VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE OBJEKTŮ POMOCÍ KINECTU

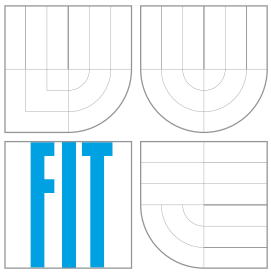
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ŘEHÁNEK

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE OBJEKTŮ POMOCÍ KINECTU

OBJECT DETECTION USING KINECT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN ŘEHÁNEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL ŠPANĚL, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

S příchodem zařízení Kinect se otevřely možnosti, jak jednoduše využít hloubku obrazu ve zpracování obrazu. Cílem této práce je popsat metodu, kterou jsem navrhnul pro rozpoznávání a detekci objektů v hloubkové mapě. Pro rozpoznávání objektů použiji metodu Bag of Words, ve které jako deskriptor hloubkové mapy použiji metodu Spin Image. Spin Image je jeden z několika přístupů k popisu hloubkové mapy, které ve své práci popíši. O vyhledání objektu v obraze se postará metoda klouzajícího okna, která je vylepšena o využití hloubkové informace pro zrychlení prohledávání.

Abstract

With the release of the Kinect device new possibilities appeared, allowing a simple use of image depth in image processing. The aim of this thesis is to propose a method for object detection and recognition in a depth map. Well known method Bag of Words and a descriptor based on Spin Image method are used for the object recognition. The Spin Image method is one of several existing approaches to depth map which are described in this thesis. Detection of object in picture is ensured by the sliding window technique. That is improved and speeded up by utilization of the depth information.

Klíčová slova

Detekce objektů, hloubkový obraz, hloubková mapa, RGB-D, rozpoznávání v obraze, deskriptory, Spin Image, balík slov, klouzající okno.

Keywords

Object detection, depth image, depth map, RGB-D, image recognition, descriptors, Spin Image, bag of words, sliding window.

Citace

Martin Řehánek: Detekce objektů pomocí Kinectu, diplomová práce, Brno, FIT VUT v Brně, 2012

Detekce objektů pomocí Kinectu

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D.

.....
Martin Řehánek
16. května 2012

Poděkování

Tímto bych chtěl poděkovat Washingtonské univerzitě za poskytnutí jimi vytvořené datové sady hloubkových map. Hlavně bych chtěl poděkovat panu Ing. Michalu Španělovi, Ph.D. za rady a pomoc při tvorbě mé práce.

Obsah

Kapitola 1

Úvod

Problém klasifikace a rozpoznávání objektů v obraze je aktuální a komplexní úkol, který stále prochází vývojem. Vzdálený cíl, ke kterému tento vývoj nejspíše směřuje, je dokázat zpracovat obrazové informace stejně rychle, plynule a univerzálně, jako to dokáže lidský mozek. I přes současný technologický rozmach je tato vize nejspíše desítky let vzdálená. Přesto si dnešní způsoby klasifikace a rozpoznávání obrazu v mnoha ohledech berou za vzor principy, kterými lidský mozek dokáže obraz zpracovat. Tyto postupy bylo potřeba přizpůsobit možnostem současné techniky, která nedosahuje ani takového výpočetního výkonu, ani takové úrovně paralelismu, jaké je schopen lidský mozek.

Lidský mozek má podstatnou výhodu. Pracuje s hloubkou obrazu zcela přirozeně a s rozpoznáváním objektu mu to v mnoha ohledech pomáhá. Tato možnost byla pro programátory do nedávna nedostupná, z pohledu komplikovanosti, výpočetní náročnosti nebo finanční nákladnosti. Změna nastala s příchodem zařízení Microsoft Kinect, který byl najednou lehce k dostání pro kohokoliv a odstraňoval problémy spojené se získáním hloubky obrazu. Toto zařízení otevřelo nové možnosti pro zpracování obrazu, které dává hloubka obrazu. Najednou lze nalézt na internetu mnoho ukázek, jak využít hloubku obrazu, ať už pro jeho prvoplánové využití, k ovládání her pohybem, tak i u vědeckých projektů, jako je robotika. Proto jsem se rozhodl napsat diplomovou práci, kde využiji hloubkovou informaci získávanou z Kinectu, a použiji ji pro detekci objektů v obraze, aby snad našla využití v robotice.

Cílem mé diplomové práce je navrhnout metodu, která bude schopna v časovém úseku řádu sekund vyhledat v obraze objekt a rozpoznat, zda se jedná hledaný objekt nebo hledaný druh objektu. Jako základní prvek pro rozpoznávání použiji deskriptor Spin Image, který popisuje vztah mezi hlavním bodem a body kolem něho na objektu v prostoru. Jelikož samotný Spin Image nedokáže univerzálně popsat celý objekt, využiji metodu Bag of Words (balík slov), díky které dokážu popsat kompletní objekt a získám tím data, které mohu klasifikovat. O třídění těchto dat se postará SVM klasifikátor. V neposlední řadě, aby bylo co rozpoznávat, je potřeba účinně procházet scénu. O to se postará metoda klouzajícího okna, kterou vylepším o využití hloubkové informace pro zrychlení prohledávání.

Text mé diplomové práce je rozdělen do několika kapitol. V následující kapitole vysvětlím, jak se rozlišují objekty, neboli klasifikují, a zároveň několik metod představím. Ve 3. kapitole se pokusím vysvětlit metody, které hledají objekty v obraze a jak se popisují v hloubkové mapě. Ve 4. kapitole popíši, jak funguje Kinect a kde se využívá. V 5. kapitola rozepíši navrhnutí metody pro detekci objektu v obraze pomocí Kinect. V 6. kapitole přistoupím k popisu programové implementace metody. V předposlední 7. kapitole shrnu získané vlastnosti metody a výsledky jejího testování. Svou práci uzavřu v 8. kapitole.

Kapitola 2

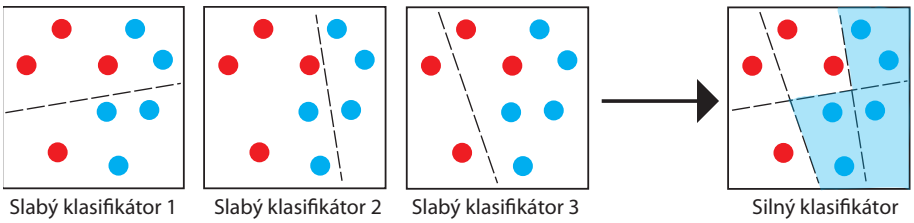
Klasifikace a rozpoznávání

Stejně jako mozek dokáže rozpoznávat objekty v obraze pomocí učení se, také současné metody počítačového rozpoznávání objektů se snaží přistupovat k problému stejným způsobem, tedy strojově se učit neboli klasifikovat. Mezi jedny z nejrozšířenějších klasifikátorů patří strojové učení AdaBoost [10] (Adaptive Boosting), neuronové sítě [25] a Support Vector Machine [27] [5](dále také SVM). Proto v této kapitole každou tuto metodu krátce vysvětlím a nakonec více vysvětlím metodu SVM, kterou dále budu používat.

2.1 Druhy klasifikátorů

AdaBoost

Tento algoritmus pro strojové učení popsali Yoav Freund a Robert Schapire ve své práci [10]. Je to algoritmus, který využívá množství jednoduchých klasifikátorů pro jejich spojení do jednoho výkonného klasifikátoru, jak ukazuje obr. 2.1. Tento algoritmus je adaptivní ve smyslu upravování důležitosti jednotlivých jednoduchých klasifikátorů podle výsledků předchozích provedených rozpoznávání. Problém tohoto algoritmu je jeho citlivost na šum v datech. Na druhou stranu nemá tendenci se přetrénovat, což je problém u většiny učících se algoritmů.



Obrázek 2.1: Schematické znázornění principu AdaBoost [převzato z internetu]

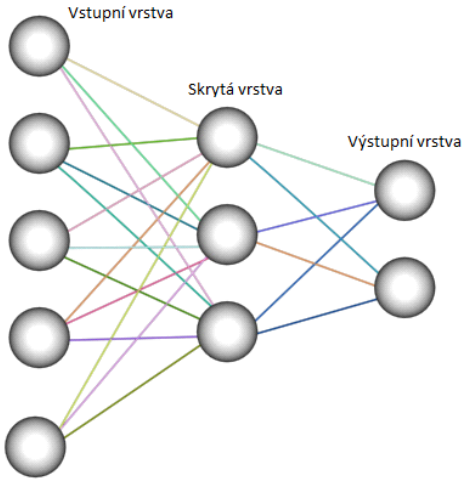
Slabé klasifikátory se volají v AdaBoost v každé sérii po cyklech $t = 1, \ldots, T$. Pro každé volání se distribuuje váha Dt, která reprezentuje důležitost tohoto klasifikátoru v datové sadě, na které se algoritmus učí. V každém kole se váha každého špatně klasifikovaného vzorku zvýší a váha správně rozpoznaného vzorku sníží, a proto se nový klasifikátor zaměřuje jen na vzorky, které se zatím nepodařilo správně rozpoznat.

Neuronové sítě

Tato metoda vychází z principu, jak pracuje neuronová síť v lidském mozku. Neuronové sítě [25] představují skupinu vzájemně propojených umělých neuronů a pro zpracování informace využívají silu propojení jednoduchých rozhodovacích jednotek. Ve většině případů je umělá neuronová síť flexibilní systém, který si distribuuje nastavení neuronu v průběhu učící fáze. Moderní neuronové sítě jsou nelineární nástroje pro vytváření statistických dat. Nejčastěji jsou používány pro vytváření vztahu mezi vstupem a výstupem nebo pro hledání podobností ve vstupních datech.

V praxi neuronové sítě představují propojení mezi neurony v různých vrstvách. Jako příklad může být tří vrstvý systém. První vrstva obsahuje vstupní neurony, které posílají data přes synapse do druhé skryté vrstvy. Druhá vrstva potom posílá své výsledky přes synapse do třetí vrstvy neuronů, která je výstupní. Samozřejmě je možné vytvořit neuronové sítě s více skrytými vrstvami a taktéž s větším množstvím vstupních a výstupních neuronů. Samotné neurony ukládají hodnoty váhy, která upravuje hodnoty při výpočtech. Tato váha se získává při učení, které se dělí na učení s učitelem a bez učitele.

Při učení s učitelem dostává neuronová síť sadu ukázkových párů $(x, y), x \in X, y \in Y$ a pak je cílem najít funkci $f : X \to Y$, která by co nejlépe dokázala X převést na Y. Při učení bez učitele, má neuronová síť pouze na vstupu sadu nějakých dat a ty se snaží co nejlépe roztřídit na několik skupin.



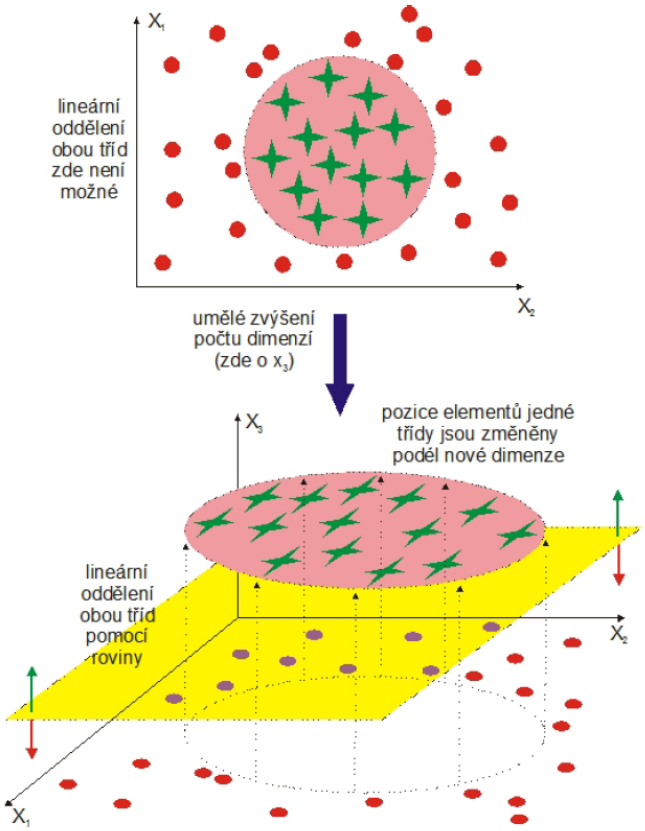
Obrázek 2.2: Schematické znázornění umělé neuronové sítě

Support Vector Machine

Alternativou k neuronovým sítím jsou takzvané podpůrné vektory [27] [5] (support vector machine, SVM), které tvoří určitou kategorii tzv. jádrových algoritmů (kernel machines). Tyto metody se snaží využít výhody poskytované efektivními algoritmy pro nalezení lineární hranice mezi třídami a zároveň jsou schopny reprezentovat vysoce složité nelineární funkce. Základním principem je převod původního prostoru do jiného, vícerozměrného, kde již lze od sebe třídy oddělit.

Princip je v podstatě jednoduchý, jak ukazuje obr. 2.3. Máme dvě třídy v dvourozměrném prostoru, které jsou nelineárně odděleny kružnicí. Přidáním dalšího rozměru vznikne

možnost posunout prvky třídy v kružnici po nově přidané ose x_3. Díky tomuto můžeme obě třídy jednoduše oddělit rovinou rovnoběžnou s rovinou danou osami původního prostoru x_1 a x_2.

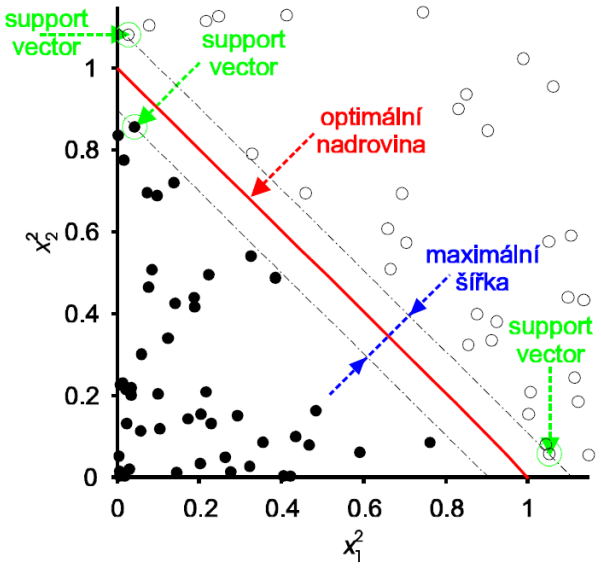


Obrázek 2.3: Princip vzniku lineárního oddělení dvou tříd s nelineárními hranicemi pomocí přidání další dimenze.[Převzato z opory MUNI]

Problém je, jak zjistit, kam umístit tuto lineární hranici z hlediska co nejefektivnější kategorizace budoucích dat. Řešení je ovšem komplikováno ještě faktem, že v případě d-rozměrného prostoru je potřeba lineární oddělovač definovat rovnicí, která má d parametrů, takže hrozí, že dojde ke ztrátě obecnosti klasifikátoru přetrénováním. Uvedená potíž je proto důvodem k tomu, aby se použila metoda jádrových funkcí, která se snaží najít optimální lineární oddělovač. Tedy takový, který se snaží najít rovinu, která bude co nejdál od nejbližšího bodu z pozitivních příkladů a zároveň co nejdál od nejbližšího bodu z negativních příkladů, jak ukazuje obr. 2.4. Těmto nejbližším bodům se říká podpůrné vektory (support vector).

Matematicky popíšeme práci SVM takto. Lineární SVM hledají nadrovinu $\langle w, x\rangle + b = 0$, která maximalizuje vzdálenost od bodů z lineárně separovatelné trénovací množiny $\{(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)\}$, kde $x_1 \in \mathbb{R}^n$ je trénovací vzor a $y_i \in \{+1, -1\}$ je jeho identifikátor třídy. Problém hledání nadroviny maximalizující vzdálenost od tříd lze převést na hledání hodnot parametru α_i, které maximalizují výraz

$$\overrightarrow{\alpha} = argmax_{\overrightarrow{\alpha}}(\sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle) \tag{2.1}$$



Obrázek 2.4: Optimální oddělovací hranice.[Převzato z opory MUNI]

přičemž platí omezení

$$\alpha_i \leq 0 \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \tag{2.2}$$

Běžně nelze očekávat, že lineární oddělovač bude nalezen přímo v originálním vstupním prostoru x. Lineární oddělovač lze hledat ve vícerozměrném prostoru $F(x)$ tak, že se nahradí člen $\langle x_i, x_j \rangle$ členem $\langle F(x_i), F(x_j) \rangle$. Tento výraz se nazývá jádrová funkce $K(x_i, x_j)$. V souvislosti s algoritmy SVM je to funkce, která může být aplikovaná na dvojice vstupních dat. Existuje řada jádrových funkcí, zde uvedu jen několik základních:

- lineární $K(x_i, x_j) = x_i^T x_j$
- polynomiální $K(x_i, x_j) = (\gamma x_i^T x_j + r)^r, \gamma > 0$
- radiální $K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2), \gamma > 0$
- sigmoidní $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

2.2 Trénovací data

Trénovací data jsou důležitou součástí výše popsaných klasifikací a bez ní by byly tyto metody nepoužitelné. Zároveň je důležité se nejdřív rozhodnout, jakým způsobem k strojovému učení přistoupit. Jestli jako učení s učitelem nebo bez učitele. Pro oboje učení je vhodné všechna data předpřipravit a odstranit u nich všechny rušivé elementy, které by mohly znehodnotit učení. Zároveň je potřeba pro učení s učitelem si připravit tzv. label, který říká algoritmu, jestli vstup je hledaný objekt nebo není. V neposlední řadě je užitečné mít dostatečně velikou datovou sadu, aby bylo možno ji rozdělit na dvě skupiny. Kde v první skupině se bude trénovat a na druhé se bude testovat, jak úspěšně proběhlo trénování a jak dobře funguje rozpoznávací metoda.

Kapitola 3

Detekce objektu v obraze

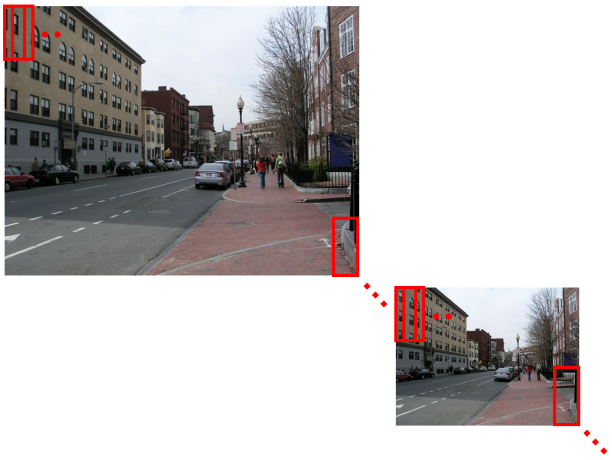
Díky klasifikaci sice dokážeme rozpoznat objekt z obrazu, ale samotný obraz není nikdy tak jednoduchý, aby se dalo očekávat, že v něm bude jen rozpoznávaný objekt. Každá scéna, je předávaná počítači jako chaos pixelů, které mohou reprezentovat neurčitý počet objektů různých tvarů a velikostí. A s tímto problémem je potřeba si nějak poradit. V současné době se v počítačovém vidění řeší tento problém dvěma různými přístupy. První je přístup "klouzajícího okna"(v anglické literatuře Sliding window approaches) a druhý přístup je "balík slov"(v anglické literetuře Bag of Word nebo BoW). Oba tyto přístupy dále popíši.

Zatím jsem popisoval všechny metody v obecné rovině, protože se dají použit mnoha způsoby a v různých kombinacích, ale vždycky slouží k roztřídění nebo detekování v obraze. Aby mohl program detekovat, potřebuje mít způsob, jak jednotně popsat data, které mu přichází na vstup. Těmto metodám se říká deskriptory nebo vlastnosti obrazu (feature). Jelikož má diplomová práce je o detekci objektu pomocí Kinect, což znamená nejen detekce v 2D obraze, ale i v hloubce obrazu, napíši v této kapitole i něco o využití hloubky obrazu (v literatuře mužeme narazit na různé názvy jako např.: depth map, range image, RGB-D, 2.5D) a o speciálních deskriptorech, které se používají pro práci v hloubkové mapě.

3.1 Metoda klouzajícího okna

Tato metoda je založená na použití hrubé síly, kdy se po obraze pohybuje identifikační okno, ve kterém se snažíme rozpoznat hledaný objekt. Toto prohledávání neprobíhá pouze pro jednu velikost objektu, ale musejí se vyhledávat i různě veliké objekty. Tím vzniká problém, kdy pro každou velikost objektu by bylo potřeba natrénovat nový klasifikátor a to pro každou velikost objektu, kterou chci vyhledat. Tento problém zbytečně komplikuje prohledávání, a proto se řeší přístupen z opačné strany. Místo změny velikosti hledaného objektu se změní velikost celého prohledávaného obrazu, což je jednodušší a rychlejší. Více se lze dočíst v článcích [22] [16].

