Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Sledování pohybu objektu ve 3D pomocí hloubkového snímače

Plzeň, 2016 Filip Berka

P R O H L Á Š E N Í

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne .....

Abstrakt

Tato práce se zabývá sledováním ruky v hloubkových obrazech. V teoretické části je nejprve popsán použitý senzor Kinect v2. Dále práce popisuje algoritmy počítačového vidění a strojového učení, které byly použity v části praktické.

Praktická část popisuje postup při řešení úlohy a shrnuje dosažené výsledky. Úloha byla řešena metodou klasifikace pixelů hloubkových obrazů do tříd ruky a pozadí náhodným rozhodovacím lesem. Postup při řešení spočíval ve vytvoření trénovací množiny hloubkových obrazů a označení pixelů, které byly součástí ruky. Toto označení pak sloužilo jako informace učitele. Analýzou změny hloubky v okolí takto označených pixelů byly vytvořeny příznakové vektory. Ty byly dále použity k trénování klasifikátorů. Pro nejsložitější uvažované nastavení klasifikátoru bylo na testovací množině dosaženo téměř 90 % F1-score. Klasifikátory byly též aplikovány na hloubkové obrazy plynoucí z Kinectu v reálném čase.

Klíčová slova: Kinect, OpenCV, Sledování ruky, Počítačové vidění, Klasifikace, Náhodný rozhodovací les

Abstract

This thesis concerns with hand tracking in depth images. In the theoretical part at first there is a description of Kinect v2 sensor which was used to acquire the depth images. Then it describes algorithms of computer vision and machine learning which were used in the practical part.

The practical part of the thesis describes solution of the problem and summarizes achieved results. The problem was solved by classification of depth image pixels into the hand class and the background class using Random Decision Forest. The solution is based on creation of a dataset of depth images and labeling of pixels which were part of the hand. This labeling was used as the teacher information. Analysing the change in depth in the neighbourhoods of labeled pixels the feature vectors were created. They were then used to train the classifiers. For the most complicated considered settings of the classifier nearly 90 % F1-score was reached on the test set. The classifiers were also used on the depth images streaming from Kinect in real-time.

Key words: Kinect, OpenCV, Hand tracking, Computer vision, Classification, Random Decision Forest

Poděkování

Děkuji panu Ing. Marku Hrúzovi PhD. za pomoc při vedení diplomové práce. Dále děkuji Ing. Janě Berkové, která mi pomohla při zdlouhavé přípravě dat. Tomáši Berkovi a Kristýně Kaslové děkuji za jejich čas při tvorbě výsledných demonstračních videí.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"(CESNET LM2015042), is greatly appreciated.

Obsah

1 Úvod **1**
1.1 Řešený problém . . . 1
1.2 Současný stav řešení . . . 2

2 Kinect **3**
2.1 Funkce . . . 3
2.2 Propojení s počítačem . . . 5

3 Zpracování obrazu **8**
3.1 Digitální obraz . . . 8
3.1.1 Definice a digitalizace . . . 8
3.1.2 Barevné modely . . . 10
3.1.3 Vzdálenost a okolí . . . 12
3.2 Použité algoritmy . . . 13
3.2.1 Continuously Adaptive Mean Shift . . . 13
3.2.2 Prahování . . . 15
3.2.3 Hledání kontur . . . 16
3.2.4 Mediánový filtr . . . 16
3.2.5 Morfologické operace . . . 17
3.2.6 Graph Cut . . . 19
3.2.7 Flood Fill . . . 22

4 Klasifikace **23**
4.1 Náhodný rozhodovací les . . . 24
4.1.1 Rozhodovací strom . . . 24
4.1.2 Rozhodovací les . . . 26
4.1.3 Náhodný rozhodovací les . . . 27
4.2 Hodnocení kvality klasifikátoru . . . 27
4.2.1 Matice chyb . . . 28
4.2.2 Podtrénování a přetrénování . . . 29

5 Sledování ruky v hloubkové mapě **30**
5.1 Vytvoření datasetu . . . 31
5.1.1 Získávání hloubkových obrazů a prvotních labelů ruky . . . 31
5.1.2 Zpřesňování labelů ruky . . . 32
5.1.3 Výběr labelů pozadí . . . 33
5.2 Extrakce příznakových vektorů . . . 35
5.3 Trénování a testování klasifikátoru . . . 37

5.4 Post processing 38
5.5 Real-time 39

6 Problémy při řešení **42**
6.1 Složitost příznakových vektorů 42
6.2 Přítomnost druhé ruky v hloubkových obrazech 42
6.3 Časové a paměťové nároky 43
6.4 Scéna sledování 43
6.5 FPS real-time aplikace 44

7 Závěr **45**

Literatura **46**

A Dataset **48**

B Sledovací program **49**

C Demonstrační videa **51**

Seznam obrázků

1.1 Porovnání ground-truth labelů a labelů odvozených RDF [1]. 2

2.1 Umístění komponent v Kinectu 2 [3]. 4
2.2 Hloubkové mapa. 6
2.3 3D hloubková mapa. 6
2.4 Rozdílné FOV hloubkového a barevného snímače. 7
2.5 Změna barevného obrazu po registraci. 7

3.1 Možné struktury vzorkování [6]. 9
3.2 Porovnání vzorkování 256x256 a 64x64 [7]. 9
3.3 Porovnání kvantování 256 hodnot (8 bitů) a 4 hodnoty (2 bity) [7]. . 9
3.4 Barevný model RGB. 10
3.5 Tóny šedi. 10
3.6 Barevný model HSV. 11
3.7 Vzdálenosti pixelů [7]. 12
3.8 4okolí a 8okolí [7]. 13
3.9 Binární prahování. 16
3.10 Nalezené kontury. 16
3.11 Filtrace mediánem [7]. 17
3.12 Strukturní elementy [7]. 17
3.13 Ukázka eroze [14]. 18
3.14 Ukázka dilatace [14]. 18
3.15 Ukázka otevření [14]. 19
3.16 Ukázka uzavření [14]. 19
3.17 Segmentace algoritmem Graph Cut [16]. 20
3.18 Ukázka funkce algoritmu Graph Cut [17]. 22
3.19 Ukázka funkce algoritmu Flood Fill. 22

4.1 Rozdělený příznakový prostor a rozhodovací strom [17]. 25
4.2 Vliv počtu stromů na rozdělení příznakového prostoru [20]. 26
4.3 Vliv hloubky stromů na rozdělení příznakového prostoru [20]. 27
4.4 Podtrénování a přetrénování v problému klasifikace. 29

5.1 Schéma postupu řešení úlohy. 30
5.2 Výseč prvotní podoby označení ruky. 32
5.3 Výseče jistého popředí a jistého pozadí před a po erozi. 32
5.4 Výseč finální podoby označení ruky dosažené algoritmem Grab Cut. . 33
5.5 Výseč finální podoby označení ruky dosažené algoritmem Flood Fill. . 33
5.6 Výsledek prvních experimentů a označení zástupců třídy pozadí. . . . 34

5.7 Aspekty extrakce příznakových vektorů. 36
5.8 Výběr vzdálených pixelů. 36
5.9 Výsledek klasifikace. 37
5.10 Výsledek klasifikace po post processingu. 39
5.11 Vizualizace pohybu ruky ve videu. 41
5.12 Ukázka různých pozic ruky při provozu real-time aplikace. 41

6.1 Problém druhé ruky v trénovacích obrazech. 43
6.2 Nevhodná scéna sledování. 44

Seznam tabulek

2.1 Srovnání parametrů Kinectu 1 a Kinectu 2 [3]. 4

4.1 Matice chyb. 28

5.1 Výsledky testování s různým nastavením parametrů lesa. 38
5.2 Výsledky testování s různým nastavením parametrů lesa po post processingu. 39

Kapitola 1

Úvod

Sledování objektu je definováno jako znalost o pozici objektu v každém obrazu video sekvence. Video sekvencí se rozumí množina po sobě jdoucích obrazů. Lze tedy předpokládat, že pozice objektu v aktuálně zpracovávaném obrazu bude přibližně ve stejných místech jako v obraze předešlém. Tento koncept se při sledování dá využít.

Úloha sledování objektů je velmi často řešeným problémem počítačového vidění kvůli svému využití v průmyslu či bezpečnostních službách. Proto byl vyvinut velký počet algoritmů, jež ji řeší. Častými metodami detekce a sledování jsou například prahování, odčítání pozadí, optický tok, SIFT, SURF, Cam Shift, Particle Filter a další. Rovněž se dá využít algoritmů z oblasti strojového učení a klasifikace. Všechny tyto metody však nejsou obecně vhodné pro každý typ problému.

Často sledovaným objektem je ruka kvůli mnohému využití v HCI - Human Computer Interaction. Tedy v oblasti přirozené interakce člověka s počítačem. Tento problém je studován od 90. let minulého století a stále není jednoznačně určená nejlepší metoda řešení. Jedním způsobem sledování ruky jsou rukavice s elektromechanickými senzory, které měří pozice a natočení jednotlivých kloubů s vysokou přesností. Nicméně takové rukavice ovlivňují přirozený pohyb ruky a jsou drahé. Je tedy potřeba řešení založené na obrazech.

Sledování ruky z obrazů je obecně obtížná záležitost. Ruka je artikulovaný objekt s mnoha stupni volnosti, což znamená velký počet možných póz. Metody pro sledování ruky jsou typicky kategorizovány do dvou skupin: apperance-based a model-based. Apperance-based metody odhadují stav ruky analýzou obrazu a hledání takových příznaků, které by mohly odpovídat ruce. Model-based metody odhadují stav ruky snahou najít parametry předem definovaného modelu (typicky 25 až 50 stupňů volnosti), aby pak co nejlépe odpovídal ruce v obraze. Obě metody se snaží odhadnout plný stav ruky, nicméně v případě apperance-based metod se tak často nestane a jsou odhadnuty pouze některé parametry, mluví se o nich tedy jako o 2D hand tracking metodách.

1.1 Řešený problém

Cílem této práce bylo seznámit se se senzorem Kinect v2, propojit ho s počítačem, navrhnout metodu sledování pohybu objektu ve 3D a tuto metodu s podporou

knihovny OpenCV naprogramovat. Jako objekt sledování byla zvolena lidská ruka bez barevně odlišitelných vlastností nebo rukavic. Rukou se zde rozumí pouze část lidské horní končetiny od zápěstí po prsty.

1.2 Současný stav řešení

Řešená úloha v této práci se opřela o část článku *Real-time continuous pose recovery of human hands using convolutional networks* [1] od autorů J. Tompson, M. Stein, Y. Lecun a K. Perlin. Tato práce se zabývá v prvé řadě sledováním lidské ruky metodou klasifikace jednotlivých pixelů hloubkových obrazů získaných hloubkovým senzorem. Obrazy jsou dále přivedeny na vstup konvoluční neuronové sítě, která slouží ke klasifikaci pózy ruky a odhadnutí 42 parametrů nastavení ruky. Těchto 42 parametrů je poté použito pro vykreslení 3D modelu ruky. Ten by pak v ideálním případě měl odpovídat skutečné ruce.

