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

MOBILNÍ APLIKACE PRO VYHLEDÁNÍ KNIHY V REGÁLU

MOBILE APP FOR LOOKING UP A BOOK IN A SHELF

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN ŠVEC**
AUTHOR

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

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Švec Martin**
Obor: Informační technologie
Téma: **Mobilní aplikace pro vyhledání knihy v regálu**
Mobile App for Looking Up a Book in a Shelf
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou návrhu a vývoje mobilních aplikací; zaměřte se na platformu Android.
2. Seznamte se s problematikou počítačového vidění na mobilních zařízeních; zaměřte se na algoritmy vyhledání na základě vizuální podobnosti.
3. Vyhledejte a analyzujte existující aplikace řešící podobný problém.
4. Navrhněte a prototypujte způsob interakce s aplikací a jednotlivé prvky uživatelského rozhraní.
5. Vyhledejte, navrhněte a experimentálně zhodnoťte potřebné algoritmy počítačového vidění.
6. Navrhněte a implementujte řešenou aplikaci.
7. Testujte vytvořenou aplikaci na uživatelích a iterativně ji vylepšujte.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- 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

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 5, značné rozpracování bodu 6.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá návrhem nástroje pro vyhledávání knihy pomocí kamery na zařízení s operačním systémem Android. Součástí práce je implementace prototypu aplikace, která navržený nástroj využívá. Nástroj používá algoritmy počítačového vidění z knihovny OpenCV.

Abstract

This bachelor thesis deals with the design of a book search tool using a camera on an Android device. Part of the thesis is the implementation of the prototype application that uses the designed tool. The tool uses computer vision algorithms from the OpenCV library.

Klíčová slova

hledání knihy, klíčové body, Android, SIFT, SURF, ORB, homografie

Keywords

find book, keypoints, Android, SIFT, SURF, ORB, homography

Citace

ŠVEC, Martin. *Mobilní aplikace pro vyhledání knihy v regálu.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro vyhledání knihy v regálu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Švec
16. května 2018

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, prof. Ing. Adamovi Heroutovi Ph.D. za odborné rady, cenné připomínky a celkovou pomoc při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Hledání knihy nejen v neseřazené nebo neznámé knihovně může někdy trvat celkem dlouho, zvlášť pokud si nepamatujeme či nevíme, jaký hřbet požadovaná kniha má, proč tedy nevyužít zařízení, které nosí spousta lidí denně u sebe, ke zkrácení a ulehčení této činnosti. Vyhledat záznam hledané knihy v databázi trvá pár okamžiků a následně vyhledání požadované knihy v polici mezi ostatními pomocí kamery na mobilním zařízení a algoritmů pro rozpoznávání objektů by mohlo být rychlejší, než procházení knihovny po jednotlivých knihách a čtení jednotlivých názvů.

Tato práce se zabývá navržením, vytvořením a otestováním nástroje pro vyhledávání knih pomocí algoritmů počítačového vidění pro rozpoznávání objektů z knihovny OpenCV. První částí práce je zjištění vhodnosti různých algoritmů pro rozpoznávání objektů pro mnou navržený nástroj. V další části práce se věnuji návrhu nástroje a implementaci prototypu aplikace využívající tento nástroj.

V kapitole 2 jsou popsány základy fungování několika algoritmů počítačového vidění pro rozpoznávání objektů. V další kapitole 3 je seznámení nejen se základy operačního systému Android ale i se základními poznatky o vytváření aplikací pro tento systém. Kapitola 4 se zabývá návrhem prototypu, testováním jednotlivých možností pro detekci a rozpoznávání objektů. Následující kapitola 5 se zabývá implementací navrženého nástroje a prototypu aplikace pro zařízení se systémem Android, který nástroj využívá. V předposlední kapitole 6 uvádím některá možná zlepšení a funkce které by mohla finální aplikace obsahovat. Závěrečná kapitola 7 shrnuje obsah této práce, hodnotí výstupy vytvořeného nástroje a uvádí dosažené výsledky.

Kapitola 2

Způsoby vyhledání známých objektů v obraze

V této kapitole jsou uvedeny informace týkající se části odvětví počítačového vidění, která řeší rozpoznávání a vyhledávání objektů v obraze. V případě známých objektů je možné použít dvě různé metody a to *S*rovnávání se vzorem (*T*emplate matching) a *P*orovnávání klíčových bodů (*F*eature matching).

Vyhledávání pomocí *S*rovnávání se vzorem probíhá posouváním vzoru (menší snímek obsahující hledaný objekt) po jednotlivých pixelech prohledávaného obrazu. Na každé pozici je proveden výpočet, jehož výsledek ukazuje podobnost aktuální pozice ke vzoru. Podle [15] je tento přístup efektivní pro kontrolované prostředí s danou pozicí a osvětlením hledaných objektů.

Alternativou je najít ve snímku hledaného objektu zajímavé oblasti a následně hledat odpovídající místa v prohledávaném snímku. Tyto body označují části obrazu vhodné pro vyhledávání například svojí jedinečností vůči nejbližšímu okolí [31], jedná se například o osamocené body, rohy, hrany apod. Hlavní výhodou porovnávání klíčových bodů je podle [30] možnost nalezení objektů i při částečném překrytí nebo značných změnách měřítka a rotace. Tato metoda je pro náš případ, s proměnitelným prostředím ve kterém objekty hledáme, vhodnější.

2.1 Klíčové body a jejich deskriptory

*K*líčové body, též nazývané jako *b*ody zájmu nebo *r*ohy, v obrazu něčím vynikají oproti svému okolí, měly by být neměnné při transformaci, škálování a rotaci obrazu, minimálně ovlivnitelné šumem nebo malým zkreslením. Některých z těchto vlastností je dosaženo až při použití složitějších algoritmů pro detekci těchto bodů. Ke každému klíčovému bodu je vytvořen deskriptor, který popisuje vlastnosti tohoto bodu, jenž je získán ze skupiny okolních pixelů.

2.2 Detekce klíčových bodů a získání deskriptorů

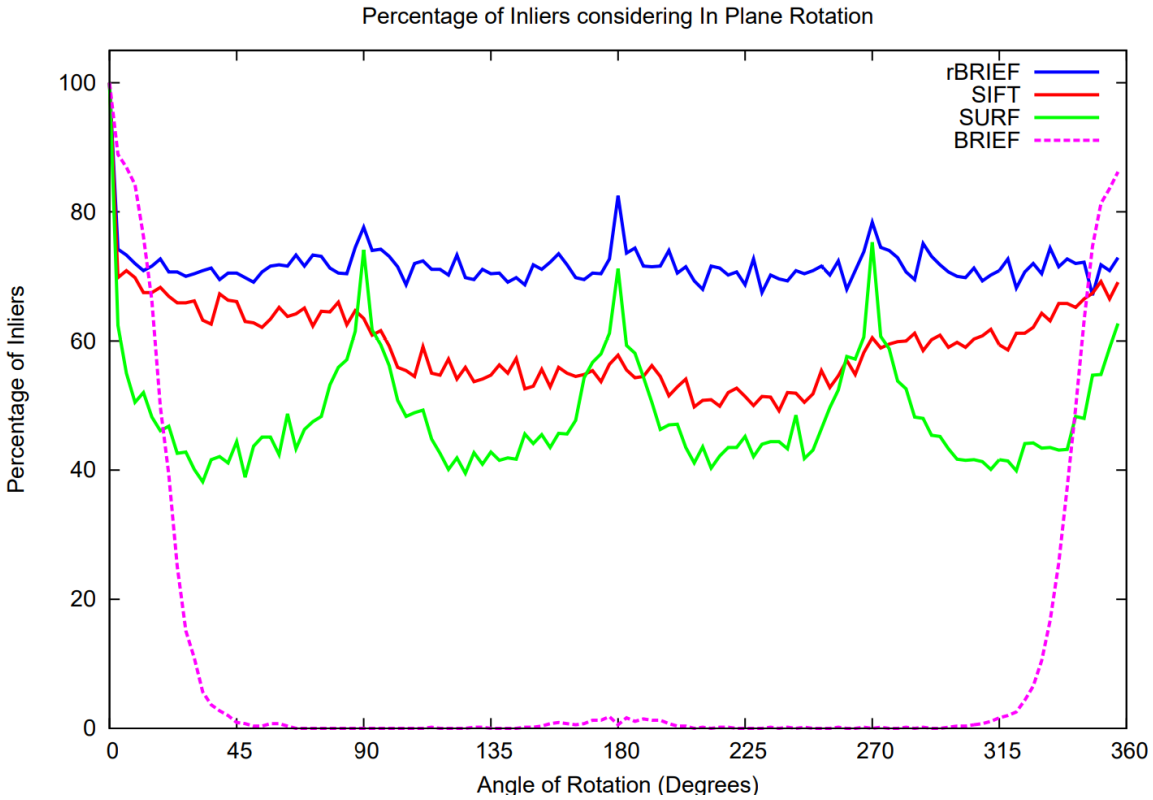
Možností jak nalézt tyto klíčové body a jejich deskriptory již existuje několik. Jeden z prvních detektorů bodů zájmu byl *H*arris Corner Detector popsaný v [11]. Pro rozpoznávání objektů byl *H*arris Corner Detector použit v [29]. Problémem tohoto detektoru bylo prohledávání obrazu pouze v jednom měřítku a při značných změnách měřítka vybíral různé

body. Detektor též neposkytuje indikaci měřítka objektu a proto je při jeho použití nutné vytváření deskriptorů a porovnávání bodů na různých škálách (zmíněno v [15]).

Právě v [15] byla představena metoda *S*cale Invariant Feature Transform (*S*IFT) vyhledávající klíčové body neměnné při změně měřítka, rotaci a odolnější vůči šumu nebo menším změnám osvětlení. Dále v [16] byla tato metoda popsána podrobněji s několika úpravami a vylepšeními.

Kvůli pomalé rychlosti *S*IFT, byl v [2] představen nový rychlejší algoritmus *S*peeded Up Robust Features (*S*URF).

Jako alternativa k výše zmíněným patentovaným algoritmům byl vytvořen *O*riented FAST and Rotated BRIEF (*O*RB), který patentován není. Tento algoritmus byl prezentován v [27].



Obrázek 2.1: Srovnání výsledků *SIFT, SIRF, BRIEF s FAST a ORB(oFAST s rBRIEF)*. Převzato z [16].

2.2.1 SIFT

Body získané algoritmem popsaném v [16] mají velkou pravděpodobnost úspěšného přiřazení k odpovídajícímu bodu při vyhledávání díky své vysoké výraznosti. V typickém obrázku je možno těchto zájmových bodů najít velké množství, a to společně s výše zmíněnými vlastnostmi vytváří základ pro rozpoznávání objektů.

Náklady na extrakci těchto bodů jsou sníženy použitím kaskádového přístupu, který aplikuje drahé operace pouze na oblasti které projdou předchozími částmi algoritmu. Hlavní fáze tohoto přístupu jsou popsány dále.

1. Zjišťování extrémů v různých měřítkách

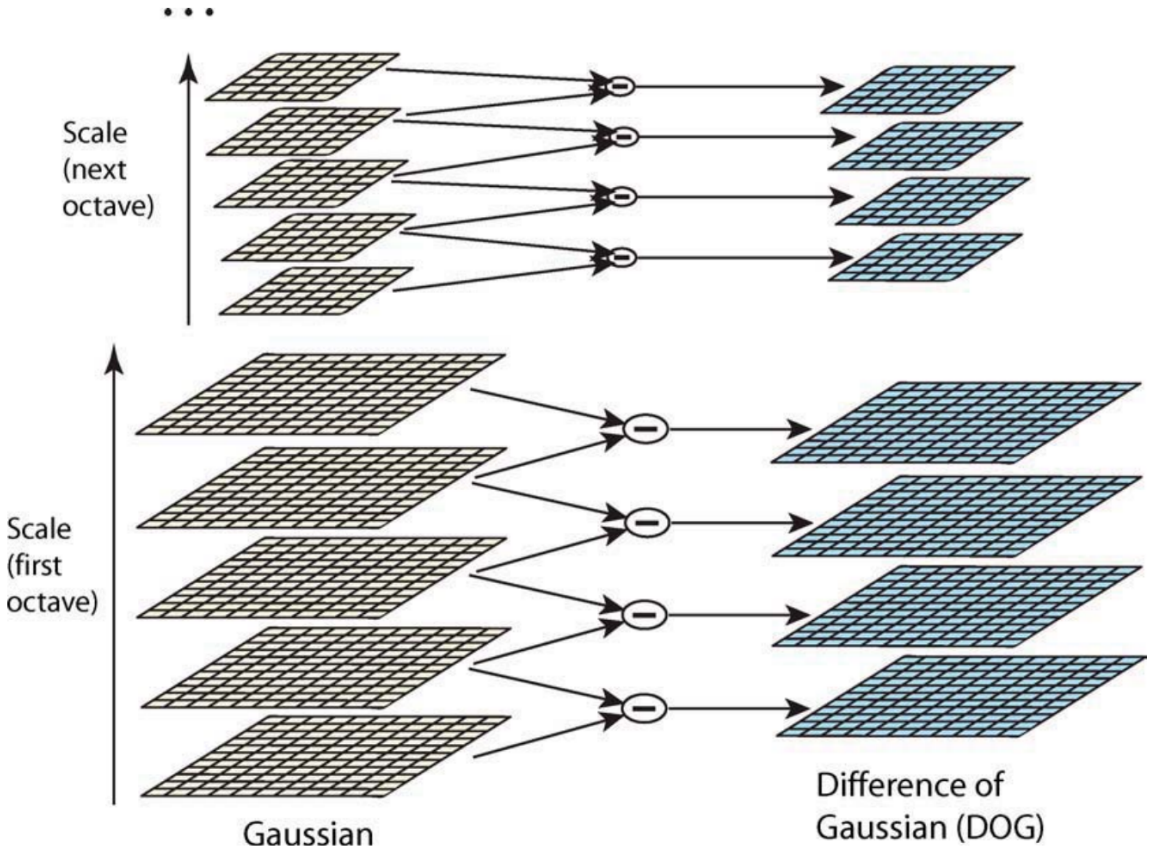
Detekce klíčových oblastí snímku, které jsou škálově neměnné lze dosáhnout hledáním stabilních zájmových bodů napříč všemi měřítky pomocí scale-space analýzy. V tomto případě je funkcí:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.1}$$

kde $G(x, y, \sigma)$ je Gaussova funkce:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2.2}$$

a $I(x, y)$ vstupní snímek. Obraz je takto opakovaně rozostřován, čímž vznikne sada scale-space snímků. Odečtením takto rozostřených snímků vznikne *Difference of Gaussians* (*DoG*) (znázorněno na obrázku 2.2).