Metoda klouzajícího okna je jen přístup, jak prohledat obraz a to ještě dost náročný. Proto je kladen důraz na vhodné a rychlé klasifikátory. Právě proto se u této metody často používá rychlý způsob klasifikace Adaboost, který jsem popsal výše. Dále se využívají různé způsoby popsání obrazu (nazývané features), jako Haar-like [18] nebo HoG [7] (Histogram of oriented gradients) metody, které se poté mohou použít v SVM klasifikátoru, který je popsány výše.

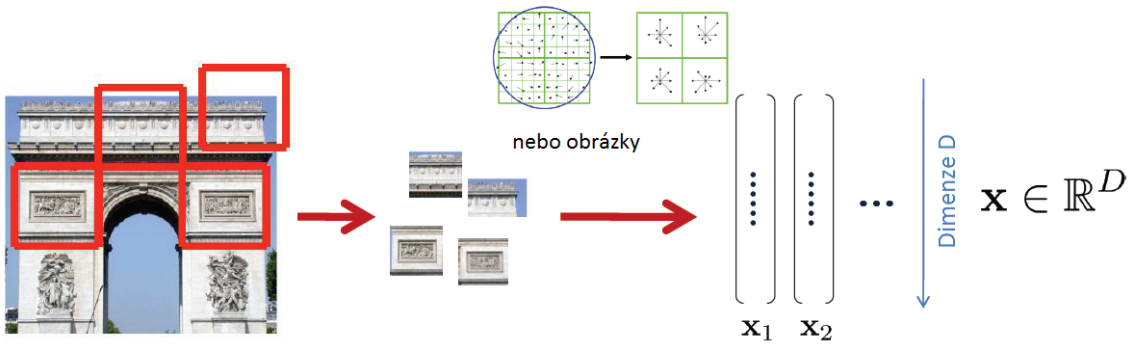


Obrázek 3.1: Průběh vyhledávání postavy pomocí klouzajícího okna.[Převzato z internetu]

3.2 Přístupy založené na BoW

Bag of Words [9] je metoda původně určená pro počítačové zpracování textu, ale nakonec se ukázal užitečný i při zpracování obrazu. V počítačovém vidění se nazývá také jako Visual codebook nebo Book of keypoints [6] a spíše než k rozpoznávání se používá ke kategorizaci obrázků.

Tato metoda pracuje tak, že se nejdříve vezme datová sada obrázků a v každém obrázku se vyhledají význačné body. Tyto body mohou být vybrány buď náhodně (pak jich musí být velké množství) nebo metodou hledání význačných bodů (např. metodou SIFT). Nyní se tyto body musí popsat nějakým deskriptorem nebo se nechají jako obrázek jeho okolí. Nad všemi získanými deskriptory se provede shlukování (clustering), díky kterému získáme jen několik reprezentujících deskriptorů, které nazýváme slova (word). Všechna tato slova tvoří dohromady kódovou knihu (codebook) nebo taky slovník.



Obrázek 3.2: Průběh zpracování obrázku pro vytvoření kódové knihy.[Převzato z internetu]

Pomocí této kódové knihy můžeme sestavit popis obrázku (feature vector), který bude reprezentován histogramem. V tomto histogramu každý sloupec popisuje, kolikrát se které slovo objevilo. Díky takovémuto druhu popisu scény v obrázku můžeme určit, jak roztřídit předloženou sadu obrázků. K samotnému třídění se používá několik metod, mezi které patří i výše popsaný SVM.

3.3 Využití hloubkové informace

2D obraz je běžně používaný vstup počítačového vidění, ale jeho používání má řadu nevýhod, které je potřeba nějak obejít nebo odstranit, jako např. ovlivnění scény různým světlem či neznalost rozložení objektů ve scéně. Získáním hloubky scény a prácí s ní se tyto problémy odstraní. Světlo nám scénu neovlivní a vyhledávání objektu je značně zjednodušeno.

Přesto má hloubková mapa svoje nevýhody, které se musí řešit kombinací s klasickým obrazem, jako například rozlišení dvou objektů stejného tvaru ale jiné textury. Dalším problémem jsou chybějící části obrazu (v anglické literatuře occlusions), které jsou způsobené kvalitou snímacího zařízení a taky faktem, že se jedná pouze o 2.5D obraz. V neposeldním případě problém překrývání objektů (v anglické literatuře clutter), kdy se muže více objektů mylně interpretovat jako jeden objekt.

Při detekci a identifikaci objektů ve scéně je hloubková mapa každopádně dalším zdrojem informací o obrazu, s kterým můžeme pracovat a získat o úroveň lepší výsledky. Záleží, jak bude informace o hloubce využita. Jestli se využije celá hloubková mapa scény pro rychlejší nalezení objektů nebo se využijí části obrazu s objekty. Potom se mohou tyto části použít k přímočarému popisu objektu nebo se použijí deskriptory pro komplexní popsání objektu v hloubkové mapě. O těchto způsobech popsání hloubkové mapy, pomocí deskriptorů, bude následující kapitola.

3.4 Deskriptory pro hloubkovou mapu

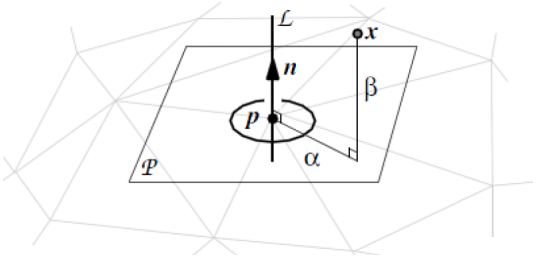
Spin-Image

Spin Image je způsob popsání povrchu tělesa, které bylo poprvé představeno Andrewem E. Johnsonem v [13] a používá se pro hledání podobných částí na povrchu a rozpoznávání objektu v 3D scéně. Tato metoda zakóduje jakýkoliv povrch zapsaný v objektově orientovaném součadném systému do pohledově orientovaného souřadného systému. Objektově orientovaný souřadný systém je systém, který je zaměřený na povrch nebo objekt, zatímco pohledově orientovaný systém je založený na úhlu pohledu na bod na povrchu. Díky tomu Spin Image získává neměnnost při jakémkoliv natočení pohledu na objekt.

Nejdůležitější částí při generování Spin Image je použití orientovaných bodů. Orientované body (označme je O) jsou body, ke kterým jsou přiřazeny směrové vektory. Povrch můžeme reprezentovat jako trojúhelníkovou síť M s vrcholy. Orientovaný bod O v povrchové síti vrcholů je definován jako 3D pozice povrchového vrcholu (označme p) a normály povrchu (označme n). Se znalostí p, n jsme schopní definovat rovinu v prostoru (označme ji P), která je kolmá na n a bod p je na jejím povrchu. Zároveň si definujeme přímku L, která prochází bodem n a má směr vektoru n. Nyní můžeme zakódovat bod x(s povrchu objektu) do válcového koordinátového systému (α, β), kde α je vzdálenost bodu x (vždy positivní hodnota) od přímky L a β je vzdálenost bodu x od roviny P (může být pozitivní i negativní hodnota). Nejlépe to půjde pochopit z obrázku 3.3. Tento postup je použit pro generování spin mapy [14], S_O. Spin mapa S_O může být reprezentovana jako funkce projekce 3D bodu x objektu do 2D souřadnic (α, β) v závislosti na hodnotách (p, n) orientovaného bodu O. Funkce projekce bude vypadat následovně:

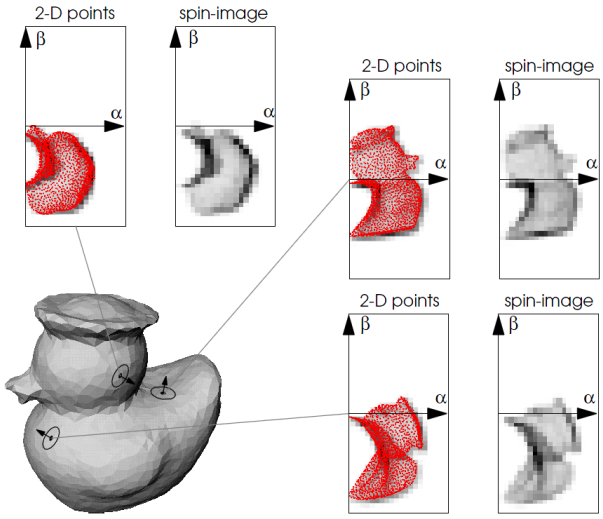
$$S_O : \mathbf{R}^3 \to \mathbf{R}^2 \qquad S_O(x) \to (\alpha, \beta) = (\sqrt{\|x - p\|^2 - (n \cdot (x - p))^2}, n \cdot (x - p))) \qquad (3.1)$$

Nyní pro získání Spin Image jednoho bodu budeme postupovat tak, že vezmeme všechny



Obrázek 3.3: Znázornění převodu x do válcového souřadného systému.[Převzato z [19]]

body povrchu objektu a použijeme na ně funkci projekce. Tímto získáme množinu dvojic (α, β). Nyní tuto množinu musíme uložit do 2D pole, které bude naším Spin Image. K tomuto účelu můžeme použít třeba bilineární interpolaci [26], která provádí interpolaci funkce dvou proměnných na pravidelnou prostorovou mřížku. Jak je zřejmé z obr. 3.4,



Obrázek 3.4: Ukázka Spin Image. U 2D points jsou červené tečky přesné hodnoty získané rovnicí 3.1 a spin-image jsou už interpolované výsledky.[Převzato z [19]]

je potřeba zavést hodnoty α_{max} a β_{max}. Pokud bychom použili všechny body objektu, zůstala by spousta nevyužitých míst prostorové mřížky, což je plýtvání prostorem. Další výhodou omezení velikosti α a β je, že při menších hodnotách je Spin Image odolnější proti problémům s překrývání nebo neúplnosti objektu. Zároveň dalším podstatným parametrem je velikost políček Spin Image (označme ho b). Pokud bychom použili příliš hustou nebo příliš řídkou prostorovou mřížku, nastal by problém, že by mohl Spin Image být paměťově a výkonově náročný nebo by mohl být příliš nepřesný pro použití. Tyto hodnoty musíme vhodně nastavit pro potřeby snímané scény.

Po zvolení těchto hodnot můžeme použít vzorce (použité v článku [19]), které vypočítají hodnoty pro velikost prostorové mřížky Spin Image

$$i_{max} = \frac{2\beta_{max}}{b} + 1 \qquad j_{max} = \frac{\alpha}{b} + 1 \tag{3.2}$$

a kam se do mřížky přiřadí vypočítaný bod z povrchu objektu

$$i = \lfloor \frac{\beta_{max} - \beta}{b} \rfloor \qquad j = \lfloor \frac{\alpha}{b} \rfloor \tag{3.3}$$

kde $\lfloor f \rfloor$ je operace zaokrouhlení f na nejbližší celou hodnotu. Přestože tato metoda na první pohled vypadá jako výpočetně náročná pro popsání celého objektu, není tomu tak. Hlavně kvůli tomu, že není potřeba popisovat všechny body povrchu. Toto si můžeme dovolit, jelikož Spin Image body v rozumně veliké oblasti jsou prakticky totožné a tudíž se dají zaměnit pouze za jeden Spin Image.

Sphere-Spin-Image

Je metoda popsaná v článcích [20] a [23] a vychází z původního Spin Image, který upravuje a vylepšuje. Jak je napsáno v [23], můžeme definovat Sphere-Spin-Image (SSI) následovně.

Máme bod p na povrchu objektu S s jeho normálovým vektorem n, které definují rovinu TP_p. Každý další bod p_k povrchu S může být popsaný dvěma parametry: 1) vzdálenost $q_k = \|p - p_k\|$; 2) vzdáleností p_k od roviny TP_p, $d_k = n \cdot (p_k - p)$, jak ukazuje obrázek 3.5(a). Dále pro každý bod p na povrchu S je dána koule s poloměrem r, jejíž střed je umístěn na bodu p (obr. 3.5(b)). Potom SSI bodu p je 2D histogram q_k a d_k tvořený body povrchu S uvnitř koule. Toto potom můžem zapsat jako

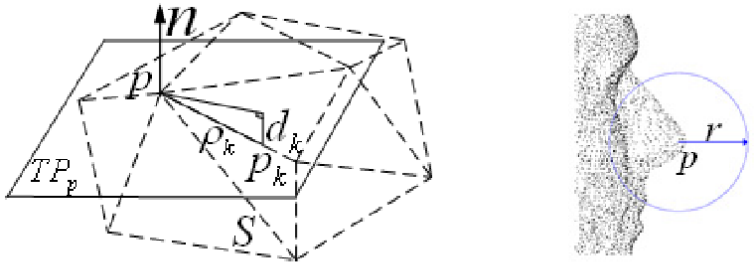
$$SSI : S^r(p) = (S^r_{ij}(p))_{m \times n, 1 \leq i \leq m, 1 \leq j \leq n} \tag{3.4}$$

$$S^r_{ij}(p) = \sum_k s^r_{ij} \tag{3.5}$$

kde

$$s^r_{ij} = \begin{cases} 1, & \text{jestliže } p_k \in S \text{ a zároveň } \|p_k - p\| \leq r \\ 0, & \text{jinak} \end{cases} \tag{3.6}$$

Sphere-Spin-Image je vlastně jen upravený Spin Image, kde se jinak počítá druhý parametr



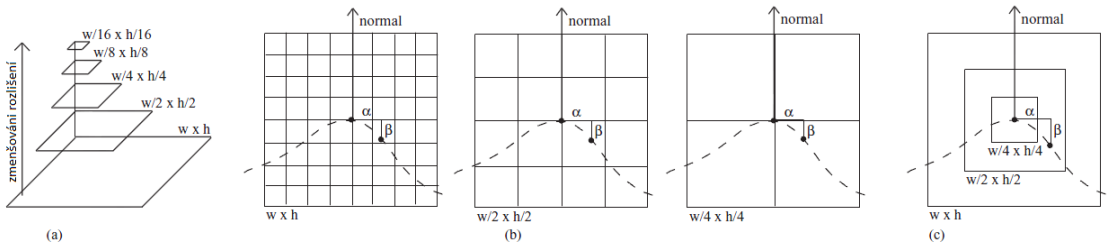
(a) (b)

Obrázek 3.5: a) znázornění vztahu v SSI. b) kružnice se středem na bodu p[Převzato z [23]]

Spin Image a omezí se počet použitých bodů pro jeho vypočítání. Tyto úpravy nevedou ani tak k zpřesnění Spin Image, ale hlavně k jeho zrychlení při vytváření Spin Image bodu.

Multi-resolution Spin Image

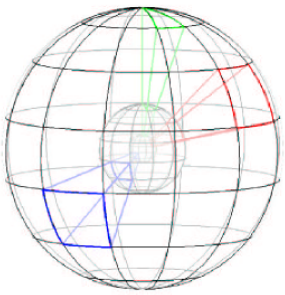
Multi-resolution Spin Image je další metoda, která upravuje klasický Spin Image a je popsána v článku [8]. Tato metoda se nesnaží ani tak zefektivnit výpočty nebo zmenšit paměťovou náročnost, ale zvýšit přesnost Spin Image přidáním více rozlišení. Aby se zachovala jednotnost stylu ukládání do paměti, klasický Spin Image se převádí do pyramidálního modelu. V každé úrovni pyramidy se zmenšuje velikost okolí, ze kterého se berou body pro počítání Spin Image a zároveň se zvyšuje velikost oken, do kterých se ukládají body v Spin Image. Názorně to lze vidět na obrázku 3.6.



Obrázek 3.6: a) Pyramida obrázku, vytvořená zvyšováním velikosti políček Spin Image. b) a snižováním podporované vzdálenosti bodů. c) Zmenšováním velikosti poliček se nevyřazují body ze Spin Image, ale až zmenšením podporované vzdálenosti ano. [Převzato z [8]]

3D Shape Context

3D Shape Context je metoda popsaná v [11], která vychází z klasické metody 2D Shape Context a rozšiřuje jí o další rozměr. U této metody se pracuje s oblastí vytýčenou koulí se středem v popisovaném bodě p a orientované podle normály N povrchu v bodě p. Tato koule se rovnoměrně rozdělí na prostory podle azimutu, výšky a ještě na poslední rozměr logaritmicky podle poloměru koule, jak je na obr. 3.7.



Obrázek 3.7: Zobrazení rozdělení koule na prostory. [Převzato z [11]]

Označíme $J+1$ rozdělení podle poloměru jako $R = \{R_0, \ldots, R_J\}$, $K+1$ rozdělení podle výšky jako $\Theta = \{\Theta_0, \ldots, \Theta_K\}$, a $L+1$ rozdělené podle azimutu jako $\Phi = \{\Phi_0, \ldots, \Phi_L\}$. Každá oblast koresponduje s jedním prvkem ve výstupním vektoru vlastnosti $J \times K \times L$. U dělení prostoru podle poloměru je R_0 rovno nejmenšímu poloměru r_{min} a R_J je rovno

maximálnímu poloměru r_{max}. Potom hranice poloměru jsou počítané podle vzorce 3.7.

$$R_j = \exp\left(ln\left(r_{min}\right) + \frac{j}{J} ln\left(\frac{r_{max}}{r_{min}}\right)\right) \tag{3.7}$$

Logaritmické členění je zvoleno pro zvýšení odolnosti algoritmu proti vadám na vstupu. Nakonec Θ a Φ jsou v rozsahu 180° a 360° děleny po s stupních. Potom pro každou oblast (j, k, l) v kouli se sčítá množství bodů s plochy objektu, které spadají do oblasti, a ve vztahu s hustotou oblasti (podle poloměru od středu koule) se počítá váha oblasti, která se ukládá do prvku výstupního vektoru $J \times K \times L$.

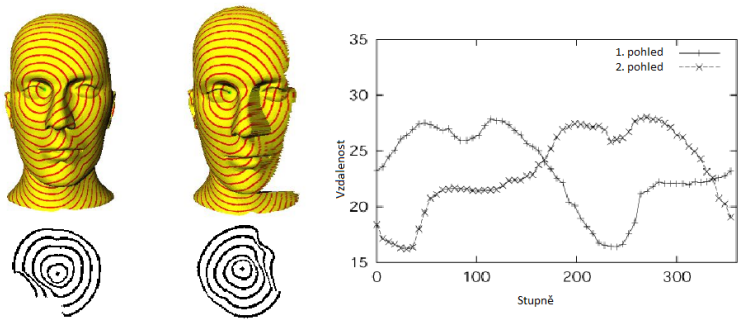
U této metody je výhodné, že díky logaritmickému členění se zvyšuje důležitost bodu blízko popisovaného bodu. Na druhou stranu je nevýhodné, že není invariantní vůči rotaci objektu, jelikož chybí druhý vektor, který by určoval natočení koule na bodu.

Harmonic shape context

Tato metoda je taktéž popsána v článku [11]. Pro její vypočítání je potřeba výstup z 3D shape context, ale poté se použijí hodnoty z oblastí pro výpočet kulové harmonické transformace pro jádro a odstraní se původní histogram 3D shape contextu. Výsledným deskriptorem je vektor popisující rozsahy transformací, které jsou poté invariantní vůči rotaci.

Point fingerprints

Tato metoda je popsána v článku [21] a je opět založená na principu promítání bodu povrchu na tečnou plochu popsanou bodem a jeho normálou. Na rozdíl od Spin Image se nepromítají všechny body, ale jen body na povrchových kružnicích (finger prints), které tvoří kontury objektu, jak je ukázáno na obrázku 3.8. Tyto kontury, které tato metoda vytváří, nemusí obsahovat jen vzdálenost od plochy, ale může se použít i jiné metody popisující vlastnosti bodu. Důležité je, že místo použití obyčejných 2D obrázků se využívají kruhové obrysy. Když se uloží informace o kontuře, každý bod v obrysu koresponduje s bodem na povrchu a může ukládat informaci o povrchu.