Ke klasifikaci pixelů v hloubkových obrazech se zde používá klasifikátor náhodný rozhodovací les (Random Decision Forest), který je rovněž vhodný pro klasifikaci jiných částí lidského těla [2]. Je použit také pro svou jednoduchou paralelizovatelnost, proto se hodí do real-time aplikací. Každý strom se skládá z uzlů, které porovnávají rozdíl hloubek aktuálního pixelu a pixelů ve stanovených posunu s natrénovaným prahem. Každý strom tak může rozhodnout o tom, zda aktuální pixel je součástí ruky nebo pozadí. Průměrováním odpovědí všech stromů lze sestavit pravděpodobnostní rozdělení pro celý obraz. V každém uzlu každého stromu se pro pixel (u, v) testují náhodné dimenze příznakových vektorů podle vztahu

$$I\left(u + \frac{\Delta u}{I(u,v)}, v\ + \frac{\Delta v}{I(u,v)}\right) - I(u,v) \geq d_t, \tag{1.1}$$

kde $I(u, v)$ je hodnota pixelu na souřadnicích (u, v) hloubkového obrazu I , Δu a Δv jsou posuny v souřadnicích a d_t je natrénovaný práh. Experimentálně bylo zjištěno, že záleží na velkém rozsahu posunů v souřadnicích, aby klasifikace dosahovala vysoké úspěšnosti. Je tomu tak, protože je potřeba pixely klasifikovat nejen na základě lokální geometrie, ale i globální.

Klasifikace pixelů ruky dosahuje na testovací sadě klasifikační chyby 4,1% při 4 stromech o hloubce 25. Tyto chyby jsou poté odstraněny mediánovým filtrem. Sledování zde funguje v reálném čase.

Cílem této diplomové práce je přiblížit se takovým výsledkům.

Obrázek 1.1: Porovnání ground-truth labelů a labelů odvozených RDF [1].

Kapitola 2

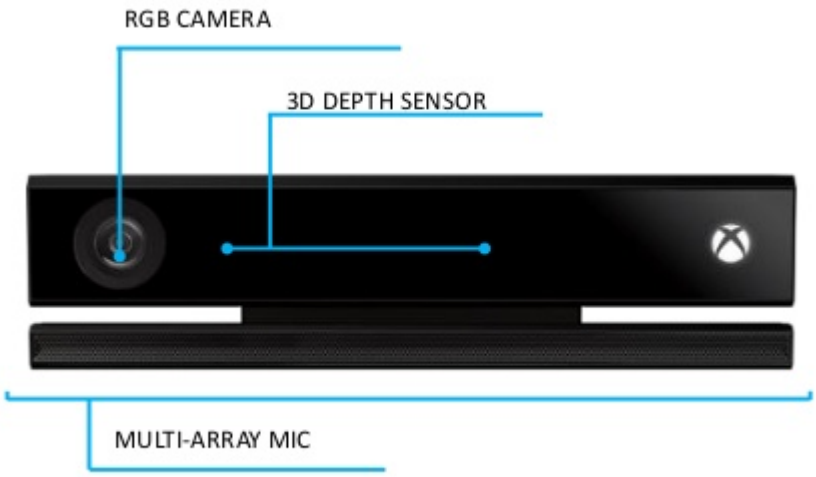
Kinect

Microsoft Kinect for Xbox One (dále jen Kinect 2) je vylepšením svého předchůdce Microsoft Kinect for Xbox 360 (dále jen Kinect 1). Oba tyto pohybové 3D senzory byly původně stvořeny pro herní průmysl ke konzolím Xbox. Od doby, co byl poprvé Kinect představen, vznikla i spousta jiných 3D senzorů od konkurenčních firem, např. Asus Xtion, Apple PrimeSense Carmine a další. Hry se ovládají přirozenými pohyby, gesty a řečí. Uživatel tedy nemusí držet žádný ovladač. Nejen hráči her však našli pro tyto senzory využití. Pro komunitu z oblasti počítačového vidění se tak vedle stereovize otevřely nové možnosti práce ve 3D prostoru.

Kinect 1 byl poprvé představen 1. června 2009 pod názvem Project Natal na roční tiskové konferenci Microsoftu Electronic Entertainment Expo. 13. června 2010 Microsoft oznámil, že název zařízení se změní na Kinect. Slovo Kinect je spojením slov kinetic (pohybový) a connect (připojit). 4. listopadu se Kinect dostal na trh v USA, o týden později pak v Evropě s cenou 149,99 dolarů. V roce 2011 vyšly i originální ovladače a Software Development Kit (SDK) pro Windows. V červenci roku 2014 začal prodej Kinectu 2 spolu s vydáním ovladačů a Kinect for Windows SDK 2.0 pro Windows.

2.1 Funkce

Na Obrázku 2.1 jsou vidět důležité části Kinectu: RGB kamera, hloubkový senzor a pole mikrofonů.



Obrázek 2.1: Umístění komponent v Kinectu 2 [3].

Tabulka 2.1 srovnává jednotlivé parametry snímačů Kinect 1 a Kinect 2.

Verze	**1**	**2**
Rozsah	0,4 - 4 m	0,4 - 4,5 m
Rozlišení RGB	640x480	1920x1080
Rozlišení hloubka	320x240	512x424
Rozlišení IR	Není k dispozici	512x424
Technologie 3D	Light coding	Time of flight
Audio	4 mikrofony 16 kHz	4 mikrofony 48 kHz
USB	2.0	3.0
Počet skeletonů	2 (+4)	6
Počet kloubů	20	25
Sledování ruky	Externě	Ano
Sledování obličejů	Ano	Ano + výrazy
FOV	57° H 43° V	70° H 60° V
Naklánění	Motorizované	Manuální

Tabulka 2.1: Srovnání parametrů Kinectu 1 a Kinectu 2 [3].

Kinect 1 funguje na principu pojmenovaném light coding. Jeho hloubkový senzor má 2 části: infračervený zdroj a snímač infračerveného záření. Infračerveným zdrojem je laser, který vysílá pseudonáhodnou strukturu světla do zorného pole (FOV - field of view) snímače. Tuto strukturu snímač zpětně detekuje a vyhodnocuje tak tzv. hloubkovou mapu podle následujícího pravidla. Pokud jsou světelné tečky nahuštěné blízko sebe a jsou malé, objekt je daleko. Pokud jsou tečky velké a dále od sebe, objekt je blízko. Kinect 1 v této práci nebyl použit.

Kinect 2 funguje na principu zvaném time of flight. Nejjednodušší verze tohoto

principu používá světelné pulzy infračerveného zdroje. Laser je na krátkou dobu zapnut - krátký pulz osvítí scénu a odrazí se od objektů v zorném poli. Odražené světlo pak zachytí senzor. Závisle na vzdálenosti objektů, světlo dopadne se zpožděním. Rychlost světla je přibližně

$$c \approx 3 \cdot 10^8 \ ms^{-1}. \tag{2.1}$$

Proto je toto zpoždění velmi krátké. Například objekt vzdálený 2,5 m zpozdí světlo o

$$t_D = 2 \cdot \frac{D}{c} = 2 \cdot \frac{2{,}5 \ m}{3 \cdot 10^8 \ ms^{-1}} = 16{,}66 \ ns. \tag{2.2}$$

Délka světelného pulzu ovlivňuje rozsah, který kamera může zachytit. Pro pulz dlouhý 50 ns je tam maximální hloubka omezena

$$D_{max} = \frac{1}{2} \cdot c \cdot t_0 = \frac{1}{2} \cdot 3 \cdot 10^8 \ ms^{-1} \cdot 50 \ ns = 7{,}5 \ m. \tag{2.3}$$

Z tohoto důvodu je ozařovací jednotka nejdůležitějším dílem systému. Pouze se speciálními LED diodami nebo lasery je možné dosáhnout tak krátkých pulzů.

Pixel se skládá z fotocitlivého elementu (např. fotodiody). Tam se převádí příchozí světlo na proud. Ten je poté naveden na přepínač, který proud vede dále do dvou paměťových elementů (např. kondenzátory), které se chovají jako sumační jednotky. Přepínač je řízen pulzem se stejnou délkou jako světelný pulz, přičemž jeden jeho konec je zpožděn přesně o šířku pulzu. Závisle na zpoždění odraženého světla, pouze část projde prvním koncem přepínače do prvního paměťového elementu S_1 a zbytek projde druhým koncem přepínače do paměťového elementu S_2. Hloubka je pak vypočtena závisle na poměru těchto dvou hodnot ze vztahu

$$D = D_{max} \cdot \frac{S_2}{S_1 + S_2}. \tag{2.4}$$

Ku příkladu je-li $S_1 = 0,66$ a $S_2 = 0,33$, pak hloubka

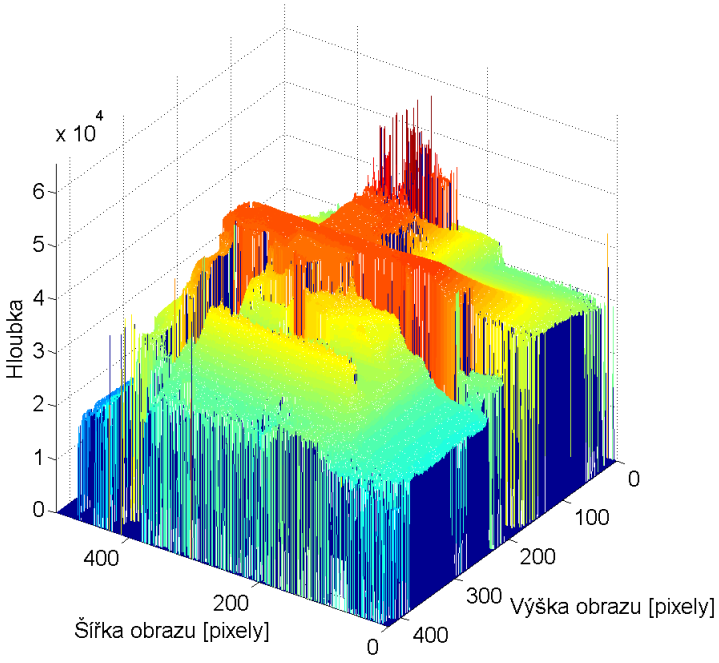
$$D = 7{,}5 \ m \cdot \frac{0{,}33}{0{,}33 + 0{,}66} = 2{,}5 \ m. \tag{2.5}$$

Výsledkem je 2D hloubková mapa. Je to obraz, jehož každý pixel obsahuje hodnotu vzdálenosti od senzoru. Hloubková mapa se dá zobrazit ve 2D jako mapování hloubky na barevnou škálu, například v tónech šedi. Příklad 16bitové (65536 barev) hloubkové mapy je na Obrázku 2.2. Další možností zobrazení hloubkové mapy je 3D graf. Příklad takového grafu je na Obrázku 2.3. Více o time of flight v [4].

2.2 Propojení s počítačem

Microsoft původně pro Kinect 1 nevydal ovladače ani SDK. Počítačová komunita tedy nemohla Kinect nijak využít. Na to zareagovala společnost Adafruit Industries vyhlášením soutěže o naprogramování open source ovladače o odměnu 1000 dolarů. Po tom, co tuto soutěž Microsoft odsoudil, vzrostla tato odměna na 2000 dolarů. 6 dní po vydání Kinectu hacker jménem Hector tuto soutěž vyhrál a získal tak nyní

Obrázek 2.2: Hloubkové mapa.



