram

Vyvíjená aplikace bude využívat právě zmíněný MVVM návrhový vzor, který je jedním z nejpožívanějších vzorů při vývoji pro iOS. Dalším známým

vzorem je například Model-View-Controller, který nachází taktéž využití v iOS aplikacích. MVVM je však modernější a vhodnější pro tuto práci.

5.1.1 MVVM v iOS

Jedním ze stavebních kamenů tvorby iOS aplikací jsou tzv. viewcontrollery, které je nutné začlenit do MVVM návrhového vzoru. Upravený MVVM diagram, který bere v potaz viewcontrollery, je vidět na obrázku 5.2.



Obrázek 5.2: diagram Model-View-ViewModel pro iOS [27]

5.2 Uživatelské rozhraní

Návrh uživatelského rozhraní je nedílnou součástí tvorby aplikace. V této sekci budou rozebrány dílčí části návrhu uživatelského rozhraní, podle kterého se implementují jednotlivé obrazovky aplikace. Při návrhu je kladen důraz zejména na jednoduchost a intuitivnost. Celá aplikace je postavena na myšlence, že se bude jednat o jednoduchý nástroj, se kterým je uživatel schopen v malém počtu kroků zdigitalizovat libovolný dokument za použití svého mobilního telefonu. Design se snaží volně inspirovat u oficiálních iOS Human Interface Guidelines [25].

5.2.1 Galerie dokumentů

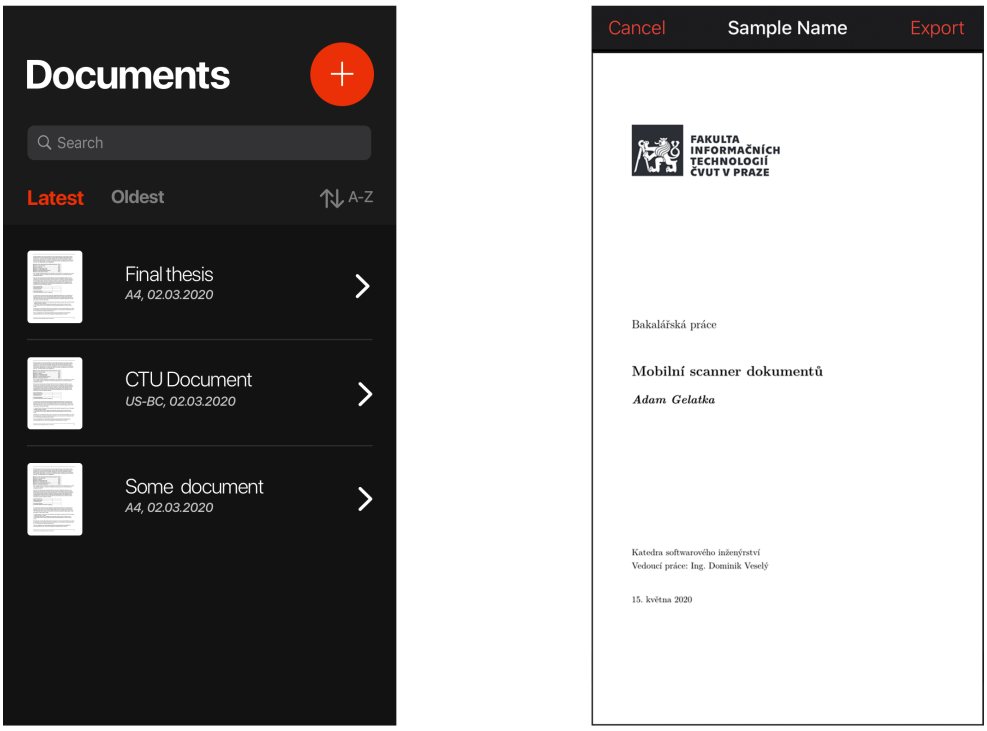
Obrazovka s galerií slouží ke správě a zobrazení všech naskenovaných dokumentů. Vzhled je částečně inspirován systémovou aplikací pro prohlížení souborů a lze jej vidět na obrázku 5.3. Obrazovka je rozdělena na dvě sekce, které jsou popsány v následujících bodech:

- **Horní ovládací panel:** V horním panelu je tlačítko pro provedení nového skenu, dále pak pole pro vyhledávání dle názvu dokumentu a panel s abecedním či časovým tříděním.

- **Panel dokumentů:** Druhá část obrazovky umožňuje procházení všech dokumentů, jejich export, mazání a zobrazení. Wireframe galerie dokumentů je viditelný na obrázku 5.3. Nabídka možností správy jednotlivých dokumentů využívá nativních nástrojů systému iOS pro správu položek v seznamu a lze ji vidět na obrázku 5.4. Nabídka se zpřístupní jednoduchým tažením zvolené položky směrem vlevo.

5.2.2 Zobrazení dokumentů

Veškeré dokumenty se zobrazují na nové obrazovce, viditelné taktéž na obrázku 5.3. Jedná se pouze o primitivní obrazovku, která slouží pouze k zobrazování dokumentů, včetně podpory více stránek. V horním panelu se nachází tlačítko pro ukončení náhledu a tlačítko pro export dokumentu.



Obrázek 5.3: Wireframe galerie dokumentů: 1) Galerie dokumentů; 2) Zobrazení dokumentů

Obrázek 5.4: Nabídka možností správy dokumentů

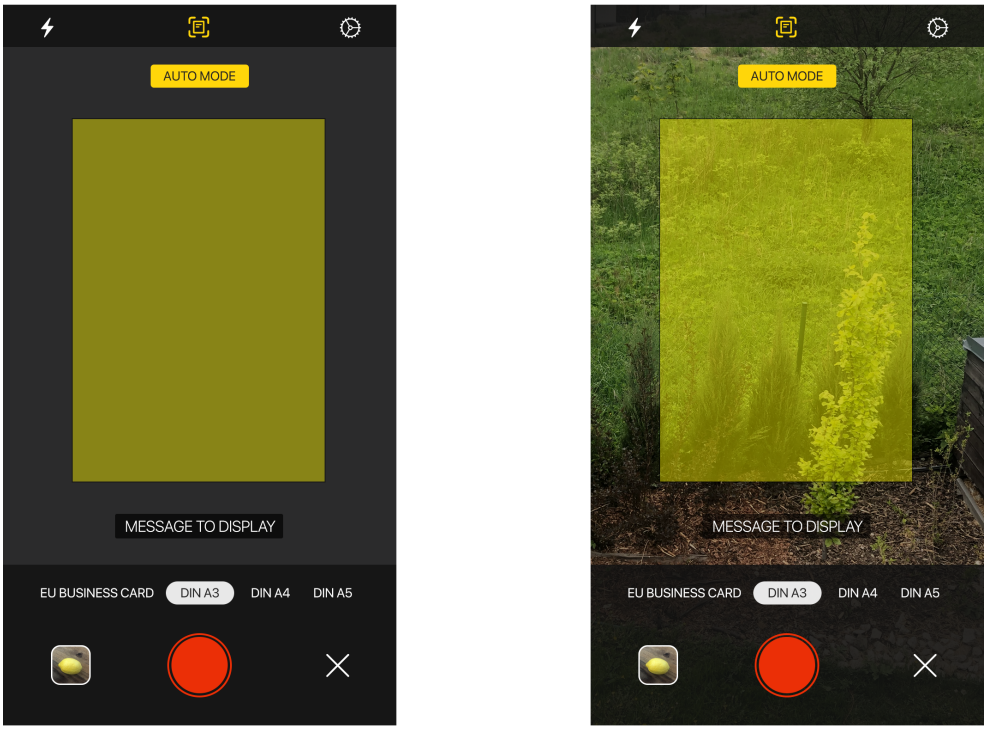
5.2.3 Skenovací obrazovka

Skenovací obrazovka slouží ke skenování dokumentů pomocí kamery mobilního telefonu. V rámci zachování čistoty designu je většina prvků poloprůhledná a návrh lze vidět na obrázku 7.2. Obrazovka se dělí na následující části:

- **Horní ovládací panel:** V horním panelu se nacházejí tlačítka pro přepínání blesku, změnu režimu skenování a otevření nastavení. Tlačítko nastavení však nemá prozatím využití a v budoucích verzích, které nebudou součástí této práce, může být případně využito.
- **Ukazatel stavu blesku a typu režimu:** Pod horním panelem je umístěn ukazatel, který informuje uživatele v případě, že je zapnutý blesk, nebo zvolený automatický režim.
- **Obraz z kamery:** Obrazová data z kamery jsou zobrazována pomocí panelu, který pokrývá pozadí celé obrazovky. V automatickém režimu navíc označuje detekovaný dokument obarvením jeho plochy do poloprůhledné žluté barvy.
- **Stav skenování a instrukce:** Ve spodní části panelu, který zobrazuje data z kamery, je umístěno textové pole, jehož úkolem je informovat uživatele o aktuálním stavu procesu skenování a popřípadě prezentuje instrukce, které mají uživatele navádět.
- **Dolní ovládací panel:** Dolní panel obsahuje menu pro výběr formátu dokumentu, dále pak tlačítko s náhledem pro výběr fotek z galerie, tlačítko pro pořízení skenu a návratu do galerie. Pro potřeby výběru formátu dokumentu bylo navrženo vlastní výběrové menu, u něhož jsou prvky rozloženy po horizontální ose. Výběr je omezen pouze na jednu vybranou možnost v jeden okamžik a zároveň nemůže nastat situace, při které by nebyla vybrána žádná možnost.

5.2.4 Editační obrazovka

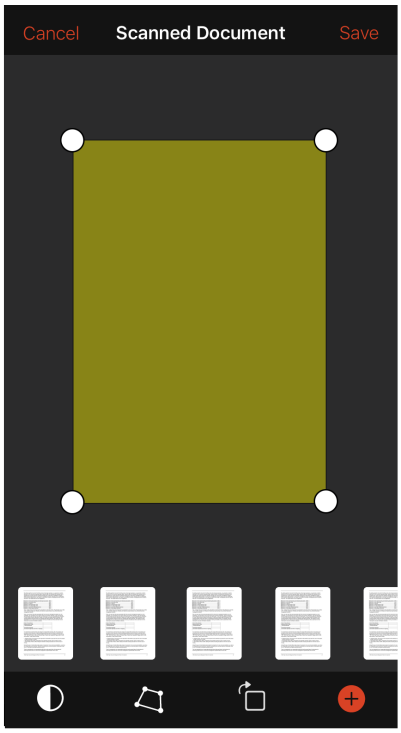
Editační obrazovka slouží k úpravě pořízeného skenu a návrh lze vidět na obrázku 5.6. Dělí se opět na sekce, které jsou popsány v bodech v následujících odstavcích.



(1) (2)

Obrázek 5.5: Wireframe skenovací obrazovky: 1) Bez pozadí; 2) S pozadím, které znázorňuje průhlednost prvků

- **Horní ovládací panel:** V horním panelu se nacházejí tlačítka pro návrat na obrazovku s kamerou, uložení skenu a zároveň název dokumentu, který lze editovat.
- **Náhled a editor kontur:** V centrální sekci se nachází náhled upraveného dokumentu a v případě editování kontur poskytuje jednoduchý nástroj, který slouží k výběrů ohraničení dokumentu ve scéně.
- **Výběr strany dokumentu:** V dolní části editoru kontur se nachází posuvník s miniaturami dokumentů a slouží k přepínání mezi jednotlivými stránkami dokumentu, za předpokladu, že jich je více než jedna.
- **Dolní ovládací panel:** V dolním panelu se se nacházejí tlačítka pro ovládání hlavních funkcionalit celé obrazovky. Patří mezi ně možnost přechodu do šedotónového obrazu, rotace, přechod do režimu editování kontur a na závěr možnost naskenování další strany dokumentu.



Obrázek 5.6: Wireframe editoru dokumentů

Kapitola 6

Realizace

Kapitola realizace se soustředí na implementaci skenovacího algoritmu a vývoj iOS aplikace, která bude algoritmus integrovat a demonstrovat jeho funkcionalitu ve formě plnohodnotné skenovací aplikace. První část této kapitoly detailně popisuje skenovací algoritmus včetně implementace. Jak již bylo řečeno, práce se soustředí spíše na vývoj skenovacího algoritmu a není nutné do hloubky rozebírat vývoj samotné iOS aplikace. Proto budou v druhé sekci této kapitoly rozebrány pouze určité části implementace, které jsou buďto specifické pro tvorbu aplikace na skenování dokumentů, a nebo svým konceptem vybočují od standardních prvků klasické aplikace. Příkladem může být vlastní horizontální menu pro výběr formátu dokumentu, nebo editor kontur.

6.1 Algoritmus detekce dokumentů

V této kapitole se práce zaobírá implementací algoritmu, který bude sloužit k hledání obsahu v obraze, konkrétně prvků mající rysy dokumentu. Kromě samotné detekce a s ní související stabilizace skenování musí algoritmus řešit i následné doostření obsahu. Při návrhu je kladen důraz na funkčnost a časovou složitost všech částí algoritmu. V následujích sekcích budou rozebrány nejen samotné fáze, ale i všechny potřebné parametry dílčích operací včetně výběru ideálních hodnot, které budou jako celek tvořit nastavení celého algoritmu.

6.1.1 Preprocessing

Zcela první fází algortimu je tzv. preprocessing, který upraví vstupní obrazová data do podoby, ze které je možné snadno izolovat veškeré kontury. Výstupem této fáze jsou tedy data ve formě binární obrázku, kde jsou bílou barvou vyznačeny hrany a černou vše ostatní. Implementovanou funkci preprocessingu lze vidět na obrázku 6.1.1.4. Požadované podoby obrazových dat se dosáhne postupnou aplikací kroků popsaných v následujících odstavcích.

6.1.1.1 Snížení rozlišení

Většina grafických operací, resp. jejich časová složitost zcela závisí na velikosti upravovaného obrázku. Je tedy důležité změnit rozlišení tak, aby byly veškeré operace méně náročné a zároveň nedošlo ke ztrátě informací v obraze. Škálování je provedeno tak, že se bude měnit šířka obrázku a pomocí zachovaného poměru stran se dopočítá výška. Testovaný rozsah horizontálního rozlišení je 400-800 pixelů. Z těchto hodnot se nejvíce osvědčila šířka 450 pixelů, která měla pozitivní rychlostní dopad na nadcházející grafické operace a zároveň nedocházelo k zániku důležitých dat.