Obrázek 3.8: Povrchové kruhy na dvou tvářích, jejich čistě kruhová reprezentace kontur a graf porovnávající kontury z 3. kruhu.[Převzato z [21]]

Normal-based signature

Tato metoda byla definována v článku [17] takto. Máme bod s blízko povrchu S, který má atributy: pozice p, normála n a rozsah h. Pro výpočet jeho podpisu (signature) nejdřív musíme definovat $N \times M$ bodů, které jsou v kruhové oblasti na tečné ploše bodu s, podle vzorce:

$$x_{ij} = p + \frac{2ih}{N}\left(\cos\left(\frac{2\pi j}{M}\right)u + \sin\left(\frac{!\pi j}{M}\right)v\right) \tag{3.8}$$

Kde $i = 1, \ldots, N$, $j = 1, \ldots, M$, u a v jsou dvě ortogonální souřadnice tečné roviny. Pro každý bod x vypočítáme odpovídající normálu, které je váženým součtem normál v okolí bodu x. Nyní pole $N \times M$ reálných čísel $\mathbf{R}_{N\times M}$ bude určeno jako projekce pole normál na tečnu plochy bodu s. Nakonec se podpis vypočítá použitím diskrétní Cosinové transformace ve směru N a diskrétní Fourierovy transformace ve směru M na pole $\mathbf{R}_{N\times M}$. Výsledkem je nové pole, které má $n \times m$ prvků. Prvky tohoto pole převedeme na absolutní hodnoty a poté je zapíšeme do jednorozměrného pole od horního levého prvku, které bude tvořit výstup této metody.

Další používané deskriptory

Jako další možný deskriptor lze použít Local surface descriptor, popsaný v článku [4]. Využívá se zde dříve popsané metody popisu 3D povrchu, kde se popisují vlastnosti okolí bodu povrchu a z nich se získá 2D histogram, typ povrchu a jeho těžiště.

Další metoda popsaná v článku [3] se nazývá Depth kernel descriptor. Tento způsob popisu 3D objektu využívá několik jednoduchých metod pro popsání celého objektu k vypočítání jádra pro SVM a potom jeho využití ke klasifikaci objektu. Využívá metody popisu velikosti objektu, tvaru objektu, modifikovaný Spin Image, a nakonec hrany objektu.

Poslední metoda popsaná v [12] se nazývá Local feature histograms. Tato metoda opět vychází z použití několika jednoduchých způsobů popsaní 3D objektu. Využívá 2D histogram hloubky obrazu, povrchové normály a zakřivení povrchu. Narozdíl od předchozí metody se tady z těchto vlastností nedělá jádro SVM, ale prostě se jen vytvoří vektor, který se porovnává klasickými nebo statistickými metodami.

Kapitola 4

Kinect a jeho využití v počítačovém vidění

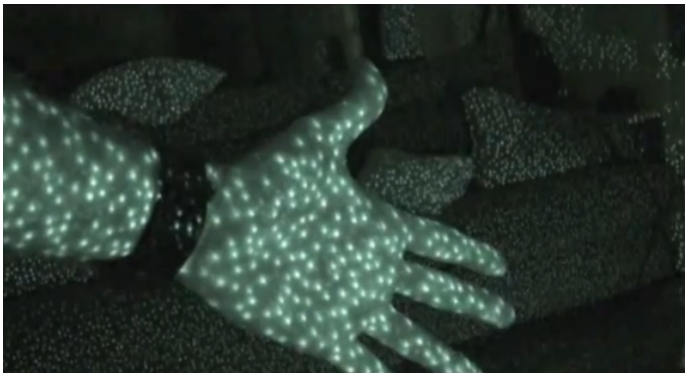
Za poslední desetiletí pokročila kategorizace a detekce objektu v obraze o notný krok dopředu. V současné době jsme svědky zrodu nové generace snímacích zařízení, která jsou schopná zaznamenávat nejen barevný obraz, ale i jeho hloubku, a přitom jsou dostupné pro běžné použití. Prvním takovým zařízením je Microsoft Kinect(obr. 4.1), uvedený do prodeje 4. listopadu 2010.

Obrázek 4.1: Zařízení Kinect.

Technologie použitou v Kinect vynalezli v roce 2005 vědci Zeev Zalevsky, Alexander Shpunt, Aviad Maizels a Javier Garcia ve společnosti PrimeSence. Kinect jako takový byl poprvé představen na zahraničním herním veletrhu E3 v roce 2009 pod názvem "projekt Natal". Tehdy Microsoft představil několik prvních věcí, které bude možno díky tomuto zařízení realizovat, jako např. snímání pohybu člověka a namapování na jeho tělo kostru pro další využití jako ovládacího prvku. Na tomto veletrhu Microsoft také prohlásil, že projekt Natal bude zařízení pouze pro tehdy novou konzoli nové generace Xbox 360. I přesto po oficiálním vydání Kinectu začaly ve světě vznikat neoficiální ovladače, které umožňovaly použití Kinectu na PC. Nakonec 16. června 2010 zveřejnil Microsoft SDK pro nekomerční využití (více informaci se lze dočíst na [24]).

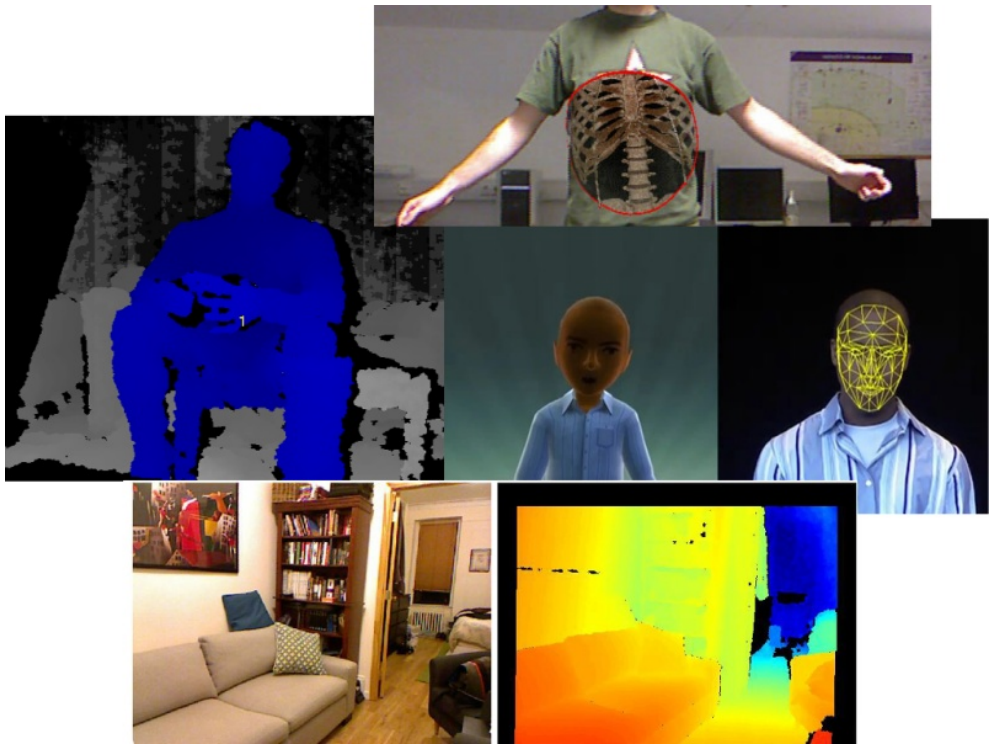
Zařízení Kinect je komplexní přístroj, který obsahuje kameru pro normální RGB obraz, kameru pro infračervený obraz, sadu mikrofonů pro stereo zvuk a nejdůležitější infračervený projekční laser. Samotné získávání hloubky obrazu funguje tak, že pomocí infračerveného laseru se pseudonáhodně promítají body do scény (Obr. 4.3). Toto mračno bodů se potom použije ke stereo triangulaci vzdálenosti od kamery. Pro triangulaci je potřeba dvou snímků. Toto řeší Kinect tak, že první snímek je obraz z infra kamery a místo druhého snímku se použije znalost rozložení promítnutých pseudo náhodných bodů na scénu. Tato triangulace je realizována procesorem přímo v Kinectu, takže konečným výstupem ze zařízení je hloubková mapa o rozlišení 640x480 při frekvenci 30 Hz.

Samotný Kinect v dnešní době začíná mít široké uplatnění. V robotice se používá pro



Obrázek 4.2: Ukázka promítaných bodu ze zařízení Kinect, zachycené pomocí infračervených brýlí. [Převzato z internetu]

orientaci robotu v prostoru nebo k vyhledávání objektů a jejich rozpoznávání. Díky tomuto zařízení začala nová epocha interakce s počítačem, prostřednictvím pohybů a gest, což umožňuje nové možnosti rozšířené realitě v počítači. Dalším využitím v počítačové grafice je získávání mračna bodů, ze kterých se vytváří 3D objekty. V neposlední řadě se Kinect hojně využívá v biometrických systémech pro identifikaci osob, například při rozpoznávání obličeje. Microsoft dál pokračuje ve vývoji Kinectu a objevují se zprávy, že nový Kinect 2, určený pro novou konzoli Xbox 720, by měl zvládat odezírat ze rtů. I další firmy si uvědomili potenciál takovéhoto zařízení a proto nedávno vydala firma Asus své zařízení Xtion přímo určené pro osobní počítače, které ovšem bylo taktéž vyvíjeno firmou PrimeSence.



Obrázek 4.3: Ukázka vyžití a výstupů z Kinect. [Převzato z internetu]

Kapitola 5

Návrh detektoru objektů s využitím Kinectu

V této kapitole spojím vybrané nejvhodnější metody, které jsem popsal výše v kapitolách 2 a 3, do jedné a to tak, abych vytvořil metodu, která by rozpoznávala objekty v obraze za využití informace o hloubce obrazu. Tento návrh rozepíši do několika bodů:

- Zpracování dat z Kinectu a jejich kalibrace.
- Metoda pro detekci objektu:
  - Spin Image
  - Bag of Words
  - Sliding Window
- Datová sada

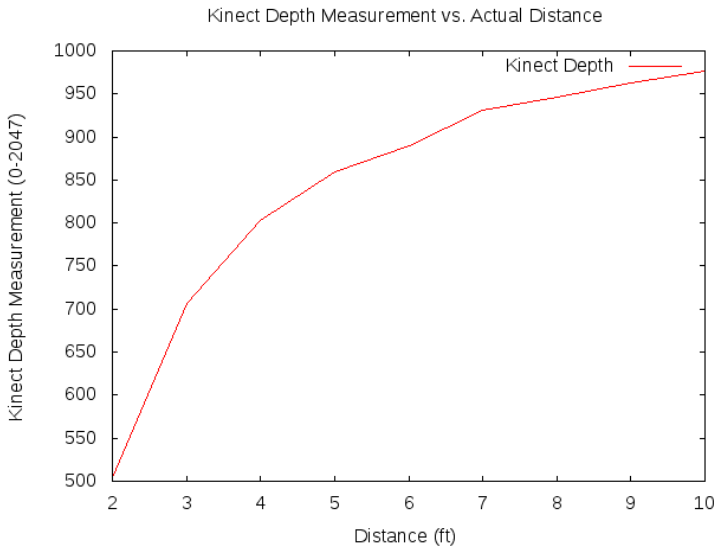
V prvním kroku popíši, jak bude potřeba zpracovat data, která přichází z Kinect. Poté přistoupím k samotné mu návrhu metody, kde rozepíši jednotlivé fáze detekce. Nakonec přistoupím k popsání a navrhnutí datových sad potřebných k detekci.

5.1 Zpracování dat z Kinect

Prvním krokem při detekci obrazu je zpracování dat přicházejících ze snímacího zařízení tak, aby byla vhodně uzpůsobena pro další zpracování. Výstupními daty, která přichází ze zařízení Kinect, je matice celých kladných čísel o velikosti 640x480. Tato čísla vycházející ze zařízení nejsou přesné hodnoty vzdáleností od obrazu. Než je možné z této hodnoty získat opravdovou prostorovou hloubku, je potřeba jí normalizovat, protože se její průběh mění se vzdáleností a není lineární, jak je vidět na obrázku 5.1. Normalizace hodnot je vyjádřena vztahem:

$$d = \frac{1}{8} * (doff - kd) \tag{5.1}$$

Kde d je normalizovaná hodnota, kd je hodnota přicházející z Kinectu, $doff$ je specifická hodnota kompenzace pro Kinect, která většinou mívá hodnotu kolem 1090. Násobí se 1/8, protože kd nejspíše nabývá hodnoty v jednotkách 1/8 pixelu. Potom matematický vztah



Obrázek 5.1: Graf vztahu mezi daty z výstupu Kinect a reálně naměřenými vzdálenostmi. [Převzato z http://mathnathan.com/2011/02/03/depthvsdistance/]

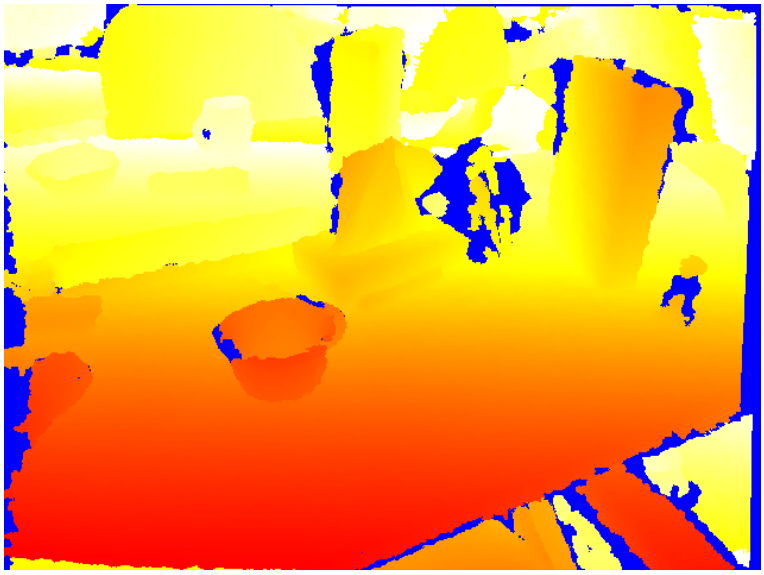
mezi normalizovanou hodnotou ze zařízení Kinect a reálnou hloubkou obrazu se zapíše takto:

$$z = \frac{b * f}{d} \tag{5.2}$$

Kde z je hloubka obrazu (v metrech), b je horizontální vzdálenost mezi snímacími kamerami (v metrech), f je běžná ohnisková vzdálenost Kinect kamery (v pixelech) a nakonec d je normalizovaná hodnota z Kinect(v pixelech).

Tímto postupem je možné získat reálnou hloubku obrazu v každém pixelu. Problém je v tom, že Kinect není dokonalé zařízení pro snímání hloubky obrazu. Jelikož pracuje na principu využití infračerveného laseru, je možné, že některé materiály laser pohltí nebo úplně odrazí, takže se ztratí bod, podle kterého se trianguluje hloubka. Následkem čehož vznikají nežádoucí artefakty v hloubkové mapě (jak jde vidět na obrázku 5.2). Pro odstranění těchto artefaktů využiji metodu, kterou byla použita v články [15]. Touto metodou je Median velikosti 5x5, který se aplikuje pouze na místa, kde chybí informace o hloubce. Medián vlastně využije okolí bodu pro aproximaci možné hodnoty.

Posledním krokem při zpracování hloubkové mapy z Kinectu je její převedení do mračna bodů. Tento krok je velice důležitý, jelikož pokud využiji pouze hloubkovou mapu, pracoval bych s paralelní projekcí, ale Kinect snímá perspektivně. To znamená, že bych se připravil o reálnou velikost objektů v závislosti na hloubce obrazu, což je velice důležitá informace při popisu objektu ve 3D prostoru. Jak jde vidět na obrázku 5.3 a), tak dva různé objekty převedené do 3D prostoru z výstupních dat Kinectu, jsou v jakékoliv hloubce stále stejně veliké, což ale neodpovídá snímané scéně. Ve snímané scéně na obrázku 5.3 b) jde vidět, že ve větší hloubce je malý objekt z hloubkové mapy ve skutečnosti větší. To znamená, že každý bod v obrazu se bude muset přepočítat ve vztahu ke své vzdálenosti a poloze na



Obrázek 5.2: Hloubková mapa získaná z Kinect. Modré části jsou oblasti, kde nebyla získána hloubková informace.

hloubkové mapě. Tento vztah popisují přibližně vzorce:

$$x' = \frac{x}{H_o} * z_{xy} \tag{5.3}$$

$$y' = \frac{y}{H_o} * z_{xy} \tag{5.4}$$

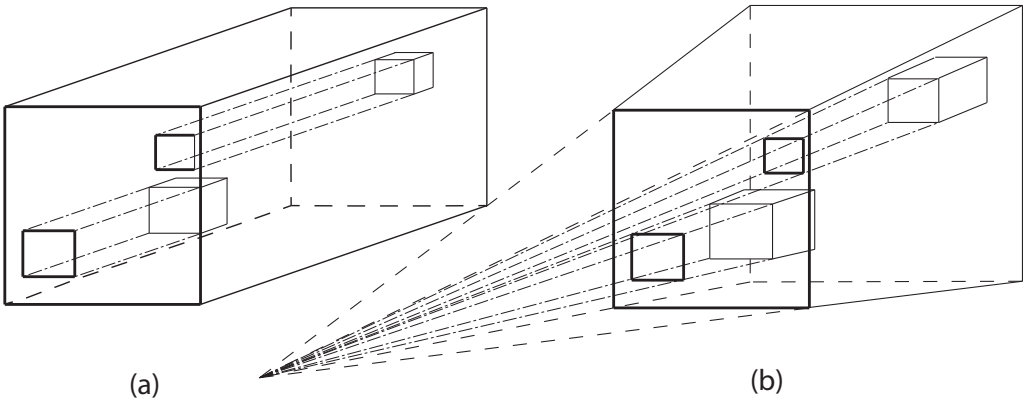
Kde x a y jsou souřadnice bodu v hloubkové mapě, H_o je hloubka ostrosti objektivu kamery, která je u zařízení Kinect přibližně 580 pixelů nebo ve stupních 57,5°, z_{xy} je hloubka bodu na souřadnicích x, y a x', y' jsou reálné souřadnice bodu v prostoru. Díky tomuto přepočtu získáme reálné současnice $[x', y', z_{xy}]$ bodu v 3D prostoru scény a tudíž můžeme vypočítat vzdálenosti mezi body, což je stěžejní vlastnost, kterou potřebujeme pro výpočet Spin Image.

5.2 Metoda pro detekci objektu

Metoda, kterou budu realizovat, bude založena na kombinaci dvou přístupů a to na propojení Bag of Words a klouzajícího okna. Jako ústřední deskriptor použiji Spin Image (dále také SI). Tento deskriptor jsem zvolil, protože je to ve světě nejklasičtější metoda, se kterou se porovnávají i nově vyvinuté metody a zároveň jsem na tuto metodu nikde nenarazil v české literatuře.

Spin Image

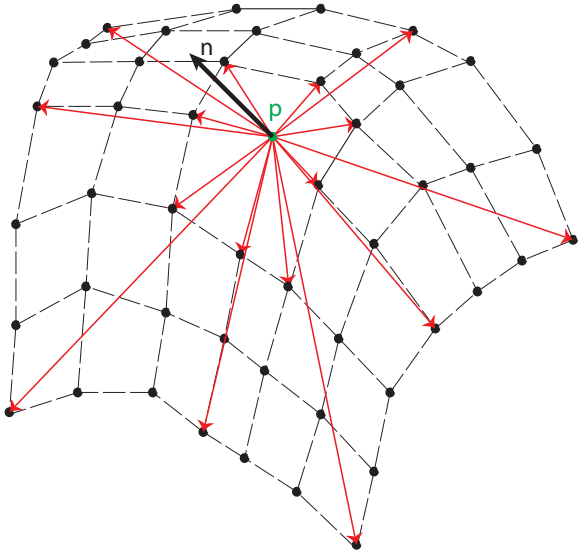
Realizace deskriptoru Spin Image není jen jednoduché použití vzorců, které jsem popsal v kapitole 3.4 na straně 9. Je důležité brát zřetel na několik faktorů, které ovlivňují správnost výstupního deskriptoru, rychlost vytvoření deskriptoru a rychlost pozdějšího používání



Obrázek 5.3: a) objekty z hloubkové mapy bez započítání perspektivy. b) Scéna jak je snímaná zařízením Kinect.

deskriptoru. Tyto faktory jsou: správné určení normály povrchu, omezení počtu bodů pro zpracování a správné nastavení parametrů Spin Image.
Správné určení normály povrchu je velmi důležité, protože Spin Image popisuje vzdálenost okolních bodů od roviny a přímky procházející popisovaným bodem. Tato přímka a rovina jsou definovány právě normálou, a pokud je špatně určená, pozbývá Spin Image veškerých výhodných vlastnosti, které by jinak měl (jako je například invariance vůči rotaci a stálost při nedostatku dat ze vstupu). Proto pro co nejpřesnější určení normál využiji 16 vektorů od popisovaného bodu k bodům v okolí tak, jak je zobrazeno na obrázku 5.4. Na tyto vektory použiji vektorový součin, kterým získám jejich kolmice. Průměrem těchto kolmic získám hledanou normálu. Směr konečné normály (nad či pod plochu) je důležitý jen v tom smyslu, že musí zůstat stejný u všech bodů Spin Image. Jinak změna tohoto směru nemá vliv na vlastnosti Spin Image.