Obrázek 2.3: 3D hloubková mapa.

již 3000 dolarů. Po vydání dalších open source ovladačů Microsoft prohlásil, že byl jen zvědav, s čím komunita přijde.

Ke Kinectu 2 ovladače i SDK vyšly spolu se zařízením. V této práci byl však použit ovladač od organizace OpenKinect. Ovladač se nazývá libfreenect2 [5] a podporuje přenos RGB, IR a hloubkových obrazů. Dále pak podporuje metodu registrace mezi RGB a hloubkový obrazem.

Registrace se stará o vyřešení problému nestejného FOV barevného a hloubkového obrazu. Jelikož je barevná kamera umístěna v senzoru na jiném místě než hloubkový snímač, jednotlivé pixely obou obrazů si neodpovídají. Toto je vyřešeno geometrickou transformací barevného obrazu - homografií. Na Obrázku 2.4 jsou obrazy

obou snímačů tak, jak je Kinect původně vrací. Na Obrázku 2.5 je vidět změna barevného obrazu. Po registraci si již souřadnice obou obrazů do jisté míry odpovídají.

Obrázek 2.4: Rozdílné FOV hloubkového a barevného snímače.

Obrázek 2.5: Změna barevného obrazu po registraci.

Kapitola 3

Zpracování obrazu

Počítačové vidění je vědní oblast zabývající se porozumění informací obsažených v kamerou zachycených obrazech. Patří sem digitální zpracování obrazu, kde se uplatňuje především teorie zpracování signálu. Vstupem a výstupem jsou obrazová data nebo jejich jednoduché charakteristiky. Patří sem například: filtrace šumu, transformace obrazu a jednoduché metody hledání objektů v obraze. Dále sem patří složitější přístupy, ve kterých se uplatňuje především algoritmů strojového učení. Například vytváření modelů, sledování, klasifikace a detekce jevů. Příbuznými obory jsou umělá inteligence, strojové učení, zpracování signálu, statistika, optika.

Tato kapitola uvede základní informace o zavedených pojmech používaných v počítačovém vidění a popíše jednotlivé algoritmy z této oblasti použité v této práci.

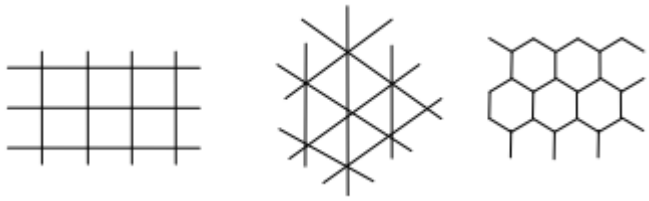
3.1 Digitální obraz

Základním a nejdůležitějším pojmem počítačového vidění je digitalizovaný obraz.

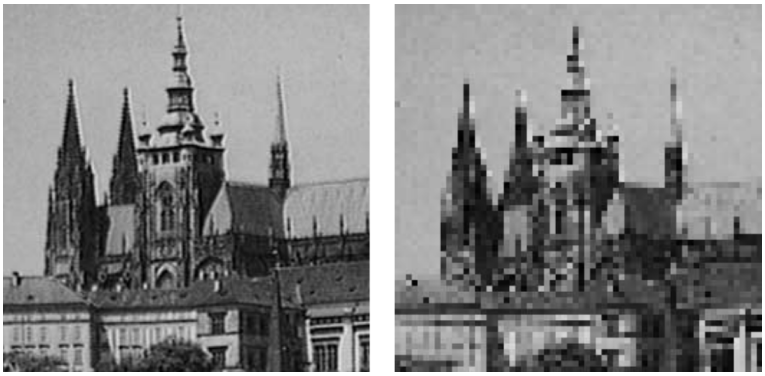
3.1.1 Definice a digitalizace

Obraz může být považován za signál o více rozměrech. Matematickým modelem obrazu je obrazová funkce. Tou se rozumí spojitá funkce dvou proměnných $f(x, y)$, kde (x, y) označují souřadnice v obrazové rovině. Funkce může být rozšířena o souřadnici času, pak se označuje $f(x, y, t)$. Hodnoty této funkce odpovídají fyzikální veličině, které zařízení snímá. Může jít o jas, teplotu a jiná záření, může ovšem jít třeba i o hloubku, tedy vzdálenost od kamery. Je-li potřeba pracovat s obrazem na počítači, je potřeba ho digitalizovat. Tento proces má 2 fáze: vzorkování a kvantování.

Vzorkování obrazů je proces rozdělení obrazové funkce na buňky podle některé ze struktur na Obrázku 3.1. Používají se skoro výhradně čtvercové mřížky, nicméně dá se vzorkovat i na trojúhelníky či šestiúhelníky. Jedna buňka takto diskretizovaného obrazu se nazývá pixel (z anglického jazyka - picture element). Při vzorkování je potřeba myslet na Nyquist-Shannonův vzorkovací teorém. Ten říká, že vzorkovací frekvence musí být větší nebo rovna než je dvojnásobek nejvyšší frekvence v signálu. V obraze toto znamená, že velikost vzorku (pixelu) musí být alespoň dvakrát menší než nejmenší detail, který je ještě potřeba zachytit.



Obrázek 3.1: Možné struktury vzorkování [6].



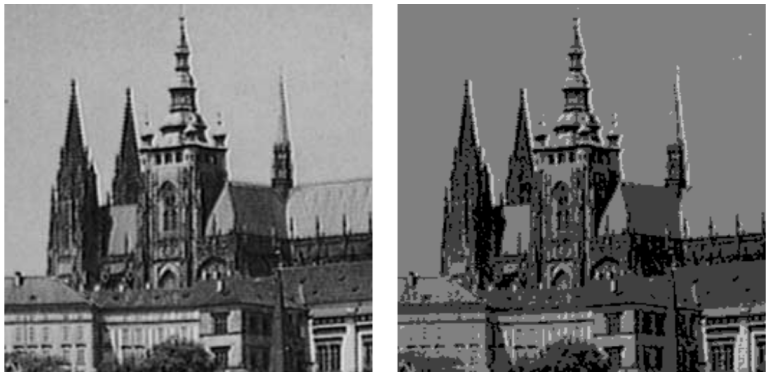
Obrázek 3.2: Porovnání vzorkování 256x256 a 64x64 [7].

Kvantování je proces rozdělení oboru hodnot obrazové funkce na K hodnot. Tyto hodnoty od sebe mohou být vzdálené ekvidistantně i neekvidistantně. Neekvidistantní rozdělení je vhodné, pokud je potřeba docílit vyšší přesnosti v okolí nějaké specifické hladiny z důvodu rozlišení drobných detailů.

Velikost rozsahu diskretizovaného oboru hodnot závisí na počtu bitů, které jsou pro zakódování hodnot použity. Pro b bitů bude pak K hodnot, podle

$$K = 2^b. \tag{3.1}$$

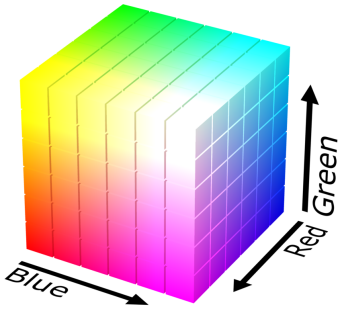
Do každé buňky matice se uloží hodnota od 0 do $K-1$. Více o obrazové funkci v [8].



Obrázek 3.3: Porovnání kvantování 256 hodnot (8 bitů) a 4 hodnoty (2 bity) [7].

3.1.2 Barevné modely

V případě snímání barevných obrazů se často používá barevný model RGB. Jde o aditivní způsob míchání barev, který se používá v monitorech a projektorech. Pro získání požadované barvy se pro každý pixel skládají mohutnosti tří základních barev: R - červené (red), G - zelené (green) a B - modré (blue). Mohutnost jednotlivých tzv. kanálů se udává v procentech nebo za použití barevné hloubky. Například pro 8bitové obrazy může mohutnost jednotlivých kanálů nabývat hodnot od 0 do 255. Barevný model RGB lze zobrazit jako krychli znázorněnou na Obrázku 3.4.



Obrázek 3.4: Barevný model RGB.

Často je potřeba obrazy převést z barevného modelu RGB do stupňů šedi (grayscale). Intuitivně se nabízí řešení

$$Y = \frac{R + G + B}{3}. \tag{3.2}$$

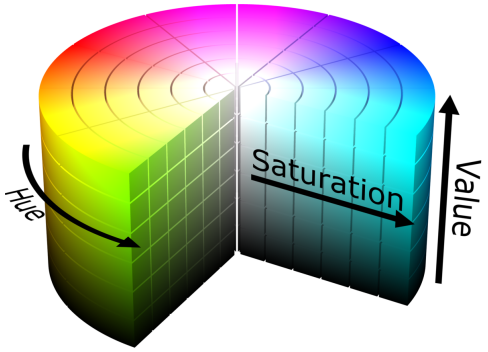
Takový přístup však nerespektuje vyšší citlivost lidských očí na zelenou složku a používá se proto váženého součtu

$$Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B. \tag{3.3}$$

Hodnota 0 ve stupních šedi znamená černou barvu. Hodnota 255 by pak pro 8bitový obraz znamenala barvu bílou. Další barvy mezi nimi by odpovídaly různým odstínům šedi jako například na Obrázku 3.5.

Obrázek 3.5: Tóny šedi.

Dalším často používaným barevným modelem je HSV. Model HSV nejvíce odpovídá lidskému vnímání barev. Jde o skládání tří vlastností barev (nikoliv samotných barev jako v případě RGB). Tyto složky jsou: H - odstín (hue), S - saturace (saturation) a V - jas (value). Model HSV lze zobrazit jako válec znázorněný na Obrázku 3.6. Hodnota odstínu je od 0 ° do 360 ° a představuje obecně, o jakou barvu se jedná. Hodnoty saturace a jasu se udávají v procentech nebo opět za použitá barevné hloubky. Saturace představuje sytost barvy. Čím je větší sytost, tím je



Obrázek 3.6: Barevný model HSV.

barva „čistší“. Jas vyjadřuje množství bílého světla.

Pro převod 8bitových obrazů z barevného modelu RGB do HSV se používá následující postup. Normalizace intervalů

$$R' = \frac{R}{255}, \tag{3.4}$$

$$G' = \frac{G}{255}, \tag{3.5}$$

$$B' = \frac{B}{255}. \tag{3.6}$$

Zjištění maxima a minima a jejich rozdílu

$$C_{max} = \max(R', G', B'), \tag{3.7}$$

$$C_{min} = \min(R', G', B'), \tag{3.8}$$

$$\Delta = C_{max} - C_{min}. \tag{3.9}$$

Výpočet složky H

$$H = \begin{cases} 0 & , \Delta = 0, \\ 60 \cdot \frac{G'-B'}{\Delta} + 0 & , C_{max} = R' \text{ a } G' \geq B', \\ 60 \cdot \frac{G'-B'}{\Delta} + 360 & , C_{max} = R' \text{ a } G' < B', \\ 60 \cdot \frac{B'-R'}{\Delta} + 120 & , C_{max} = G', \\ 60 \cdot \frac{R'-G'}{\Delta} + 240 & , C_{max} = B'. \end{cases} \tag{3.10}$$

Výpočet složky S

$$S = \begin{cases} 0 & , C_{max} = 0, \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0. \end{cases} \tag{3.11}$$

Výpočet složky V

$$V = C_{max}. \tag{3.12}$$

Více o barevných modelech v [9].