6.1.1.2 Převod snímku do stupnice šedi

Příchozí snímek z kamery je v barevném RGB, potažmo BGR[3] prostoru. Všechny následující operace je vhodné z hlediska rychlosti a i principu provádět na jednokanálovém snímku. Z tohoto důvodu je nutné převést obrázek do stupňů šedi pomocí OpenCV funkce `cvtColor(...)`. Kromě vstupní a výstupní matice je navíc potřeba pouze druh konverze. Jedná se o vskutku jednoduchou metodu, u které není nutné nic víc analyzovat.

6.1.1.3 Gaussovo rozostření

Platí, že čím ostřejší snímek je, tím více hran v něm detektor nalezne. To má však negativní rychlostní dopad na algoritmy, které pracují právě s těmito hranami. Při skenování dokumentů je dobré předpokládat, že část scény bude zaplňovat text. Je tedy vhodné použít Gaussovo rozostření, které zredukuje počet detailů v obraze, zejména zmíněný text. Operace rozmazání je prováděná z toho důvodu, aby se omezil celkový počet hran na místech, která na hledání dokumentu nemají vliv. Zároveň však dochází k redukci šumu v obraze, který má obecně špatný vliv na detektory hran. OpenCV implementace poskytuje funkci Gaussovského rozmazání `GaussianBlur(...)` s následujícími důležitými parametry:

- **Velikost kernelu:** Důležitým parametrem je velikost kernelu, která je zadávána zvlášť pro horizontální a vertikální osu. Díky dříve získaným poznatkům je zvolena matice typu 5×5. Je tu ovšem možnost parametr velikosti vynechat a nechat jej dopočítat z hodnoty *Sigma* z definice 2.3.2.

- **Směrodatná odchylka:** Hodnotu směrodatné odchylky není nutné zadávat, neboť OpenCV implementace ji dopočítá na základě zadané velikosti kernelu. Využívá k tomu vzorec z definice 2.3.2.

[3]OpenCV využívá BGR model, avšak v rámci práce to nebude rozdíl a bude zaměňován za RGB

- **Typ okraje:** Je také možné definovat, jak budou řešeny okrajové podmínky konvoluce. Implicitním nastavením je zrcadlení, které vynechává přímý okraj. Obecně je tato metoda nejčastěji používána. V rámci redukce detailů v obraze se osvědčila a není nutné ji měnit.

6.1.1.4 Cannyho detekce hran

Stěžejním krokem celého detekčního algoritmu je aplikace Cannyho detektoru hran na snímky, které již prošly preprocessingem a vyznačují se šedotonovým prostorem a potlačenými detaily. Díky těmto vlastnostem vstupních snímků je Cannyho detektor schopen realtivně úspešně extrahovat, hrany. V rámci implementace lze použít knihovní funkci `Canny(...)`, které kromě vstupní a výstupní matice požaduje dvě prahové hodnoty, které jsou využity při procesu prahování s hysterezí. Volba obou hodnot vychází z článku od Adriana Rosebrocka [28], kde byl spodní práh volen jako $lowerTreshold = 75$ a horní $upperTreshold = 100$. Obě hodnoty prošly snižováním a zvyšováním zhruba o 30 jednotek. Výsledkem však bylo zjištění, že původní volba ze zmíněného článku podávala nejlepší výsledky. Mezi nepovinné parametry patří velikost kernelu pro Sobelův operátor a norma Gradientu. Velikost jádra Sobelova operátru je implicitně nastavená na 3 a není nutno ji měnit, stejně jako normu gradientu.

```
void preProcess(Mat src, Mat &dst) {
    Mat imageDownSized,imageGrayed, imageBlurred;
    // New height constant
    double newHeight = [scanSettings downsized_image_height];
    // Downsize the image in order to make calculations easier.
    resize(src, imageDownSized, cv::Size(src.cols *
    (newHeight / double(src.rows)), newHeight));
    // Change image color to grayscale
    cvtColor(imageDownSized, imageGrayed, cv::COLOR_BGR2GRAY);
    // Apply Gaussian Blur
    GaussianBlur(imageGrayed, imageBlurred, cv::Size(5, 5), 0);
    // Aplly Canny edge detection
    Canny(imageBlurred, dst, 75, 100);
}
```

Obrázek 6.1: Kompletní funkce preprocessingu

6.1.2 Převod hran na kontury

Výstupem fáze preprocessingu je snímek, ve kterém jsou bílou barvou vyznačeny hrany a černou zbylá plocha. Je potřeba mít možnost s jednotlivými hranami pracovat a to s jejich stávající reprezentací není možné. Na převod

z hran na kontury se používá knihovní funkce `findContours(...)`, jež má na vstupu binární obraz a na výstupu vektor detekovaných kontur v matematické podobě. Mezi důležité parametry patří:

- **Mód:** Parametr módu úzce souvisí s nepovinným sekundárním výstupem, vektorem zvaným hierarchie. Ten vypovídá o topologii obrazu a není v rámci této práce relevantní. Z toho důvodu se mód nastavuje na možnost `CV_RETR_LIST`, při které dochází k nalezení všech kontur bez výstupního vektoru hierarchie.
- **Metoda:** Metoda pojednává o způsobu, jak bude algoritmus nakládat s převodem. Možností je například zachování všech bodů, avšak při hledání útvarů jako jsou dokumenty stačí udržovat pouze rohové body hran, což dělá `CV_CHAIN_APPROX_SIMPLE`.
- **Offset:** Nepovinný parametr, tzv. Offset, říká o kolik se vzniklá kontura posune oproti předlohové hraně. Implicitně je parametr nastaven na nulový posun a není jej potřeba pro potřeby této práce měnit.

6.1.3 Aproximace

Veškeré získané kontury nemusejí nutně odpovídat obrysu objektu ve scéně. Tento jev může nastat z důvodu nepřesnosti jakéhokoliv předchozího kroku algoritmu, nebo také jako důsledek různorodých jevů ve scéně, mezi které nejčastěji patří například stíny, poškození dokumentu nebo ohnuté rohy. Právě kvůli těmto jevům může být výsledný obrys dokumentu jiného, než obdélníkového tvaru. Tomu je samozřejmě nutné předejít. Je vhodné u každé nalezené kontury provést úpravu, při které dochází k aproximaci původních křivek jinými, jednodušími křivkami. Tento výrok lze alternativně vyložit jako redukci detailů v kontuře. Tento proces zastupuje knihovní funkce `approxPolyDP(...)`, jejíž algoritmus je založen na Ramer–Douglas–Peucker algoritmu [29]. Aproximační funkce má následující parametry:

- **Uzavřenost:** Parametr uzavřenosti vypovídá o tom, zda-li je aproximovaná kontura uzavřená. V případě hledání dokumentu tomu tak vždy bude a je tedy trvale nastavená na uzavřené kontury.
- **Přesnost aproximace:** Hodnota přesnosti aproximace udává maximální vzdálenost mezi původní křivkou a její aproximací. Platí, že čím vyšší hodnota je volena, tím bude aproximace méně přesná v tom smyslu, že bude ignorovat více detailů. Nejčastěji se volí v závislosti na délce kontury. Vztah je velmi jednoduše definován formulí:

  $$\texttt{c*arcLength(...)} \ ,$$

  kde funkce `arcLength(...)` vrací délku dané kontury a konstanta c hodnotu škáluje. K nejlepší aproximaci dokumentů docházelo s konstantou

$c = 0,02$. To znamená, že maximální vzdálenost mezi původní a aproximovanou křivkou jsou 2% délky dané kontury.