Spin Image popisuje vztah všech bodů objektu, v mém případě všech bodů hloubkové mapy, k jednomu bodu. Jak už jsem psal u popisu Spin Image, tento přístup je dost náročný na paměť a přitom zbytečně, protože se vzdáleností od bodu klesá důležitost bodů. Proto je Spin Image omezen parametry α_{max} a β_{max}, které zmenšují Spin Image. Tímto přístupem se sice sníží paměťová náročnost, ale nesníží se výkonová náročnost, jelikož je stejně stále potřeba vypočítat vztah ke všem bodům a až potom nastupuje toto omezení, co se uloží. Z tohoto důvodu je vhodné omezit prostor bodů, které se budou využívat pro výpočet Spin Image, do pokud možno co nejmenšího výřezu z celkové hloubkové mapy ještě předtím než se začne samotný Spin Image počítat. Aby bylo možné tento výřez vytvořit, je důležité si uvědomit, že Spin Image,omezený α_{max} a β_{max} je v 3D prostoru válec se středem v bodě p určený směrem normály n a s výškou β_{max} a poloměrem α_{max} jak je ukázané na obrázku 5.5. Pokud tento válec převedeme do 2D prostoru a získáme jeho maximální a minimální rozměry, můžeme účinně minimalizovat prostor, v kterém se bude provádět výpočet Spin Image. Výhodou toho, že budeme omezovat hloubkovou mapu, je možnost zanedbat osu Z, jelikož ta míří prakticky kolmě k hloubkové mapě. Pro výpočet tedy stačí



Obrázek 5.4: Vektory potřebné pro získání normály bodu na povrchu objektu.

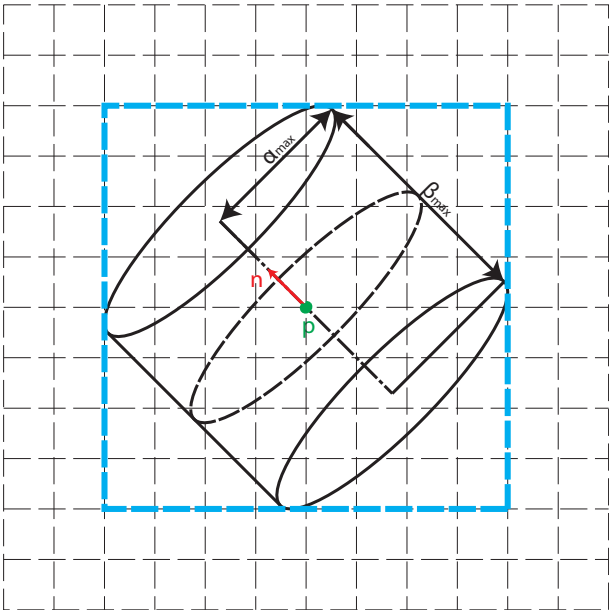
získat středy spodku a vrchu válce a potom je použít jako středy pro elipsy, jejíž šířka bude rovna $2 * \alpha_{max}$. Jelikož vypočítání parametrů pro elipsy a poté získání maxim X, Y os k výpočtu výřezu z hloubkové mapy není zcela triviální problém, raději použiji místo elips, pro výpočet jednodušší kružnice. Tento způsob vypočtu výřezu sice zanedbává několik věcí (jako je perspektiva, která má vliv i na válec Spin Image, a elipsy při výpočtu), ale přesto sníží počet výpočtů, v případě výpočtu Spin Image z hloubkové mapy přímo z Kinectu v rozlišeni 640x480, v řádu statisíců výpočtů.

Posledním krokem při návrhu Spin Image je vhodné zvolení jeho parametrů. Jak jsem již psal v kapitole 3.4, Spin Image potřebuje pro výpočet tři konstanty a to α_{max}, β_{max} a b. α_{max} a β_{max} v zásadě jen omezují prostor, v kterém se Spin Image počítá. Zvětšením těchto hodnot Spin Image nezíská žádnou výhodu, jen zvětší výpočetní a paměťovou náročnost. Extrémním snížením může jen uškodit vlastnostem Spin Image, proto není až tak potřeba testovat tyto hodnoty. Nechám je nastavené na jednu hodnotu a to 40 pixelů. Zbylý parametr b má vliv na mnohem víc věcí, jak jsem se zmiňoval výše v kapitole 3.4. Tento parametr ovlivňuje kvalitu výstupního Spin Image a to jak po stránce přesnosti, tak po stránce paměťové náročnosti. Dále teké ovlivňuje i rychlost výpočtu metod, které budou v mém návrhu využívat Spin Image, protože tyto metody pracují rychleji se vstupy, které jsou méně rozměrné. Proto parametr b otestuji pro několik hodnot a to pro $2,7$; $3,5$ a $4,3$.

Metoda klíčových slov(Bag of Words)

Tato metoda je v principu popsaná v kapitole 3.2 na straně 8. Jako hlavní deskriptor použiji samozřejmě Spin Image, který je ústředním prvkem mé práce. Zároveň abych mohl porovnat výsledky této metody s využitím Spin Image pro detekci z hloubkové mapy, použiji jako referenční deskriptor SIFT, který se u této metody běžně používá v případě detekce z klasického RGB obrazu.

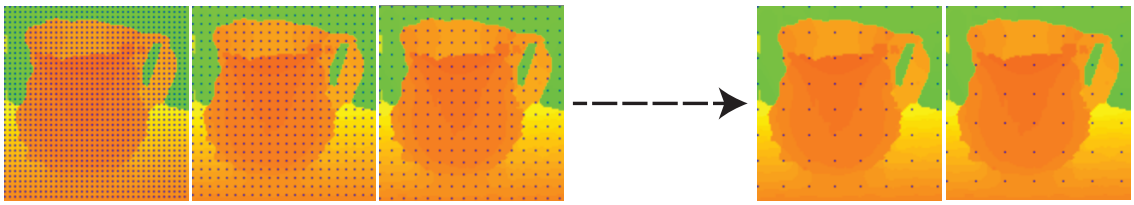
Jako první krok této metody je vytvoření slovníku, který bude tvořit ústřední prvek vy-



Obrázek 5.5: Využívaný prostor Spin Image ve 3D převedený do 2D prostoru, pro určení výřezu používaných bodu z hloubkové mapy.

užívány pro popis celého objektu. Rozhodnul jsem se vytvořit slovník, který bude obsahovat 500 slov. Pro jeho vytvoření použiji metodu K-means, která z několika set tisíc deskriptorů, které mu předložím, vytvoří jen potřebných 500 deskriptorů, které budou reprezentovat slova ze slovníku. Tyto deskriptory získám rovnoměrným rozprostřením na obrázcích z datové sady, o které budu mluvit dále.

Dalším krokem bude naučení rozpoznávání objektu. Protože pro Spin Image zatím neexistuje detektor význačných bodů, rozhodl jsem se použít pro Spin Image i pro SIFT rovnoměrné rozprostření bodů po obrazovce, které se budou popisovat těmito deskriptory. Přitom pro lepší zjištění vlastností této metody jsem se rozhodl použít několik velikostí rozestupů mezi body a to 3, 5, 7 až 17 pixelů mezi body, jak je ukázané na obrázku 5.6. Jak jsem již zmiňoval v popisu Bag of Words, z těchto vzniklých Spin Image vytvořím za pomoci předpřipraveného slovníku histogram. Tímto postupem vytvořím řadu histogramů, které budou popisovat jak obrázky hledaného objektu, tak i obrázky různých jiných objektu, a na těchto histogramech naučím klasifikátor rozpoznat hledaný objekt. Jako klasifikátor pro tuto metodu jsem si zvolil SVM metodu klasifikace, která se u metody balíků slov používá nejčastěji.

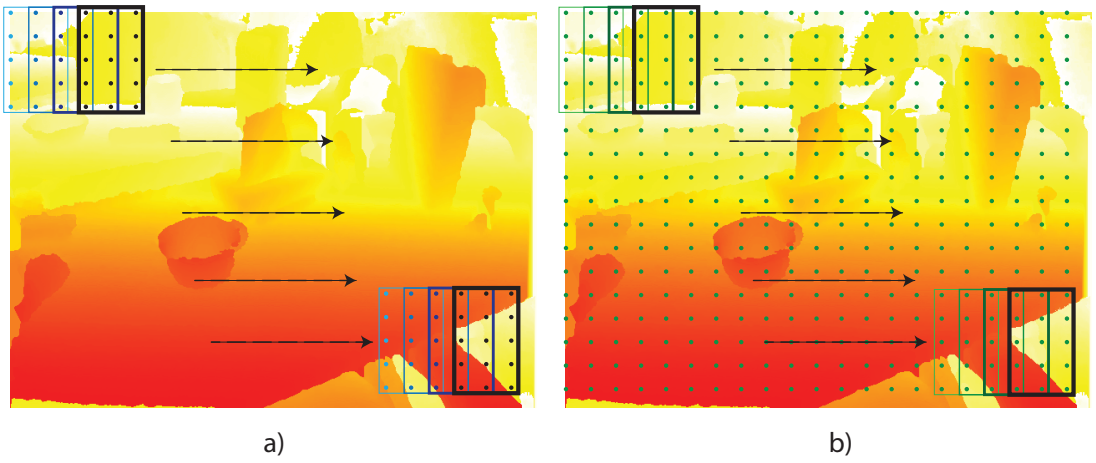


Obrázek 5.6: Rozprostření bodů na obrázku s mezerama 3, 5, 7 až 15, 17.

Metoda klouzajícího okna

Metoda Bag of Words už sama o sobě stačí k tomu, abych mohl rozpoznávat objekty, ale v praxi ještě není příliš použitelná. Pokud bych jí takto použil, musely by všechny objekty být ideálně ve stejné vzdálenosti od Kinectu. Navíc bych nedokázal rozpoznat více objektů než jeden. Ještě k tomu by obraz musel být ořezaný jen na ten objekt, který budu identifikovat, protože Kinect dokáže nejblíž zaostřit přibližně na vzdálenost 60cm a v tom případě menší objekty (než třeba hrnek) by v obraze nezabraly celou plochu. Tudíž je potřeba použít metodu, která by tyto problémy odstranila a tím by byla schopná pracovat v realističtějších podmínkách. Proto jsem si zvolil metodu klouzajícího okna, která bude odstraňovat popsané problémy. Tuto techniku jsem popsal zevrubně v kapitole 3.1 na straně 7. Přitom jí budu muset navrhnout specificky pro metody, principy a datovou sadu, které budu používat ve své celkové metodě.

Metoda klouzajícího okna, jak jsem psal výše v kapitole 3.1, pracuje s oknem, které se systematicky pohybuje po obrázku a ve vnitřku okna se provádí detekce. Budu používat rovnoměrné rozložení deskriptoru po obrázku a ty teprve využívat pro identifikaci objektu, o čemž jsem se zmiňoval u metody Bag of Words. Což znamená, že v klasickém případě metody klouzajícího okna bych měl v každém okně vypočítat znovu všechny deskriptory, jak je ukázáno v obrázku 5.7 (a). Což je neefektivní. Proto tuto metodu upravím tak, že se okno nebude pohybovat po obrázku jako, takovém, ale bude se pohybovat po výsledcích deskriptorů uložených k jednotlivým rovnoměrně rozprostřeným bodům. Díky tomuto přístupu se budou deskriptory počítat jen jednou pro každý bod, který se bude popisovat, jak je ukázáno na obrázku 5.7 (b).



Obrázek 5.7: a) metoda klouzajícího okna, při které by se v každém okně znovu počítaly nové deskriptory. b) upravená metoda, v které okno klouže už po výsledcích deskriptorů

Dalším krokem u mého návrhu této metody je změna velikosti vyhledávacího okna. Pro nasnímání co nejvíce hloubkových detailů je datová sada, kterou budu používat, snímaná z největší možné blízkosti, jakou Kinect dovolí. Proto moje metoda bude postupovat od největšího vyhledávajícího okna, to znamená od nejmenšího celkového obrazu, po nejmenší vyhledávací okno, to znamená po největší celkový obraz.

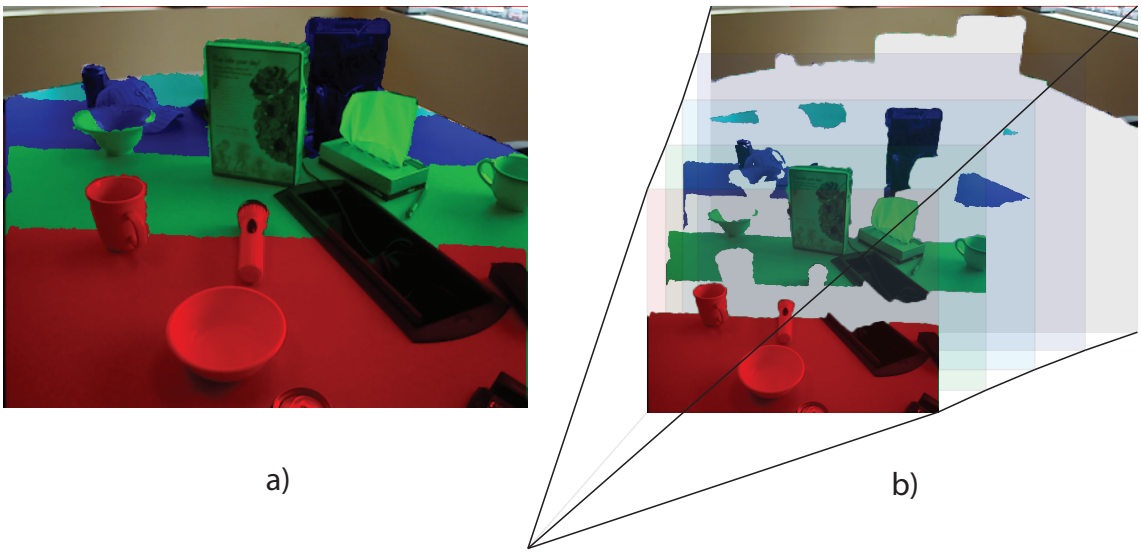
V tomto bodě bych chtěl přistoupit k popisu mého vylepšení, které významně sníží výpočetní náročnost metody klouzajícího okna. Tento přístup je založený na využití hloub-

kové informace obrazu. Klasické procházení obrazu oknem a poté zvětšení obrazu a znovu procházení oknem je dost časově náročné. Přitom se prochází stále ty samé oblasti jen ve větším obrazu, což je výrazně neefektivní prohledávání. Než použít tuto metodu je dobré se zeptat: "Proč je potřeba procházet obraz s menšími okny?". Odpověď je jednoduchá, protože v obraze se můžou vyskytovat menší stejné objekty. Proč jsou potom menší? Protože perspektiva (jak sem vysvětloval výše při kalibraci Kinectu) vzdálené objekty zmenšuje, tudíž důvod proč hledáme stejné, ale menší objekty, v obraze je ten, že tyto objekty jsou dále od obrazovky. Když si toto uvědomíme, zjistíme, že nemusíme procházet oknem obraz pokaždé všude, ale stačí nám obraz na základě hloubkové mapy získané z Kinectu rozdělit na oblasti s podobnou vzdáleností. Pro získání těchto oblastí stačí jen prahováním rozdělit hloubkovou mapu po vzdálenostech, v kterých se bude procházet určitými velikostmi oken. Takto rozložený obraz lze vidět na obrázku 5.8(b). Samozřejmě potom zvětšování jednotlivých prohledávacích úrovní je závislé na perspektivě a potom vypočet velikostí úrovní je následující:

$$x_{1..n} = \frac{x_0}{H_o} * z_{1..n} * 2 \tag{5.5}$$

$$y_{1..n} = \frac{y_0}{H_o} * z_{1..n} * 2 \tag{5.6}$$

Kde $x, y_{1..n}$ jsou rozlišení pro úrovně $1..n$, x_0, y_0 jsou hodnoty rozlišení Kinectu, takže 640 a 480, H_o je hloubka ostrosti Kinect, takže 580 v pixelech, $z_{1..n}$ je hloubka, v které bude umístěna prohledávácí úroveň.

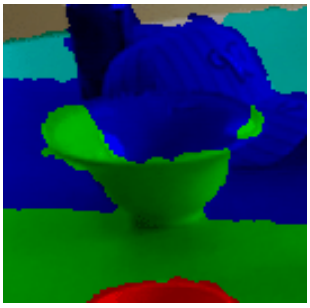


Obrázek 5.8: (a) Celistvý obraz barevně rozdělený na úrovně. (b) Rozložení obrazu na úrovně, s ovlivněním úrovní perspektivou.

Velikost mezer mezi úrovněmi je dobré zvolit na základě přibližné velikosti prohledávaného objektu. Neměly by být ani menší než hledaný objekt, čímž by část objektu mohla chybět při jeho rozpoznávání, ani moc velké, protože potom by detekce neměla v okně dost informací o objektu. Samozřejmě i přes vhodně nastavenou velikost mezer mezi úrovněmi může nastat případ, jako na obrázku 5.9, kdy se objekt rozpůlí mezi úrovněmi. Tento problém jde vyřešit částečným překryvem úrovní. Dále je vhodné omezit počet úrovní, ve kterých se bude prohledávat, protože obraz s Kinectu sice můžeme zvětšovat do nekonečna,

ale žádné nové informace z něho nedostaneme. Proto tyto data o příliš vzdálených objektech nejsou použitelná pro detekci.

S využitím tohoto přístupu rozdělení obrazu na úrovně k prohledávání na základě hloubkové mapy snížím počet oken, ve kterých se detekuje, z řádu tisíců oken na několik stovek oken (při posouvání okna po třetinách délky okna a projití čtyř úrovní zvětšení obrazu). Samozřejmě počet oken je ovlivněn natočením kamery, pozicí kamery a samotným rozložením objektu ve scéně, díky které se může změnit kolik se ve skutečnosti budu procházet oken. Tato metoda rozhodně vždycky ušetří detekci oken, ve kterých nemůže být objekt velikosti prohledávajícího okna. Zároveň tato metoda nejenom detekuje čtverec v obrazu, kde by se měl vyskytovat objekt, ale přitom přibližně určí kvádr v prostoru vůči kameře, kde se objekt vyskytuje.

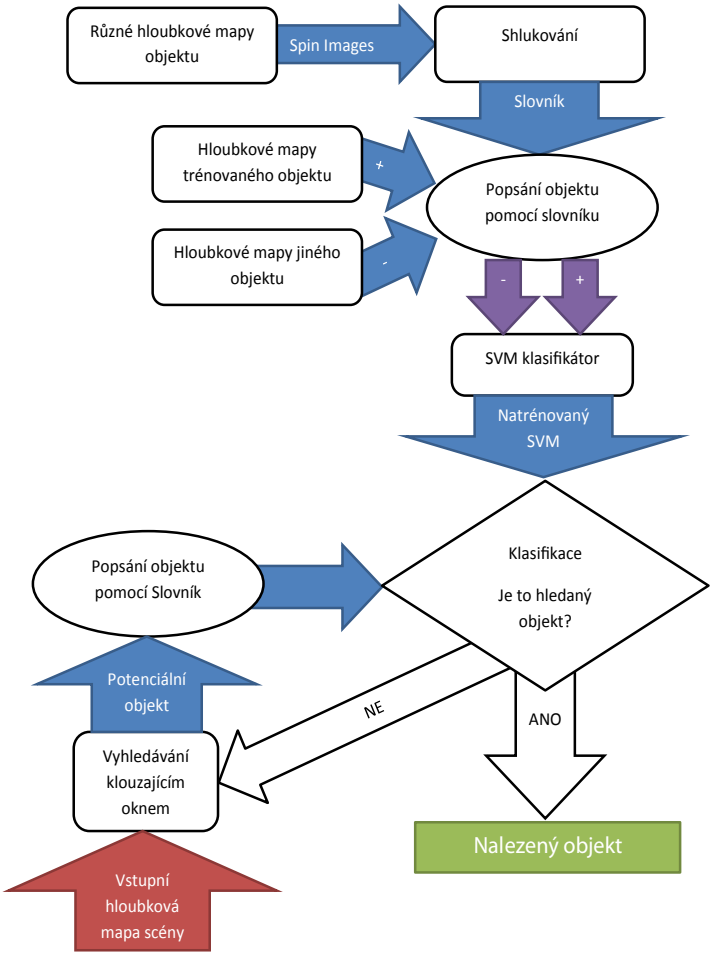


Obrázek 5.9: Při rozdělování obrazu může nastat rozpůlení objektu mezi dvěma úrovněmi.