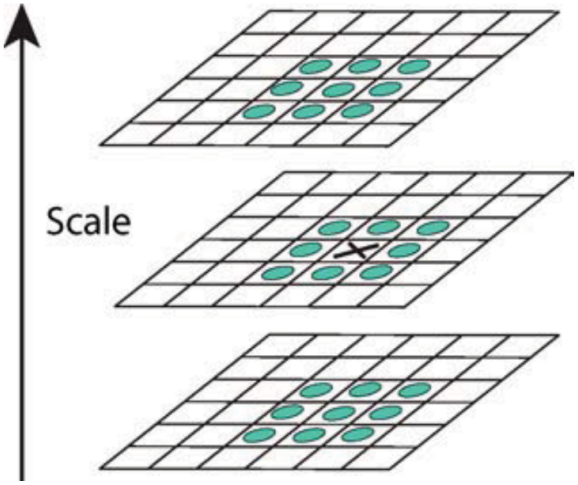


Obrázek 2.2: Sestavení scale-space za pomocí DoG. Převzato z [16].

Lokální extrémy v *DoG* (pixely s odlišující se hodnotou od svého nejbližšího okolí nejen na svém snímku ale i na sousedních snímcích scale-space (Obrázek 2.3) jsou vybrány jako kandidáti na klíčové body.

2. Lokalizace klíčových bodů

Z nalezených kandidátů na klíčové body je nutné odstranit nestabilní body, ty s nedostatečným kontrastem nebo ležící podél hran. Klasifikace těchto bodů proběhne na základě



Obrázek 2.3: Extrém z DoG je získán porovnáním pixelu (X) s jeho 26 sousedy. Převzato z [16].

tvaru proložené 3D kvadratické funkce do lokálních bodů. Je zde využit Taylorův rozvoj nad $D(x, y, \sigma)$:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{2.3}$$

ve které se D a jeho deriváty vyhodnocují v místě klíčového bodu se souřadnicemi $\mathbf{x} = (x, y, \sigma)^T$. Umístění extrému dané posunutím $\hat{x}$ je získáno položením zderivované funkce podle $\mathbf{x} = 0$ čímž se dospěje k:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}} \tag{2.4}$$

V případě $\hat{x} > 0.5$ v jakémkoliv směru, extrém leží blíže k jinému bodu, je klíčový bod vyměněn za tento a zopakuje se aproximace Taylorovým rozvojem. Finální posunutí $\hat{x}$ je přičteno k souřadnicím zkoumaného bodu. Pro odstranění bodů s malým kontrastem je využita funkční hodnota D(x), která se získá pomocí:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}} \tag{2.5}$$

Absolutní hodnota $D(\hat{x})$ se porovná s prahem, a tím dojde k vyřadění bodu. Dále je potřeba vyřadit body ležící na hranách. Takový klíčový bod má velké zakřivení kolmé na hranu a malé podél hrany. Tyto zakřivení lze získat z Hessovy matice $\mathbf{H}$ velké 2x2 spočítané v místě a škále klíčového bodu.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2.6}$$

Náročnému výpočtu vlastních hodnot matice $\mathbf{H}$, které jsou přímo úměrné hlavním zakřivením D, se lze vyhnout, protože je potřebný pouze jejich poměr. Nechť je α vlastní hodnota $\mathbf{H}$ z větší velikostí a β z menší. Poté můžeme spočítat součet těchto vlastních hodnot na hlavní diagonále matice a součin jako determinant:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{2.7}$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \tag{2.8}$$

Ve vzácných případech, kdy je determinant záporný, je bod vyřazen. Nechť r je poměr mezi vlastními hodnotami tak že $\alpha = r\beta$. Poté

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{2.9}$$

závisí spíše na poměru vlastních hodnot než na jejich jednotlivých hodnotách. Velikost $\frac{(r+1)^2}{r}$ je minimální při stejných hodnotách a zvětšuje se s r. A proto je možné pro zkontrolování poměru hlavních zakřivení vůči prahu r pouze provést

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{2.10}$$

Body které neprojdou touto kontrolou jsou vyfiltrovány jako nevhodné.

3. Přiřazení orientace

Přidělování orientace ke klíčovému bodu umožňuje, aby byl bod invariantní vůči rotaci obrazu. Orientace bodu je rozhodnuta na základě histogramu vytvořeného z orientací gradientů vzorkových bodů z okolí klíčového bodu. Šířka intervalů histogramu orientací je 10°, pro celý kruh jich tedy je celkem 36. Velikost gradientů jednotlivých pixelů je spočítána pomocí funkce:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.11}$$

jejich orientace se získá funkcí:

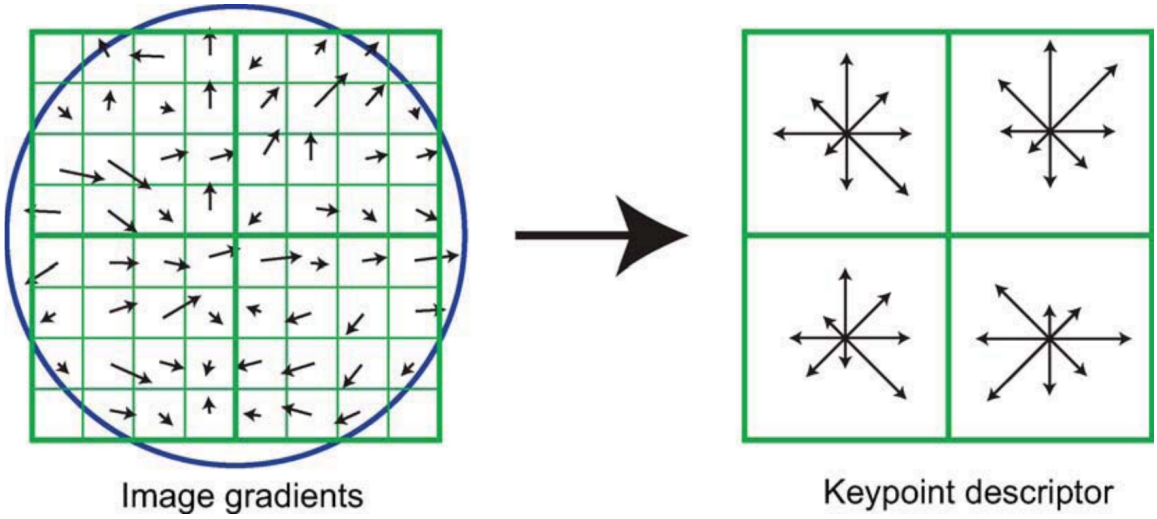
$$\theta(x,y) = tan^{-1}(\frac{(L(x,y+1) - L(x,y-1))}{L(x+1,y) - L(x-1,y)}) \tag{2.12}$$

kde L(x, y) je obrázek rozmazaný Gaussovou funkcí, jehož měřítko má nejblíže k měřítku klíčového bodu, což dále zajistí nezávislost na škále. Každý prvek vkládaný do histogramu je vážen na základě jeho hodnoty. Nejvyšší sloupec histogramu pak značí orientaci klíčového bodu. Pokud některý nižší interval přesahuje minimálně 80 % sloupce nejvyššího, je pro tuto orientaci vytvořen další klíčový bod na stejném místě. Posledním krokem této části je proložení paraboly třemi nejbližšími body histogramu pro každou takto zvolenou orientaci, které vede ke zvýšení přesnosti.

4. Získání deskriptorů

Poslední částí tohoto algoritmu je ke každému bodu získat vlastnosti jeho okolí, deskriptor. Tento deskriptor klíčového bodu je vytvořen z gradientů a orientací bodů v okolí. Tyto gradienty jsou znázorněny na obrázku 2.4 vlevo jako malé šipky.

Následně se využije Gaussova vážící funkce s hodnotou σ rovnající se polovině okna deskriptoru pro určení váhy každého bodu okolí (tato funkce je znázorněna na obrázku 2.4 jako kruh). Cílem této funkce je zamezit náhlým změnám deskriptoru při malé změně pozice okna a klást větší důraz na body bližší ke středu než na ty vzdálenější. Deskriptor je poté na obrázku 2.4 vyobrazen vpravo, pro umožnění posunů pozic gradientů jsou vytvořeny histogramy nad oblastmi 4×4 v okolí bodu, díky čemuž gradient přesunutý i o 4 pozice bude stále náležet stejnému histogramu. Obrázek 2.4 zobrazuje 8 směrů pro každý orientovaný



Obrázek 2.4: SIFT deskriptor o velikosti 2×2 získaný z oblasti 8×8. Převzato z [16].

histogram s délkou šipek odpovídající velikosti třídy histogramu. Na obrázku 2.4 je ukázán deskriptor 2×2, ale lepších výsledků je dosaženo s 4×4.

Je důležité se vyhnout veškerým efektům ohraničení, díky kterým se deskriptor náhle změní, jako například přesunutí z jednoho histogramu do jiného. K zamezení se využívá trilineární interpolace, která zařídí distribuci hodnoty každého gradientu do přilehlých oddílů histogramů.

Deskriptor je vytvořen jako vektor obsahující hodnoty ze všech histogramů. Těchto hodnot je z pole 4×4 celkem 128 pro každý klíčový bod. Aby deskriptor maximálně ignoroval efekty světelných změn, jsou jeho hodnoty normalizovány na jednotkovou délku.

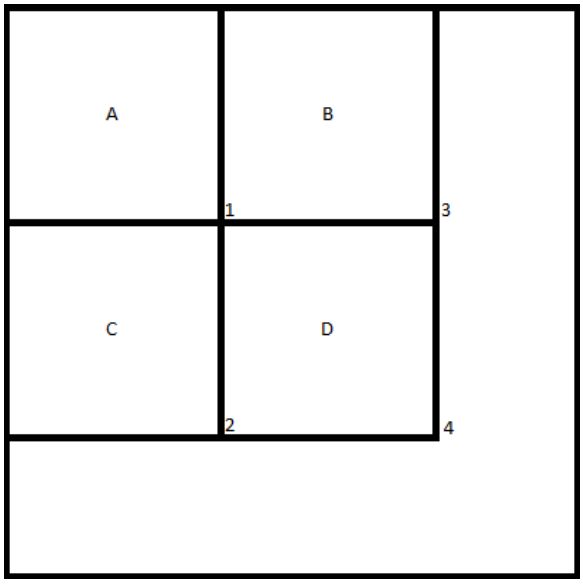
2.2.2 SURF

Při vytváření metody *SURF* si autoři jako hlavní cíl stanovili dosáhnout co nejvyšší rychlosti při detekci klíčových bodů a extrakci jejich deskriptorů (publikována v [2]). Algoritmus je v mnohém inspirován výše zmíněným *SIFT*.

Detekce klíčových bodů je prováděna detektorem *Fast-Hessian*, který pro zrychlení využívá integrálních obrazů definovaných v [32]. Detektor je založený na Hessově matici, na rozdíl od *SIFT* využívá determinant matice jak pro výběr lokace, tak měřítka. Pomocí integrálního obrazu lze konstantní rychlostí získat informace o intenzitě jakékoliv oblasti snímku s nutností znát pouze krajní body této oblasti. Tomuto bodu je následné přiřazena orientace a nakonec je klíčový bod popsán vektorem vlastností, který je menší než u metody *SIFT*.