6.1.4 Optimalizace

Veškeré kroky procesu hledání dokumentu v obraze jsou aplikovány na jednotlivé snímky z kamery a to nepřetržitě. To ovšem přináší rizika spjatá zejména s rychlostí celého algoritmu. V nejhorším případě by mohlo docházet k tomu, že celkový čas potřebný na výpočet všech kroků bude natolik velký, že náhled kamery neposkytne zobrazení v reálném čase. Jinými slovy dojde ke snížení snímkovací frekvence pod určitou hranici, pod kterou by se v rámci plynulosti obrazu dostat neměla.

Omezení počtu má kladný rychlostní dopad na algoritmy, které s konturami pracují. Omezení probíhá tak, že je vybrán pouze určitý počet kontur s největší plochou v obraze. Předpokládá se, že velké kontury mají velkou šanci býti dokumentem oproti konturám s malou plochou. V praxi se nejvíce osvědčilo omezení na pět kontur s největší plochou. Tento krok je možné provést bez negativního dopadu na úspěšnost skenování právě díky tomu, že je velmi nepravděpodobné, aby hledaný dokument nebyl jednou z největších kontur.

6.1.5 Algoritmus ustálení

Při běhu algoritmu dochází k nepřetržitým pokusům o nalezení dokumentu ve scéně. V ideálním případě nedochází při absenci dokumentu k nalezení kontur a naopak pokud se hledaný objekt v obraze nachází, je detekován pouze jeho obrys. To však není v praxi možné a přesto, že algoritmus konturu cílového dokumentu najde, dochází příležitostně k chybné detekci, tj. nalezení obrysu, který nepatří hledanému dokumentu.

Algoritmus, který byl navržen pro vyloučení chybných detekcí, je postaven na výpočtu obsahu jednotlivých kontur. Základem je fronta o rozměru n prvků, která uchovává obsahy posledních n kontur. Dále bude v popisu algoritmu použita hodnota odchylky $deviation = 0.1 \times AVG$, kde AVG je průměr obsahů fronty. Druhou potřebnou hodnotou je konstanta prahu chybných skenů $s = 10$. Chod algoritmu je rozdělen do kroků, které jsou popsány v následujících podsekcích.

6.1.5.1 Inicializace

Během procesu skenování přicházejí z kamery jednotlivé snímky, které mají díky detekčnímu algoritmu přidělenou konturu potenciálního dokumentu. Při inicializaci je z každé příchozí kontury vypočítán její obsah pomocí knihovní metody `contourArea(...)` a vložen do fronty. Proces inicializace probíhá do té doby, dokud fronta není zcela zaplněna. Jakmile k zaplnění dojde, je inicializace dokončena a algoritmus je připraven k vyhodnocování správnosti kontur.

6.1.5.2 Vyhodnocování správnosti kontur

Po inicializaci je algoritmus schopen rozhodovat, zdali lze konturu považovat za použitelnou, nebo se naopak jedná o chybnou detekci. Selekce probíhá na základě výpočtu průměrného obsahu všech záznamů ve frontě. U každé nově příchozí kontury je vypočítán její obsah a porovnán s průměrem fronty. Kontura je vyhodnocena kladně v případě, že se její obsah neodchyluje od průměru fronty o více jak zmíněnou odchylku *deviation*, tedy o 10% procent, v opačném případě je zahozena.

Zmíněná hranice tolerance je zcela variabilní a lze ji libovolně změnit. Čím je konstanta vyšší, tím je proces skenování rychlejší, avšak méně úspěšný. Zmíněných 10% vychází z volby hodnoty *deviation*, která se v praxi osvědčila. V případě kladně vyhodnocené příchozí kontury může detekční algoritmus předat právě získaná data jako výsledek a ukončit svoji činnost.

6.1.5.3 Opětovná inicializace

Je patrné, že výše zmíněný algoritmus má slabinu. Problém nastává tehdy, kdy je inicializace provedena na jiné scéně, než ve které se nachází samotný dokument. Dojde k tomu, že průměr obsahů kontur ve frontě bude zcela rozdílný od kontury nově objeveného dokumentu ve scéně. Z těchto důvodu je nutné zavést metodu, která v určitý okamžik provede novou inicializaci algoritmu.

Řešení je snadné a je plně dostačující, aby algoritmus zaznamenával počet chybných skenů, tedy zahozených kontur. V případě, že hodnota překročí určitou hranici, dojde k vyčistění fronty a nové inicializaci. Při implementaci byla zvolena hodnota $c = 10$, tedy opětovná inicializace proběhne po deseti neúspěšných detekcích.

6.1.6 Výběr nejostřejšího snímku

Druhým nežádoucím jevem jsou rozmazané snímky, u kterých obsah nedosahuje požadované ostrosti a tím by snižoval čitelnost samotného skenu. Kdyby skenovací algoritmus nezohledňoval ostrost pořízených snímku, mohlo by docházet k produkci rozostřených skenů. Algoritmus, který se zmíněnému problému snaží zamezit, funguje na jednoduchém principu, při kterém je do pole zaznamenáno m snímku se schválenými konturami a následně je z nich vybrán ten, který má nejvyšší ostrost. Algoritmus výběru nejostřejšího snímku probíhá v krocích popsaných v následujících podsekcích.

6.1.6.1 Inicializace

Během inicializace algoritmu dochází k postupnému vkládání snímků s kladně vyhodnocenou konturou do pole. Jakmile má struktura všechny prvky zaplněné, je inicializace dokončená.

```
bool isStable(vector<vector<Point>> & contour, Mat& imageSrc){
    if(contour.size() < 1)
        return false;

    if( wrongScansCount >= [scanSettings wrong_scans_treshold]){
        scannedImagesQueue.clear();
        wrongScansCount = 0;
    }
    auto actualArea = contourArea(Mat(contour[0]));

    if(scannedImagesQueue.size() < [scanSettings image_queue_size] ){
        // Init the queue with values
        insertToQueue(actualArea);
        return false;
    } else {
        // Queue is initialized, do the main work
        double avgArea = 0.0;
        analyzeQueueData(avgArea);
        if((avgArea >= actualArea
        - [scanSettings image_area_difference_treshold])
        &&(avgArea <= [scanSettings image_area_difference_treshold]
        + actualArea)){
            // Contour OK
            return true;

        } else {
            // Contour NOT OK
            wrongScansCount++;
            return false;
        }
    }
}
```

Obrázek 6.2: Algoritmus ustálení v jazyce C++

6.1.6.2 Výběr nejostřejšího snímku

Pro proběhnutí inicializace má algoritmus k dispozici strukturu plnou snímků s přidruženými konturami. Algoritmus pro výběr nejostřejšího snímku využívá metodu výpočtu koeficientu ostrosti 2.8.1 a aplikuje ji na každý snímek zvlášť. Ze všech snímků v poli vybere ten, který dosahuje nejvyšší hodnoty ostrosti ve srovnání s ostatními. Z toho plyne, že je vybrán pouze nejostřejší snímek.

```
func insertImage(image: UIImage, contour: Contour?,
status: ContourStatus?){
        if(status == nil || status == ContourStatus.Stable){

            stableImages.append((image, contour))
            if(stableImages.count >= 5){
                let stable = stableImages.max{
                OpenCVService.calculateBlurnessIndex($0.0) <
                OpenCVService.calculateBlurnessIndex($1.0) }
                if let stable = stable {
                    selectedImage.value = stable
                }
                stableImages.removeAll()
            }
        } else {
            stableImages.removeAll()
        }
}
```

Obrázek 6.3: Algoritmus pro výběr nejostřejšího snímku v jazyce Swift

6.1.6.3 Opětovná inicializace

Dle definice použité metody 2.8.1 je výsledná hodnota relativní a závislá na scéně. Z toho plyne nutnost patřičné reakce na změnu scény. Pokud by algoritmus změny prostředí ignoroval, mohla by nastat situace, kdy je pole vyplněné snímky s rozdílnými konturami.