Shrnutí celé metody

V souhrnu tedy můj návrh metody pro detekci bude takový, že nejdřív vytvoří datovou sadu zcela náhodných objektů. Na těchto datech rovnoměrně vyberu body, které popíši Spin Image. Na množinu všech Spin Image použiji shlukování, kterým získám jen několik SI, které budou reprezentovat slova ze slovníku. Nyní použiji datovou sadu pro učení SVM. Obrázky v nich popíši pomocí vytvořeného slovníku. Takto popsanou datovou sadu předložím SVM, která se díky ní naučí rozlišit jen ty objekty, které hledáme. Nyní přichází na řadu upravená metoda klouzajícího okna, ve které se nejdřív obraz scény rozdělí na úrovně podle hloubky obrazu, pak se zvětší podle hloubky a nakonec se jednotlivé úrovně rovnoměrně popíší deskriptory. Okno bude přecházet přes výsledky deskriptoru a pomocí slovníku se bude vytvářet histogram výskytů slov, který se předloží SVM pro rozhodnutí, zda je v tomto okně hledaný objekt. Lépe to bude pochopitelné z diagramu metody 5.10.

Díky použití metody klouzajícího okna by takto navržená metoda měla vyhledat všechny stejné objekty v obraze. Zároveň díky invarianci Spin Image vůči rotaci nebude potřeba řešit natočení vyhledávajícího okna. Bohužel Spin Image nelze použít přímo jako vstup do klasifikátoru, proto je spravován přes metodu balíků slov. Díky použití metody balíku slov by se zároveň mělo snížit množství porovnávání Spin Image pro identifikaci objektu (v porovnání se základními metodami, kdy se porovnává každý SI ze scény s každým SI z popisu objektu). Na druhou stranu samotný výpočet Spin Image je relativně komplikovaný a až implementace ukáže, jak rychle bude celá metoda fungovat.



Obrázek 5.10: Diagram navrhnuté metody detekce objektu.

5.3 Datová sada

Jako datovou sadu využiji sadu [15], která mi byla poskytnuta z Washingtonské university. Tato sada obsahuje 300 objektů (obr. 5.11) denní potřeby snímané pomocí kamery vyrobené firmou PrimeSense, která získává hloubkový obraz v rozlišení 640x480 stejně jako Kinect, z několika uhlů a dohromady tvoří 250 000 RGB-D obrázků. Tyto objekty lze běžně najít doma nebo v kancelářích, neboli v prostředích, kde by se mohl pohybovat robot využívající detekci v obraze. Všechny objekty byly zároveň i vyfoceny kamerou s vyšším rozlišením (1600x1200) pro lepší využití při detekci pomocí klasických metod, pracujících pouze s RGB obrazem. Zároveň tato sada obsahuje i soubor s maskou pro získání čistě objektu k lepšímu trénování. Současně univerzita vytvořila několik testovacích scén, na kterých lze otestovat vytvořený detektor.

V mém projektu budu využívat několik speciálně uzpůsobených datových sad pro jednotlivé části detekce v hloubkové mapě. První datovou sadou je skupina asi tisíce obrázků, která bude sloužit pro vytvoření slovníku. Tato sada bude univerzální pro všechny případně

Obrázek 5.11: Ukázka několika objektu z datové sady.[Převzato z [15]]

trénované objekty, takže v ní bude pokud možno co nejvíc absolutně různorodých obrázků objektů a pokud možno i obrázků, na kterých bude různé prostředí bez dalších objektů. Účelem je, aby vznikl materiál pro vytvořeni co nejuniverzálnějších slov pro slovník.

Další datová sada bude pro trénování SVM klasifikátoru. V této sadě budou pozitivní a negativní případy výskytu objektů. Pozitivní množina bude obsahovat hledané objekty ze všech možných úhlů, aby se SVM mohlo co nejlépe rozhodnout, a bude tvořit přibližně jednu třetinu celé trénovací datové sady. Negativní množina bude obsahovat pokud možno co nejvíc nejrůznějších obrázků objektů. Důležité je, aby byly co nejrůznější, protože musí pokrýt co nejvíc možností, které by mohly přijít na SVM a které nesmí byt klasifikovány jako hledaný objekt. Negativní množina bude zabírat zbývající část trénovací sady a dohromady bude tvořit množinu asi 1500 obrázků.

Poslední datová sada je testovací a bude pro vyhodnocení účinnosti rozpoznávání za pomocí balíku slov a SVM klasifikátoru. Tato sada bude mít přibližně 4500 obrázků a stejně jako testovací bude sada rozdělena na pozitivní a negativní množiny, ale narozdíl od ní bude pozitivní část zabírat asi jednu pětinu celku, protože v reálných podmínkách bude mít většinou na vstupu negativní hodnoty, a proto je potřeba přesně zjistit, jak na ně bude metoda reagovat. V této sadě bude pokud možno co nejvíc různorodých obrázků, než jaké byly v trénovací sadě, aby se zjistilo, jak univerzálně bude metoda fungovat.

Kapitola 6

Implementace

V této kapitole se zaměřím na popis samotné realizace své metody v programu pro demonstraci její funkčnosti, jak jsem jí popsal v návrhu v kapitole 5. Implementaci popíši v několika bodech:

- Implementační nástroje.
- Realizace detekční metody:
  - Spin Image
  - Vytvoření slovníku
  - Natrénování klasifikátoru
  - Detekce objektu

Nejdříve napíši něco o nástrojích, které v programu využívám. Pak přistoupím k popsání jednotlivých částí kódu, v kterých se zastavím u důležitých nebo problematických částí a popíši, jak jsem je řešil.

6.1 Implementační nástroje

Výše popsanou metodu realizuji v programovacím jazyku C/C++. Jako vývojové prostředí budu používat Microsoft Visual Studio 2010 kvůli své robustnosti a možnostem debuggingu. Pro zpracování obrazu využiji rozsáhlé knihovny OpenCV [2] verze 2.3.1, které obsahuje optimalizované nástroje pro co nejefektivnější práci s obrazem. Mezi metody, které budu z OpenCV využívat patří, datové typy, dále metody pro shlukování a také metody pro porovnávání vektorů. Zároveň budu psát kód Spin Image v takové formě, aby byl co nejlépe použitelný v knihovnách OpenCV, abych zároveň mohl lépe využít prostředky z OpenCV. Dalším podstatným nástrojem, který budu využívat je OpenMP [1], což je sada příkazů pro kompiler, která umožňuje jednoduchým způsobem počítat části programu na více vláknech současně.

6.2 Realizace detekční metody

Celá implementace je rozdělená do čtyř důležitých jednotlivých kódů. První je `SpinImage.h`, další je `DictData.h`, pak následuje `ClassifierSVM.h` a nakonec `ObjectDetector.h`. O jednotlivých částech kódu se dále podrobněji rozepíšu. Všechny tyto části se postupně využívají

v `main.h`. Zároveň všechny části vyžívají podpůrné funkce z `Additional.h`, která obsahuje funkce pro práci s 3D souřadnicemi, třídu pro zjednodušení práce se starším formátem pro načítání obrázků `IplImage` a nakonec třídu starající se o načítání parametrů pro práci programu. Pro načítání parametrů ze souboru jsem se rozhodl z důvodu velikého množství parametrů, které jsou potřeba pro správně nastavení jednotlivých částí programu. Tato třída `ProgramSetting` využívá vestavěnou knihovnu z OpenCV pro práci s XML formátováním.

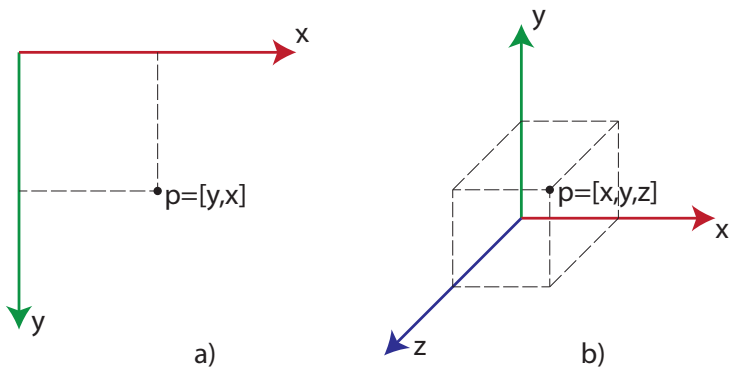
Spin Image

Spin Image je realizovaný třídou `SpinImage`, která taktéž obstarává všechnu práci s hloubkovou mapou. Proto tuto třídu můžu rozdělit na dvě fáze: na přípravu vstupních dat v konstruktorech a výpočet Spin Image v metodách.

- Konstruktor `SpinImage(IplImage* depthImage)`. V tomto konstruktoru probíhá zpracování vstupních dat. Typem vstupního parametru je `IplImage`, tento formát pro práci s obrazem sice patří do starší verze OpenCV, ale nezjistil jsem postup, jak je načíst přímo do formátu `Mat`. Datová sada s hloubkovou mapou, kterou používám, nebyla uložena v RAW formátu vycházejícího přímo z Kinectu, ale už ve formátu z části kalibrovaných dat, ve kterých jsou už uložené reálné vzdálenosti v milimetrech. To sice zjednodušilo práci, ale zároveň uložením takovýchto hodnot do formátu .png zkomplikovalo zpětné načtení. Zatím jediný účinný způsob jak se dostat k těmto datům se zdá být funkce `CvScalar`. Přímo při získávání hloubky jednotlivých bodů je převádím pomocí funkce `depthToWorld` do trojrozměrného prostoru a ukládám je do datového typu `Mat`, ve formátu který popíši dále. Zároveň tímto převodem do `Mat` se zbavím závislosti na starém, pomalém a komplikovanějším formátu uložení dat hloubkové mapy. Po této operaci mám hotové mračno bodů. Jak jsem psal v kapitole 5.1, v hloubkových mapách ze zařízení Kinectu se vyskytují artefakty prázdných míst. Tato místa vyplním pomocí funkcí `void medianFillIn(Mat& pointCloud)`, která je vyplní pomocí upraveného Median filtru. Po této operaci mám připravená data pro výpočet Spin Image.

- Konstruktor `SpinImage(Mat& pointsCloud)`. Přes tento konstruktor získávám už přímo připravená data, ze kterých se už dá přímo počítat Spin Image. Tímto ideálním vstupem je mračno bodů (points cloud), pro které využívám širokých možností datového typu `Mat` z OpenCV. Díky tomuto datovému typu můžu nastavit prvky matice `Mat` na typ `CV_32FC3`, následkem čehož se jednotlivé prvky chovají jako `Point3_<float>`, neboli tří rozměrný vektor popisující bod v prostoru.

- Metoda `Mat getSpinImage(Point coordinate)`. Tato metoda je druhá fáze výpočtu Spin Image a stará se o výpočet jednoho jediného Spin Image. Tato funkce má dvě části. První je výpočet normály bodu. Při realizaci této části jsem narazil na drobný problém, který mi nejednou zkomplikoval realizaci nechtěnými chybami. Z nějakého důvodu mají starší metody v OpenCV prohozené pořadí zadávání koordinát ([y,x]) v maticích a nové metody mají pořadí matematicky správně ([x,y]). Ještě k tomu OpenCV počítá s tím, že obraz má souřadnice [0,0] v levém horním rohu, což znamená, že oproti klasické matematice má opačnou hodnotu osy y. Což v případě, že jsem potřeboval ze systému 6.1(a) v ideálním případě pracovat v systému 6.1(b), způsobovalo značné komplikace a časté překlepy, které zapříčinily nesmyslné výsledky normály bodu a tudíž i špatné výsledky Spin Image. Naštěstí tyto problémy nastávaly jen u počítání normály.



Obrázek 6.1: (a) souřadný systém s jeho zápisem u OpenCV. (b) souřadný systém a jeho zápis, který používám.

Druhá část je už samotný výpočet spin image. Nejdříve se vypočítá výřez z obrazu, v kterém se bude počítat Spin Image. Potom přijde samotný výpočet, který je zapsaný následujícím kódem:

```
Vectors objPoint;
float alpha,beta,tmp;
for (int yy = sizeYmin; yy < sizeYmax; yy++){
 for (int xx = sizeXmin; xx <sizeXmax; xx++){
  objPoint = F3dVectors(pointsCloud.at<Point3_<float>>(yy,xx).x,
                        pointsCloud.at<Point3_<float>>(yy,xx).y,
                        pointsCloud.at<Point3_<float>>(yy,xx).z);
   //výpočet vzdálenosti bodu
   tmp = GetF3dVectorsLength( objPoint - point );
   //vypočet bety
   beta = normal * ( objPoint - point );
   //vypočet alfy
   alpha = sqrt(( tmp * tmp ) - ( beta * beta ));
   // zapiš do Spin Image pokud se tam vlezeš
   if(alphaMax > alpha && betaMax > fabs(beta)){
   spinImage.at<float>(0,(int)floor((betaMax - beta) / b)
                       * jMax + (int) floor( alpha / b)) +=1;
  }
 }
}
return spinImage;
```

Kvůli optimalizaci výkonu jsem provedl pár drobných změn oproti výše popsanému způsobu výpočtu Spin Image. Zaprvé jsem se pokusil maximálně omezit množství numerických operací s plovoucí desetinou čárkou, proto si nejdřív vypočítám `tmp` a `beta` které by se stejně pro výpočet `alpha` musely znovu počítat víckrát. Možná to je jen malá úspora času na jeden Spin Image, ale když v průměru je potřeba 15 000 Spin Image na jeden obrázek z Kinect a každý spin Image potřebuje vypočítat `alpha` a `beta` minimálně 1600 krát (při minimální velikosti okna pro počítání 40 na

40 pixelů) dělá to 2,5 milionu výpočtů, kdy už se počítá každé násobení v takto velikém cyklu. Dalším zrychlením je ignorování přesnější aproximace zápisu do Spin Image, protože distribuce chyby by byly další numerické operace, které by zpomalovaly výpočet velkého množství Spin Image. Bohužel je tato optimalizace na úkor přesnosti Spin Image. Výsledek generování Spin Image ukládám do jednorozměrné matice, ve stejném tvaru v jakém se ukládají výstupy z ostatních deskriptorů používaných v OpenCV.

- Metoda `Mat getAllSpinImages()`. Tato funkce využívá metodu `Mat getSpinImage(Point coordinate)` a pomoci ní generuje Spin Image buď pro všechny body načteného obrázku nebo body rovnoměrně rozprostřené po obrazu se zadanou mezerou anebo pro určité body určené pomocí OpenCV vektoru typu `KeyPoint`. Výsledek této funkce je matice čísel, kde každý řádek reprezentuje jeden Spin Image. V této funkci využívám paralelního zpracování pomocí OpenMP. Díky tomu pokud má procesor víc jader, počítám každý Spin Image v jednom vlákně. Paralelizování této části skoro zdvojnásobilo rychlost výpočtu na mém dvou jádrovém procesoru. Zkoušel jsem použit OpenMP přímo ve funkci `Mat getSpinImage(Point coordinate)` při cyklu pro počítání jednotlivých vztahů mezi body, ale na tomto místě to nemělo významný vliv na zrychlení výpočtu. Přibližně jen o 10%.

Vytvoření slovníku

První využití Spin image je potřeba v třídě `DictData`, v jejíž metodách se připravují slovníky pro BoW.

- Metoda `Mat computeSIDictionary(vector<ObdImage*>& imagePath, int clusteSize)`. Metoda provádí generování slovníku. Princip pro generování slovníku je veskrze jednoduchý. Načtený obrázek popíši rovnoměrně rozprostřenými Spin Image. Výsledné Spin Image předložím OpenCV třídě `BOWKMeansTrainer` pomocí metody `add`, která si je ukládá pro pozdější provedení K-means shlukování. Takto zpracuji všechny obrázky z datové sady pro vytvoření slovníku. Jakmile mám hotové všechny Spin Image, spustím metodu `Mat cluster()`, která provede shlukování a navrátí hotový slovník. V tomhle bodě ale nastaly problémy. Hlavní a základní problém, který spíš předpokládám, protože do OpenCV kódu nevidím, je, že OpenCV clustering potřebuje mít všechna data na jednom místě a to dokonce na jednom souvislém místě. Díky tomu mi celkem náhodně padal program v případě, že jsem potřeboval více než 0,7GB paměti, což jsem se Spin Image potřeboval celkem pravidelně. Moje datová sada má kolem 900 obrázků, z nich získám asi 250 000 Spin Image (mohl jsem získat i víc, ale to už mi program začal padat) a přitom jeden Spin Image je přibližně 500 32bitových float hodnot, což dohromady dělá asi 500MB paměti jen čistě pro Spin Image. Tohle je maximální hodnota, kterou se mi podařilo bez problémů alokovat. Přitom velikost výstupního vektoru je právě velká nevýhoda spin Image, protože zabírá příliš mnoho paměti a tím i zpomaluje další zpracování. Což v praxi znamenalo, že vytvořit slovník s 500 slovy trvalo přibližně 8 hodin výpočetního času.

Natrénování klasifikátoru

Po připravení slovníku můžu přistoupit k natrénování SVM klasifikátoru pro rozpoznávání objektů z obrazu. Tuto část metody provádím v třídě `ClassifierSVM`. Jako SVM klasifi-

kátor používám SVM, které je implementováno přímo v knihovnách OpenCV. Ale než se začne učit rozpoznávat objekty, je potřeba připravit data.

- Metoda `void trainSVMClassifierDepth(CvSVM& svmDepth)`. Slouží k natrénování klasifikátoru pro Bag of Words. Původní plán byl využít metod OpenCV pro Bag of Words, které OpenCV obsahuje. Bohužel jsem zjistil, že nejsou tak univerzální, jak jsem si myslel, a nepodporují jiné deskriptory, než které má v sobě implementované OpenCV. Tudíž jsem si musel napsat sám tvoření vět pro SVM z balíků slov. Naštěstí OpenCV obsahuje metody pro porovnávání seznamů n-dimensionálních vektorů, jenž navrátí vektor, který s kterým vektory si jsou nejbližší. K tomuto přerovnávání jsem použil OpenCV třídu `FlannBasedMatcher` a pro samotné porovnávání se používá metoda `match`. Výsledek operace se uloží do vektoru `matches`, ve kterém jsou uložené datové struktury obsahující kromě vzdálenosti, jak jsou si podobné deskriptory, hlavně indexy na slova ze slovníku, s kterými jsou jednotlivé deskriptory nejpodobnější. Díky výsledku této operace je už pak jednoduché vytvořit histogram výskytu slov ze slovníku v obrázku, který se předloží k učení SVM. Jediná nevýhoda této metody je, že je relativně pomalá (asi 0,25s na obrázek), ale v této části programu rychlost není tak důležitá, protože se trénování SVM provádí takzvaně off-line, neboli ještě předtím než je potřeba detekce. V dalších částí už budu používat rychlejší metodu porovnávání.

  Nyní, když mám vytvořené pro všechny obrázky z trénovací sady jejich histogramy neboli věty ze slovníku, můžu je předložit SVM společně s označením, které histogramy jsou hledané objekty a které nejsou. Jelikož správné nastavení SVM pro co nejlepší klasifikaci je velice složitá záležitost, která by vystačila na samostatnou práci, pro správné nastavení a naučení klasifikátoru použiji metodu `train_auto`. Tato metoda pro správné určení parametrů používá cross-validaci, která rozdělí trénovací data na dvě části a potom se na jedné učí a na druhé kontroluje výsledky, díky čemuž snižuje šanci na přetrénování klasifikátoru.

Detekce objektu

Nyní předchozí části kódu zužitkuji v kódu pro detekci v obraze a syntetické zhodnocení rozpoznávání objektu. Tato část je napsaná v třídě `ObjectDetector`. Přitom ji lze rozdělit na dvě hlavní části: na metodu pro syntetické testování schopností rozpoznat objekt `testDetectSI/SIFT` a metodu pro vyhledání a rozpoznání objektu v komplexním obraze `detectInImgSI`.