Integrální obraz

Integrálním obrazem pro bod $\mathbf{x} = (x, y)$ je suma všech pixelů vstupního obrazu od levého horního rohu obrazu po bod $\mathbf{x}$. Obrázek 2.5 ukazuje jednoduchý princip získání sumy intenzit pro jakýkoliv čtverec z jednou spočítaného integrálního obrazu. Doba výpočtu takových oblastí je nezávislá na jejich velikostech.



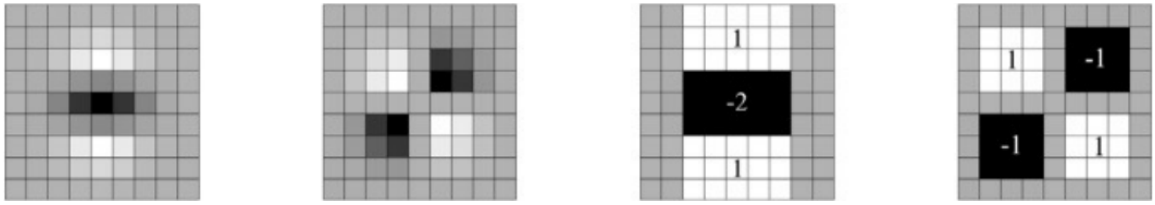
Obrázek 2.5: Pro výpočet sumy intenzit v jakémkoliv obdélníku z integrálního obrazu stačí tři jednoduché sčítací operace, pro obdélník $D = 4 + 1 - 3 - 2$, kde hodnota v bodě 1 je součet pixelů v obdélníku A, v bodě 2 je to A + B, ve 3 A + C a pro bod 4 je hodnota rovna A + B + C + D. Převzato z [32].

Detektor Fast-Hessian

Pro bod $\mathbf{x} = (x, y)$ v obraze I je Hessova matice $H(\mathbf{x}, \sigma)$ v bodě $\mathbf{x}$ a měřítku σ:

$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathrm{x}, \sigma) & L_{xy}(\mathrm{x}, \sigma) \\ L_{xy}(\mathrm{x}, \sigma) & L_{yy}(\mathrm{x}, \sigma) \end{bmatrix} \tag{2.13}$$

kde $L_{xx}(\mathrm{x}, \sigma)$ je konvoluce obrazu I v bodě $\mathbf{x}$ s Gaussovým jádrem druhé derivace $\frac{\partial^2}{\partial x^2} g(\sigma)$. Obdobně pro $L_{xy}(\mathbf{x}, \sigma) a L_{yy}(\mathbf{x}, \sigma)$. Pro další zjednodušení oproti *SIFT* je využita ještě větší aproximace za pomoci obdélníkových filtrů. Gaussovy funkce jsou optimální pro scale-space analýzu, ale jejich zpracování je pomalé, a proto je nutná jejich diskretizace a oříznutí, jak lze vidět na obrázku 2.6.



Obrázek 2.6: Zleva: diskretizované a oříznuté druhé derivace Gaussovy funkce ve směrech y a xy a následná aproximace obdélníkovými filtry. Šedé oblasti jsou rovné nule. Převzato z [2].

Scale-space je implementován jako obrazové pyramidy. Obrazy jsou rozostřeny Gaussem a poté pod-samplovány k dosažení vyšších vrstev pyramidy. Použití obdélníkových filtrů umožní použití jakéhokoliv filtru na původní obraz (dokonce lze pracovat i paralelně), na rozdíl od postupného používání stejného filtru na výstup dříve filtrované vrstvy. Z tohoto

důvodu je scale-space analyzován změnou filtru namísto iterativního redukování velikosti obrazu. Výstup předchozích filtrů je považován na základní vrstvu scale-space. Následující vrstvy jsou získány filtrováním obrazu s postupně většími maskami.

Po vytvoření scale-space jsou klíčové body lokalizovány stejným způsobem jako u *SIFT*, napříč všemi měřítky se naleznou lokální maxima a jsou porovnány se svými 26 sousedy.

Orientace

Aby byly body nezávislé na rotaci, je potřeba jim přidělit orientaci. Jako první jsou spočítány Haar-wavelet hodnoty [1] ve směru x a y uvnitř kruhového okolí klíčového bodu, jehož velikost je $6s$, kde s je měřítko ve kterém byl klíčový bod detekován. U větších měřítek je opět využito integrálního obrazu ke zrychlení výpočtu. Výsledky jsou váženy Gaussovou funkcí vycentrovanou do klíčového bodu. Výsledná orientace se získá jako suma všech hodnot posuvného orientačního okénka, které pokrývá úhel $\frac{\pi}{3}$. Hodnoty ve směru x a y jsou sečteny a dva takovéto sečtené výsledky vytváří nový vektor. Ze všech takovýchto vektorů se vybere nejdelší, jehož orientace určuje orientaci klíčového bodu.

Tvorba deskriptoru

První částí pro vytvoření deskriptoru je sestavení čtvercové oblasti vycentrované okolo klíčového bodu. Oblast o délce strany $20s$ je rozdělena na 16 menších podoblastí, které uchovávají informace o okolí klíčového bodu, vyobrazeno na obrázku 2.7. V každé oblasti jsou následně spočítány d_x a d_y, Haar-wavelet hodnoty pro horizontální a vertikální směr, pro několik pravidelně rozmístěných bodů. Filtry pro získání těchto hodnot jsou také natočeny ve směru orientace klíčového bodu a jsou velké $2s$. Následně jsou hodnoty d_x a d_y váženy Gaussovou funkcí centrovanou na klíčový bod pro zlepšení odolnosti vůči geometrickým deformacím a odstranění lokalizačních chyb. Dále jsou hodnoty d_x a d_y v každé podoblasti sečteny a vytvoří se vektor $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. Spojením těchto vektorů ze všech 16 podoblastí vznikne vektor deskriptoru dlouhý 64 hodnot.

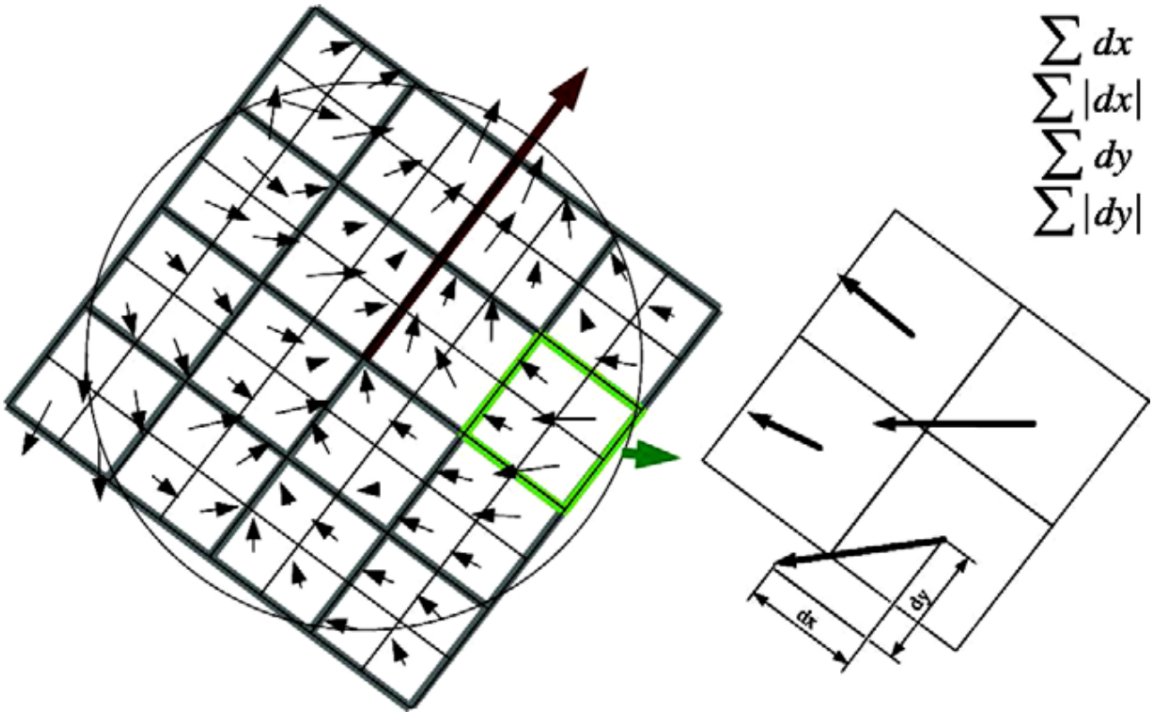
2.2.3 ORB

Metoda *ORB* představená v [27] používá modifikovaný detektor *FAST* (představen v [25]). Použitá modifikace se nazývá *oFAST* a přidává orientaci k nalezeným bodům. Jako deskriptor je použit modifikovaný *BRIEF* (který je publikován v [8]), pojmenovaný *rBRIEF*, jenž do původního také přidává informaci o rotaci.

oFAST

Oriented Features from Accelerated Segment Test je modifikace široce využívaného detektoru *FAST*, který ale neurčuje orientaci nalezených klíčových bodů. *FAST* detektor jako argument přijímá hodnotu prahu t, který určuje minimální rozdíl intenzit mezi kandidátem na klíčový bod a body na kružnici se středem v tomto bodě. Bod projde testem pokud se intenzita předem daného počtu souvislých bodů liší od intenzity bodu zkoumaného o t. Výběr takového bodu je vidět na obrázku 2.8, Vysokorychlostní test kontroluje pouze čtyři body rozdělující kruh na čtvrtiny, ze kterých tři body musí projít, čímž rychle vyřadí podstatnou část kandidátů. Až poté kontroluje všechny body kružnice.

V případě *ORB* je použit *FAST-9*, který používá kruh s poloměrem 9. Protože *FAST* nevytváří žádnou hodnotu měřící rohovitost, a proto detekuje velké množství bodů podél



Obrázek 2.7: *SURF* deskriptor využívá čtvercovou oblast rozdělenou na 4 × 4 podoblastí, okolo klíčového bodu. Hodnoty deskriptoru jsou počítány s ohledem na orientaci klíčového bodu. Převzato z [1].

hran, je pro *ORB* dále použita Harrisova míra rohovitosti, která seřadí nalezené klíčové body, kterých algoritmus nalezne cíleně více než je požadováno, a poté je vybrán požadovaný počet bodů ze začátku seznamu.

Protože *FAST* sám o sobě nehledá body na různých měřítkách, je v *ORB* využita škálová pyramida a klíčové body se hledají na každé úrovni této pyramidy.

Pro určení orientace je využit centroid intenzity [24], který předpokládá, že intenzita rohu je ofset z jeho středu a že tento vektor může být použit k určení orientace. Pokud momenty bodu jsou:

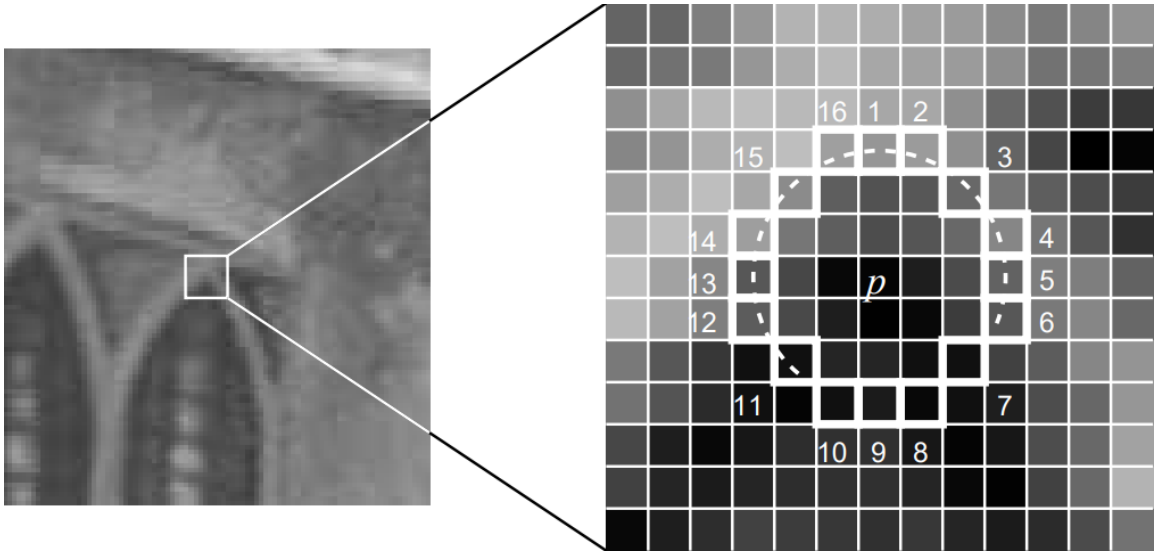
$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2.14}$$

tak centroid lze pomocí těchto momentů nalézt:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2.15}$$

Následně je zkonstruován vektor ze středu klíčového bodu do centroidu a orientaci bodu získáme:

$$\theta = atan2(m_{01}, m_{10}) \tag{2.16}$$



Obrázek 2.8: Test kandidáta na klíčový bod p, zvýrazněny jsou pixely využité v testu, čárkovaný oblouk prochází 12 pixely, které jsou světlejší než bod p minimálně o hodnotu prahu. Převzato z [25].

rBRIEF

BRIEF deskriptor [8] je bitový řetězec popisující klíčový bod sestavený z množiny testů intenzit dvou pixelů. Popis bodu je definován jako vektor n binárních testů:

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; x_i, y_i) \tag{2.17}$$

kde $\tau(\mathbf{p}; x_i, y_i)$ je:

$$\tau(\mathbf{p}; x_i, y_i) := \begin{cases} 1, & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0, & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases} \tag{2.18}$$

Pro distribuci testů je využita Gaussova distribuce okolo středu bodu. V *ORB* se využívá délka vektoru 256. Před provedením testů je důležité provézt vyhlazení obrazu k odstranění šumu. V implementaci *ORB* je vyhlazení obrazu provedeno použitím integrálního obrazu, kde každý testovací bod je 5×5 okno, které nahradí oblast 31×31. Tyto rozměry byly vybrány provedením experimentů. Přidání rotace do tohoto deskriptoru je provedena natočením *BRIEF* podle orientace klíčového bodu. Pro sadu n binárních testů na lokaci(x_i, y_i) definujeme matici $2 \times n$:

$$S = \begin{bmatrix} \mathrm{x}_1 & \dots & \mathrm{x}_n \\ \mathrm{y}_1 & \dots & \mathrm{y}_n \end{bmatrix} \tag{2.19}$$

Využitím orientace klíčového bodu θ a odpovídající matici R_θ se vytvoří natočená matice S_θ z matice S

$$S_\theta = R_\theta S \tag{2.20}$$

natočený BRIEF operátor je tedy:

$$f_n(\mathbf{p}, \theta) := f_n(\mathbf{p}) | (\mathbf{x}_i, \mathbf{y}_i) \in S_\theta \tag{2.21}$$

Úhel je následně diskretizován do sekcí po násobcích 12 stupňů.