Tento problém lze vyřešit pozorováním, zdali během inicializace pole nedojde k zápornému vyhodnocení kontury. Pokud tak nastane, stačí všechny prvky z pole odstranit a spustit opětovnou inicializaci.

6.2 Postprocessing a transformace

Výstupem operací zmíněných v předchozích odstavcích je kontura zachycující ohraničení detekovaného dokumentu a původní snímek, tj. ten, který byl pořízen v době nalezení kontury a splňuje náležitosti dokumentu.

V poslední fázi algoritmu je nutné z obrazových dat a příslušné kontury získat obraz nový, který vznikne transformací vstupního obrazu dle daných rohových bodů obrysu. Proces, který docílí požadovaného výsledku, se dělí na kroky popsané v následujících kapitolách.

6.2.1 Ustanovení poloh rohových bodů

Algoritmem získaná kontura se skládá ze čtyř rohových bodů, které ohraničují plochu nalezeného dokumentu. Jednotlivé body je však nutné kromě jejich pozice v obrazové matici identifikovat tak, aby bylo jasné, který z nich je levý horní, pravý horní, pravý dolní a levý dolní. K rozpoznání slouží jednoduchá kalkulace, která za levý horní bod považuje ten, který má z dané čtveřice nejmenší součet polohy na ose x a polohy na ose y. Dále pravý horní bod se vyznačuje nejmenších rozdílem pozičních hodnot obou os v absolutní hodnotě a levý dolní naopak rozdílem největším.

Tento postup není bezchybný a v určitých případech by mohlo dojít k chybnému rozpoznání. To však v případě této práce téměř nehrozí, neboť je dokument skenován plošně tak, aby k tomuto jevu nedošlo. Pokud by však uživatel chybu dokázal vyvolat, tak jednal tak, že by sken stejně nebyl použitelný a algoritmus rozpoznávání by ho vyloučil.

6.2.2 Perspektivní transformace

K provedení perspektivní transformace dokumentu a jeho následnému ořezu, je zapotřebí spočítat velikost takto nově vzniklých obrazových dat. K tomu slouží jednoduchý výpočet, který je založen na nalezení největších vzdáleností mezi jednotlivými body na příslušných osách. Díky těmto již získaným údajům lze pomocí jednoduchých matematických operací[4] sestavit novou matici, na kterou je potřeba původní obrázek transformovat. O perspektivní transformaci se stará knihovní funkce `getPerspectiveTransform(...)`, které je v argumentech předána již nově vytvořená matice cílových rozměrů. Výstupem je samotná transformační matice, která uchovává údaje potřebné k transformaci obrazových dat. Právě tato matice je následně použita ve funkci `warpPerspective(...)`, které je předán vstupní obraz, transformační matice a rozměry nového obrazu. Výsledkem posloupnosti všech zmíněných operací je izolovaný, perspektivně transformovaný dokument.

6.2.3 Doostření obrazu

V rámci skenování dokumentů je zcela zásadní čitelnost naskenovaného obsahu. Na míře čitelnosti se z velké části podílí ostrost obrazových dat. Je tedy žádoucí provést doostření na již naskenovaný obsah tak, aby bylo docíleno lepší čitelnosti. Proces je založen pouze na jednoduché konvoluční operaci s vhodně zvoleným kernelem.

Za pomocí knihovní funkce `filter2D(...)`, lze na libovolná obrazová data aplikovat vlastní konvoluční matici. Kromě vstupních dat a kernelu umožňuje funkce upřesnění okrajových podmínek a změnu středu matice. Implicitní na-

[4]Jedná se o jednoduché sestavení matice na základě vypočítaných dat, není nutné dále rozebírat.

stavení není však nutné pro použití při doostřování měnit a stačí pouze zvolit vhodnou matici [30], která je zachycena na obrázku 6.5. Implementovanou funkci doostření lze naopak vidět na obrázku 6.4.

```
+ (UIImage*)sharpenImage:(UIImage*)image{
    cv::Mat src,dst;
    UIImageToMat(image, src);
    // Create a kernel
    Mat kernel = (Mat_<double>(3,3)
    << -1, -1, -1, -1, 9, -1, -1, -1, -1);
    // Convolute given kernel with image
    cv::filter2D(src,dst, -1, kernel);
    return MatToUIImage(dst);
}
```

Obrázek 6.4: Algoritmus pro doostření snímku v Objective-C++

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Obrázek 6.5: Kernel doostření

6.3 Implementace iOS aplikace

Jedním z cílů této práce je implementace skenovací aplikace pro iOS platformu, která bude používat navržený skenovací algoritmus. Následující sekce se soustředí na výběr šablony projektu, integraci skenovacího algoritmu, přístup k programování a na další nedílné součásti vývoje. Dále je přiblížena implementace specifických prvků, které byly vytvořeny přímo pro použití v této aplikaci.

6.3.1 Šablona projektu

Celý projekt aplikace je postaven na MVVM šabloně od firmy Ackee, dostupné na veřejném Ackee Github profilu [31]. Šablona je určena pro projekty s návrhovým vzorem MVVM. Je založena na reaktivním programování, flow coordinators, dependency injection, ...

```
bottomSectionContainer.snp.makeConstraints { (make) in
    make.top.equalTo(topSectionContainer.snp.bottom).offset(8)
    make.left.right.equalToSuperview()
    make.bottom.equalTo(self.view.safeAreaLayoutGuide.snp.bottom)
}
```

Obrázek 6.6: Ukázka použití frameworku SnapKit v jazyce Swift

6.3.2 Použití OpenCV ve Swiftu

Vzhledem k tomu, že implementovaný skenovací algoritmus je napsán v jazyce C++, neboť používá knihovnu OpenCV, resp. její iOS verzi, je nutné zařídit, aby iOS aplikace psaná ve Swiftu dokázala využívat veškeré knihovní funkce. To jinými slovy znamená, že je potřeba vytvořit přemostění mezi jazyky Swift a C++.

Bohužel zmíněné přemostění nelze učinit napřímo, ale je nutné ještě využít třetího jazyka, kterým je Objective-C. Ten má tu výhodu, že podporuje kód psaný v C++ a zároveň jej lze snadno přemostit do Swiftu.

6.3.2.1 Objective-C++

Lze snadno zajistit, aby soubor s Objective-C kódem podporoval a kompiloval i jazyk C++. To se provede tak, že se u cílového Objective-C souboru změní původní přípona `.m` na `.mm`. Díky tomu vznikne nový Objective-C soubor, který však podporuje i syntaxi jazyka C++.

6.3.2.2 Bridging header

Veškerou funkcionalitu napsanou v Objective-C++ lze zpřístupnit Swiftu. Stačí vytvořit Bridging Header, jenž je hlavičkovým souborem, do kterého stačí pomocí direktivy `#import` vložit hlavičkové soubory, které mají být viditelné ve Swiftu. V případě této práce se v přemosťovací hlavičce nachází import souboru, který implementuje skenovací algoritmus a ostatní grafické funkce.