- Metoda `void testDetectSI(CvSVM& svm,ofstream& fl, vector<ObdImage*>& imagePath)`. Samotné syntetické testování je předložení testovací datové sady natrénovanému SVM klasifikátoru. Přitom každý obrázek z datové sady se musí převést na histogram výskytu deskriptoru stejně jako při trénování SVM, které jsem popsal výše. Ovšem narozdíl od trénování používám jinou metodu pro porovnávaní vektorů. Místo metody `FlannBasedMatcher` používám třídu, která je optimalizovaná pro porovnávání stále s jedním seznamem vektorů, `flann::Index`, taktéž z knihoven OpenCV. V mém případě v konstruktor této třídy používám pro nastavení metodu `cv::flann::KDTreeIndexParams()`, která připraví paralelní prohledávání pomocí k-dimensionálních stromů. Díky této metodě se rapidně zvýšila rychlost porovnávání a to o jeden řád s 0,25s na 0,02s na obrázek. Na druhou stranu by tato metoda měla být méně přesná, ale to jsem zatím v mé metodě nezpozoroval. Toto je taky důvod, proč jsem ji použil až v poslední fázi

programu, protože u trénování mi nejde o rychlost, ale o přesnost. U syntetického testování mi sice nejde o rychlost, ale pro co nejpodobnější výsledky s reálným požitím, používám tuto metodu i v této části. Z metody pro testování mám dva druhy výstupů. Prvním výstupem je soubor obsahující vzdálenosti od nadroviny SVM pro vytvoření ROC křivky a druhým je soubor obsahující dobu výpočtu a počty správného přijetí, špatného přijetí, správného odmítnutí a špatného odmítnutí. K výsledkům metody se budu vyjadřovat v další kapitole.

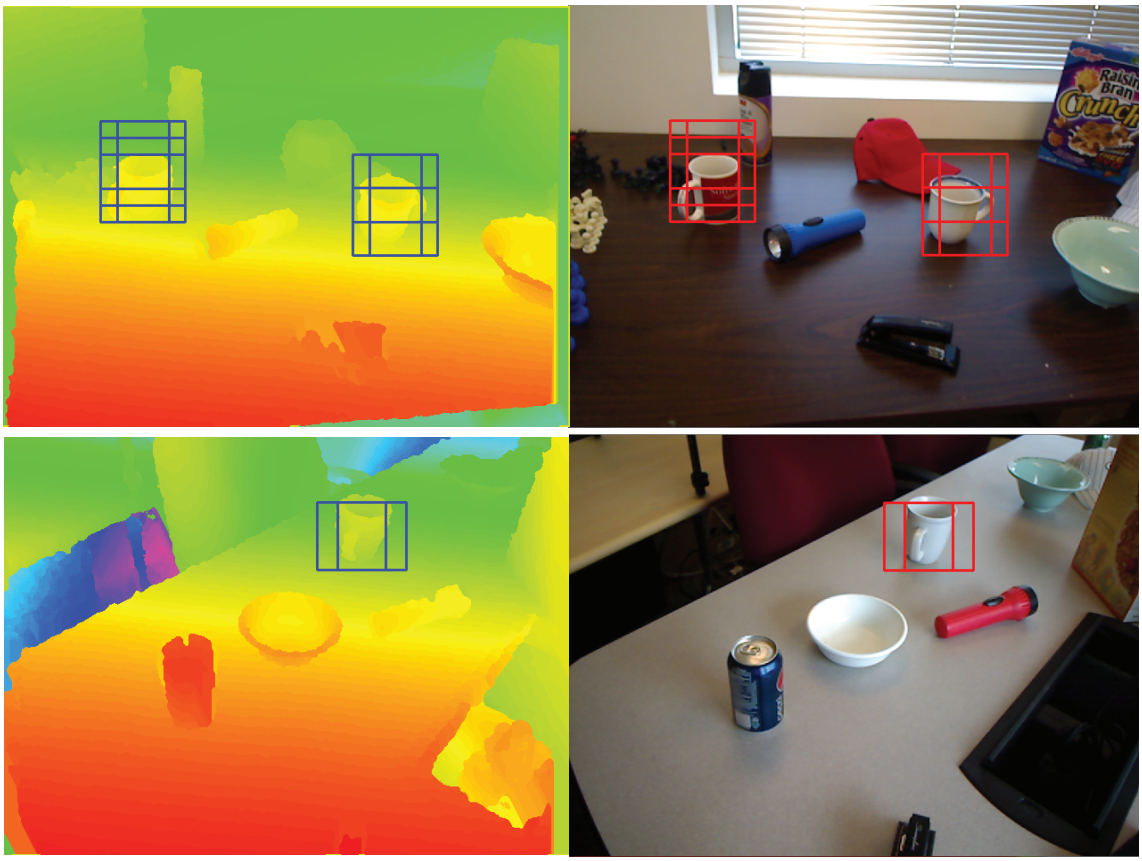
- Metoda `void detectInImgSI(CvSVM& svm, IplImage*depthImg, IplImage*rgbImg)`. Tato metoda v praxi ukazuje, jak navrhnutá metoda funguje v reálných podmínkách. Tunkce má tři části, v kterých se zpracovává hloubkový obraz se scénou. První částí je příprava dat pro zpracování. V této části rozděluji hloubkovou mapu na jednotlivé úrovně podle hloubky a zároveň zvětšuji rozlišení hloubkové mapy podle hloubek jednotlivých úrovní. Toto zvětšování jsem měl původně v plánu provádět přímo na hloubkové mapě, kterou dostávám na vstup, ale při použití funkce `cvResize` vznikaly ve výstupní hloubkové mapě podivné artefakty, jak jde vidět na obrázku 6.2. Předpokládám, že tyto artefakty vznikly netypickým uložením částečně kalibrované hloubkové mapy do formátu .PNG a potom jejím opětovným načtením do typu `IplImage`(o důvodu proč načítám do tohoto formátu jsem psal výše). Tento problém jsem vyřešil tak, že jsem nejdřív načetl hloubkovou mapu třídou `SpinImage`, kterou jsem popsal výše. Konstruktor této třídy vytvoří point cloud, který je reprezentovaný OpenCV maticí `Mat`, která už jde bezpečně zvětšit pomocí funkce `resize`. Tento zvětšený point cloud mohu potom vložit jako vstupní parametr konstruktoru dalšího `SpinImage`. Pro každý tento point cloud v oblasti jeho úrovně hloubky obrazu nageneruji rovnoměrně rozprostřené `KeyPoint`. `SpinImage` a `KeyPoint` jednotlivých úrovní prohledávání si uložím do vektoru pro použití v další fázi. Druhá fáze je generování Spin Image. Podle

Obrázek 6.2: Artefakty vzniklé změnou rozlišení vstupní hloubkové mapy.

keypointu, které jsem si nageneroval v předchozí fázi, si vyrobím pro jednotlivé úrovně Spin Image. Tyto výsledné deskriptory si taktéž uložím do vektoru pro zpracování v poslední fázi.

Závěrečná fáze je hledání objektů v obraze . Tato fáze má několik úrovní zabezpečení, aby se SVM rozpoznávání předkládalo co nejmenší množství oken. První úrovní je

samozřejmě omezení prohledávání pouze na oblasti dané hloubkou obrazu. Prohledávání úrovní se provádí v cyklu a jelikož Spin Image, ve kterém využívám všechna jádra procesoru, jsem prováděl v předchozí fázi, můžu si dovolit zde opět použít OpenMP pro paralelizaci výpočtu. Tudíž každá úroveň prohledávání se provádí v různých vláknech. Každý obraz úrovně se prochází zleva do prava od vrchu dolů oknem, které se posunuje po určitých krocích. Pokud přibližný střed tohoto okna spadá do oblasti hloubky určené pro tuto úroveň, pustí se toto okno k dalšímu zpracování. Pro takovéto okno se vyhledají všechny Spin Image podle jejich keypointu, které do tohoto okna spadají. Po této operaci se ještě ale nepřistupuje k rozpoznání. Nejdřív toto okno musí splnit podmínku, aby obsahovalo dostatečné množství Spin Image. Jakmile i tuto podmínku splní, může přistoupit k identifikování. Toto identifikování již probíhá stejně, jak jsem napsal v předchozích odstavcích u testování detekce. Pokud je objekt nalezen, uloží se jeho informace do struktury `DetectObj`, která obsahuje souřadnice rohů okna přepočítaných na původní velikost obrázku, velikost okna a úroveň, ve které byl nalezen. Takto uložená okna jsou potom vložena do vektoru, který je zároveň výsledkem vyhledávání v hloubkové mapě. Tato okna jsou potom zakreslena do obrázku scény a výsledek vypadá jako na obrázku 6.3.



Obrázek 6.3: Vystup z detektoru se zvýrazněnými úrovněmi prohledávání.

Kapitola 7

Testování a popis vlastností metody

V ideálním případě bych měl hodnotit výstup na konci celé metody, jak tato metoda funguje jako celek. Bohužel upravené metodě klouzajícího okna lze jen obtížně exaktně změřit její účinnost, protože rozpoznávání objektů v komplexní scéně, ve které by se vyhledávalo, činní z tohoto problému v celku subjektivní hodnocení. Proto tuto kapitolu rozdělím na dvě části. Na popsání naměřených hodnot rozpoznávání objektů a popsání vlastnosíi metody s detekci v obraze.

Protože budu měřit rozpoznávání čistě na natrénovaném SVM klasifikátoru bez vyhledávání, tak budu předkládat klasifikátoru čistě datovou sadu, kterou jsem popsal výše. Testovat metodu budu dvěma způsoby a oba způsoby s využitím jak Spin Image tak i SIFT pro doplňující porovnání účinnosti stejného principu, ale na datech z RGB obrazu.

Obrázek 7.1: Objekty, které budu vyhledávat.

V prvním způsobu testování budu hledat jakýkoliv objekt jednoho druhu objektu, a to na různých druzích hrnků, které jsou vidět na obrázku 7.1(a). Aby bylo vidět, že metoda opravdu dokáže univerzálně rozpoznat hrnky, v trénovací sadě úmyslně vynechám dva druhy hrnků, které se objeví až v testovací sadě. Toto testování by mělo být relativně jednodušší pro metodu se Spin Image, protože samotná metoda balíků slov je spíše určená pro identifikování objektů z obrazu, které by mohly být podobné a ne pro přesné identifikování jednoho určitého objektu. Druhý způsob testování bude založený na rozpoznání určitého objektu. Tímto objektem bude krabice od potravin, která je vidět na obrázku 7.1(b). Tento druh testu bude pro mojí metodu rozpoznávání náročnější, protože pro Spin Image tento

objekt není příliš tvarově výrazný, a taky v testovací sadě budou podobné krabice jen trošku jinak veliké. V tomto testu by měl mít výhodu právě SIFT, který se specializuje na detekci určitých objektů, ale který je na druhou ochuzený o detektor významných bodů, který mu výrazně pomáhá.

7.1 Naměřené hodnoty rozpoznávání objektů

Jak jsem psal výše, testoval jsem rozpoznávání libovolných hrnků a jedné specifické krabice. Nejdřív se vyjádřím k naměřeným hodnotám prvního testování. Výsledky, které ukazují grafy ROC křivek 7.4, 7.5, 7.6, jsou opravdu velmi dobré i přes přidání nových hrnků do trénovací sady. Musel jsem dokonce udělat jen výřez levého horního rohu ROC křivky, aby bylo vidět lépe detaily křivek. Tento výborný výsledek jde samozřejmě zdůvodnit syntetickou testovací a trénovací datovou sadou, kterou jsem vyrobil a použil. Tyto trénovací a testovací datové sady sice neobsahují úplně stejně snímané objekty, ale rozdíly mezi sadami jsou jen v maličko jiném úhlu snímání objektu. Na druhou stranu testovací sada obsahuje mnoho nových negativních objektů a také dva druhy hrnků, které v trénovací sadě nebyly a mohly by tedy rozpoznávání zhoršit, což se ale nestalo. Zároveň hrnek má dost specifický a výrazný tvar, tudíž se dá lépe rozpoznat od ostatních objektů.

Zpátky ale k vlastnostem Spin Image. Při měření jsem měnil vzdálenost mezi deskriptory po 3, 5, 7, 9, 11, 13, 15 a 17 pixelech. Dále jsem také měnil konstantu b, která zvětšuje a zmenšuje rozlišení Spin Image, na hodnotu 2,7; 3,5 a 4,3. Přitom jsem nechal u všech měření stejně nastavenou hodnotu α_{max} a β_{max} na 40 voxelech (α_{max} a β_{max} jsou prostorové rozměry, jak jsem psal v 3.4). Jak lze vidět na grafech **??** a), b), c) změnou vzdálenosti mezi deskriptory se snižuje i přesnost rozpoznání, na druhou stranu tato změna je pozvolná a relativně malá. Původně jsem měl obavy, že při mezeře 11 pixelů bude mít klasifikátor příliš málo informací pro správné rozhodnutí. Přitom tyto výsledky dokazují vlastnost Spin Image, že Spin Image jsou si v okolí podobné, tudíž se nemusí vyskytovat blízko u sebe. Na druhou stranu se zvětšujícími mezerami stoupá náchylnost k nedostatku Spin Imagu, které byly vypočítány na objektu. Toto může vysvětlovat, proč se některé křivky u větších mezer skoro překrývají.

Zajímavým poznatkem z grafů je vliv konstanty b na metodu. Podle předpokladu i měření tato konstanta ovlivňuje, jak prudce ztrácí metoda přesnost s rostoucí mezerou mezi deskriptory. Co je překvapující, že tento průběh není úplně lineární. Z grafů vyplývá, že konstanta nastavená na 3,5 je trošičku lepší než 4,3 i 2,7. U hodnoty 4,3 se to předpokládalo, protože Spin Image s rostoucí konstantou b ztrácí na přesnosti. Zato u hodnoty 2,7 je zhoršení kvality překvapující.

Nakonec, když porovnám Spin Image a SIFT v tomto druhu testu, tak SIFT, jak jde vidět na grafu 7.3 a), je prakticky nepoužitelný a to i přes to jakou datovou sadou byl trénován a testován. Tento výsledek je ale pochopitelný, protože SIFT se zaměřuje na objekty s texturou a tudíž dokáže spíše rozpoznat jen jeden určitý objekt.

Přistupme k druhému druhu testu. V tomto testu měla metoda rozpoznat jednu určitou krabici. Jak jde vidět na grafech 7.7, 7.8, 7.9, metoda si opět vedla výborně a to dokonce lépe než SIFT 7.3 b). Přitom vše v tomto testu napomáhalo SIFT pro lepší detekci. Jednalo se o jeden určitý objekt a tento objekt má výraznou a jedinečnou texturu. Přitom Spin Image měl velkou nevýhodu. Jednalo se o jednoduchý kvádr bez něčeho specifického a měl se rozlišit od krabic, které mají jen rozdílnou velikost, a dalších objektů. Samozřejmě metoda detekuje trošku hůře než při předchozím testu, ale to je za těchto podmínek pochopitelné.

FA:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11	FR:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11
Gaps 3	1,56	6,94	11,62	17,61	25,31	Gaps 3	4,86	0,13	0,00	0,00	0,00
Gaps 5	0,00	1,90	5,49	10,06	14,02	Gaps 5	95,52	3,71	0,13	0,00	0,00
Gaps 7	0,00	0,22	1,64	5,94	9,23	Gaps 7	100,00	56,52	5,37	0,13	0,00
Gaps 9	0,00	0,00	0,33	2,15	5,32	Gaps 9	100,00	97,19	56,14	4,09	0,26
Gaps 11	0,00	0,00	0,00	0,53	1,87	Gaps 11	100,00	100,00	98,47	42,20	6,01

a)

FA:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11	FR:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11
Gaps 3	0,72	4,04	10,67	16,22	17,56	Gaps 3	0,51	0,00	0,00	0,00	0,00
Gaps 5	0,00	1,28	5,80	9,20	11,43	Gaps 5	100,00	3,45	0,00	0,00	0,00
Gaps 7	0,00	0,20	1,98	5,80	8,11	Gaps 7	100,00	43,73	0,77	0,26	0,00
Gaps 9	0,00	0,03	0,50	2,15	4,63	Gaps 9	100,00	89,26	40,66	4,73	0,77
Gaps 11	0,00	0,00	0,03	0,67	2,01	Gaps 11	100,00	98,21	84,27	41,56	2,81

b)

FA:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11	FR:	SVM 3	SVM 5	SVM 7	SVM 9	SVM 11
Gaps 3	1,20	7,58	8,89	17,06	18,12	Gaps 3	2,81	0,00	0,00	0,00	0,00
Gaps 5	0,00	1,37	4,93	8,72	11,87	Gaps 5	100,00	4,48	0,00	0,00	0,00
Gaps 7	0,00	0,00	1,95	5,52	8,42	Gaps 7	100,00	86,70	2,17	0,00	0,00
Gaps 9	0,00	0,00	0,45	2,15	5,07	Gaps 9	100,00	100,00	41,56	2,81	0,13
Gaps 11	0,00	0,00	0,00	0,59	2,15	Gaps 11	100,00	100,00	87,21	38,11	2,56

c)

Obrázek 7.2: Tabulky ukazující vliv různých velikostí mezer na rozpoznávání krabice pro konstanty b a) 2,7; b) 3,5; c) 4,3. FA je špatné přijetí a FR je špatné odmítnutí v procentech. Gaps značí velikost mezery mezi deskriptory a SVM je klasifikátor pro jaké mezery byl natrénovaný.

Přitom u tohoto testu zůstaly některé vlastnosti, které jsem výše popsal u předchozího testování, prakticky totožné. Opět platí, že nejlepší výsledky dosahuje metoda s konstantou b nastavenou na 3,5. Další zajímavá vlastnost je viditelná v tabulce 7.2. Tato tabulka ukazuje, jak se metoda chová v případě, že se sníží nebo zvýší množství deskriptorů, což může nastat při metodě klouzajícího okna, když se okno netrefí rovnou doprostřed objektu. Tabulky ukazují, že preventivním snížením velikosti mezer mezi deskriptory o jeden pixel získám prakticky jistotu, že bych neodmítnul správný objekt a přitom zvýším pravděpodobnost, že bych špatně přijal objekt, jen o jedno nebo dvě procenta. Tato vlastnost ovšem klesá s rostoucí mezerou při prénování klasifikátoru.

7.2 Vlastnosti metody s detekci v obraze

Jak jsem již psal v úvodu, exaktní přesné změření vylepšené metody klouzajícího okna je relativně komplikované, ale cvičným zkoušením v mém programu mohu zhodnotit alespoň jeho vlastnosti. Rozhodně můžu říct, že tato metoda zrychlila klasickou metodu klouzajícího okna, ale nedá se jednoduše říct, o kolik ji zrychlila. U klasické metody je jednoduché spočítat, kolik oken musel program zpracovat, ale u mé metody už to nejde, protože počet oken se už odvíjí od druhu scény. Pokud je celá scéna jen ve vzdálené úrovni hloubky, musí metoda projít velké množství oken. Pokud je scéna v blízké úrovni hloubky stačí projít jen několik oken. Každopádně moje implementace dokáže scénu, která má přibližně pod úhlem 45° rovinu s objekty a prohledává ve 4 úrovních hloubky po 25cm, zanalyzovat za přibližně 3-5s a to na počítači s dvou jádrovým procesorem Core2Duo P8600 2,4GHz.

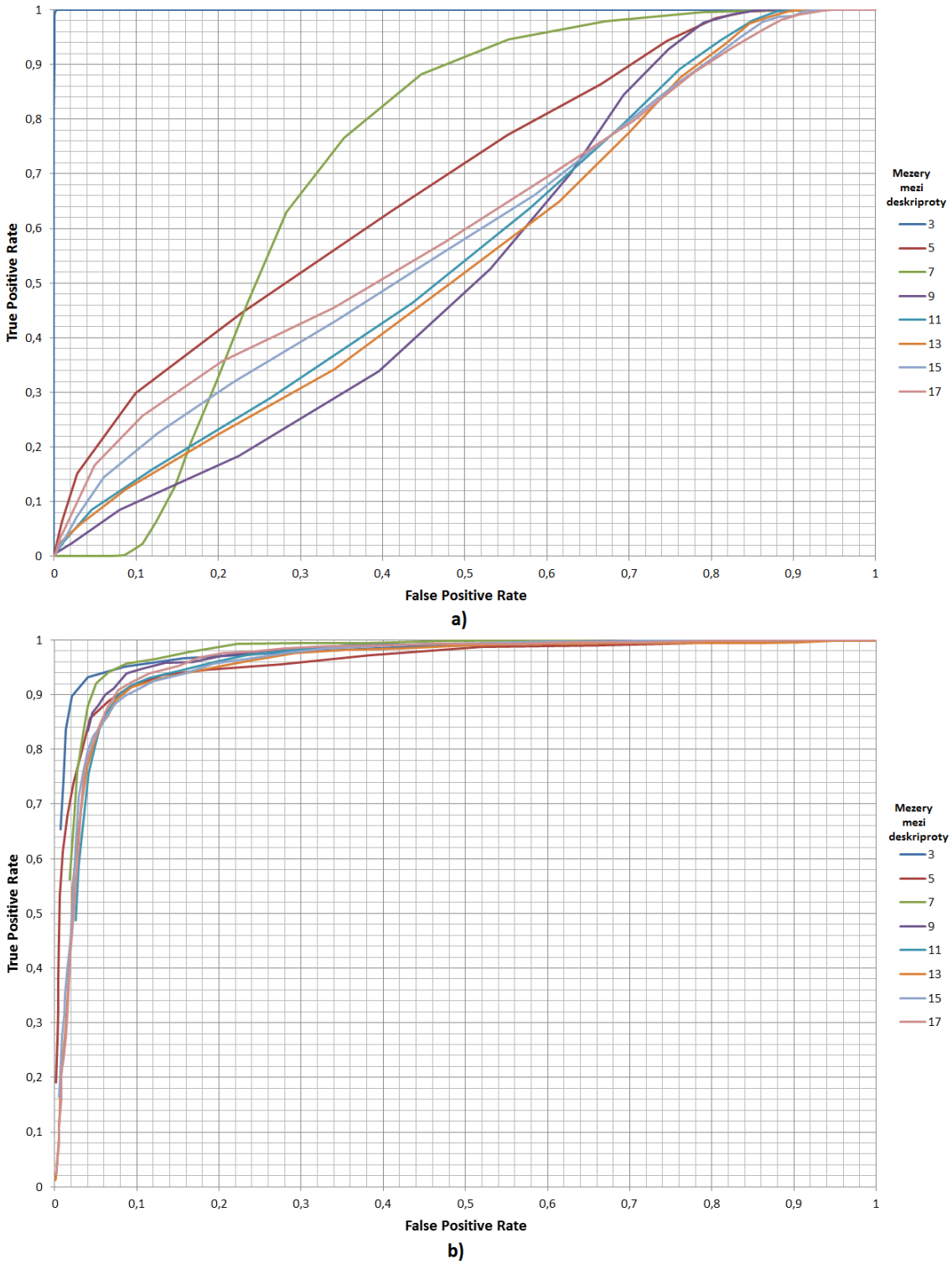
Na výkonnějším 4 jádrovém 3,2GHz AMD procesoru toto trvalo pod 2s. Přitom parametry detekce a prohledávaní byly nastaveny tak, aby nalezla co nejvíc objeků a co nejrychleji.