3.1.3 Vzdálenost a okolí

Digitalizovaný obraz je matice pixelů. Je vhodné zde zavést pojem vzdálenost pixelů. Funkce D se nazývá vzdáleností, právě tehdy když

$$D(p,q) \geq 0, \text{ speciálně } D(p,p) = 0 \text{ (identita)}, \tag{3.13}$$

$$D(p,q) = D(q,p), \text{ (symetrie)}, \tag{3.14}$$

$$D(p,r) \leq D(p,q) + D(q,r), \text{ (trojúhelníková nerovnost)}. \tag{3.15}$$

Ve čtvercové mřížce pak můžeme definovat několik druhů vzdáleností.

Euklidovskou vzdálenost se vypočte jako

$$D_E((x,y),(h,k)) = \sqrt{(x-h)^2 + (y-k)^2}. \tag{3.16}$$

Tato metoda výpočtu vzdálenosti vrací neceločíselné výsledky. Výpočet odmocniny se v některých zejména real-time aplikacích může jevit jako zbytečně výpočetně složitý. Proto se ke vzdálenosti přistupuje i z jiných pohledů.

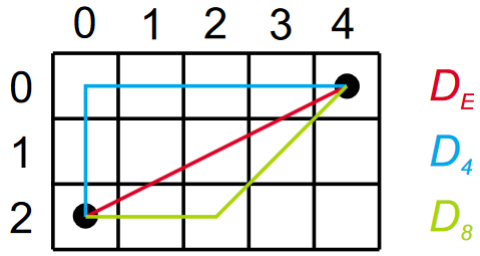
Manhattan distance neboli vzdálenost městských bloků vyjadřuje minimální počet operací do čtyř směrů k přechodu ze startovního do koncového bodu. Vypočte se jako

$$D_4((x,y),(h,k)) = |x-h| + |y-k|. \tag{3.17}$$

Chessboard distance neboli šachová vzdálenost vyjadřuje minimální počet operací, které by provedla figurka krále ve hře šachy. Vypočte se jako

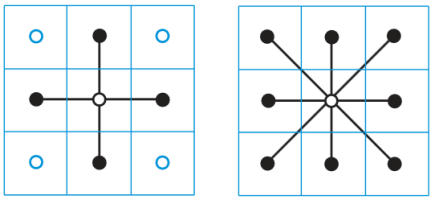
$$D_8((x,y),(h,k)) = \max(|x-h|, |y-k|). \tag{3.18}$$

Příklad všech 3 vzdáleností je na Obrázku 3.7.



Obrázek 3.7: Vzdálenosti pixelů [7].

Okolí je množina pixelů, složená z centrálního pixelu (tzv. reprezentativní bod) a jeho sousedů ve vzdálenosti 1. Ty pixely, které splňují tuto vlastnost, spolu sousedí. Nejčastějšími okolími jsou 4okolí a 8okolí. Jsou na Obrázku 3.8. Více o obraze v [8] a [7].



Obrázek 3.8: 4okolí a 8okolí [7].

3.2 Použité algoritmy

V této podkapitole budou obecně popsány algoritmy počítačového vidění, které byly v této práci použity.

3.2.1 Continuously Adaptive Mean Shift

Continuously Adaptive Mean Shift, také známý jako Cam Shift, je založen na algoritmu Mean Shift, který byl poprvé představen roku 1975 autory Fukunaga a Hostetler [10].

Algoritmus Mean Shift byl vyvinut jako metoda pro odhad hustoty pravděpodobnosti. V oblasti počítačového vidění jej ale lze též využít jako neparametrickou iterativní shlukovací metodu, která z náhodně vybraných bodů dokonverguje do lokálních maxim hustotní funkce, k čemuž nepotřebuje informace o počtu shluků a nevymezuje jejich tvar.

Mějme n bodů $x_i, i = 1, ..., n$ v d-dimenzionálním prostoru R^d, kernel $K(x)$ a rádius okénka h. Odhad hustoty v bodě x je pak

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right). \tag{3.19}$$

Pro radiálně symetrické kernely je možné použít profil kernelu $k(x)$

$$K(x) = c_{k,d} k(||x||^2), \tag{3.20}$$

kde $c_{k,d}$ je normalizační konstanta zajišťující, že integrál z $K(x)$ je 1. Lokální maxima (vrcholy) hustoty jsou body, v nichž je gradient hustoty nulový. Gradient odhadu hustoty

$$\begin{aligned} \nabla f(x) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x - x_i) g\left(\left\|\frac{x - x_i}{h}\right\|^2\right) \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)\right] \left[\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x\right], \end{aligned} \tag{3.21}$$

kde $g(s) = -k'(s)$. První člen je odhad hustoty. Druhým členem je posun střední hodnoty, tedy Mean Shift.

$$m_h(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x. \tag{3.22}$$

Tento vektor vždy ukazuje směr největšího růstu hustotní funkce.
Algoritmus Mean Shiftu se dá shrnout:

1. Volba kernelu, velikosti okénka h a počátečního bodu prohledávání x^0,
2. Výpočet Mean Shift vektoru $m_h(x^t)$,
3. Posunutí okénka $x^{t+1} = x^t + m_h(x^t)$,
4. Opakování bodu 2 a 3, dokud není dosaženo konvergence.

Algoritmus je možné využít při sledování. Na začátku se sestaví barevný model sledovaného objektu v podobě histogramu. Poté je sestavena pravděpodobnostní mapa $I(x, y)$ aktuálního snímku. Mean Shift nalezne vrchol této mapy s počátečním bodem vrcholu z minulého snímku.

Metoda Cam Shift byla představena autorem Garym Bradskim roku 1998 [11]. Jde o modifikaci metody Mean Shift ke sledování. Výhodou v úloze sledování je aktualizace okénka na příslušnou velikost a natočení. Nejprve je opět potřeba sestavit barevný model sledovaného objektu v podobě histogramu. Poté se provede jedna či více iterací Mean Shiftu. Tím je nalezen vrchol hustoty pravděpodobnosti. Dále je v okénku vypočítán nultý moment

$$M_{00} = \sum_x \sum_y I(x, y). \tag{3.23}$$

První momenty pro x a y

$$M_{10} = \sum_x \sum_y xI(x, y), \tag{3.24}$$

$$M_{01} = \sum_x \sum_y yI(x, y), \tag{3.25}$$

kde $I(x, y)$ je pravděpodobnost výskytu sledovaného objektu na pozici (x, y). Z těchto 3 hodnot je vypočítán centroid nového okénka

$$x_c = \frac{M_{10}}{M_{00}}, \tag{3.26}$$

$$y_c = \frac{M_{01}}{M_{00}}. \tag{3.27}$$

Orientace objektu se vypočte pomocí druhých momentů

$$M_{20} = \sum_x \sum_y x^2 I(x, y), \tag{3.28}$$

$$M_{02} = \sum_x \sum_y y^2 I(x, y). \tag{3.29}$$

Jsou zavedeny substituce

$$a = \frac{M_{20}}{M_{00}} - x_c^2, \tag{3.30}$$

$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c\right), \tag{3.31}$$

$$c = \frac{M_{02}}{M_{00}} - y_c^2. \tag{3.32}$$

Pak délka, šířka a orientace hlavní osy výsledné elipsy jsou

$$\xi = \sqrt{\frac{(a+b) + \sqrt{c^2 + (a-b)^2}}{2}}, \tag{3.33}$$

$$\eta = \sqrt{\frac{(a+b) - \sqrt{c^2 + (a-b)^2}}{2}}, \tag{3.34}$$

$$\theta = \frac{1}{2}\arctan\frac{b}{a-c}. \tag{3.35}$$

Poté je znovu vypočítán Mean Shift s novým okénkem, dokud není dosaženo požadované přesnosti. Algoritmus je vhodné použít pro sledování objektu barevně odlišného od prostředí, ve kterém se pohybuje. Díky dynamické změně sledovacího okénka vrací nejen jeho pozici, ale i velikost a natočení.

3.2.2 Prahování

Prahování je jednou z nejstarších, nejjednodušších a nejpoužívanějších metod segmentace. Tato metoda je velmi rychlá, a proto vhodná pro použití v real-time aplikacích. Používá se pro rozlišení objektů s odlišnými jasovými hodnotami obrazové funkce. Problémem zůstává volba prahu. Často není možné určit ho automaticky (například Otsuovým algoritmem [12]). Proto se práh určuje na základě experimentů.

Základní funkce prahování je

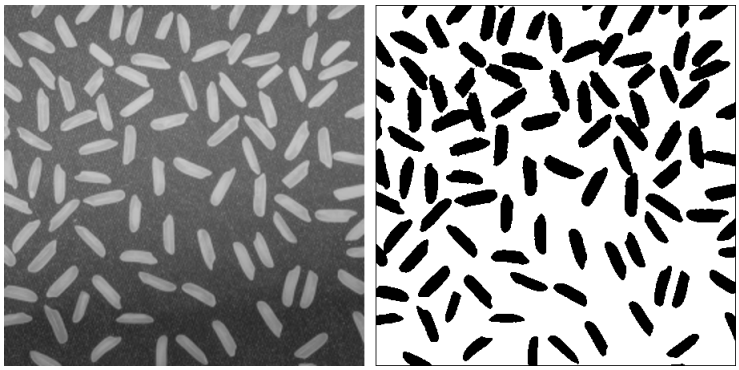
$$f(x,y) = A \text{ pro } f(x,y) > t, \tag{3.36}$$

$$f(x,y) = B \text{ pro } f(x,y) \leq t, \tag{3.37}$$

kde $f(x,y)$ hodnota obrazové funkce na pixelu (x,y), A a B jsou nově přiřazené hodnoty a t je práh. Rozlišuje se několik druhů prahování:

- Binární, $A = maximum$, $B = 0$,
- Binární inverzní, $A = 0$, $B = maximum$,
- Oříznutí, $A = t$, $B = 0$,
- Prahování na nulu, $A = f(x,y)$, $B = 0$,
- Prahování na nulu inverzní, $A = 0$, $B = f(x,y)$.

Výsledkem binárního prahování je binární obraz. Je to takový obraz, jehož obrazová funkce $f(x,y)$ nabývá pouze dvou hodnot - 1 nebo 0. Na Obrázku 3.9 je vidět, že jednotlivá zrnka rýže tvoří oblasti (černá) a zbytek obrazu je pozadí (bílá). Oblastí se rozumí množina souvislých bílých pixelů.