6.3.2.3 Tvorba grafického rozhraní

K tvorbě grafického rozhraní celé aplikace se využívá postup pomocí kódu a framework UIKit. U veškerých grafických prvků je nutné definovat vlastnosti týkající se pozice a velikosti. K tomu je možné využít například framework Auto Layout od společnosti Apple. Auto Layout bohužel není nejsnazší na používání a proto existují alternativy, kterou je například, v této práci využívaný, SnapKit. Ukázku deklarace grafického prvku je možné vidět na obrázku 6.7 a ukázku použití prvků z knihovny SnapKit na obrázku 6.6.

```
let searchTextField: UITextField = {
    let textField = SearchTextField()
    textField.theme.makeTextField()
    textField.placeholder = "Search"
    return textField
}()
```

Obrázek 6.7: Ukázka definice textového pole v jazyce Swift

6.3.2.4 Reaktivní programování

V rámci tvorby celé aplikace byl využíván reaktivní přístup k programování, který je popsán v sekci 3.6.1. Většina stuktur a obecně dat, které se často mění, je deklarována jako `MutableProperty` a díky tomu se lze snadno přihlásit k odběru veškerých změn. V určitých částech projektu jsou navíc ještě využívány datové typy `Signal` a `SignalProducer`. Ukázku použitých datových typů z frameworku ReactiveSwift lze vidět na obrázku 6.8.

```
// Mutable property
var documents = MutableProperty<[Document]>([])
// Signal
let (imageFeed, imageFeedInput) = Signal<UIImage, Never>.pipe()
```

Obrázek 6.8: Ukázka použití typu MutableProperty a Signal v jazyce Swift

6.3.3 Specifické prvky aplikace

V rámci realizace aplikace bylo využíváno standardních prvků pro vývoj iOS aplikací, které není nutné s ohledem na zadání této práce popisovat. Z toho důvodu je v této sekci rozebrána implementace prvků, které jsou specificky a z určitého pohledu zajímavě navržené pro použití v této aplikaci.

6.3.3.1 Horizontální menu

Horizontální posuvné menu pro výběr formátu dokumentu je založeno na třídě `UICollectionView`, dále jen collection view, která slouží k prezentování kolekce dat. Jednotlivé položky menu jsou uloženy v poli a pomocí vhodně nastaveného collection view zobrazeny, viz implementace na obrázku 6.9. Při interakci formou tažení je důležité, aby se po ukončení posunu automaticky vybral nejbližší prvek ke středu menu. K nalezení nejbližší možnosti slouží funkce na obrázku 6.10. Druhým způsobem, jak lze vybrat položku v menu, je prosté kliknutí.

```
var collectionView: UICollectionView = {
    // Collection View Layout
    let layout = UICollectionViewFlowLayout()
    // Collection View
    let collectionView = UICollectionView(frame: .zero,
    collectionViewLayout: layout)
    // Register Cell
    collectionView.register(OptionCell.self,
    forCellWithReuseIdentifier: "OptionCell")

    // Set parameters
    layout.scrollDirection = .horizontal
    collectionView.backgroundColor = UIColor.white
    collectionView.showsHorizontalScrollIndicator = false
    collectionView.decelerationRate = UIScrollView.
    DecelerationRate.fast
    return collectionView
}()
```

Obrázek 6.9: Nastavení třídy UICollectionView pro účely horizontálního menu

```
func getNearestCellToCenter(scrollView: UIScrollView)
-> IndexPath? {
    let middlePoint =  UIScreen.main.bounds.width / 2
    guard let layout = collectionView.collectionViewLayout as?
    UICollectionViewFlowLayout else {return nil}
    let spacing = layout.minimumInteritemSpacing/2

      // Simply check both directions for neighbours
    let path = getCellPath(at: CGPoint(x: middlePoint,
    y: frame.midY))
    ?? getCellPath(at: CGPoint(x: middlePoint + spacing,
    y: frame.midY))
    ?? getCellPath(at: CGPoint(x: middlePoint - spacing,
    y: frame.midY))
    return path
}
```

Obrázek 6.10: Nastavení UICollectionView() pro účely horizontálního menu v jazyce Swift

6.3.3.2 Editor kontur

Důležitou součástí procesu skenování je režim editace kontur dokumentu. Uživatel je nucen editor využít v případě, že skenování probíhalo v manuálním režimu a kontura nebyla automaticky detekována. V opačném případě je editace kontury zcela dobrovolná a uživatel tak učiní pouze v případě, že došlo k nedokonalé automatické detekci.

Editor je implementován tak, že se jedná o oddělený `UIView`, při jehož zobrazení aplikace nepřejde na jinou obrazovku a pouze dojde k překrytí stávajícího `UIImageView`, který má na starost zobrazování náhledu a originální fotky. Přechod do editačního režimu kontur zapříčiní zobrazení útvaru, který znázorňuje plochu dokumentu a umožňuje uživateli pomocí čtyř rohových bodů provádět úpravy. Součástí editoru je lupa, která vždy zachycuje oblast upravovaného rohového bodu tak, aby uživatel i přes přítomnost jeho palce v dané oblasti viděl, kam přesně bod umísťuje. Klíčovou částí implementace je přepočet polohy jednotlivých rohových bodů souřadnice ve zdrojovém obrázku. Funkci, která se o přepočet stará, lze vidět na obrázku 6.11.

```
func convertContourCoords(contour: Contour, from: CGSize,
to: CGSize ) -> Contour {
    let sizeRatio: CGFloat = rawImage.size.width /
    self.frame.width

    let yOffset: CGFloat = ((rawImage.size.height
    - self.frame.height * sizeRatio ) / 2.0)

    return Contour(
    topLeft: CGPoint( x: contour.topLeft.x * sizeRatio,
    y: (contour.topLeft.y * sizeRatio + yOffset )),
    topRight: CGPoint(x: contour.topRight.x * sizeRatio,
    y: contour.topRight.y * sizeRatio + yOffset ),
    botRight: CGPoint(x: contour.botRight.x * sizeRatio,
    y: contour.botRight.y * sizeRatio + yOffset  ),
    botLeft: CGPoint(x: contour.botLeft.x * sizeRatio,
    y: contour.botLeft.y * sizeRatio + yOffset ))
}
```

Obrázek 6.11: Přepočet souřadnic rohových bodů editoru kontur na polohu ve zdrojovém obrázku v jazyce Swift

6.4 Dokumentace

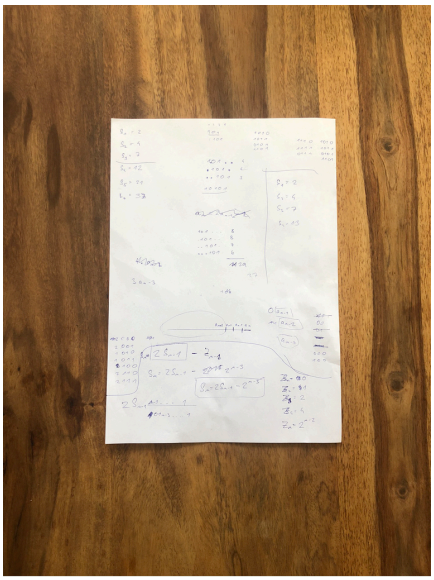
Vývojové prostředí Xcode nativně podporuje dokumentační komentáře, které jsou během vývoje chytře zobrazovány. Veškerá dokumentace celé aplikace je postavena právě na tomto způsobu. Dokumentace není nikterak rozsáhlá a to z toho důvodu, že vyprodukovaná aplikace je uzavřená a nejedná se o knihovnu, kterou by mohli využívat další vývojáři.