7.3 Zhodnoceni naměřených hodnot

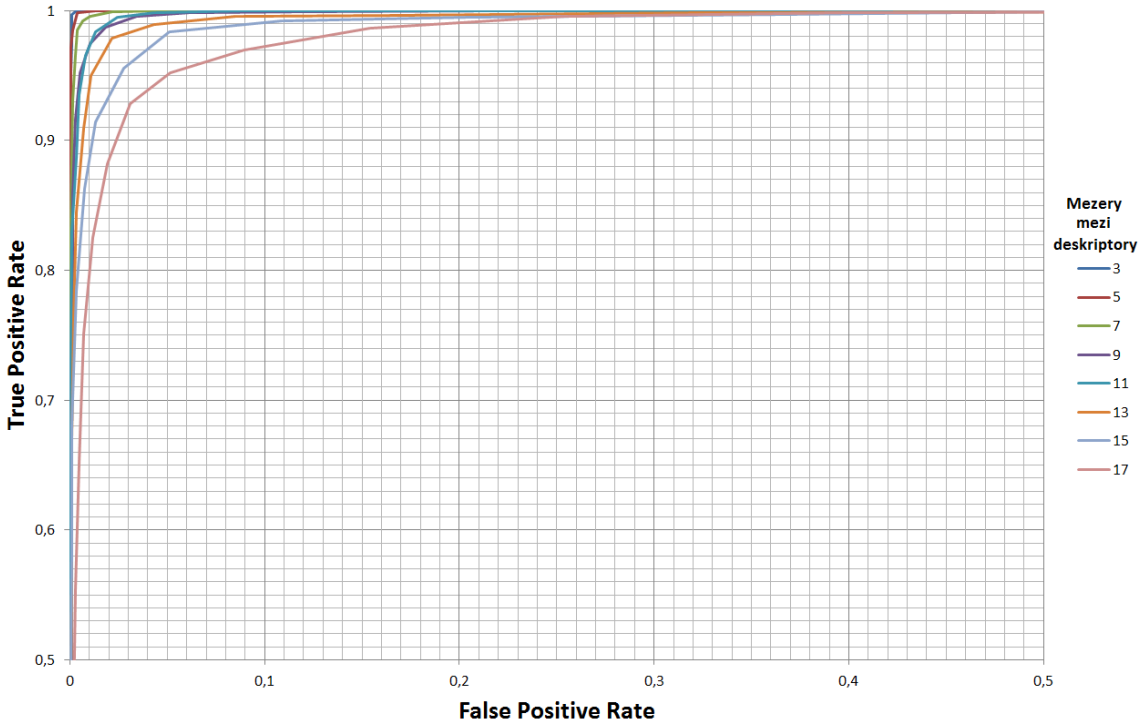
Výsledky ukázaly, že metoda Spin Image je robustní a přesný způsob popisu hloubkové informace, který díky absenci vnějších vlivů na hloubkovou mapu (na rozdíl od RGB obrazu) vykazuje lepších výsledků než klasické přístupy. Zároveň oproti SIFT dokáže popsat větší okolí a tím snižuje množství deskriptorů, které je potřeba pro popis objektů. V kombinaci s Bag of Word vykazuje výborných výsledků rozpoznávání objektů při syntetickém testování a velmi dobrých při rozpoznávání v reálné scéně. Bohužel nevýhodou Spin Image je jeho velká výpočetní náročnost. Tuto vlastnost kompenzuje pouze (výše zmíněný) větší prostor, který Spin Image popisuje. Pomalost algoritmu je způsobena obrovským množstvím výpočtů s plovoucí desetinou čárkou. Proto by bylo výhodné počítat tyto části pomocí procesorů na GPU.

Pokud bych měl zhodnotit metodu rozpoznávání při reálném použití, projevily se u ní vlastnosti, které u syntetického testování nešlo zjistit. Bag of Word maže kontext mezi deskriptory a jediný, který zůstává, je uvnitř Spin Image. Proto se někdy stává, že v reálné scéně, ve které jsou různé objekty u sebe, identifikuje špatně objekt. Tuto komplikaci nelze již tak jednoduše odstranit a bylo by potřeba použít jiný přístup než Bag of Word. Další problémy při rozpoznávání způsobuje moje metoda klouzajícího okna. Zaprvé rozdělování podle hloubky způsobuje dost často obrys kolem objektu, tudíž při rozpoznávání chybí metodě část informací, které jsou sice při detekci zbytečné, ale u trénování byly zahrnuty. Druhý problém je velikost kroku vyhledávacího okna. Pokud je krok příliš malý, hledání trvá dlouho. Pokud je moc velký, okno nemusí obsahovat celý objekt a tudíž je horší ho rozpoznat. Dalším problémem je pro metodu identifikování hodně vzdálených objektů, protože vzdálenější objekty mají už nepřesnou hloubkovou mapu, ale toto je vlastnost Kinectu, se kterou se nedá nic moc dělat. V neposlední řadě je nevýhodou použití rovnoměrného rozprostření bodů, které může způsobovat nedostatek deskriptorů na objektu a tím možnost špatného rozhodnuti. Toto by bylo dobré řešit technikou, která by dokázala v hloubkové mapě vyhledat význačné body.

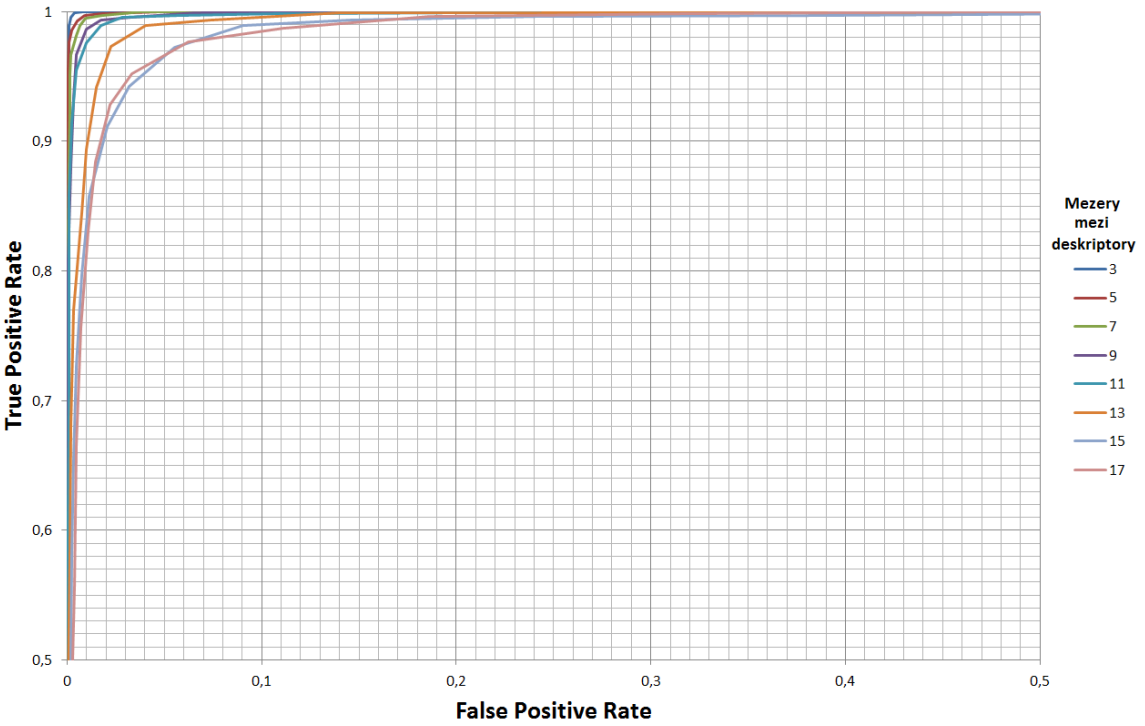
Ale i přes všechny tyto problémy je metoda schopna s patřičným nastavením parametrů vyhledat a rozpoznat hledané objekty ve scéně v řádu několika sekund. Přitom testování ukázalo, že metoda je schopná v některých případech rozpoznat i objekty, které jsou částečně zakrytý jiným objektem.



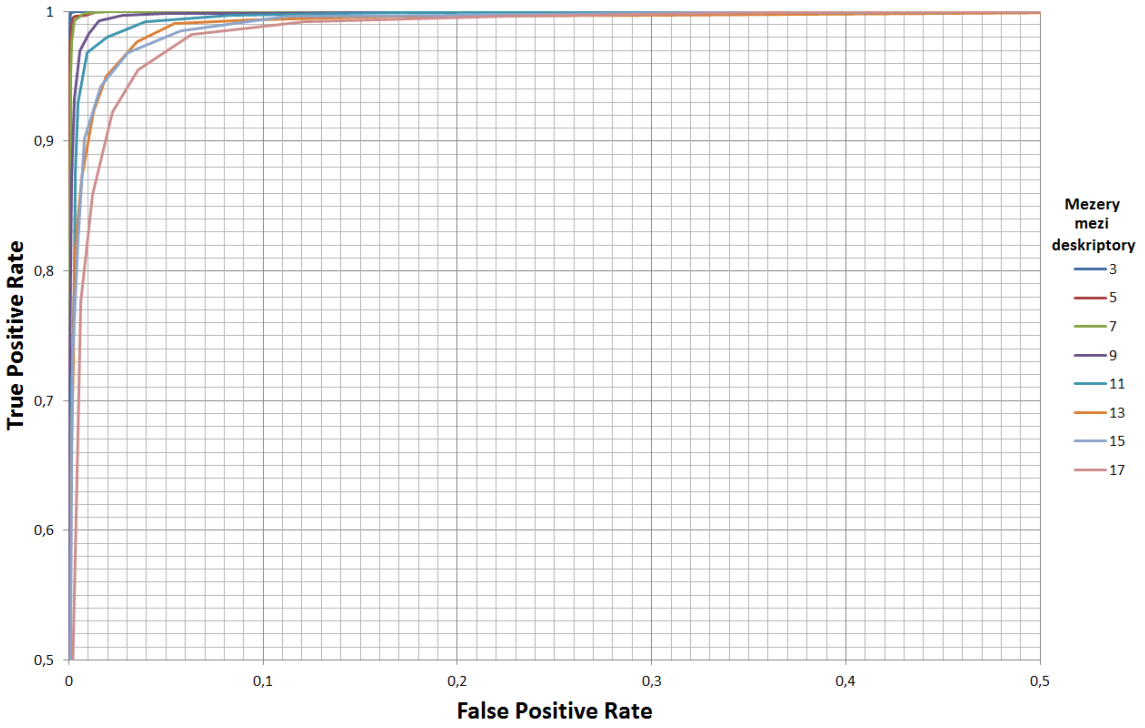
Obrázek 7.3: ROC křivky pro natrénované SVM s SIFT deskriptorem. a) rozpoznávání hrnků. b) rozpoznávání krabice



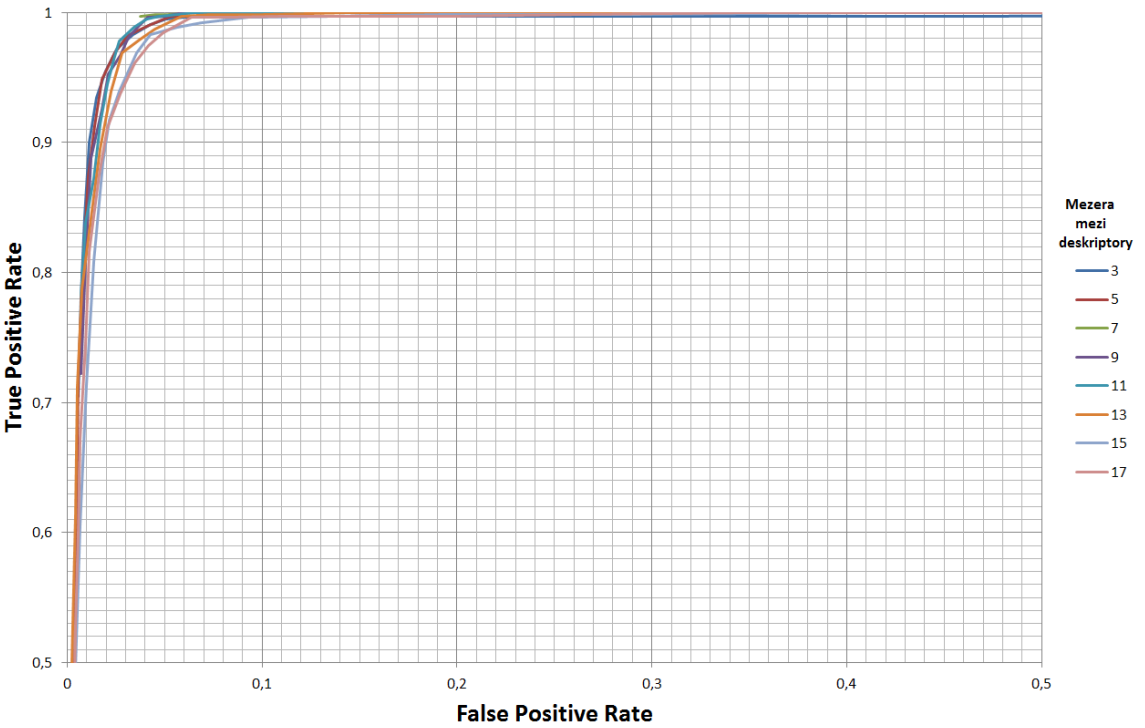
Obrázek 7.4: ROC křivky pro natrénované SVM na rozpoznání hrnků s různými mezerami a nastavenou konstantou b na 2,7



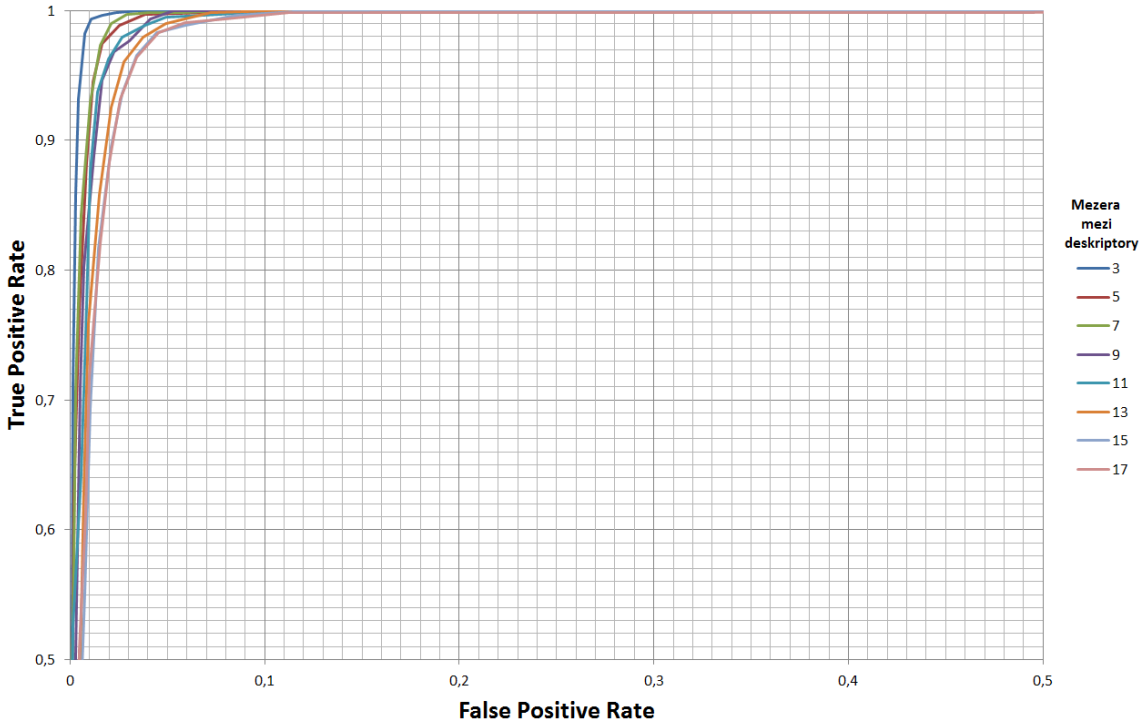
Obrázek 7.5: ROC křivky pro natrénované SVM na rozpoznání hrnků s různými mezerami a nastavenou konstantou b na 3,5



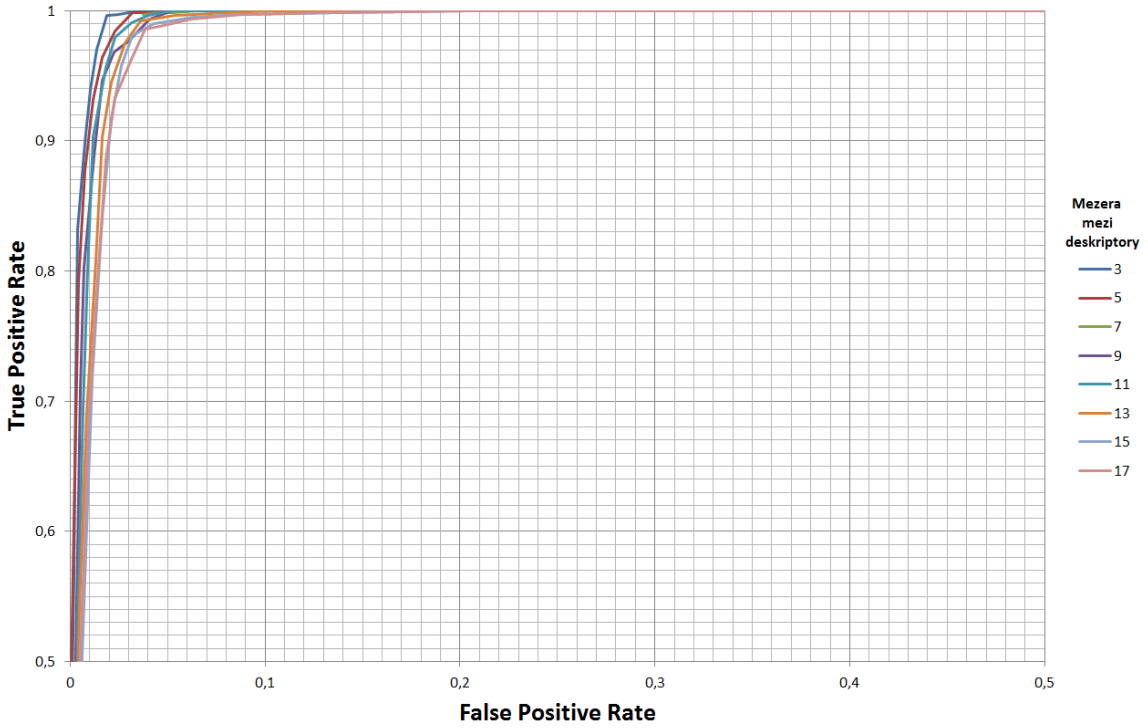
Obrázek 7.6: ROC křivky pro natrénované SVM na rozpoznání hrnků s různými mezerami a nastavenou konstantou b na 4,3



Obrázek 7.7: ROC křivky pro natrénované SVM na rozpoznání krabice s různými mezerami a nastavenou konstantou b na 2,7



Obrázek 7.8: ROC křivky pro natrénované SVM na rozpoznání krabice s různými mezerami a nastavenou konstantou b na 3,5



Obrázek 7.9: ROC křivky pro natrénované SVM na rozpoznání krabice s různými mezerami a nastavenou konstantou b na 4,3

Kapitola 8

Závěr

Má práce se zabývá popisem a využitím informace o hloubce obrazu pro detekci objektů. Cílem práce bylo navrhnout metodu, která bude schopná v rozumném časovém úseku vyhledat a rozpoznat objekt v obraze s pomocí hloubkové mapy získané ze zařízení Kinect. Takovouto metodu jsem navrhnul, realizoval, otestoval a zhodnotil v této práci. Čímž se mi podařilo splnit zadání diplomové práce.