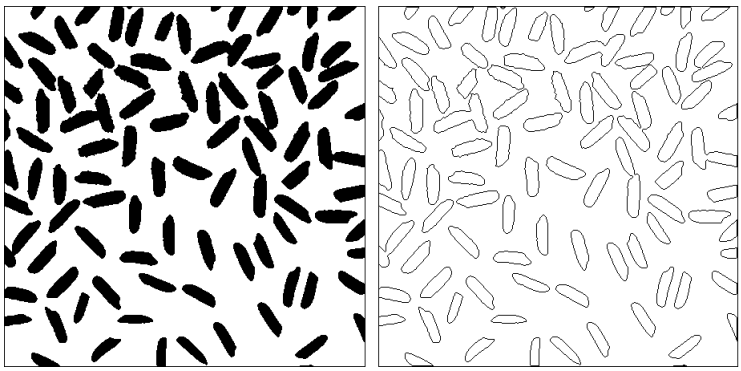


Obrázek 3.9: Binární prahování.

3.2.3 Hledání kontur

Kontury jsou obrysy kolem oblastí nalezených například v binárních obrazech. Kontury jsou jedním z možných způsobů popisu oblasti. Díky jejich znalosti se dají vypočítat další vlastnosti oblastí jako například obsah, obvod, orientace, podlouhlost, výstřednost či nekompaktnost. Jedna z metod, která dokáže kontury vyhledat, byla popsána roku 1985 autory S. Suzukim a K. Abem [13].

Hlavním principem hledání kontur je sledování hranice mezi oblastí a pozadím. Algoritmus funguje tak, že prochází vstupní binární obraz po řádcích a hledá bod náležící hranici. Pokud žádný takový bod není nalezen, algoritmus končí. Pokud takový bod je nalezen, je této hranici přiděleno identifikační číslo. Dále je mu přiděleno identifikační číslo hranice nadřazené, jestliže existuje (jestliže se například jedná o díru v oblasti). Od tohoto bodu se poté sleduje hranice. To znamená postupné prohledávání 8okolí a jestliže najde další hraniční bod, přiřadí mu stejná identifikační čísla a sledování hranice pokračuje tímto bodem. Výstupem jsou tedy sekvence souřadnic jednotlivých hranic a k nim informace o jejich hierarchii. Hierarchii je vhodné využít, jsou-li ve vstupním binární obraze například oblasti v dírách oblastí nadřazených. Na Obrázku 3.10 jsou vidět kontury jednotlivých zrnek rýže.



Obrázek 3.10: Nalezené kontury.

3.2.4 Mediánový filtr

Mediánový filtr je jednou z nelineárních technik filtrace šumu. Je vhodný především k odstranění náhodného šumu. Nevýhodou je, že porušuje tenké čáry a ostré

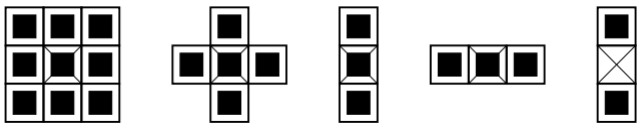
rohy. Také může změnit tvar nalezených oblastí. Filtr funguje tak, že pro každý pixel z obrazu vybere jeho čtvercové okolí o předem určené velikosti. Hodnoty z tohoto okolí jsou seřazeny a prostřední hodnota je vybrána a zapsána do aktuálně zpracovávaného pixelu. Na Obrázku 3.11 je vidět přidání umělého šumu a jeho následné odstranění mediánovým filtrem s velikostí okénka 3×3.

Obrázek 3.11: Filtrace mediánem [7].

3.2.5 Morfologické operace

Matematická morfologie je nástrojem pro úpravu binárních obrazů. Morfologické operace pomáhají k odstranění šumu, zjednodušení tvaru, rozdělení oblastí na více částí, spojení částí v jednu oblast, zvýraznění struktury tvaru (skeleton), zaplnění děr v nalezených oblastech a podobně.

Tyto operace pracují tak, že binární obraz procházejí pixel po pixelu a ke každému vždy přiloží tzv. strukturní element. Na Obrázku 3.12 jsou znázorněny obvykle používané příklady strukturních elementů. Přeškrtnutá buňka strukturního elementu se nazývá aktuální nebo reprezentativní bod. Tento bod při přiložení strukturního elementu vždy překrývá aktuálně zpracovávaný pixel. Po přiložení struk-



Obrázek 3.12: Strukturní elementy [7].

turního elementu se pro aktuální pixel provede relace mezi obrazem a strukturním elementem. Výsledek této relace se zapíše do výsledného binárního obrazu. Základní morfologické operace jsou eroze a dilatace.

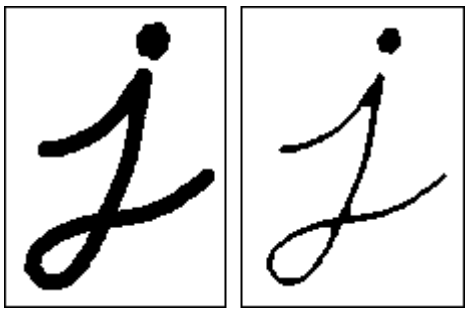
Eroze je operací, která se používá ke zmenšení oblastí v binárním obraze, kompletní odstranění malých oblastí či rozpojení dvou spojených oblastí. Strukturní element je přikládán postupně ke každému pixelu v obraze. Jestliže se všechny pixely obrazu na přiloženém místě shodují s pixely strukturního elementu, aktuální pixel je zachován. Pokud tomu tak není, aktuální pixel je erodován neboli vynulován a začerněn.

Matematicky se jedná o Minkowského rozdíl a zapisuje se jako

$$X \ominus B = \bigcap_{b \in B} X_{-b} \tag{3.38}$$

nebo

$$X \ominus B = \{p \in E^2 : p = x + b \in X, \forall b \in B\}. \tag{3.39}$$



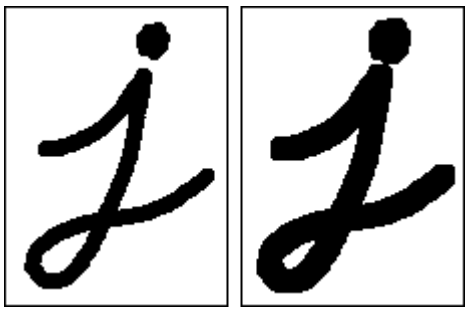
Obrázek 3.13: Ukázka eroze [14].

Dilatace je duální (nikoliv však inverzní) operací k erozi. Používá se ke zvětšení oblastí v binárním obraze, spojení několika malých oblastí nebo zaplnění děr. Strukturní element je přikládán postupně ke každému pixelu v obraze. Jestliže se aktuální pixel obrazu shoduje s počátečním pixelem strukturního elementu, pak se strukturní element "obtiskne"do výsledného obrazu. Pokud se neshodují, neděje se nic.
Matematicky se jedná o Minkowského součet a zapisuje se jako

$$X \oplus B = \bigcup_{b \in B} X_b \tag{3.40}$$

nebo

$$X \oplus B = \{p \in E^2 : p = x + b,\ x \in X, b \in B\}. \tag{3.41}$$

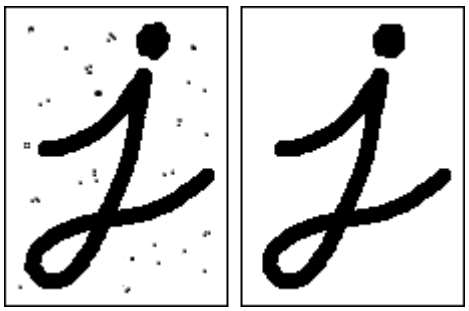


Obrázek 3.14: Ukázka dilatace [14].

Další morfologické operace jsou kombinací eroze a dilatace, nejčastějšími kombinacemi jsou otevření a uzavření.

Otevření je operace, která provede nejprve erozi a pak dilataci. Je vhodná pro odstranění šumu. Eroze odstraní malé oblasti a jednu vnější vrstvu velkých oblastí. Poté je aplikována dilatace a vnější vrstva velkých objektů je navrácena.
Matematicky se zapisuje jako

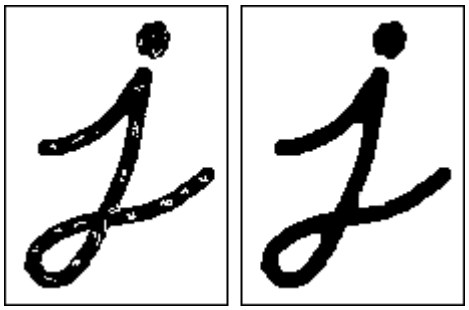
$$X \circ B = (X \ominus B) \oplus B. \tag{3.42}$$



Obrázek 3.15: Ukázka otevření [14].

Uzavření je operace, která provede nejprve dilataci a pak erozi. Je vhodná pro odstranění děr v oblastech. Dilatace díry zaplní a zvětší oblasti o jednu vrstvu. Poté je aplikována eroze a vrstva navíc zmizí.
Matematicky se zapisuje jako

$$X \bullet B = (X \oplus B) \ominus B. \tag{3.43}$$



Obrázek 3.16: Ukázka uzavření [14].

Pomocí eroze je též možné získat konturu oblasti. Jestliže byl použit strukturní element o plném 8okolí, pak budou oblasti zmenšeny o jednu vrstvu. Odečteme-li vzniklý binární obraz od původního binárního obrazu, získáme její hranici. Více o morfologických operací v [8], [7] a [14].

3.2.6 Graph Cut

Graph Cut neboli řez grafem je jednou z metod segmentace objektů. Byl představen v roce 2001 autory Y. Y. Boykov a M. P. Jolly [15]. Princip spočívá v tom,

že se s obrazem pracuje jako s grafem. Graf je uspořádaná dvojice

$$G = (V, E), \tag{3.44}$$

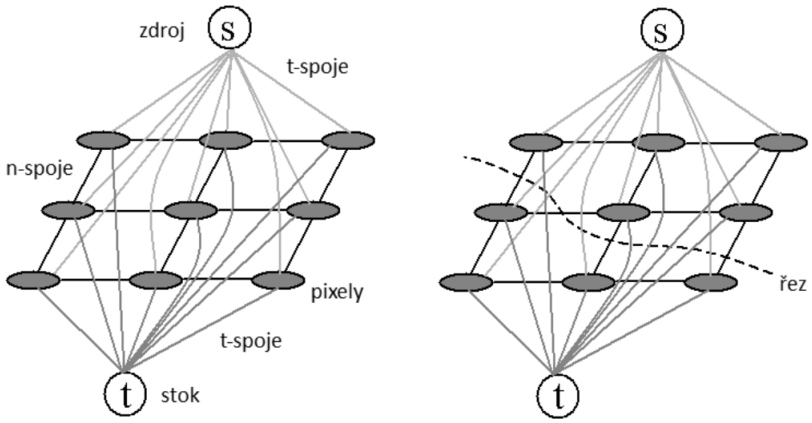
kde V je množina vrcholů (vertices) grafu a E je množina hran (edges) grafu, které spojují vrcholy. Tento graf je ohodnocený, to znamená, že každý vrchol a každá hrana mají přiřazenou svou hodnotu - váhu. Vrcholy grafu jsou vytvořeny z pixelů obrazu, graf tedy bude mít stejnou velikost jako obraz. Do grafu se přidávají další dva vrcholy zdroj s (source) a stok t (terminal). Množina všech vrcholů je pak $V = P \cup \{s, t\}$, kde P je množina vrcholů představující pixely. Množina všech dvojic sousedících pixelů $\{p, q\}$ je N. Hrany v grafu vzniknou mezi sousedními vrcholy, přičemž se uvažuje 4okolí. Hrany mezi vrcholy se nazývají n-spoje (podle anglického slova neighbour). Hrany mezi vrcholy a zdrojem s nebo stokem t se nazývají t-spoje.