Jednou z dobrých vlastností *BRIEF* je velká rozdílnost jednotlivých popisů klíčových bodů, díky které lze tyto body dobře rozlišovat. Další požadovanou vlastností je, aby testy byly vzájemně nezávislé, protože každý test se podílí na výsledku. Tento *natočený BRIEF* je v obou vlastnostech horší. Ke zvýšení rozdílnosti jednotlivých popisů a ke snížení korelace testů v *natočený BRIEF* byla vyvinuta metoda pro vybrání dobré podmnožiny testů. Ze všech testů jsou vybrány ty, které jsou nejvíce rozdílné, vzájemně nezávislé a s průměrem 0.5. Tento výsledný deskriptor je *rBRIEF*.

2.3 Vyhledání odpovídajících deskriptorů

Vyhledávání objektu probíhá pomocí porovnávání deskriptorů klíčových bodů ze zdrojového obrázku a z prohledávaného snímku. Podle druhu deskriptoru jsou použity různé varianty porovnávání těchto bodů [12]. Vektorově založené deskriptory (*SIFT*, *SURF*) využívají hledání nejbližšího souseda pomocí Euklidovské vzdálenosti. Vzdálenost dvou vektorů $d(a, b)$ je:

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2.22}$$

Tyto nejbližší deskriptory navíc musí, aby byly označeny jako odpovídající, mít rozdíl jejich vzdáleností takový, aby nepřesáhl zvolenou prahovou hodnotu. Variantou na tuto metodu je místo rozdílů vzdáleností porovnávat s prahovou hodnotou poměr vzdáleností mezi prvním a druhým nejbližším deskriptorem [17].

Binárně založené deskriptory jsou porovnávány pomocí Hammingovy vzdálenosti [20]. Pro výpočet této vzdálenosti je použita exkluzivní disjunkce, jejíž výstup je sečten. Tato operace zahrnuje pouze manipulaci s bity, a proto může být provedena velmi rychle.

Pro vybrání bodů pro porovnávání zvažujeme dva přístupy poskytované knihovnou OpenCV.

2.3.1 Vyhledání hrubou silou (Brute Force)

Základní a nejednoduší možností je *Vyhledání hrubou silou*, ve kterém dochází k porovnávání každého bodu z jednoho snímku s každým bodem nalezeným ve snímku druhém. Výhodou této metody je, že nevytváří žádné struktury pro vyhledávání vhodných kandidátů na porovnání, ale tím že porovnává každý bod s každým je velmi náročná pro velké množství bodů.

2.3.2 FLANN

Fast Library for Approximate Nearest Neighbors(*FLANN*) ([18], [20]) je sada algoritmů pro vyhledávání podobných deskriptorů řízená algoritmem pro automatický výběr vyhledávacího algoritmu a nastavení jeho vlastností. Algoritmy v této sadě jsou optimalizované pro rychlé vyhledávání nejbližšího souseda ve velkých datových sadách deskriptorů vysokých dimenzí. Velké množství bodů zpracuje mnohem rychleji než *vyhledávání hrubou silou*. Použité algoritmy pro vyhledávání jsou ve *FLANN Náhodně rozdělený k-dimenzionální strom* (*The Randomized k-d Tree Algorithm*) a *Prioritně prohledávací strom k-průměrů* (*The Priority Search K-Means Tree Algorithm*). V [19] byl přidán algoritmus vhodný pro binární deskriptory *Hierarchický strom shluků* (*The Hierarchical Clustering Tree*).

Náhodně rozdělený k-dimenzionální strom

Tento algoritmus sestavuje několik náhodně rozdělených k-dimenzionálních stromů, které jsou paralelně prohledávány. Tyto k-dimenzionální stromy jsou sestaveny podobně jako klasické [5] s tím rozdílem, že u klasických k-dimenzionálních stromů jsou data rozděleny na dimenzi s největším rozptylem, zatímco pro zde použité náhodně rozdělené k-dimenzionální stromy je dělící dimenze vybrána z N_D dimenzí s největším rozptylem. V případě *FLANN* je $N_D = 5$.

Při prohledávání k-dimenzionálního lesa je využita jedna prioritní fronta, která prochází všechny k-dimenzionální stromy a je seřazena podle zvětšující se vzdálenosti každé položky fronty k rozhodovací hranici tak, že prvně budou během hledání prohledány nejbližší listy všech stromů.

Jakmile je datový bod porovnán s aktuálně hledaným bodem, je označen, aby nebyl znovu porovnáván v jiném stromu. Stupeň aproximace je určen počtem všech navštívených listů ve všech stromech a jsou vráceni kandidáti na nejbližšího souseda nalezené v tomto počtu listů.

Prioritně prohledávací strom k-průměrů

Jedná se o vylepšenou verzi prohledávání stromu k-průměrů pomocí strategie best-bin-first [3].

Prioritně prohledávací strom K-průměrů je sestaven rozdělením datových bodů na každé úrovni do K rozdílných oblastí pomocí shlukování k-průměrů, a poté rekurzivně rozdělováním těchto bodů stejným způsobem, dokud není počet bodů v oblasti menší než K.

Strom je prohledáván procházením stromu od kořenu k nejbližšímu listu. Na každém uzlu následuje větev s nejbližším středem shluku k hledanému bodu, neprozkoumané větve do prioritní fronty. Tato fronta je seřazena podle zvyšující se vzdálenosti od hledaného bodu k hranici větve přidávané do fronty. Po původním průchodu algoritmus postupně prochází větve ve frontě.

Počet shluků K je parametrem algoritmu nazývaný jako *větvící faktor* (*branching factor*) a má velký vliv na výkon prohledávání. Dalším parametrem je počet iterací shlukového cyklu, kdy omezení iterací i na malé číslo má jen malý vliv na výsledky vyhledávání, ale značně sníží dobu potřebnou na sestavení stromu oproti stromu s konvergencí všech shluků.

Hierarchický strom shluků

Pro binární deskriptory není vhodná nebo efektivní spousta algoritmů vytvořených pro vektorové deskriptory. Proto byl vytvořen nový algoritmus vhodný pro rychlé vyhledávání těchto deskriptorů.

Při sestavování stromu je vybráno K náhodných bodů jako středy shluků, poté jsou všechny body přiřazeny k nejbližšímu středu. Takto vzniklé shluky jsou uzly stromu a dále rekurzivně rozdělovány stejným způsobem, dokud ve shluku není méně bodu než je určeno maximální velikostí listu. Tyto uzly se stanou listy. Tímto je sestaven Hierarchický strom shluků, v tomto algoritmu je těchto stromů vytvořeno několik a jsou prohledávány paralelně.

Prohledávání stromu začne průchodem všech stromů, kdy se jako následující uzel vybírá ten, který je nejblíže hledanému bodu a ostatní jsou vloženy do prioritní fronty. Když je dosaženo listu stromu, jsou všechny jeho body prohledány lineárně. Jakmile jsou všechny stromy jednou prohledány, algoritmus pokračuje výběrem nejbližšího uzlu z fronty, od kte-

rého poté pokračuje průchod stromem stejným způsobem. Prohledávání končí ve chvíli, kdy je prohledán maximální počet bodů daný argumentem algoritmu.

2.3.3 Ratio test

Pro odstranění špatně spojených deskriptorů byl v článku [16] představen test, který po vybrání pro každý bod dvou nejvíce odpovídajících bodů z druhé sady vyřadí jako neodpovídající ty, jež mají poměr vzdáleností větší než určená hranice. V článku [16] to byla hranice 0.8, a díky tomu bylo odstraněno 90 % špatných spojení a pouze 5 % správných.

2.4 RANSAC

Algoritmus *RANSAC* (*RANdom SAmple Consensus*) [9] se používá pro výpočet parametrů matematického modelu v datech, které obsahují velké množství chybových bodů nazývaných *outliners*, ostatní body, které odpovídají modelu, jsou nazývány *inliners*. Jedná se o iterativní metodu s určitou pravděpodobností nalezení správného výsledku, která se s každou iterací zvyšuje. Oproti ostatním metodám pro výpočet parametrů modelu, jako je například *metoda nejmenších čtverců*, které se snaží model napasovat na všechny dodané data, *RANSAC* náhodně vybírá nejmenší možný počet bodů potřebných pro získání parametrů modelu. I jeden chybový bod zamíchaný mezi dobré body může v některých ostatních metodách způsobit dosažení nepoužitelných výsledků. Proto je vhodnější pro automaticky získávané body, které obsahují spoustu těchto *outliners* použít právě *RANSAC* .

Jedna iterace algoritmu zahrnuje náhodný výběr minimálního počtu bodů pro získání parametrů modelu, výpočet těchto parametrů pro vybrané body, získání *inliners* pro tento model a výpočet poměru získaných *inliners* vůči všem bodům, který rozhodne zda algoritmus spočítá parametry modelu pro všechny *inliners*, vrátí je jako výsledek a ukončí se v případě, že tento poměr přesáhne prahovou hodnotu. V opačném případě algoritmus pokračuje další iterací. Jako *inliners* jsou označeny body, které náleží do aktuálního modelu nebo jsou v jeho blízkém okolí.

2.5 Homografie

Homografie (projektivita, nebo projektivní lineární transformace) je mapování bodů z jedné roviny π do roviny druhé π'. Tato sekce vychází z [4]. Mapování těchto bodů lze zapsat jako $x'_i = Hx_i$, kde x'_i a x_i jsou odpovídající body z obou rovin a H transformační matice homografie o rozměrech 3×3. Cílem je získat tuto transformační matici. Homogenní souřadnice w'_i určuje měřítko.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2.23}$$

$$x'_i = Hx_i = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} wx'_i \\ wy'_i \\ w \end{bmatrix} \tag{2.24}$$

Roznásobením vztahu a po menších úpravách dostaneme dvě lineární rovnice:

$$h_{11}x + h_{12}y + h_{13} - h_{31}x'x - h_{32}x'y - h_{33}x' = 0 \tag{2.25}$$

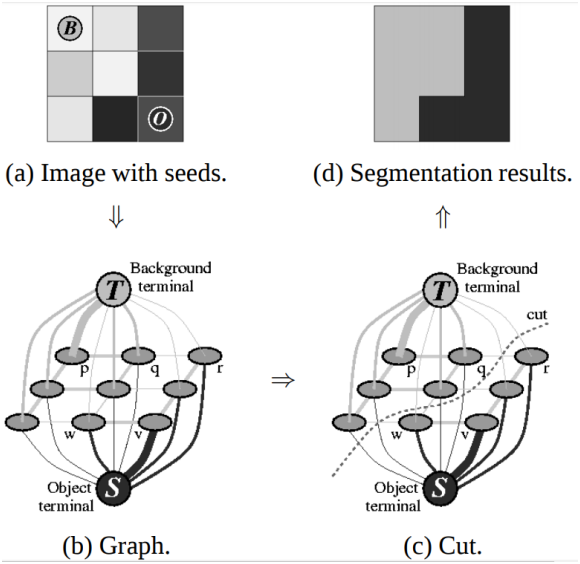
$$h_{21}x + h_{22}y + h_{23} - h_{31}y'x - h_{32}y'y - h_{33}y' = 0 \tag{2.26}$$

Složení těchto rovnic tvoří výslednou matici homografie pro n dvojic (nejčastěji n = 4). Řešením je nulový prostor této matice.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 & -y'_2 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx'_n & -y_nx'_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -x_ny'_n & -y_ny'_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0 \tag{2.27}$$

2.6 GrubCut

Segmentační algoritmus *GrubCut* představený v [26] slouží k oddělení popředí od pozadí při co nejmenší interakci s uživatelem. Jedná se o iterační algoritmus založený na přístupu segmentování obrazu pomocí *graph cut* publikovaném v [6].