Kapitola 7

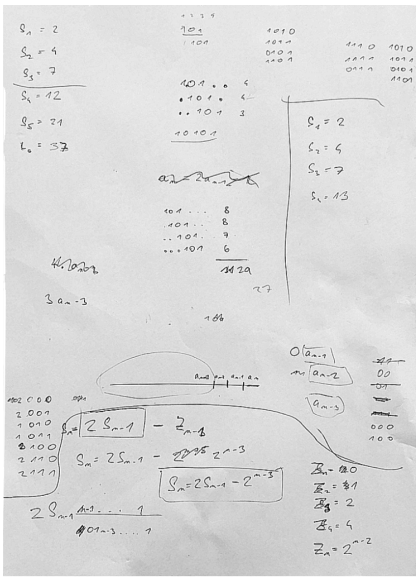
Použití a testování

V této kapitole se práce soustředí na ukázku naskenovaných dokumentů a zejména na testování skenovacího algoritmu. V následujících odstavcích jsou k vidění ukázky výstupů a popis způsobu testování algoritmu včetně výsledků.

7.1 Ukázka naskenovaných dokumentů



(1)



(2)

Obrázek 7.1: Ukázka skenu formátu A4: 1) Originální snímek; 2) Sken vytvořený aplikací

(1) (2)

Obrázek 7.2: Ukázka skenu US vizitky : 1) Originální snímek; 2) Sken vytvořený aplikací

7.2 Vlivy parametrů algoritmu

Volba parametrů jednotlivých dílčích kroků algoritmu zmíněných v předchozích odstavcích ovlivňuje nejen spolehlivost skenovací algoritmu, ale i jeho časovou složitost. Veškeré testování probíhalo na zařízení typu iPhone 8. Testovány byly následující parametry:

- Rozlišení obrázku
- Velikost kernelu Gaussovského rozostření
- Prahy Cannyho detektoru hran
- Konstanta, která slouží v výpočtu aproximace tvarů, více v sekci 6.1.3.

7.2.1 Spolehlivost algoritmu

První vlastnost, tedy spolehlivost algoritmu, nelze snadno popsat vztahem úspěšných skenů k dané volbě parametru. Je to z toho důvodu, že úspěšnost ovlivňují i jiné faktory, než samotná volba nastavení. Mezi nejzásadnější vlivy

patří pozadí scény, které úzce souvisí s kontrastem, dále pak světelnost a dokonce i vzhled dokumentu. Ke komplexnímu testování by bylo nutné využít velký počet testerů a širokou škálu druhů prostředí. Takto detailní testování není součástí zadání této práce, ale i přesto bylo zaznamenáno, zdali algoritmus s konkrétním nastavením dokázal dokument ve scéně detekovat a oskenovat. Samotné testování probíhalo na šedém pozadí s jemným vzorkem, které je vhodné pro řádné testování algoritmu. Výsledky jsou zaneseny v tabulkách 7.1, 7.2, 7.3 a 7.4.

7.2.2 Časová složitost algoritmu

Druhou důležitou vlastností navrženého algoritmu je již zmíněná časová složitost, která musí být v ideálním případě natolik rychlá, aby nedocházelo ke snižování snímkovací frekvence náhledu kamery. Jedním z přístupů testování by mohlo být měření průměrné hodnoty počtu snímků za vteřinu, avšak během testování se ukázalo, že i přes ideální průměrnou frekvenci docházelo k pocitu neplynulosti.

Řešením není tedy pozorování průměrných, nýbrž nejnižších hodnot, tzv. propadů. Právě ty způsobují uživateli pocit neplynulosti. Za propady lze považovat veškeré rázové odchylky snímkovací frekvence od její průměrné hladiny, která má být dle nastavení algoritmu na hodnotě 30 FPS.

Samotné testování probíhalo tak, že byl před kameru umístěn obraz se složitým vzorem, tedy takový, který disponuje velkým počtem ostrých hran. Tento specifický vzhled byl zvolen z důvodu, že pro celý algoritmus je výpočetně náročně pracovat se složitým obrazem.

Otestovat se musí zejména ty parametry, které ovlivňují části algoritmu běžící v reálném čase. Naopak kroky, mezi které patří například transformace a doostření, není nutné testovat, neboť jsou v průběhu analýzy obrazu použity pouze jednou. Vztahy mezi testovanými parametry algoritmu, nejnižší naměřenou snímkovací frekvencí a správnou detekcí jsou zaneseny v následujících tabulkách:

Tabulka 7.1: Závislost velikosti obrazových dat na snímkovací frekvenci

Rozlišení obrázku	200×113	400×225	1000×563	1920×1080
Min. FPS	27	25	24	21
Správná detekce	Ne	Ano	Ano	Ne

Tabulka 7.2: Závislost velikosti Gaussovského kernelu na snímkovací frekvenci

Velikost kernelu	3×3	5×5	7×7	9×9	11×11
FPS	27	27	27	27	26
Správná detekce	Ne	Ano	Ano	Ano	Ano

Tabulka 7.3: Závislost prahů Cannyho detektoru hran na snímkovací frekvenci

Dolní práh	0	20	50	75	80
Horní práh	150	130	110	100	90
FPS	25	26	27	27	27
Správná detekce	Ne	Ano	Ano	Ano	Ne

Tabulka 7.4: Závislost přesnosti aproximace kontur na snímkovací frekvenci

Konstanta aprox.	0,01	0,02	0,05	0,08	0,12
FPS	27	27	27	27	27
Správná detekce	Ne	Ano	Ano	Ne	Ne

7.2.3 Poznatky

Z výsledků testování byla vytvořena sada nastavení, která je použita při realizaci aplikace, resp. skenovacího algoritmu. Nastavení je popsané v sekci 6. Z tabulky 7.1 je patrné, že při volbě příliš malého rozlišení nebyl dokument ve scéně detekován z důvodu zániku informací. Naopak volba velkého rozlišení měla negativní dopad na snímkovací frekvenci a navíc nedocházelo k detekci dokumentu z důvodu prezence velkého počtu detailů v obraze, které algoritmus matou. Z tabulky 7.2 je patrné, že volba velikosti kernelu Gaussovského rozostření měla zanedbatelný dopad na snímkovací frekvenci. Nejmenší z testovaných kernelů neumožňoval správnou detekci. Z testování volby prahů Cannyho detektoru hran nevzešlo nic směrodatného a prahy byly zvoleny podle jedné z možností, která prošla testováním a splňovala úspěšnou detekci. Posledním testovaným parametrem byla konstanta, která vypovídá o procentuálním rozdílu mezi předlohou kontury a její aproximací. Z tabulky 7.4 je patrné, že příliš přesná aproximace zamezovala úspěšné detekci, stejně jako aproximace s největší benevolencí.

Závěr

V práci byly prozkoumány techniky zpracování obrazu ve smyslu skenování obsahu z obrazových dat kamery mobilního telefonu. U jednotlivých postupů byly porovnány jejich vlastnosti a vybráno nastavení, které bylo nejvhodnější pro výsledný algoritmus. Podle získaných poznatků byl navržen algoritmus pro skenování obsahu z obrazových dat, který je schopen vyhodnocování v reálném čase.