Řez grafem znamená rozdělení množiny vrcholů V na podmnožiny, které jsou každá propojeny s jedním z uzlů s nebo t

$$S \subset V, s \in S, \tag{3.45}$$

$$T = V - S, t \in T. \tag{3.46}$$

Řez je množina hran, které před řezem spojovaly právě takové dva vrcholy, že jeden vždy patřil do množiny S a druhý do T. Jinými slovy to znamená, že neexistuje cesta mezi vrcholy s a t, protože hrany mezi množinami S a T byly odstraněny řezem.



Obrázek 3.17: Segmentace algoritmem Graph Cut [16].

Pro nalezení řezu je definována segmentační energie E a binární vektor A, který označuje jednotlivé pixely čísly 1 nebo 0 podle toho, zda náleží k objektu či k pozadí. Vektor A tedy určuje výsledek segmentace. Segmentační energie je závislá na podobě vektoru A

$$E(A) = \lambda \cdot R(A) + B(A), \tag{3.47}$$

kde

$$R(A) = \sum_{p \in P} R_p(A_p), \tag{3.48}$$

$$B(A) = \sum_{\{p,q\} \in N} B_{p,q} \cdot \delta_{A_p \neq A_q} \tag{3.49}$$

a

$$\delta_{A_p \neq A_q} = \begin{cases} 1 & \text{pokud } A_p \neq A_q, \\ 0 & \text{pokud } A_p = A_q. \end{cases} \tag{3.50}$$

$R(A)$ se nazývá region term a představuje energetickou cenu přiřazení pixelu do jedné ze dvou výsledných množin. $B(A)$ se nazývá boundary term a představuje energetickou cenu hrany dvou sousedních pixelů. $\lambda \geq 0$ je koeficient významnosti $R(A)$ pro výslednou segmentaci. Hodnoty $R_p(A_p)$ popisují konkrétní postihy energie při přiřazení pixelu p do množiny objektu, když $A_p = 1$, či pozadí, když $A_p = 0$. Používá se například záporný logaritmus pravděpodobností s jakou pixel p s danou intenzitou I_p patří k objektu nebo pozadí

$$R_p(A_p = 1) = -\ln P(I_p | A_p = 1), \tag{3.51}$$

$$R_p(A_p = 0) = -\ln P(I_p | A_p = 0). \tag{3.52}$$

Pak platí, že čím nižší pravděpodobnost, tím větší energie. Hodnoty $B_{p,q} \geq 0$ popisují konkrétní postihy nesouvislosti mezi pixely p a q. $B(A)$ roste pouze v případě, že p a q patří do jiného segmentu. Používá se exponenciální funkce

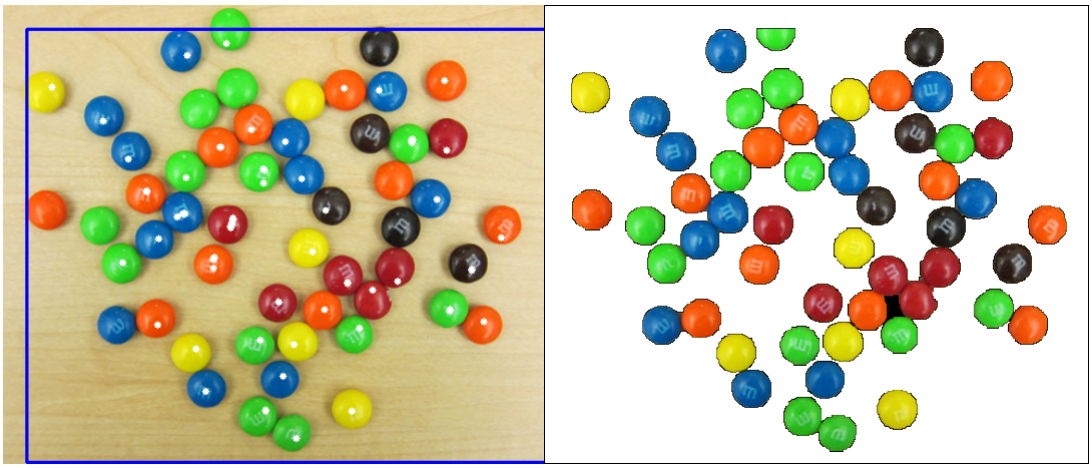
$$B_{p,q} = \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{\|p, q\|}, \tag{3.53}$$

kde σ^2 představuje celkový rozptyl jasu v obraze a $\|p, q\|$ je vzdálenost mezi pixely. Pro například jasově velmi podobné pixely je hodnota $B_{p,q}$ obvykle veliká. Naopak, pokud si pixely nejsou podobné, je hodnota menší.

Cílem algoritmu je najít takový vektor A, který minimalizuje kritérium energie. To je problém hledání minimálního řezu, respektive maximálního toku grafem. Max-flow/Min-cut problém řeší například Ford-Fulkersonův algoritmus. Tento algoritmus se zjednodušeně dá shrnout do několika kroků:

1. Nastavení nulového toku ve všech hranách,
2. Nalezení cesty mezi vrcholy s a t (tzv. augmenting path), která má ještě kapacitu, pokud taková cesta už neexistuje, konec,
3. Odečtení minimální kapacity cesty z kroku 2 ze všech jejích hran, respektive zvýšení jejich toku,
4. Pokračování krokem 2.

Pro hledání cesty v kroku 2 mohou být použity algoritmy Breadth-first Search nebo Depth-first Search. Postupným odečítáním kapacity v hranách se některé hrany zruší. Tyto hrany jsou součástí minimálního řezu grafem. Jak již bylo řečeno, minimální řez minimalizuje energetické kritérium segmentace. Výsledkem je vektor A označující, které pixely p patří do množiny objektu a které do množiny pozadí. Na Obrázku 3.18 je ukázka segmentace lentilek v barevném obraze.
Více informací o algoritmu Graph Cut v [15] nebo [8]. Více informací o Ford-Fulkersonově algoritmu v [18].



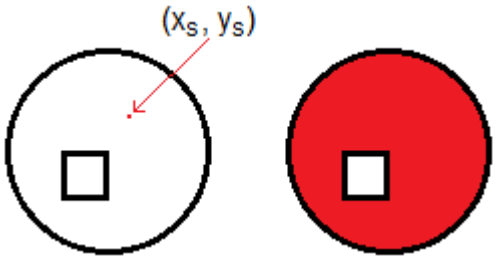
Obrázek 3.18: Ukázka funkce algoritmu Graph Cut [17].

3.2.7 Flood Fill

Algoritmus Flood Fill vyplňuje spojitelné pixely začínaje z uživatelem specifikovaného bodu. Spojitelnost je zde chápána jako jasová podobnost startovního bodu, vymezená jasovým intervalem kolem jasu startovního bodu. Uživatel tedy musí tento interval definovat volbou horního a spodního prahu. Pixel (x, y) bude patřit do nové vybarvené oblasti, pokud

$$f(x_s, y_s) - lb \leq f(x, y) \leq f(x_s, y_s) + ub, \tag{3.54}$$

kde (x_s, y_s) je startovní bod, $f(x, y)$ je hodnota obrazové funkce v bodě (x, y), lb je spodní práh a ub je horní práh. Algoritmus takto prozkoumává všechny pixely kolem startovního bodu, dokud již neexistují další cesty, kterými by se oblast mohla rozrůstat. Algoritmus je vhodné využít v případě, je-li potřeba označit nějakou oblast. Na Obrázku 3.19 je ukázka obarvení všech bílých pixelů uvnitř kruhu, které spolu sousedí.



Obrázek 3.19: Ukázka funkce algoritmu Flood Fill.

Kapitola 4

Klasifikace

Klasifikace je druh problému strojového učení a umělé inteligence, které má za úkol určit, do které kategorie neboli třídy dané pozorování patří. Metody klasifikace nacházejí využití v mnoha oblastech, například v lékařské diagnostice, počítačovém vidění, rozpoznávání řeči, robotice a podobně. Třída je souhrnné pojmenování pro množinu předmětů či jevů, které se vyznačují určitými společnými rysy. Třídění lze provádět z různých hledisek závisle na tom, jaký problém je řešen. Aby bylo možno předměty nebo jevy klasifikovat, je potřeba je nejdříve vhodně popsat. Je tedy vybrána jistá množina základních vlastností předmětu a tyto se složí do struktury podle metody klasifikace. Tyto metody lze rozdělit na dvě kategorie.

První kategorií jsou metody příznakové. Popisy předmětů mají u těchto metod číselný charakter a nazývají se příznaky. Většinou se měří více nežli jeden příznak předmětu, výsledkem měření je tedy uspořádaný vektor, jehož jednotlivé dimenze jsou tvořeny příznaky. Tento vektor se též nazývá obraz a často se podle toho používá termín rozpoznávání obrazů.

Druhou kategorií jsou metody strukturální. Ty popisují předměty jako řetězce symbolů zvaných primitiva. Primitiva jsou jednoduché nečíselné popisy elementárních vlastností předmětu. V této práci nebyly strukturální metody použity.

Výběr příznaků je v podstatě expertní záležitostí, to znamená, že by je konstruktér klasifikátoru měl volit na základě zkušenosti nebo důkladné analýzy problému. Příznaky se označují $x_1, x_2, ..., x_n$ a tvoří dohromady vektor $x^T = [x_1, x_2, ..., x_n]$. Příznakový prostor X je pak n-rozměrný a geometrickou interpretaci příznakového vektoru v takovém prostoru si lze představit jako bod. Při klasifikaci do R tříd se indikátory jednotlivých tříd značí $\omega_1, \omega_2, ..., \omega_R$. Klasifikace znamená přiřazení každému bodu $x \in X$ jeden z indikátorů tříd ω_r.

Lze předpokládat, že klasifikátor bude tím přesnější, čím lépe jsou příznaky vybrány. Při jejich vhodném výběru pak v příznakovém prostoru X jednotlivým třídám odpovídají shluky příznakových vektorů. Podobnost těchto vektorů v tomto prostoru se dá vyjádřit geometrickou vzdálenost mezi nimi. Rozhodovací pravidla rozdělují prostor X na R vzájemně disjunktních podprostorů, které poté odpovídají jednotlivým třídám. Rozhodovací pravidlo má obecně tvar

$$\omega = d(x, q), \tag{4.1}$$

kde $\omega \in \{\omega_1, \omega_2, ..., \omega_R\}$ je výsledný indikátor třídy přiřazené vstupnímu příznakovému vektoru x. Je to funkce vstupu x a aktuálního nastavení parametrů klasifikátoru q. Pro každý podprostor pak existuje takové pravidlo nabývající hodnoty

odpovídající třídy.