Obrázek 2.9: Příklad segmentace obrázku o velikosti 3 × 3. Převzato z [6].

Vstupem pro algoritmus je obrázek, na kterém je pomocí obdélníku uživatelem určeno, kde se nachází popředí, pixely v tomto výřezu jsou označeny jako neznámé a ostatní jako jisté pozadí. Dále uživatel může definovat, co je jisté pozadí a popředí ve výřezu, tyto pixely jsou pak také označeny jako známé a neměnné. Algoritmus poté vytvoří *Gaussian Mixture Model* (*GMM*) [22] k vymodelování popředí a pozadí. Podle poskytnutých dat *GMM* označí ostatní pixely jako pravděpodobné pozadí nebo popředí na základě jejich vztahu ke známým pixelům. Z vytvořeného rozložení pixelů je vytvořen graf, kde každý uzel reprezentuje pixel, navíc jsou vytvořeny dva další uzly, pozadí a popředí, se kterými jsou všechny body reprezentující pixely spojeny. Váhy hran spojující tyto uzly je určena pravděpodobností příslušnosti pixelu k pozadí nebo popředí. Váhy hran mezi sousedními pixely jsou určeny na základě jejich podobnosti. Následně je graf rozdělen pomocí algoritmu nejmenšího řezu na

grafy pozadí a popředí. Cena řezu se spočítá jako součet vah odstraněných hran. Algoritmus pokračuje přiřazením nových hodnot do GMM a iteruje dokud klasifikace konverguje. Klasifikace a rozdělení je zobrazeno na obrázku 2.9

2.7 Knihovna OpenCV

Knihovna *OpenCV* [7] je multiplatformní (podporuje Windows, Linux, Mac OS, iOS a Android) knihovna počítačového vidění vydaná pod licencí BSD a napsaná v optimalizovaném C a C++ s možností využít vícejádrové zpracování. Knihovna má rozhraní pro C++, Javu a Python. Knihovna je navržena pro výpočetní efektivnost se silným zaměřením na aplikace v reálném čase. Hlavním cílem *OpenCV* je poskytnout jednoduše použitelné společné rozhraní počítačového vidění. Knihovna obsahuje velké množství funkcí z různých oblastí zpracování obrazu.

Kapitola 3

Vývoj aplikací pro OS Android

Tato kapitola se věnuje základům operačního systému Android, způsobu vývoje aplikaci pro tento systém, základním prvkům aplikací. Dále je zde popsáno použití knihovny OpenCV aplikacemi na tomto operačním systému. Tato kapitola čerpá z [10], [13] a [23].

3.1 OS Android

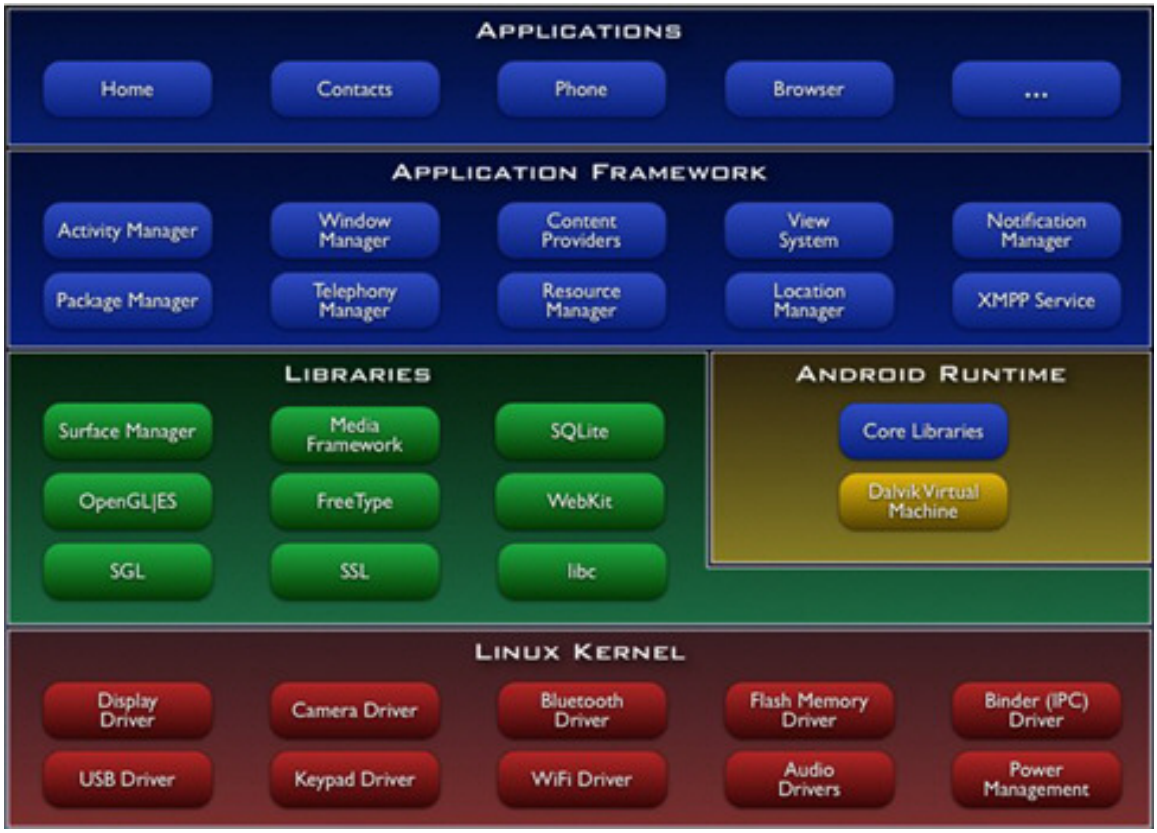
Operační systém Android je systém postavený na linuxovém jádře s otevřeným zdrojovým kódem. Android představila skupina společností *Open Handset Alliance*, která si klade za cíl urychlit inovace v mobilních zařízeních a nabídnout spotřebitelům bohatší, levnější a lepší mobilní zážitek. Android poskytuje kompletní set softwaru, operační systém, middleware a klíčové aplikace pro mobilní zařízení.

Linuxové jádro systému zajišťuje zabezpečení systému, správu procesů a paměti, přístup k síti a ovladačům senzorů a komponent zařízení. Komunikace aplikací s jádrem probíhá pomocí *Android API*, které je napsané v jazyce *Java*. Výhodou linuxového jádra je přenositelnost na různé hardwarové architektury právě díky použití Linuxu pro hardwarovou abstrakci. Android je neustále vyvíjený, ale jeho aktualizace nejsou globální. Z toho důvodu je vždy mezi uživateli rozšířeno několik verzí. Z tohoto důvodu je pro vývojáře důležité se rozhodnout pro jaké verze API bude svoji aplikaci vyvíjet, porovnat potencionální zisk oproti nákladům na zajištění kompatibility.

Architektura operační systému se skládá z několika vrstev které jsou znázorněny na obrázku 3.1. Vrstvy nejsou striktně odděleny ale prolínají se mezi sebou. Pro zajištění bezpečnosti jednotlivé aplikace běží jako samostatné procesy řízené linuxovým jádrem.

Aplikace jsou vyvíjeny v jazyce *Java* a pro jejich běh je nutný virtuální stroj. Protože *Java VM* je vytvořen s úmyslem co nejširšího použití byl vytvořen virtuální stroj zaměřený pouze na mobilní zařízení *Dalvik VM*. Při vývoji tohoto stroje byly brány ohledy na životnost baterie a výpočetní výkon cílových zařízení. Vedlejším efektem změny virtuálního stroje bylo dosažení lepších licenčních podmínek, zatímco veškeré nástroje, knihovny patřící k jazyku *Java* a jazyk samotný je zdarma, *Java VM* už volně šiřitelný není. Zdrojový kód v jazyku *Java* je stále překládaný do *Java byte kódu*, až ten je následné přeložen do *Dalvik byte kódu*, který je interpretován v *Dalvik VM*.

Aplikace jsou šířeny v jediném souboru *aplication package* (*APK*). Hlavní tři komponenty ze kterých se tento soubor skládá jsou spustitelný soubor pro *Dalvik VM*, takzvané *resources* a nepovinný nativní kód jako jsou třeba knihovny v jazyce C/C++. *Resources*



Obrázek 3.1: Architektura OS Android. Převzato z [28].

jsou statická data potřebná pro běh aplikace, například, zvuky, obrázky, videa, rozvržení oken pro různá rozlišení, texty a překlady.

3.2 Vývoj aplikací

Aplikace jsou tedy vyvíjeny v jazyce *Java*, kromě tohoto jazyka je v aplikacích použit ještě značkovací jazyk XML, ve kterém jsou určovány rozložení jednotlivých oken aplikace. Také je v tomto jazyce základní soubor aplikace *AndroidManifest.xml*, jenž uvádí informace o aplikaci, použité komponenty, kompatibilní verze API a specifikuje potřebná oprávnění pro běh aplikace. Některá vývojová prostředí umožňují vytváření jednotlivých rozložení oken, tak zvaných *layoutů*, bez znalosti XML pomocí uživatelsky přívětivého grafického rozhraní. Příkladem takového rozhraní může být využití metody *táhni a pusť*.

3.2.1 Komponenty aplikací

Hlavními komponentami aplikací pro operační systém Android jsou *aktivity* (*Activity*), *záměry* (*Intents*), *služby* (*Services*) a *poskytovatelé obsahu* (*Content providers*). Tyto komponenty jsou mezi sebou volně vázané.

Aktivity

V aplikaci reprezentují prezentační vrstvu, jsou to základní vizuální komponenty, jednotlivé obrazovky aplikace. Aplikace se obecně skládá z několika *aktivit*, kdy je jedna z nich určena jako hlavní, která se spouští ihned po spuštění. Každá *aktivita* může spouštět další *aktivity*. Vždy když je spuštěna nová *aktivita*, systém pozastaví činnost předchozí *aktivity* a uloží ji na zásobník *aktivit*. Při stisknutí tlačítka zpět je obnovena nejvrchnější *aktivita* na zásobníku.

Záměry

Jsou zprávy mezi ostatními komponentami. Způsobí spuštění aplikace, spouští nebo vypínají služby, přepínají aktivity. *Záměry* jsou asynchronní, tedy kód který je odesílá nemusí čekat na jejich dokončení. *Záměr* se obecně skládá z činnosti která se má vykonat, parametru pro tuto činnost a aplikace která má tuto akci provést.

Záměry se rozdělují na explicitní a implicitní. Explicitní *záměr* výslovně stanovuje co má požadovanou akci provést, naopak implicitní pouze říká jakou požaduje činnost a výběr aplikace je na uživateli. Například při *záměru* otevření webové stránky se uživateli nabídnou všechny aplikace umožňující otevřít webovou stránku a je na něm kterou si zvolí.

Služby

Služby jsou součásti aplikace, které provádějí dlouho trvající operace a mohou běžet v pozadí. *Služby* dokáží provádět stejně činnosti jako *aktivity* akorát nemají žádné uživatelské rozhraní a při přepnutí *aktivit*, ani během přechodu do jiné aplikace, se neuspávají.

Poskytovatelé obsahu

Poskytovatelé obsahu pracují s daty a zpřístupňují je ostatním aplikacím. Každá aplikace v systému Android běží ve vlastním *sandboxu*, takže všechna data jedné aplikace jsou naprosto izolovaná od ostatních aplikací. Ačkoliv jde malé množství dat sdílet pomocí *záměrů*, *poskytovatelé obsahu* jsou mnohem vhodnější variantou.

3.3 NDK

Android *NDK* (*Native Development Kit*) [14] je sada nástrojů pro implementaci aplikací pro systém Android nejen v jazyce Java ale i v jazyce C/C++. Této možnosti se využívá ke zvýšení výkonu nebo použití již implementovaných knihoven z jazyka C/C++. Použití NDK je doporučeno pouze v nutných případech a je na zvážení vývojáře zda NDK použije. Jeho využití nemusí pokaždé zvýšit výkon, ale složitost aplikace je jeho použití zvýšena vždy.

3.4 JNI

JNI (*Java Native Interface*) [14] je programový rámec který dovoluje kódu v jazyku Java běžícímu v java virtuálním stroji (*JVM*) volat a být volán nativními aplikacemi a knihovnami v jazyce C/C++. Rozhraní *JNI* lze také využít v aplikaci pro Android, pokud se vývojář rozhodne použít *NDK* pro implementaci některých částí v C/C++, kdy kód běží ve virtuálním stroji Dalvik. *JNI* kromě použití knihoven C/C++ a zvýšení výkonu umožňuje provedení úkolů specifických pro použitou platformu nebo přistupování k systémovým

zařízením. Použití *JNI* má ovšem také nevýhody a to možnou ztrátu přenositelnosti a bezpečnosti aplikace.