Při svém studiu tématiky práce jsem se nejvíc zaměřil na způsoby, jak účinně popisovat hloubkovou mapu. Díky tomu jsem byl schopen sepsat přehled metod pro popis 3D prostoru a popsat jejich principy a základní vlastnosti. Přitom jsem zjistil, že tento obor není novou látkou, ale spíš znovu objevenou s příchodem zařízení Kinect, protože před tím měření hloubky obrazu bylo příliš náročné, než aby mělo masivnějšího uplatnění. Zároveň jsem při studiu narazil na metodu Spin Image, která mnohokrát sloužila jako metoda, se kterou se porovnávaly ostatní metody. Proto jsem se jí rozhodnul použít jako deskriptor pro svou navrhovanou metodu.

Po vybrání deskriptoru přišel na řadu komplexní návrh metody a její implementace. Hlavně návrh metody byl pro mě obtížný, protože kvůli mým malým zkušenostem se zpracováním obrazu bylo obtížné odhadnout, jestli metoda v celku bude fungovat. Nakonec jsem zvolil metodu Bag of Words společně s SVM klasifikátorem. Po implementaci a otestování těchto dvou metod v kombinaci se Spin Image, jsem zjistil, že tento přístup vykazuje výborné rozpoznávací vlastností, ať už pro rozpoznání určitého objektu, tak pro rozpoznání skupiny podobných objektů. Těchto výsledků bylo dosáhnuto hlavně díky Spin Image, který dokáže velmi přesně popsat povrchy objektů. Přidáním vylepšené metody prohledávání scény jsem získal komplexní metodu pro hledání a rozpoznávání objektů ve scéně, která je natolik rychlá, že je použitelná i v reálnějších podmínkách.

Samozřejmě metoda není dokonalá a je u ní stále co zlepšovat. Mezi zlepšení do budoucna by bylo použití vylepšených variant deskriptoru Spin Image, které by zrychlily rozpoznávání bez ztráty přesnosti, nebo nahradit Spin Image nějakým modernějším deskriptorem hloubkové mapy. Dalším vylepšením by mohlo být použití sofistikovanějšího způsobu vyhledávání. Napadá mě přístup, který by místo 2D okna používal, 3D kostku, která by se pohybovala po hloubkové mapě. Ale to už je problém pro další zkoumání a vývoj, který by na této metodě mohly dále proběhnout.

Na závěr bych dodal, že diplomová práce nakonec splnila má očekávání, že do jisté míry spojí dva obory počítačové grafiky. 3D grafiku díky potřebě myslet v dalším rozměru a zpracování obrazu pro nutnost analýzy obrazu. Zároveň tato práce přinesla i něco nového do oblasti zpracování obrazu.

Literatura

[1] OpenMP [online]. http://openmp.org/wp/, Duben 2012 [cit. 2012-04-23].

[2] OpenCV [online]. http://opencv.willowgarage.com/wiki/, Srpen 2011 [cit. 2011-12-10].

[3] Bo, L.; Ren, X.; Fox, D.: Depth Kernel Descriptors for Object Recognition [online]. http://www.cs.washington.edu/homes/lfb/paper/iros11.pdf, December 2010 [cit. 2011-12-10].

[4] Chen, H.; Bhanu, B.: 3D free-form object recognition in range images using local surface patches [online]. http://dl.acm.org/citation.cfm?id=1020930#, 20 April 2006 [cit. 2011-12-10].

[5] Cortes, C.; Vapnik, V.: Support-Vector Networks [online]. http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf, 1995 [cit. 2011-18-10].

[6] Csurka, G.; Dance, C. R.; Fan, L.; aj.: Visual Categorization with Bags of Keypoints [online]. http://www.cs.cmu.edu/ẽfros/courses/AP06/Papers/csurka-eccv-04.pdf, 2004 [cit. 2011-18-10].

[7] Dalal, N.; Triggs, B.: Histograms of Oriented Gradients for Human Detection [online]. http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf, 2005 [cit. 2011-18-10].

[8] Dinh, H. Q.; Kropac, S.: Multi-Resolution Spin-Images [online]. http://hqdinh.com/papers/Dinh_multires_spinimages_CVPR06.pdf, 22 June 2006 [cit. 2011-12-10].

[9] Fei-Fei, L.; Perona, P.: A Bayesian Hierarchical Model for Learning Natural Scene Categoriesn [online]. http://vision.stanford.edu/documents/Fei-FeiPerona2005.pdf, June 2005 [cit. 2011-18-10].

[10] Freund, Y.; Schapirei, R.: A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting [online]. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1213564, September 20. 1995 [cit. 2011-12-10].

[11] Frome, A.; Huber, D.; Kolluri, R.; aj.: Recognizing Objects in Range Data Using Regional Point Descriptors [online]. http://www.cs.jhu.edu/ misha/Papers/Frome04.pdf, [cit. 2011-12-10].

[12] Hetzel, G.; Leibe, B.; Levia, P.; aj.: 3D Object Recognition from Range Images using Local Feature Histograms [online].
http://www.mmp.rwth-aachen.de/publications/pdf/hetzel-range-cvpr01.pdf, 2001 [cit. 2011-12-10].

[13] Johnson, A. E.; Hebert, M.: Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes [online].
http://www.cs.jhu.edu/m̃isha/Papers/Johnson99.pdf, 1999 květen[cit. 2011-12-10].

[14] Johnson, A. E.; Hebert, M.: Surface Matching for Object Recognition in Complex 3-D Scenes [online].
http://www-robotics.jpl.nasa.gov/publications/Andrew_Johnson/aejIVC1998.pdf, Summer 1998[cit. 2011-12-10].

[15] Lai, K.; Bo, L.; Ren, X.; aj.: A Large-Scale Hierarchical Multi-View RGB-D Object Dataset [online].
http://www.cs.washington.edu/homes/xren/publication/lai_icra11_rgbd_dataset.pdf, May 2011 [cit. 2011-12-10].

[16] Lampert, C. H.; Blaschko, M. B.; Hofmann, T.: Beyond SlidingWindows: Object Localization by Efficient Subwindow Search [online].
http://www.kyb.mpg.de/fileadmin/user_upload/files/publications/pdfs/pdf5070.pdf, 2008 [cit. 2011-18-10].

[17] Li, X.; Guskov, I.: 3D object recognition from range images using pyramid matching [online]. www.eecs.umich.edu/g̃uskov/li-guskov07.pdf, 14-21 Oct. 2007 [cit. 2011-12-10].

[18] Lienhart, R.; Maydt, J.: An Extended Set of Haar-like Features for Rapid Object Detection [online].
http://www.multimedia-computing.de/mediawiki//images/c/c3/Icip2002.pdf, 2002 [cit. 2011-18-10].

[19] Papadimitrious, G.: Spin Images [online].
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0910/SpinImages.pdf, 2000[cit. 2011-12-10].

[20] Ruiz-Correat, S.; Shapirot, L. G.; Meli, M.: A New Signature-Based Method for Efficient 3-D Object Recognition [online].
http://www.cs.washington.edu/homes/shapiro/cvpr01_paper155.pdf, 2001 [cit. 2011-12-10].

[21] Sun, Y.; Paik, J.; Koschan, A.; aj.: Point Fingerprint: A New 3-D Object Representation Scheme [online].
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1213564, Aug. 2003 [cit. 2011-12-10].

[22] Viola, P.; Jones, M.: Robust Real-time Object Detection [online].
http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf, July 13, 2001 [cit. 2011-18-10].

[23] Wang, Y.; Pan, G.; Wuand, Z.; aj.: Sphere-Spin-Image: A Viewpoint-Invariant Surface Representation for 3D Face Recognition [online]. http://www.springerlink.com/content/vur8a2yd4cjl0fqh/fulltext.pdf, 2004 [cit. 2011-12-10].

[24] Wikipedia: Kinect [online]. http://en.wikipedia.org/wiki/Kinect#cite_note-releasedate-1, December 2011 [cit. 2011-12-10].

[25] Wikipedia: Artificial neural networke [online]. http://en.wikipedia.org/wiki/Artificial_neura_network, March 2009 [cit. 2011-12-10].

[26] Wikipedia: Bilinear interpolation [online]. http://en.wikipedia.org/wiki/Bilinear_interpolation, Nov 2010 [cit. 2011-12-10].

[27] Zjevík, O.: Impelnentace a testování SVM [online]. http://am.vsb.cz/theses/bakalari/2011/pdfs/zje002.pdf, 2011 [cit. 2011-18-10].

Příloha A

Obsah CD

- *Dokumentace*
  - *Dokumentace.pdf*
  - *Plagat projektu.pdf*
- *Dokumentace Zdroj*
  - *Grafy* - zpracované tabulky a grafy vystupních dat z programu
  - *Latex* - zdrojové kódy latexu i s obrázky
- *Program*
  - *Data* - složka obsahující vytvořené slovníky, natrénované klasifikátory a výstupní data s testování.
  - *DataSets* - složka obsahující všechny datové sady.
  - *TestDetectScene* - složka obsahující ukázkové sceny pro vyhledávání a rozpoznávání hrnků.
  - *setting.xml* - konfigurační soubor programu viz. manual.
  - *SpinImage.exe* - samotný program
- *Program Zdroj*
  - *SpinImage* - zdrojové kody, knihovny a soubory pro Microsoft Visual Studio 2010

Příloha B

Manual

Toto je zevrubný manuál k demonstračnímu programu, který ukazuje funkčnost metody pro detekci a rozpoznávaní objektu s pomoci deskriptoru Spin Image. Zároveň poskytuje možnost porovnání rozpoznávacích schopností Spin Image v hloubkové mapě, proti SIFT v RGB obraze za stejných podmínek.

Parametry programu

- *h* :nápověda.
- *s* :vypíše nastavení pro všechny částí detekce podle konfiguračního souboru.
- *default* :vygeneruje defaultní konfigurační soubor.
- *dSIFT* :vytvoří slovník pro SIFT, z datové sadě nastavené v konfiguračním souboru, a uloží ho podle nastaveni v konfiguračním souboru.
- *dSI* :vytvoří slovník pro SI, z datové sadě nastavené v konfiguračním souboru, a uloží ho podle nastaveni v konfiguračním souboru.
- *SVMSIFT* :natrénuje klasifikátor na datové sadě (nastavené v konfiguračním souboru) s použitím SIFT pro mezery 3-5-7-9-11-13-15-17.
- *SVMSI* :natrénuje klasifikátor na datové sadě (nastavené v konfiguračním souboru) s použitím SI pro mezery 3-5-7-9-11-13-15-17 a pro konstantu b (nastavenou konfiguračním souboru).
- *detect* :demonstrace funkčnosti cele metody i s vyhledáváním v obraze, která analyzuje obrázek podle nastavení konfiguračního souboru.
- *detectAll* :demonstrace funkčnosti cele metody i s vyhledáváním v obraze, která analyzuje všechny obrázky stejného jmena.
- *lSVM [první pos.] [poslední pos.]* :vytvořeni souboru s označením positivních a negativních vstupu pro trénovaní klasifikátoru.
- *lTD [první pos.] [poslední pos.]* :vytvořeni souboru s označením positivních a negativních vstupu pro testovaní rozpoznávaní.

- *TDSI* :testovaní rozpoznávacích schopnosti metody na datové sadě (nastavené v konfiguračním souboru) s detektorem Spin Image pro mezery 3-5-7-9-11-13-15-17 a konstantou b (získané s konfiguračního souboru).

- *TDSIFT* :testovaní rozpoznávacích schopnosti metody s detektorem SIFT pro mezery 3-5-7-9-11-13-15-17, na datové sadě nastavené v konfiguračním souboru.

Popis jednotlivých parametru v konfiguračním souboru

- *data_Paths:* nastaveni cest k datovým sadám.
  - *dict_SI_Data_Path:* cesta k RGB-D data setů pro slovník.
  - *dict_SI_Data_Name:* hromadný název ve tvaru [název]_[cislo].png.
  - *svm_SI_Train_Data_Path:* cesta k RGB-D data setů pro klasifikátor.
  - *svm_SI_Train_Data_Name:* hromadný název ve tvaru [název]_[cislo].png.
  - *test_SI_Detect_Data_Path:* cesta k RGB-D data setů pro test.
  - *test_SI_Detect_Data_Name:* hromadný název ve tvaru [název]_[cislo].png.
  - *dict_SIFT_Data_Path:* cesta k RGB- data setů pro slovník.
  - *dict_SIFT_Data_Name:* hromadný název ve tvaru [název]_[cislo].png.
  - *svm_SIFT_Train_Data_Path:* cesta k RGB data setů pro klasifikátor.
  - *svm_SIFT_Train_Data_Name:* hromadný název ve tvaru [název]_[cislo].png.
  - *test_SIFT_Detect_Data_Path:* cesta k RGB data setů pro test.
  - *test_SIFT_Detect_Data_Name:* hromadný název ve tvaru [nazev]_[cislo].png.
  - *path_Dictionary:* cesta ke slovníkům(soubory musí být už vytvořené)«¡endl;
  - *path_SVM_Train:* cesta k SVM(soubory musí být už vytvořené)
  - *path_Test_Output_SI:* cesta k souboru pro uloženi výstupu testování SI(soubory musí být už vytvořené)
  - *path_Test_Output_SIFT:* cesta k souboru pro uloženi výstupu testování SIFT (soubory musí být už vytvořené)
  - *path_Test_Detect_Scene:* cesta k souboru obsahujícím sceny pro testování
- *settings_SI:* hlavni nastaveni SI deskriptoru.
  - *b_SI:* nastaveni konstanty b pro všechny ostatní proměnné.
  - *alpha_Max_SI:* velikost α_{max}.
  - *beta_Max_SI:* velikost β_{max}.
- *settings_Method:* nastaveni různých částí programu.
  - *gapsSI:* velikost mezer mezi deskriptory při tvořeni slovníku a u natrénovaného klasifikátoru.
  - *dictionary_Size:* počet slov ve slovníku.
  - *dictionary_SI_File:* soubor kam se uloží slovník SI a odkud se bude načítat (číslo velikosti b konstanty se samo dopíše podle *b_SI*).
  - *dictionary_SIFT_File:* soubor kam se uloží slovník SIFT a odkud se bude načítat.

  - *label_SVM_File:* označeni positivních a negativních obrázku pro trénování SVM.
  - *label_Test_Detect_File:* označeni positivních a negativních obrázku pro testovaní rozpoznávacích vlastnosti metody.
  - *train_Depth_SVM_File:* soubor odkud se bude načítat natrénovaný SVM klasifikátor pro Spin Image (číslo velikosti b konstanty i velikost mezer se samo dopise).
  - *train_RGB_SVM_File:* soubor odkud se bude načítat natrénovaný SVM klasifikátor pro SIFT(velikost mezer se sama dopise).

- *settings_Detection:* nastaveni vyhledávaní ve scéně při tomto vyhledávání se používá výhradně Spin Image.
  - *detect_Scene_Num:* číslo testované scény.
  - *detect_Scene:* jmeno obrázku, ve kterém se bude detekovat objekt, ve složce musí bý souborry tvaru [detect_Scene]_[cislo]_depth.png (soubor s RGB-D) a [detect_Scene]_[cislo].png (soubor s RGB).
  - *min_SI_in_Win:* minimální poměr k maximu deskriptoru v okně, který se ještě kontroluje.
  - *depth_Distance:* velikost hloubky jednotlivých úrovni v mm.
  - *shift_Window:* násobek šířky okna, o který se posunuje okno při prohledávání obrazu.
  - *gaps_SI_Slide_Win:* velikost mezer mezi deskriptory ve scéně.
  - *window_Size_Width:* šířka prohledávacího okna.
  - *window_Size_Height:* výška prohledávacího okna.
  - *level_Count:* počet úrovni, v kterých se bude prohledávat.
  - *show_Levels:* 1 zobrazuje barevné zvýraznění úrovní, 0 nezvýrazní úrovně.

Příloha C

Konfigrační soubor

```
<?xml version="1.0"?>
<opencv_storage>
<data_Paths>
  <dict_SI_Data_Path>DataSets\DataSetDictionary\depth\
  </dict_SI_Data_Path>
  <dict_SI_Data_Name>depthImage</dict_SI_Data_Name>
  <svm_SI_Train_Data_Path>DataSets\TrainHrnky\depth\
  </svm_SI_Train_Data_Path>
  <svm_SI_Train_Data_Name>depthImage</svm_SI_Train_Data_Name>
  <test_SI_Detect_Data_Path>DataSets\testDetectDataHrnky\depth\
  </test_SI_Detect_Data_Path>
  <test_SI_Detect_Data_Name>depthImage</test_SI_Detect_Data_Name>

  <dict_SIFT_Data_Path>DataSets\DataSetDictionary\crop\
   </dict_SIFT_Data_Path>
  <dict_SIFT_Data_Name>image</dict_SIFT_Data_Name>
  <svm_SIFT_Train_Data_Path>DataSets\TrainHrnky\crop\
  </svm_SIFT_Train_Data_Path>
  <svm_SIFT_Train_Data_Name>image</svm_SIFT_Train_Data_Name>
  <test_SIFT_Detect_Data_Path>DataSets\testDetectDataHrnky\crop\
  </test_SIFT_Detect_Data_Path>
  <test_SIFT_Detect_Data_Name>image</test_SIFT_Detect_Data_Name>
  <path_Test_Detect_Scene>TestDetectScene\
  </path_Test_Detect_Scene>

  <path_Dictionary>Data\Dictionary\
  </path_Dictionary>
  <path_SVM_Train>Data\TrainSVM_hrnky\
  </path_SVM_Train>
  <path_Test_Output_SI>Data\Outputs_Hrnky_SI\
  </path_Test_Output_SI>
  <path_Test_Output_SIFT>Data\Outputs_Hrnky_SIFT\
  </path_Test_Output_SIFT>
  <path_Test_Detect_Scene>TestDetectScene\
  </path_Test_Detect_Scene>
```

```
</data_Paths>

<settings_SI>
  <b_SI>3.5</b_SI>
  <alpha_Max_SI>40</alpha_Max_SI>
  <beta_Max_SI>40</beta_Max_SI>
</settings_SI>

<settings_Method>
  <gapsSI>11</gapsSI>
  <dictionary_Size>500</dictionary_Size>
  <dictionary_SI_File>Data\Dictionary\dict_depth
  </dictionary_SI_File>
  <dictionary_SIFT_File>Data\Dictionary\dict_RGB.dic
  </dictionary_SIFT_File>

  <label_SVM_File>DataSets\TrainHrnky\label_SVM_Hrnky
  </label_SVM_File>
  <label_Test_Detect_File>DataSets\testDetectDataHrnky\label_test_Detect_Hrnky
  </label_Test_Detect_File>

  <train_Depth_SVM_File>Data\TrainSVM_hrnky\train_svm_depthImage
  </train_Depth_SVM_File>
  <train_RGB_SVM_File>Data\TrainSVM_hrnky\svm_train_RGB
  </train_RGB_SVM_File>
</settings_Method>

<settings_Detection>
  <detect_Scene_Num>1</detect_Scene_Num>
  <detect_Scene>testScene</detect_Scene>
  <min_SI_in_Win>0.4</min_SI_in_Win>
  <depth_Distance>250</depth_Distance>
  <shift_Window>0.2</shift_Window>
  <gaps_SI_Slide_Win>11</gaps_SI_Slide_Win>
  <window_Size_Width>110</window_Size_Width>
  <window_Size_Height>110</window_Size_Height>
  <level_Count>2</level_Count>
  <show_Levels>1</show_Levels>
</settings_Detection>
</opencv_storage>
```