Klíčovým problémem je určení typu klasifikátoru, tedy konkrétního tvaru rozhodovacího pravidla pro jednotlivé třídy a jeho nastavení q. Nastavení klasifikátoru je věc učení. Je-li k dispozici množina trénovacích příznakových vektorů a zároveň informace o jejich správné klasifikaci, pak je tento problém řešen metodami učení s učitelem. Pokud chybí informace o správné klasifikaci, pak je tento problém řešen metodami učení bez učitele neboli metodami shlukové analýzy. V této práci bylo použito výhradně učení s učitelem. Více o klasifikaci v [19].

4.1 Náhodný rozhodovací les

Náhodný rozhodovací les neboli Random Decision Forest (RDF) je kolekcí rozhodovacích stromů.

4.1.1 Rozhodovací strom

Rozhodovací strom je jedním z nelineárních rozhodovacích systémů, jímž je příznakový prostor rozdělen na hyperobdélníkové subprostory odpovídající jednotlivým třídám. Klasifikátorem je orientovaný graf ve tvaru stromu, jehož jednotlivé vrcholy nebo také uzly jsou tvořeny otázkami typu ANO $\times$ NE. Z každého uzlu tedy vedou dvě větve, jedná se proto o binární strom. Otázky neboli rozhodovací pravidla porovnávají dimenze x_i příznakového vektoru s nastavenými prahy α, obecně

$$x_i \leq \alpha. \tag{4.2}$$

Rozhodovací pravidla však nemusí porovnávat jen jednu dimenzi příznakového vektoru. Mohou též mít tvar

$$\sum_i c_i x_i \leq \alpha, \tag{4.3}$$

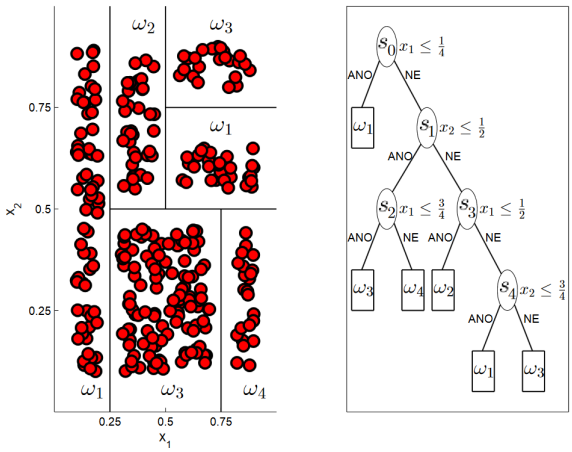
kde i jsou jednotlivé dimenze příznakového vektoru, c_i jsou váhy každého příznaku x_i a α je nastavený práh. Rozdělující nadroviny pak nejsou nutně rovnoběžné s osami příznakového prostoru jako v předchozím případě. Taková pravidla mohou mít za následek jeho přesnějšího rozdělení, nicméně konstrukce stromů se tím komplikuje. Tato pravidla proto nebudou dále uvažována.

Je-li T trénovací množina a T_s její podmnožina, která je svázána s uzlem s, pak dělící otázka v uzlu s rozdělí množinu T_s do dvou následnických subsouborů T_{s-A} a T_{s-N} spojených s následnickými uzly s_A a s_N. Pro tyto subsoubory platí

$$T_{s-A} \cap T_{s-N} = \emptyset \text{ a } T_{s-A} \cup T_{s-N} = T_s. \tag{4.4}$$

Příklad rozdělení 2D příznakového prostoru rozhodovacím stromem je na Obrázku 4.1.

Při generování rozhodovacího stromu je potřeba konkrétně specifikovat rozhodovací pravidla jednotlivých uzlů. To je buď prací expertů, kteří stanoví prahy a dimenze, nebo je možné tato pravidla určit automaticky na základě analýzy dat. Při automatickém generování se často používá snižování entropie. Entropie v uzlu s je tím větší, čím více zástupců různých tříd je v množině T_s a naopak nulová



Obrázek 4.1: Rozdělený příznakový prostor a rozhodovací strom [17].

v případě, že množina T_s obsahuje zástupce jen jedné třídy. Entropie se vypočte jako

$$I(s) = -\sum_{r=1}^{R} P(x \in \omega_r|s) log_2 P(x \in \omega_r|s), \tag{4.5}$$

kde $P(x \in \omega_r|s)$ je pravděpodobnost jevu, že vektory x uvnitř podmnožiny T_s patří do třídy ω_r pro $r = 1, 2, ...R$. Tato pravděpodobnost se vypočte jako relativní četnost počtu vektorů v T_s náležící k třídě ω_r ku počtu všech vektorů v T_s.

Cílem je určit takovou otázku, která množinu T_s rozdělí tak, aby její vzniklé podmnožiny T_{s-A} a T_{s-N} obsahovaly každá menší počet tříd než T_s. Tato otázka by měla mít za následek největší pokles entropie a tedy i optimální rozdělení množiny. Změna entropie se vypočte jako

$$\Delta I(s) = I(s) - \frac{N_{s-A}}{N_s} I(s_A) - \frac{N_{s-N}}{N_s} I(s_N), \tag{4.6}$$

kde $I(s), I(s_A)$ a $I(s_N)$ jsou entropie svázané s uzly s, s_A a s_N, přičemž s_A a s_N jsou následnické uzly uzlu s. N_{s-A} a N_{s-N} jsou počty vektorů svázaných s uzly s_A a s_N a $N_{s-A} + N_{s-N} = N_s$.

Generování stromu je ukončeno například, pokud pokles entropie bude menší než předem definovaná prahová hodnota anebo velikost množiny v nějakém uzlu je menší než předem definovaný počet. Též je možné další dělení nějakého uzlu ukončit, pokud množina svázaná s tímto uzlem obsahuje zástupce pouze jedné třídy.

Všechny koncové uzly stromu se nazývají listy stromu. Tyto listy rozhodují o tom, jakou třídu přiřadí klasifikovanému vektoru. Listy přiřazují právě takovou třídu, která je nejčastěji zastoupena v množině trénovacích dat spojených s těmito uzly. Listový vrchol s_L tedy klasifikuje do takové třídy ω_{r^*}, k níž náleží většina vektorů z množiny svázané s tímto uzlem T_{s-L}

$$\omega_{r^*} = \arg\max_r P(x \in \omega_r|s_L). \tag{4.7}$$

Klasifikace neznámého vektoru y probíhá tak, že je tento vektor přiveden na vrchol neboli kořen stromu. V kořeni odpoví na otázku a v závislosti na odpovědi se propadne do odpovídajícího následnického uzlu. Toto se opakuje, dokud vektor

nedocestuje do nějakého z listových uzlů, kde je zařazen podle příslušnosti daného listu.

4.1.2 Rozhodovací les

Nevýhodou klasifikace jedním rozhodovacím stromem je právě princip sestavování stromu. Změní-li se trénovací množina, dojde též ke změně tvaru rozhodovacího stromu. To může vést k nárůstu chyby klasifikace. Takový problém lze odstranit zapojením více stromů, tedy vytvoření rozhodovacího lesa.

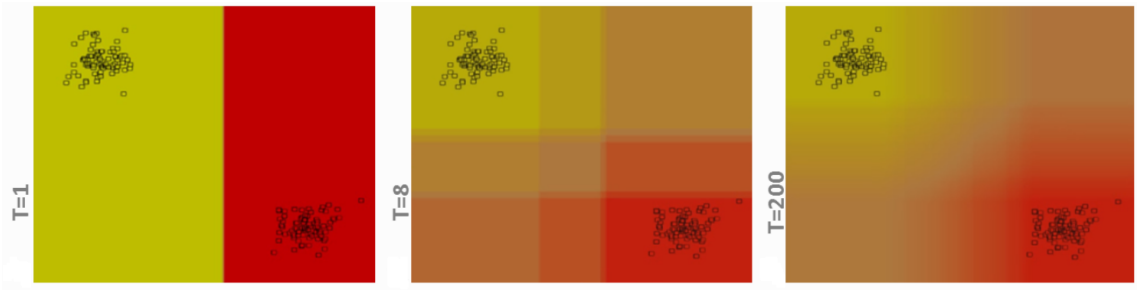
Trénovací množina T obsahující N vektorů je algoritmem bootstrap aggregating rozdělena na n množin $T_{(n)}$ obsahujících $N_{(n)}$ vektorů, přičemž $N_{(n)} \leq N$. Množiny $T_{(n)}$ jsou vytvořeny rovnoměrným náhodným výběrem vektorů z celé množiny T. Náhodný výběr zde probíhá s vracením, to znamená, že některé vektory se v jednotlivých množinách mohou opakovat, bude tomu tak přibližně pro třetinu vektorů. Pro každou množinu $T_{(n)}$ je zkonstruován rozhodovací strom způsobem popsaným výše. Neznámý vektor y je přiveden na vstup každého stromu. Všechny tyto stromy klasifikují y do některé ze tříd ω_r pro $r = 1, 2, ...R$. Výsledek klasifikace celého lesa je pak nejčastější výsledek jednotlivých stromů. Pro případ binární klasifikace (klasifikace do dvou tříd) lze vypočítat pravděpodobnostní mapu, která popíše každý bod příznakového prostoru pravděpodobností náležitosti tohoto bodu do jedné ze tříd. Je-li x bodem v příznakovém prostoru, pak pravděpodobnost třídy ω_0 bude

$$P(\omega_0|x) = \frac{n_0}{n}, \tag{4.8}$$

kde n_0 označuje počet stromů lesa, které klasifikovaly x do třídy ω_0 a n celkový počet stromů v lese. Pravděpodobnost druhé třídy ω_1 pak bude

$$P(\omega_1|x) = \frac{n_1}{n} = 1 - P(\omega_0|x). \tag{4.9}$$

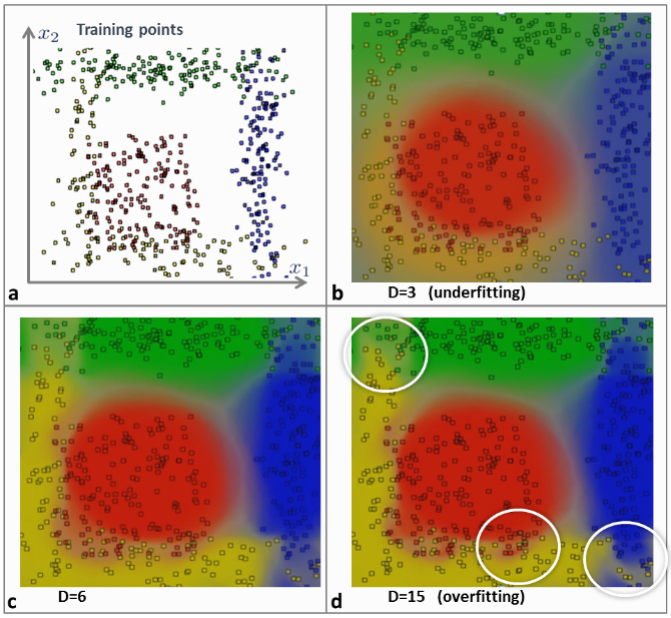
Výhodou rozhodovacího lesa nad rozhodovacím stromem je ten, že rozhodovací strom má mezi jednotlivými třídami ostrá rozhodovací pravidla daná otázkami, zatímco v lese je mezi třídami změna pozvolná. Na Obrázku 4.2 jsou příklady rozdělení příznakového prostoru do dvou tříd - žluté a červené. Pro jeden rozhodovací strom o jednom rozhodovacím pravidlu je prostor rozdělen jednou přímkou, v dalších dvou příkladech si lze všimnout skutečnosti popsané výše.