3.5 OpenCV pro Android

Pro použití knihovny OpenCV v aplikacích pro Android byl vytvořen balíček OpenCV4Android [21]. Kromě potřebných knihoven a OpenCV API obsahuje balíček sadu ukázkových aplikací využívající právě knihovnu OpenCV, dokumentaci a instalační soubor pro OpenCV Manager. Kód knihovny, který je napsán v jazyce C/C++ je sestaven do dynamické knihovny a v aplikaci pro Android jsou její funkce dostupné pomocí *Java Native Interface*.

OpenCV Manager je aplikace poskytující přístup k nativní knihovně OpenCV aplikacím na cílovém zařízení. Tato služba je pro správnou funkčnost těchto aplikací nezbytná, kromě přístupu k této knihovně zařizuje i stažení kompatibilní verze knihovny s aktuální verzí systému. Další výhodou je menší paměťová náročnost, požadované knihovny jsou v zařízení jen jednou, místo aby byly u každé aplikace která je využívá.

Kapitola 4

Návrh nástroje pro vyhledání knihy a prototypovací aplikace

Cílem mojí práce bylo prozkoumat existující možnosti rozpoznávání objektů ve videu a ověřit realizovatelnost rozpoznávání knih pomocí aplikace pro OS android prototypem takové aplikace. Tato kapitola se zabývá návrhem tohoto vyhledávacího nástroje a prototypu aplikace. Mnou navržený nástroj se skládá z několika částí. Aby mohl nástroj vyhledávat požadované knihy, musí o nich mít nějaké informace, první část nástroje se proto stará o získání a uložení těchto informací. Další část se věnuje vyhledávání a správě záznamů o jednotlivých knihách a poslední, třetí část aplikace se věnuje samotnému vyhledání knihy.

Pro vyhledávání knih bylo potřeba zvolit vhodnou metodu a také složení záznamů o knihách. Z důvodu proměnlivých podmínek, ve kterých se mohou hledané knihy nacházet, a vysoké náročnosti nejen na výpočetní výkon, ale i na paměť při ukládání celých snímků hledaných knih, byl přístup *srovnávání se vzorem* nevhodný, a proto jsem zvolil metodu *porovnávání klíčových bodů*. Dalším důležitým rozhodnutím bylo vybrat správný algoritmus pro detekci klíčových bodů a extrakci deskriptorů. Pro výběr ze zvažovaných metod nabízených knihovnou *OpenCV* *SIFT*, *SURF* a *ORB* jsem sestavil několik skriptů v jazyce Python. Dalším rozhodnutím, se kterým mi také pomohly tyto skripty, bylo zvolit algoritmus pro vyhledávání dvojic odpovídajících bodů. V tomto případě knihovna *OpenCV* nabízí dvě možnosti, a to porovnávání *hrubou silou* nebo pomocí sady algoritmů *FLANN*. Podle výsledků těchto testů jsem pro můj nástroj zvolil pro detekci klíčových bodů a extrakci jejich deskriptorů algoritmus *ORB*. Výsledky srovnání přístupů pro vyhledávání dvojic deskriptorů ukázaly, že obě varianty jsou pro *ORB* podobně rychlé, ale při vyhledávání pomocí *hrubé síly* nástroj produkuje méně falešných nálezů nebo špatně spočítaných matic homografie. V nástroji jsem vyzkoušel obě varianty a použití vyhledávání pomocí *hrubé síly* pracuje lépe.

Pro způsob pořizování dat jsem zvažoval několik možností. Všechny varianty měly společný požadavek na jednotvárné pozadí odstranitelné pomocí algoritmu *GrabCut* bez zásahu uživatele. První variantou bylo pořídit dvě a více fotografií knihy s viditelnou přední stranou a hřbetem pod různým úhlem, detekovat oba obdélníky, pomocí změny velikostí pozorované při posunutí pozorovacího úhlu rozhodnou, který z nich je hřbet knihy, v těchto částech snímků detekovat klíčové body a získat jejich deskriptory, poté provést průnik klíčových bodů z jednotlivých snímků pomocí porovnání bodů zájmu a to použitím stejného algoritmu jako při vyhledávání knihy v knihovně. Následně by se body z tohoto průniku uložily. Další variantou byla oproti první možnosti změna pozorovacího úhlu při první foto-

grafii tak, aby na fotografii kromě pozadí byl vidět pouze hřbet knihy. Po odstranění pozadí by se fotografie ořízla tak, aby obsahovala pouze hřbet knihy, velikost tohoto snímku by udávala rozměry pro ohraničení knihy pomocí *homografie*. Detekce a výběr bodů by probíhal stejně jako v první variantě. Díky použití pouze průniku nalezených bodů by bylo možné provádět odstraňování pozadí, činnost, která trvá celkem dlouho, pouze u první fotografie, nicméně tímto přístupem by se v některých případech získalo pouze minimum bodů a nebylo pak možné knihu vyhledat. Další variantou bylo použití dvou fotografií hřbetu knihy. Při pořizování snímků z jednoho místa je vhodné knihu otočit, v některých případech byly kvůli světelným podmínkám nalezeny jen body z poloviny hřbetu. Z obou fotografií se odstraní pozadí, nejlépe jednobarevná plocha barevně odlišná od knihy. Z obrázků obsahující pouze hřbet knihy se detekují klíčové body, nicméně v tomto případě se provede sjednocení nalezených bodů a výsledek uloží společně se získanými rozměry pro *homografii*. Tato možnost navíc podporuje použití *Ratio test*. Variantu zmíněnou naposledy jsem nakonec použil i v prototypu nástroje, nicméně jsem provedl velkou změnu oproti všem předcházejícím možnostem, které jsem zvažoval, a to nepoužití algoritmu *GrabCut* z důvodu velké časové velmi náročnosti a také kvůli tomu, že s nízkým počtem iterací a bez jakéhokoliv zásahu uživatele produkoval často nepoužitelné výsledky, kvůli kterým byl záznam nepoužitelný. Při testování jsem zjistil, že pokud je hřbet knihy vyfocen na vhodném pozadí, detektor v tomto pozadí nedetekuje žádné klíčové body, a tak je možné jako velikost knihy pro homografii použít ohraničení nalezených klíčových bodů. Kniha sice není při úspěšném nálezu pěkně ohraničená, ale ve většině případů použitím algoritmu *GrabCut* bylo ohraničení o dost větší než hledaná kniha.

Pro ukládání jednotlivých záznamů o knihách jsem navrhl vytvoření a využití speciální třídy pro nalezené klíčové body implementující rozhraní *Serializable* umožňující instanci této třídy změnit na sekvenci bytů a zpět. Tato sekvence se následně uloží do souboru pojmenovaného uživatelem při pořizování fotografií. Kromě klíčových bodů a jejich deskriptorů tato třída také obsahuje rozměry hřbetu knihy v pixelech na první fotografii.

Část s vyhledáváním knihy jsem navrhl jako obrazovku zobrazující poslední zpracovaný snímek pořízený kamerou a v případě nalezení dostatečného počtu odpovídajících klíčových bodů s hledanou knihou, je zde pomocí homografie vykreslen zelený obdélník, který by měl vyznačovat hledanou knihu. Pro vytřídění špatně nalezených bodů je použit *Ratio test* s vyřazením poměrů nad 0,75.

Maximální počet klíčových bodů detekovaných detektorem jsem stanovil na 500 pro každou fotografii při získávání bodů určených k uložení jako záznam o knize a 2000 pro každý snímek při vyhledávání, tedy jeden záznam o knize obsahuje maximálně 1000 bodů. Nástroj pořizuje veškeré snímky v rozlišení 1280×720. Ostatní parametry jsem nechal na výchozích hodnotách, při testování jejich variant nebylo dosaženo výrazně lepších výsledků.

4.1 Testování a srovnání algoritmů

Pro vyzkoušení, otestování a srovnání algoritmů jsem vytvořil několik skriptů v jazyce Python, k analýze jsem kromě různých výpisů a vlastního pozorování také využil profilování. Pro testování jsem vytvořil sadu videozáznamů knihoven obsahující hledané knihy v různých prostředích, například podobné knihy v okolí nebo naopak naprosto odlišné, různé osvětlení knihovny, různé vzdálenosti kamery od knih a rozdílné rozlišení. Fotografie hřbetů hledaných knih jsem také pořizoval v různém rozlišení.

4.1.1 Profilování

Jedná se o techniku dynamické analýzy kódu za běhu programu. Primárně slouží pro vyhledávání vhodných míst programu pro optimalizaci. *Profilování* sbírá a poskytuje různé statistiky o běhu programu. Získávání probíhá pomocí specializovaných nástrojů nazývaných *profilery*. Já jsem použil *profiler* ze standardní knihovny jazyka Python *cProfile*.

4.1.2 Detektory a deskriptory

Pro testování detektorů a deskriptorů jsem vytvořil dva skripty pro každý typ. První skript zpracuje dvě vstupní fotografie hřbetů knih, odstraní z nich pozadí, pomocí vybraného detektoru získá klíčové body a jejich deskriptory, které poté uloží do souboru. Druhý skript slouží už k vyhledávání záznamů knih ve videozáznamu. Nejvíce statistik je získáno z druhého skriptu. Nejvíce jsem se zaměřil na srovnávání doby po kterou běžely funkce vykonávající algoritmus pro vyhledání klíčových bodů a získání deskriptorů, dalšími pro mě podstatnými informacemi byly počty správně vyznačených knih, špatně označených knih a snímky kde se kniha vyskytovala a nebyla vůbec označena.

Dále přikládám výsledky některých testů. Pro prezentaci jsem vybral testy, které porovnávali výsledky pro různá rozlišení při hledání knihy mezi několika podobnými.

SIFT

Při testování tohoto algoritmu jsem dosáhl nejlepších výsledků co se týká kvality vyhledávání, ale je ze všech testovaných algoritmů nejpomalejší.

Rozlišení[px]	**Detekce**[s]	**Nalezeno/snímků s knihou**	**Nesprávné nálezy**
1920 × 1080	102.002	141/140	12
1280 × 720	46.454	101/110	38
720 × 480	25.474	51/85	28

Tabulka 4.1: Výsledky testů pro *SIFT*

SURF

Tento algoritmus je o něco rychlejší než SIFT pro odpovídající množství bodů, nicméně jeho výsledky byly ve většině případů o něco horší než u *SIFT* a obdobné jako u *ORB*.

Rozlišení[px]	**Detekce**[s]	**Nalezeno/snímků s knihou**	**Nesprávné nálezy**
1920 × 1080	72.946	149/140	25
1280 × 720	30.704	85/110	36
720 × 480	14.529	86/85	74

Tabulka 4.2: Výsledky testů pro *SURF*

ORB

Nejrychlejší testovaný algoritmus s uspokojivými výsledky, které byly o něco horší než při použití SIFT, ale zlepšení v oblasti rychlosti toto malé zhoršení vysoce převyšuje.

Rozlišení[px]	**Detekce**[s]	**Nalezeno/snímků s knihou**	**Nesprávné nálezy**
1920 × 1080	9.772	95/140	37
1280 × 720	6.057	65/110	41
720 × 480	3.615	0/85	0

Tabulka 4.3: Výsledky testů pro *ORB*

4.1.3 Vyhledání odpovídajících bodů

U vyhledávání odpovídajících bodů jsem se zaměřil hlavně na rychlost a poměr špatných a správných výsledků. Prezentované výsledky jsou ze stejné sady testů jako výše prezentované výsledky testování detektorů.

Brute Force

Vyhledání odpovídajících bodů při použití vyhledávání *Brute Force* bylo ve většině případů rychlejší a produkovalo méně špatných nálezů.

Rozlišení[px]	**SIFT**[s]	**SURF**[s]	**ORB**[s]	**Nalezeno/celkem** (SIFT;SURF;ORB)	**nesprávné nálezy** (SIFT;SURF;ORB)
1920 × 1080	12.338	11.012	2.305	141; 149; 95/140	12; 25; 37
1280 × 720	12.288	1.142	1.512	101; 85; 65/110	38; 36; 41
720 × 480	12.947	3.449	2.906	51; 86; 0/85	28; 75; 0

Tabulka 4.4: Výsledky testů pro Brute Force

FLANN

Výsledky testů ukázaly, že při použití sady algoritmů *FLANN* jsou v našem případě náklady na vytvoření vyhledávacích struktur větší než ušetřený čas při vyhledávání oproti *Brute Force*. Tato metoda také produkuje více falešných nálezů, ale v různých problémových situacích, jako jsou rozmazané záběry, více podobných knih nebo při vyhledávání knih s malým počtem klíčových bodů, označí správnou knihu i v případech, kdy přístup *Brute Force* dosáhl špatných výsledků.