Výsledkem práce je mobilní aplikace pro platformu iOS, která slouží jako mobilní scanner dokumentů. Proces skenování vyžaduje minimální zásah uživatele, myšleno tak, že většina kroků je prováděna automaticky. Aplikace obsahuje nespočet prvků, které jsou nad rámec původního zadání a patří mezi ně např. manuální režim, ovládání blesku, interaktivní instruktáž uživatele, řazení dokumentů dle různých kriterií, podpora nahrání fotky z galerie telefonu a mnohé další. Naopak však nebyl kladen velký důraz na podporu vzdáleného serveru, který měl být původně jiný, než použitý iCloudDrive.

Možností dalšího budoucího rozvoje aplikace je nespočet, počínaje rozšířením na nejen detekci dokumentů, ale i jejich textu. Dále by bylo přínosné vylepšení rozpoznávacího algoritmu do náročnějších podmínek, například v noci.

Bibliografie

1. *Diskrétní obraz: ZVS* [online] [cit. 2020-03-25]. Dostupné z: `http://midas.uamt.feec.vutbr.cz/ZVS/Exercise02/content_cz.php`.
2. PRATT, William K. *Digital Image Processing: PIKS Scientific Inside* [online]. USA: Wiley-Interscience, 2007 [cit. 2020-04-08]. ISBN 0471767778.
3. VALENTA, Michal. *Porovnání výkonnosti počítání konvoluce v knihovnách I3dlibsa OpenCV* [online]. Brno, 2016 [cit. 2020-03-25]. Dostupné z: `https://is.muni.cz/th/yyuf5/Bakalarska_prace.pdf`. bathesis. Masaryk University, Faculty of Informatics. Vedoucí práce Vladimír ULMAN.
4. *Cvičení 6 - Diskrétní konvoluce: Multimediální interaktivní didaktický systém* [online] [cit. 2020-02-25]. Dostupné z: `http://midas.uamt.feec.vutbr.cz/ZVS/Exercise06/content_cz.php`.
5. GEDRAITE, Estevao; HADAD, M. Investigation on the effect of a Gaussian Blur in image filtering and segmentation. In: [online]. 2011 [cit. 2020-04-11]. ISBN 978-1-61284-949-2.
6. FISHER, Robert; PERKINS, Simon; WALKER, Ashley; WOLFART, Erik. *Image processing learning resources: HIPR* [online] [cit. 2020-03-18]. Dostupné z: `http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm`.
7. *Gaussian function: Wikipeda* [online] [cit. 2020-04-13]. Dostupné z: `https://en.wikipedia.org/wiki/File:Gaussian_2d.svg`.
8. Filtering,GaussianKernel. In: *OpenCV* [online] [cit. 2020-04-02]. Dostupné z: `https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#getgaussiankernel`.
9. *The Lenna Story: www.lenna.org* [online] [cit. 2020-05-01]. Dostupné z: `http://www.lenna.org/`.

10. CHANDRAN, Saravanan. Color Image to Grayscale Image Conversion. In: [online]. 2010, s. 196–199 [cit. 2020-03-18]. Dostupné z DOI: 10.1109/ICCEA.2010.192.

11. CANNY, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 1986, roč. PAMI-8, č. 6, s. 679–698 [cit. 2020-04-11]. Dostupné z DOI: 10.1109/TPAMI.1986.4767851.

12. Contours Getting Started. In: *OpenCV* [online] [cit. 2020-04-15]. Dostupné z: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

13. SUZUKI, Satoshi; ABE, Keiichi. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing* [online]. 1985 [cit. 2020-04-15].

14. *Transformace ve 2D: is.mendelu.cz* [online] [cit. 2020-04-03]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=19974.

15. HLAVÁČ, Václav; SEDLÁČEK, Miloš. *Zpracování signálů a obrazů* [online]. Praha: ČVUT, 2007 [cit. 2020-04-09]. ISBN 978-80-01-03110-0.

16. PECH-PACHECO, J. L.; CRISTOBAL, G.; CHAMORRO-MARTINEZ, J.; FERNANDEZ-VALDIVIA, J. Diatom autofocusing in brightfield microscopy: a comparative study. In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000* [online]. 2000 [cit. 2020-03-12].

17. ROSEBROCK, Adrian. *Pyimagesearch*. Blur detection with OpenCV [online]. 2015 [cit. 2020-05-01]. Dostupné z: https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/.

18. *Documentation: Apple Developer* [online] [cit. 2020-05-02]. Dostupné z: https://developer.apple.com/documentation/.

19. *About OpenCV: OpenCV* [online] [cit. 2020-03-25]. Dostupné z: https://opencv.org/about/.

20. *About Swift: Swift.org* [online] [cit. 2020-02-02]. Dostupné z: https://swift.org/about/#swiftorg-and-open-source.

21. PAYNE, Robert; BUDELMANN, Jonas. *SnapKit* [online]. 2014 [cit. 2020-04-22]. Dostupné z: https://github.com/SnapKit/SnapKit.

22. *ReactiveSwift: ReactiveCocoa / ReactiveSwift* [online] [cit. 2020-04-13]. Dostupné z: https://github.com/ReactiveCocoa/ReactiveSwift.

23. *Realm Swift 5.0.1: Realm* [online] [cit. 2020-05-03]. Dostupné z: https://realm.io/docs/swift/.

24. *Mobile and Tablet iOS Version Market Share Worldwide: Statcounter* [online] [cit. 2020-05-22]. Dostupné z: https://gs.statcounter.com/os-version-market-share/ios/mobile-tablet/worldwide.

25. *Human Interface Guidelines: Apple Developer* [online] [cit. 2020-04-25]. Dostupné z: `https://developer.apple.com/design/human-interface-guidelines/`.

26. GREENE, Joshua; JAY, Strawn. *Design Patterns by Tutorials* [online]. 3rd. Razeware LLC, 2018 [cit. 2020-03-15]. ISBN 1950325059.

27. *Design Patterns by Tutorials: MVVM: raywenderlich.com* [online] [cit. 2020-05-15]. Dostupné z: `https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm`.

28. ROSEBROCK, Adrian. *Pyimagesearch*. How to build mobile document scanner [online]. 2014 [cit. 2020-04-03]. Dostupné z: `https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/`.

29. *OpenCV*. Contour Features [online] [cit. 2020-04-18]. Dostupné z: `https://docs.opencv.org/trunk/dd/d49/tutorial_py_contour_features.html`.

30. GARRIDO, Gabriel; JOSHI, Prateek. *OpenCV 3.x with Python By Example - Second Edition: Make the Most of OpenCV and Python to Build Applications for Object Recognition and Augmented Reality* [online]. 2nd. Packt Publishing, 2018 [cit. 2020-05-01]. ISBN 1788396901.

31. *iOS-MVVM-ProjectTemplate: AckeeCZ* [online] [cit. 2020-02-18]. Dostupné z: `https://github.com/AckeeCZ/iOS-MVVM-ProjectTemplate`.

Příloha **A**

Seznam použitých zkratek

API Application programming interface

BPMN Business process model and notation

FPS Frames per second

GUI Graphical user interface

IDE Integrated development environment

OCR Optical character recognition

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `diag` ..... adresář s diagramy
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `Gelatka_BP_mobilní_scanner_dokumentů.pdf` ..... text práce ve formátu PDF