Rozlišení[px]	**SIFT**[s]	**SURF**[s]	**ORB**[s]	**Nalezeno/celkem** (SIFT;SURF;ORB)	**nesprávné nálezy** (SIFT;SURF;ORB)
1920 × 1080	24.419	7.668	2.272	141; 139; 182/140	14; 15; 133
1280 × 720	25.072	2.607	1.736	101; 44; 205/110	30; 11; 180
720 × 480	23.810	2.815	2.476	51; 28; 195/85	30; 21; 184

Tabulka 4.5: Výsledky testů pro FLANN

Kapitola 5

Implementace prototypu aplikace

Navržený nástroj **BookFinder** jsem implementoval pomocí tří aktivit a jedné další třídy pro serializaci klíčových bodů. Hlavní aktivitou nástroje je pouze seznam uložených záznamů zobrazený jako přepínač (radio button) a tří tlačítek. Tlačítko **NEW** přepíná aplikaci do aktivity pro přidání nové knihy, tlačítko **FIND** otevře aktivitu pro hledání a odešle ji pomocí záměru informaci, ze kterého souboru má načíst hledané body. Tlačítko **DELETE** smaže vybraný záznam.

5.1 Přidání nové knihy

Aktivita pro přidání nové knihy se skládá z náhledu záběru kamery, textového pole pro pojmenování záznamu a třech tlačítek. Tlačítka **CAPTURE 1** a **CAPTURE 2** slouží ke zpracování dalšího snímku v záběru. Tlačítko **SAVE** uloží klíčové body a deskriptory, pokud jsou již vytvořené záběry tlačítky **CAPTURE 1** a **CAPTURE 2**.

Při vytváření této aktivity je nastaven náhled kamery. Dále při volání metody **onResume** je inicializován manažer knihoven *OpenCV* a knihovna samotná. Pro inicializaci závislostí na knihovně *OpenCV* pomocí metody **initializeOpenCVDependencies** je vytvořen asynchronní **callback**. Metoda **initializeOpenCVDependencies** vytváří detektor klíčových bodů a extraktor deskriptorů.

Aktivita obsahuje dále metody, které jsou volané při stisknutí tlačítek. V případě tlačítek **CAPTURE** slouží pouze k nastavení příznaku, který zařídí vyhledání klíčových bodů v dalším snímku. Metoda pro tlačítko **SAVE** zkontroluje, zda jsou již získány klíčové body z obou snímků a v kladném případě vytvoří instanci třídy **Keys**, naplní ji a uloží jako soubor jehož název načte z textového pole.

Hlavní činnost aktivity probíhá v metodě **onCameraFrame**, jež je volána pro každý snímek zachycený kamerou. Snímek zde není nijak upravován a je vždy vrácen k zobrazení. V případě nastavení některého z příznaků zachycení jsou ze snímku získány klíčové body a deskriptory. Nepoužitím algoritmu *GrabCut* k odstranění pozadí a následnému oříznutí fotografie tak, aby obsahovala pouze knihu došlo ke zjednodušení metody a rapidnímu zkrácení času nutného pro pořízení záznamu.

5.2 Vyhledání

Aktivita pro vyhledávání obsahuje pouze náhled záběru kamery, ve kterém jsou vyhledávány klíčové body porovnávány s klíčovými body hledané knihy, a proto je frekvence snímků v tomto záběru celkem nízká.

Protože tato aktivita taktéž využívá služeb knihovny, musí při svém spouštění a obnovování také nastavit používaný náhled kamery a knihovnu OpenCV pomocí jejího manažeru. Metoda pro inicializaci závislostí na knihovně OpenCV **initializeOpenCVDependencies** vytváří detektor klíčových bodů, extraktor deskriptorů a objekt provádějící porovnávání bodů. Po vytvoření detektoru nastaví jeho vlastnosti pomocí dočasného yml souboru, který vytvoří pomocí metody *writeToFile*. Kromě vytváření výše zmíněných objektů metoda **initializeOpenCVDependencies** také získává uložené body ze záznamu o hledané knize a to použitím metody **getKeys**.

Metoda **onCameraFrame** zajišťuje hlavní funkčnost nástroje, a to vyhledávání hřbetu knihy. Metoda je volána pro každý snímek a jejím prvním úkolem je získat z tohoto snímku klíčové body a jejich deskriptory. Poté provede porovnání bodů pomocí algoritmu *brute-force* s vrácením dvou nejlepších variant, jehož výstup je vytříděn použitím *Ratio testu*. Pokud je po tomto kroku dostatečný počet bodů, je pro ně spočítána matice *homografie*, podle které je provedena perspektivní transformace rohových bodů knihy do scény a nález zvýrazněn.

5.3 Ukládání klíčových bodů

Pro umožnění serializace objektu je potřeba, aby všechny jeho položky byly serializovatelné, z tohoto důvodu v této třídě **Keys** nezachovávám klíčové body a jejich deskriptory jako typy **MatOfKeyPoint** a **Mat**, ale jako sadu celých čísel a dvě bytové pole, které obsahují samotné hodnoty. Sada celých čísel definuje vlastnosti n-dimenzionálních polí **MatOfKeyPoint** a **Mat**. Protože tato třída slouží jen pro ukládání těchto bodů, není potřeba body udržovat v dále použitelné podobě. Instance této třídy je vždy použita pouze k uložení bodů do souboru nebo jejich načtení ze souboru a předání do proměnných se kterými dále opět pracuje knihovna *OpenCV*. Třída implementuje tři metody.

Metoda **setKeys** přijme jako argumenty rozměry pro homografii a dvě sady klíčových bodů s jejich deskriptory. Úkolem této metody je serializovat obdržené hodnoty a uložit je jako atributy aktuální instance. Metody **getDescriptors** a **getKeypoints** mají za úkol ze serializovaných atributů vytvořit objekt **Mat** respektive **MatOfKeyPoint** který je jejich návratovou hodnotou.

Kapitola 6

Návrhy na funkce výsledné aplikace a možnosti zlepšení

Prezentovaný prototyp aplikace obsahuje pouze základní funkčnost potřebnou pro otestování. V této kapitole uvádím několik možných rozšíření, dalších funkcí a vylepšení aplikace, kterými by mohla práce na aplikaci pokračovat.

6.1 Optimalizace výkonu

I přes použití ze zvažovaných nejrychlejšího algoritmu, nástroj je při prohledávání pořád pomalý a v porovnání s člověkem, který když se podívá jak kniha vypadá, nemá tato aplikace šanci vyhrát. Možným způsobem jak zvýšit vyhledávací rychlost by mohlo být vytvořit celou funkci v nativním kódu pomocí C++ a z části aplikace v jazyce Java ji pouze volat, místo aktuálního přístupu, kdy jsou v nativním kódu pouze funkce knihovny *OpenCV* a jejich spolupráce je zařizována v Javě.

6.2 Serverová část s veřejnou databází knih

Ukládání záznamů o jednotlivých knihách lokálně u aplikace, jak je to provedeno v prezentovaném prototypu aplikace, není ideální způsob. Tento přístup umožňuje použití aplikace jen pro osobní knihovny, a to jen na knihy, které si každý uživatel sám připraví, což je časově velmi náročná aplikace a většinu uživatelů by nejspíš odradila od používání této aplikace. Navíc o svojí vlastní knihovně má člověk alespoň minimální přehled a ví, co kde hledat, proto největší potenciál pro využití této aplikace je ve veřejných knihovnách, nebo knihkupectvích.

Odstranění této překážky by bylo možné dosáhnout vytvořením serveru pro tuto aplikaci, který by obsahoval již připravené datové sady pro jednotlivé knihy. Aplikace by poté obsahovala klientskou část komunikace, která by umožňovala nejen hledat knihy na serveru již existující, ale i možnost přidat do databáze další chybějící knihy nebo další vydání již existující knihy s jiným hřbetem. Tento přístup by tedy umožnil uživateli hledat knihy, které ještě neviděl, komunitní rozšiřování databáze, odstranil by potřebu vytváření vlastní databáze. Také by bylo umožněno rozšiřování databáze knih pomocí aplikace na počítači s větším výkonem než má mobilní zařízení, a tedy rychlejším zpracováním pomocí předem vyfocených fotek. Nicméně umožněním tohoto aplikací vytvořené v jiném jazyce než Java by vyžadovalo změnu formy ukládání dat, například ukládat klíčové body do formátu *JSON*.

6.3 Uchování podrobnějších informací o knihách

V prototypu je o knize uchováváno pouze jméno, kterým je daný záznam pojmenován. Toto je sice dostačující pro nalezení knihy, u které známe jméno, ale v případě že uživatel ví například jen autora a se jménem knihy si není jistý, nebo jde hledat další díl právě dočtené knihy, mohlo by pro aplikaci být přínosem umožňovat vyhledávat a třídit knihy podle dalších specifikacích.

Minimálně by bylo vhodné, aby bylo možné knihy vyhledávat nejen podle jména knihy, ale i autora. Dalším kritériem pro vyhledávání by mohl být žánr knihy nebo série, která se například hodí, když uživatel dočte jeden díl a jde jen vyměnit za další, nebo když si půjčuje celou sérii, a ta není v knihovně uložená na jednom místě. Někdy může nastat situace, že si uživatel chce přečíst nějakou knihu, ale neví jakou, jen má například vybraný žánr, jedním řešením této situace je dojít ke knihovně a prohlížet knihy a hledat tu, která jej zaujme, další možností je otevřít si nějaký knihovní systém, vybrat si v něm knihu a tu jít poté hledat pomocí aplikace, ale v případě že vytváříme aplikaci, je v našem zájmu, aby ji uživatelé používali, což je pravděpodobnější v případě, že kromě ní nemusí mít další aplikaci, kde by mohli procházet knihy podle různých informací, a proto by bylo lepší umožnit uživateli sednout si do křesla a vybrat si knihu již v aplikaci, kterou pak bude danou knihu hledat i v knihovně. Pro tuto činnost by bylo vhodné umožnit i vytvářet vlastní seznamy knihy a procházet poté knihy jen z toho seznamu, například knihy co má uživatel doma.

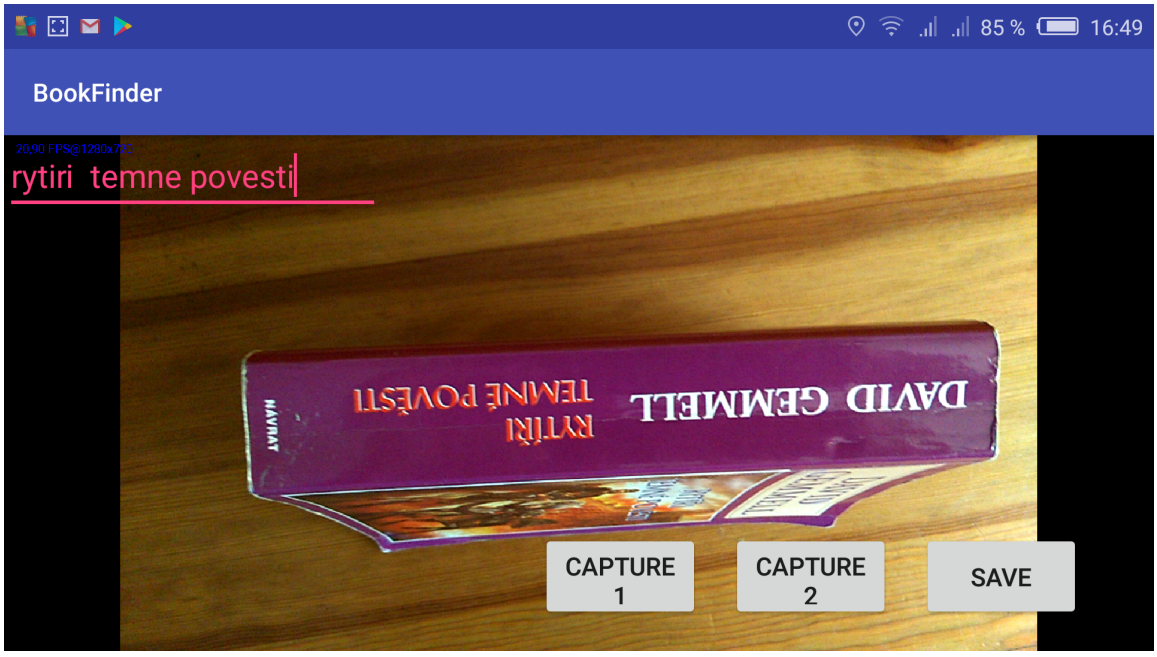
Kapitola 7

Závěr

Cílem této práce bylo navržení, vytvoření a otestování nástroje pro vyhledávání knih v knihovně pomocí kamery na zařízení se systémem Android. Navržený nástroj knihu vyhledává podle předem uloženého záznamu o knize, který obsahuje zajímavé body získané z fotografie hřbetu knihy. Nástroj byl vytvořen s pomocí knihovny OpenCV.

První část práce se zabývá popisem existujících algoritmů pro detekci objektů v obraze. Pro navrhovaný nástroj jsem zvolil metodu vyhledávání objektů pomocí klíčových bodů. Jako algoritmy pro tento nástroj byly zvažovány algoritmy poskytované knihovnou OpenCV.

Další částí práce bylo navržení samotného nástroje pro vyhledávání knih. Pro zvolení vhodného nástroje jsem vytvořil několik skriptů v jazyce Python, pomocí kterých jsem se seznámil s vyjmenovanými algoritmy a otestoval jejich použití v tomto konkrétním případě. Jako nejvhodnější se projevil algoritmus *ORB* pro detekci klíčových bodů a extrakci jejich deskriptorů a přístup *brute-force* pro vyhledávání odpovídajících bodů pomocí Hammingovy vzdálenosti. Pro otestování navrženého nástroje jsem implementoval jednoduchý prototyp

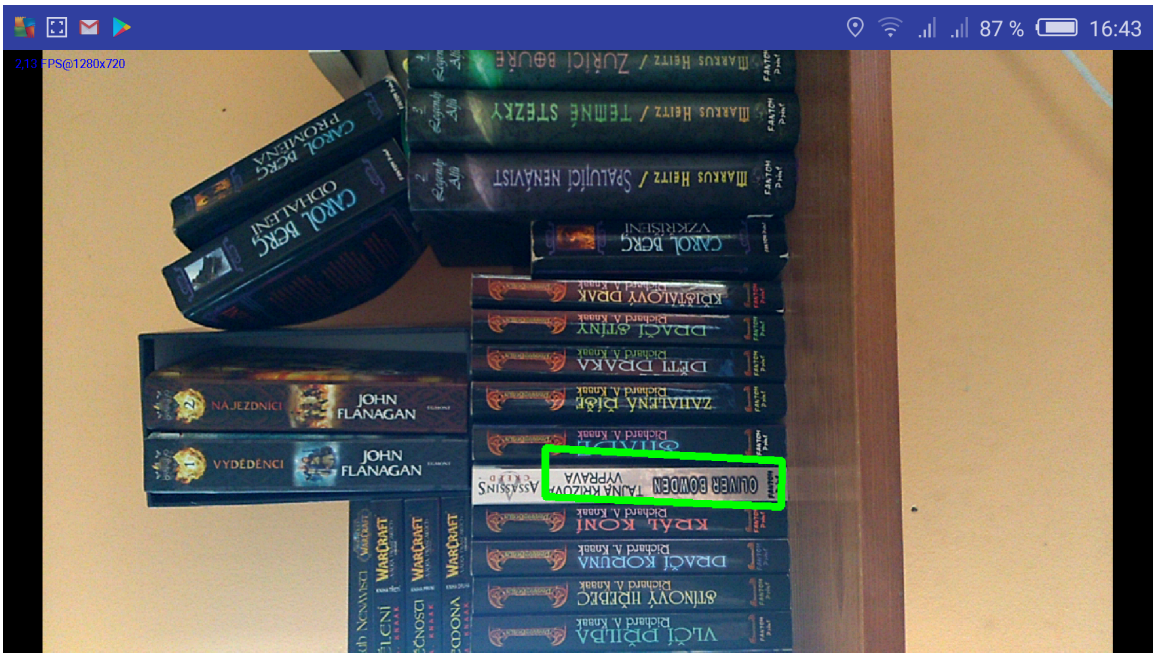


Obrázek 7.1: Pořizování nového záznamu.

Zařízení	**Procesor**	**Rychlost vyhledávání**[fps]
Nubia Z9 max	Snapdragon 8939 8j 2GHz	2.34
Xiaomi Redmi 4A	Snapdragon 425 4j 1.4GHz	4.68
Samsung Galaxy A5 2016	Exynos 7580 8j 1.6GHz	3.31
Huawei ALE-L21	Cortex-A53 8j 1.2 GHz	3.60

Tabulka 7.1: Tabulka rychlosti vyhledávání na různých zařízeních

aplikace pro systém Android, která tento nástroj využívá. Nástroj vyhledává knihy průměrnou rychlostí 3.48 snímku za sekundu, jeho úspěšnost záleží na různých faktorech, jako je osvětlení, kvalita klíčových bodů v záznamu, šířka hledané knihy, vzdálenost, na jakou se kniha hledá a jedinečnost vzhledu knihy oproti ostatním na polici v jejím okolí. Například pokud je kniha oproti ostatním výrazně jedinečná, nástroj je schopen ji najít na vzdálenost až 50 centimetrů, naopak pokud jsou knihy téměř stejné, dochází k problému že i na vzdálenost 10 centimetrů a menší zvolí knihu špatnou v 35 % případů, nebo je špatně určena matice homografie a zvýraznění vytvoří zvláštní tvary místo obtažení hledané knihy.

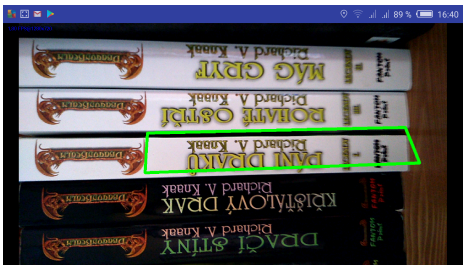


Obrázek 7.2: Aplikace je schopná najít jedinečně vypadající knihu i na 50cm.

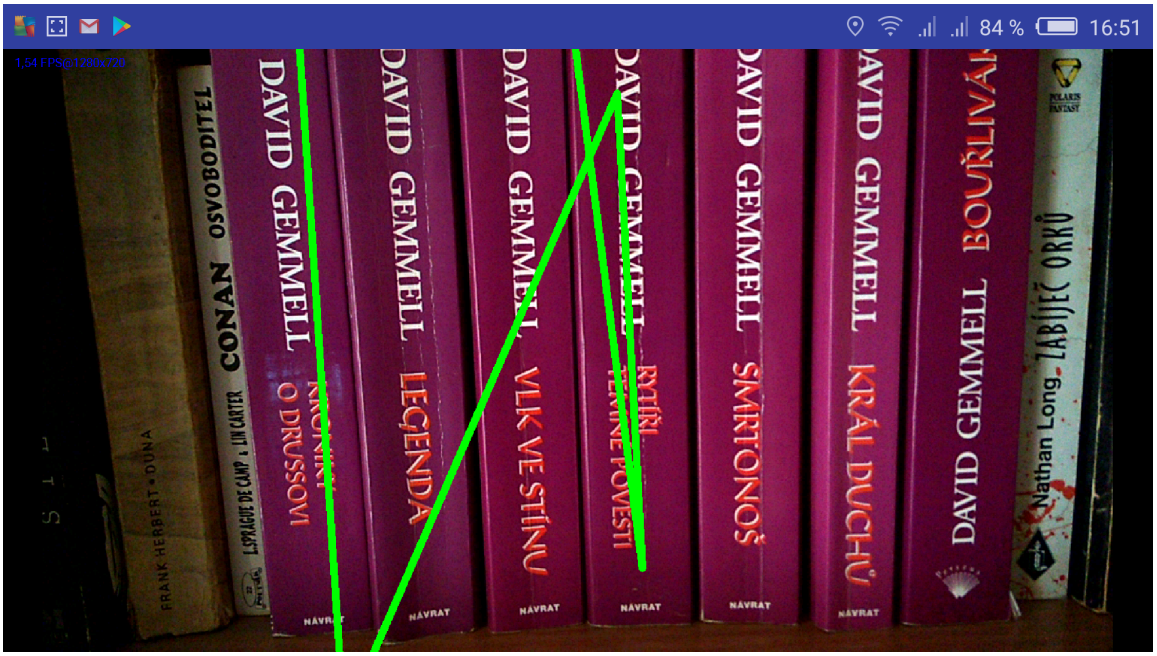
Při testování rychlosti vyhledávání oproti člověku byl nástroj běžící na zařízení Nubia Z9 max rychlejší jen v několika výjimečných případech při vyhledávání pro hledajícího člověka známé knihy i v naprosto neznámé nebo přeházené knihovně, pokud ale hledající věděl jen název knihy a jméno autora tak byl nástroj rychlejší ve více případech, ale vždy záleželo na vlastnostech hledané knihy a jejího okolí.

Nástroj by mohl v některých případech ve vhodných podmínkách být použitelný, ale pro větší uplatnění by potřeboval nějaká vylepšení, například optimalizaci pro umožnění rychlejšího vyhledávání snímků s větším rozlišením pomocí více klíčových bodů v záznamu, čímž by bylo dosaženo kvalitnějších výsledků a možnosti najít knihy i na větší vzdálenost.

Obrázek 7.3: Podobné knihy - správný nález



Obrázek 7.4: Podobné knihy - špatný nález



Obrázek 7.5: Při hledání mezi podobnými knihami, nebo při hojném výskytu podobných bodů k bodům z hledané knihy může dojít ke špatnému získání matice homografie.

Literatura

[1] Bay, H.; Ess, A.; Tuytelaars, T.; aj.: Speeded-up robust features (SURF). *Computer vision and image understanding*, ročník 110, č. 3, 2008: s. 346–359.

[2] Bay, H.; Tuytelaars, T.; Van Gool, L.: Surf: Speeded up robust features. In *European conference on computer vision*, Springer, 2006, s. 404–417.

[3] Beis, J. S.; Lowe, D. G.: Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, IEEE, 1997, s. 1000–1006.

[4] Beneda, M.: Homografie a epipolární geometrie. *Trilobit: Odborný vědecký časopis*, 2010.

[5] Bentley, J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM*, ročník 18, č. 9, 1975: s. 509–517.

[6] Boykov, Y. Y.; Jolly, M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, ročník 1, IEEE, 2001, s. 105–112.

[7] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[8] Calonder, M.; Lepetit, V.; Strecha, C.; aj.: Brief: Binary robust independent elementary features. In *European conference on computer vision*, Springer, 2010, s. 778–792.

[9] Fischler, M. A.; Bolles, R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, Elsevier, 1987, s. 726–740.

[10] Gargenta, M.: *Learning android*. O'Reilly Media, Inc., 2011.

[11] Harris, C.; Stephens, M.: A combined corner and edge detector. In *Alvey vision conference*, ročník 15, Citeseer, 1988, s. 10–5244.

[12] Hassaballah, M.; Abdelmgeid, A. A.; Alshazly, H. A.: Image features detection, description and matching. In *Image Feature Detectors and Descriptors*, Springer, 2016, s. 11–45.

[13] Jiří, V.; Miroslav, U.: *Programujeme pro Android: 2., rozšířené vydání*. Grada Publishing a.s., 2013, ISBN 9788024788548.

[14] Lee, S.; Jeon, J. W.: Evaluating performance of Android platform using native C for embedded systems. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, IEEE, 2010, s. 1160–1163.

[15] Lowe, D. G.: Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, ročník 2, Ieee, 1999, s. 1150–1157.

[16] Lowe, D. G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, ročník 60, č. 2, 2004: s. 91–110.

[17] Mikolajczyk, K.; Schmid, C.: A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, ročník 27, č. 10, 2005: s. 1615–1630.

[18] Muja, M.; Lowe, D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, ročník 2, č. 331-340, 2009: str. 2.

[19] Muja, M.; Lowe, D. G.: Fast matching of binary features. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, IEEE, 2012, s. 404–410.

[20] Muja, M.; Lowe, D. G.: Scalable Nearest Neighbor Algorithms for High Dimensional Data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 36, 2014.

[21] dev team OpenCV: OpenCV4Android SDK. 05.05.2018.
URL https://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html

[22] Reynolds, D.: Gaussian mixture models. *Encyclopedia of biometrics*, 2015: s. 827–832.

[23] Rogers, R.; Lombardo, J.; Mednieks, Z.; aj.: *Android application development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.

[24] Rosin, P. L.: Measuring corner properties. *Computer Vision and Image Understanding*, ročník 73, č. 2, 1999: s. 291–307.

[25] Rosten, E.; Drummond, T.: Machine learning for high-speed corner detection. In *European conference on computer vision*, Springer, 2006, s. 430–443.

[26] Rother, C.; Kolmogorov, V.; Blake, A.: Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (TOG)*, ročník 23, ACM, 2004, s. 309–314.

[27] Rublee, E.; Rabaud, V.; Konolige, K.; aj.: ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on*, IEEE, 2011, s. 2564–2571.

[28] Saha, A. K.: A Developer's First Look At Android. *Linux For You*, ročník 60, 2008: s. 48–50.

[29] Schmid, C.; Mohr, R.: Local grayvalue invariants for image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, ročník 19, č. 5, 1997: s. 530–535.

[30] Szeliski, R.: *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[31] Tuytelaars, T.; Mikolajczyk, K.; aj.: Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, ročník 3, č. 3, 2008: s. 177–280.

[32] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, ročník 1, IEEE, 2001, s. I–I.

Seznam příloh

Příloha A

Návod na instalaci a používání prototypu nástroje

Po nainstalování aplikace je potřeba pro funkčnost nástroje povolit přístup ke kameře.

Pro přidání nové knihy je nutné pořídit snímky pomocí tlačítek **CAPTURE 1** a **CAPTURE 2**, a poté záznam uložit tlačítkem **SAVE**. Záznam bude uložen pod názvem uvedeným v textovém poli zobrazeným při pořizování snímků, pro názvy s mezerou nebude nic uloženo.

Pro vyhledání stačí vybrat záznam v menu a stisknout tlačítko **FIND**, vyhledávaná kniha by na obrazovce měla být zvýrazněna zelenými okraji a zařízení při nálezu vibruje.