Obrázek 4.2: Vliv počtu stromů na rozdělení příznakového prostoru [20].

Dalším parametrem vedle počtu stromů je jejich hloubka. Větší hloubka stromů

znamená více uzlů čili více rozhodovacích pravidel. Tím pádem bude jeden strom dělit příznakový prostor více nadrovinami. Na Obrázku 4.3 si lze všimnout, že při volbě malé hloubky stromů může dojít k podtrénování. Naopak při příliš velké hloubce může dojít k přetrénování. Je tedy potřeba volit hloubku tak, aby les generalizoval problém a shluky se tak tvořily kompaktní.



Obrázek 4.3: Vliv hloubky stromů na rozdělení příznakového prostoru [20].

4.1.3 Náhodný rozhodovací les

Nevýhodou klasifikace obyčejným rozhodovacím lesem je, že se často může stát, že se stromy přetrénují na svou trénovací množinu. To je důsledkem porovnávání jednotlivých dimenzí příznakových vektorů stromem popořadě. Tím se zvyšuje riziko, že vzniklé stromy budou korelované a chyba klasifikace bude větší. K potlačení tohoto jevu byly v roce 2001 představeny náhodné rozhodovací lesy v práci autorů Leo Breiman a Adele Cutler [21].

Stromy jsou generovány podobně jako v případě obyčejného rozhodovacího lesa. Rozdělení trénovací množiny T se rozdělí algoritmem bootstrap aggregating stejně jako v předchozím případě. Rozdílná je konstrukce stromů. Je zvolen parametr m, jehož velikost je menší než dimenze příznakových vektorů. Rozhodovací pravidla jsou pak konstruována z m náhodně vybraných dimenzí, za jejichž použití bylo dosaženo nejlepších výsledků. Takto jsou zkonstruovány všechny stromy. Klasifikace probíhá stejně jako v případě obyčejného rozhodovacího lesa. Více o stromech v [21], [17], [20] a [19].

4.2 Hodnocení kvality klasifikátoru

Přesnost klasifikátoru vyjadřuje míru jeho schopnosti správně klasifikovat neznámá data. To jsou taková data, která nebyla použita při jeho trénování. Je-li

tedy k dispozici sada trénovacích dat, je vhodné ji před trénováním rozdělit na dvě množiny. První množina, zpravidla větší, se nazývá trénovací a je použita k natrénování vybraného druhu klasifikátoru odpovídajícím algoritmem. Druhá množina se nazývá testovací a obsahuje výše zmíněná při trénování neviděná data. Na této množině se testuje přesnost, respektive chyba klasifikace.

4.2.1 Matice chyb

Matice chyb (confusion matrix) je pro binární klasifikaci (klasifikaci do dvou tříd) tabulka o dvou řádcích a dvou sloupcích. Obsahuje počty true positives (TP), false positives (FP), false negatives (FN) a true negatives (TN). Dobrým příkladem pro vysvětlení těchto čísel je testování nemoci na lidech. Každý člověk projde testem, zda je nebo není nemocen. Test může vyjít pozitivně (klasifikuje osobu jako nemocnou) nebo negativně (klasifikuje osobu jako zdravou). Výsledky tohoto testu mohou být:

- True positive: Nemocný člověk s pozitivním testem,
- False positive: Zdravý člověk s pozitivním testem,
- True negative: Zdravý člověk s negativním testem,
- False negative: Nemocný člověk s negativní testem.

V Tabulce 4.1 je znázorněna matice chyb pro případ klasifikace do dvou tříd. V ideálním případě je matice chyb diagonální.

		Skutečnost	
		1	0
Klasifikováno	1	True positive	False positive
	0	False negative	True negative

Tabulka 4.1: Matice chyb.

Na základě těchto údajů se dají vypočítat různé míry přesnosti klasifikátoru.

Precision nebo též positive predictive value, česky přesnost, popisuje pravděpodobnost, že pozitivně klasifikované jevy jsou relevantní. Vypočítá se jako

$$\text{Precision} = \frac{\text{True positives}}{\text{počet pozitivně klasifikovaných}} = \frac{TP}{TP + FP}. \tag{4.10}$$

Recall nebo též sensitivity nebo true positive rate, česky výtěžnost, popisuje pravděpodobnost, že relevantní jevy jsou klasifikované pozitivně. Vypočítá se jako

$$\text{Recall} = \frac{\text{True positives}}{\text{počet skutečně pozitivních}} = \frac{TP}{TP + FN}. \tag{4.11}$$

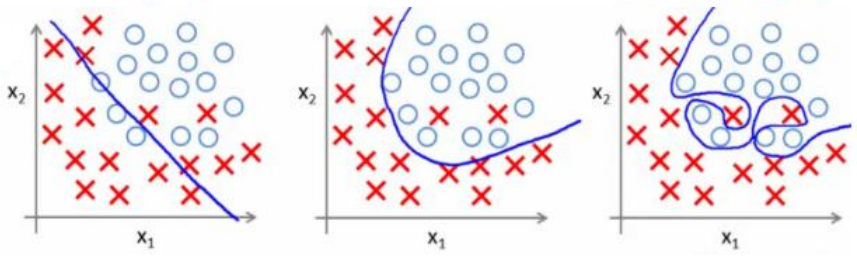
F1-score kombinuje precision a recall do jedné hodnoty a popisuje jejich harmonický průměr. Vypočítá se jako

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.12}$$

Dalšími mírami hodnocení klasifikátoru jsou například: accuracy, true negative rate, negative, false positive rate, false negative rate, false discovery rate.
Více o tomto tématu v [22].

4.2.2 Podtrénování a přetrénování

Při učení klasifikátoru může dojít k podtrénování (underfitting) nebo k přetrénování (overfitting). Oba tyto jevy vedou ke snížení přesnosti klasifikace na nových datech. Podtrénování je jev, při kterém klasifikátor neuspokojivě rozděluje jednotlivé třídy v trénovacích datech. Může být způsobeno například příliš jednoduchou strukturou klasifikátoru, kterou je poté vhodné vybrat složitější. Přetrénování je opačným jevem, při kterém dochází k tomu, že složitost klasifikátoru je natolik vysoká, že dokáže oddělit zástupce obou tříd v trénovacích datech s libovolně malou chybou. Na datech testovacích však většinou bývá nepřesný, protože ztrácí schopnost generalizace problému. Dá se vyřešit buď přidáním dalších trénovacích dat, nebo snížením složitosti klasifikátoru. Na Obrázku 4.4 je vidět podtrénovaný, normální a přetrénovaný příklad rozdělení do 2 tříd. V prvním případě dochází k velké klasifikační chybě na trénovacích datech vlivem malé flexibilitě rozdělovací křivky. V případě druhém jsou vidět 2 chyby klasifikace, nicméně rozdělovací křivka vhodně zobecňuje daný problém. Ve třetím případě je sice 100procentní úspěšnost na trénovacích datech, nicméně klasifikátor tak zřejmě ztrácí přesnost na testovací sadě.

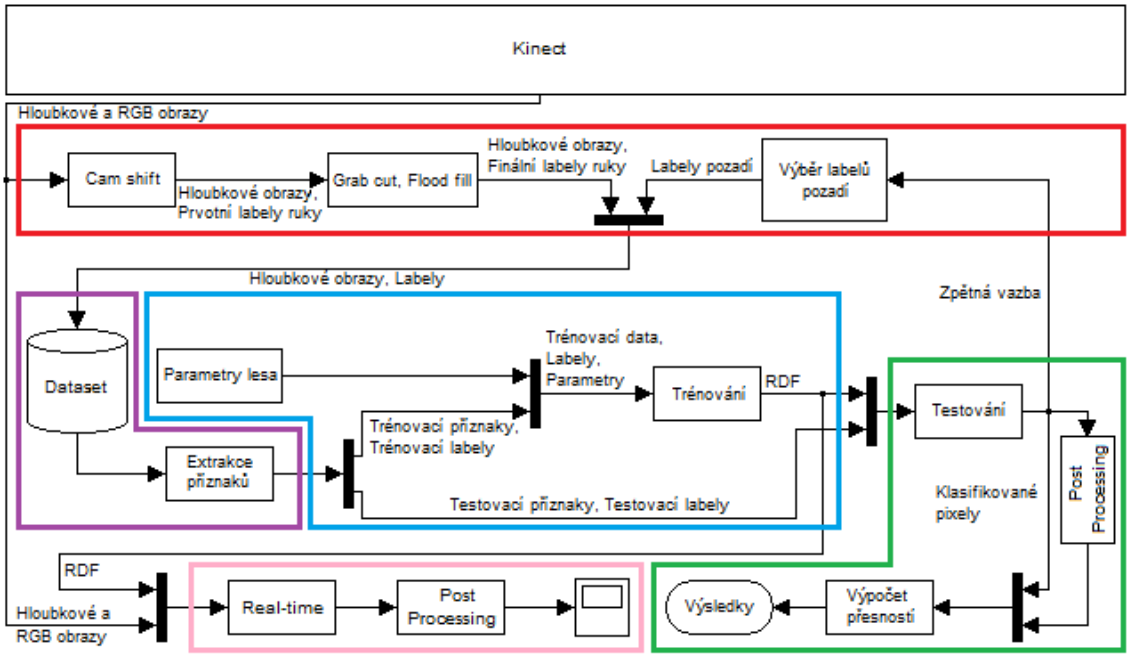


Obrázek 4.4: Podtrénování a přetrénování v problému klasifikace.

Kapitola 5

Sledování ruky v hloubkové mapě

Cílem této práce bylo sledovat ruku v hloubkovém obraze. Byl zvolen stejný postup jako v článku [1]. Jde o problém klasifikace pixelů hloubkových obrazů náhodným rozhodovacím lesem (RDF) do dvou tříd. První třídou je ruka, druhou je pozadí. Celý postup řešení by se dal shrnout následujícím schématem.



Obrázek 5.1: Schéma postupu řešení úlohy.

V první řadě byl vytvořena obsáhlá množina (dataset) hloubkových obrazů a k nim označení (labelů) pixelů, které byly součástí ruky, tedy informace učitele. Pixely, které nebyly součástí ruky, byly v prvních experimentech vybrány náhodně. Po testech s takovým výběrem se ukázalo, že je tyto pixely potřeba vybrat ručně. Tato část je v Obrázku 5.1 označena červeně.

Dalším krokem byla extrakce příznakových vektorů z datasetu. Tato část je označena fialově.

Po extrakci příznakových vektorů byl dataset vždy náhodně rozdělen na trénovací a testovací množinu v konstantním stanoveném poměru. Z trénovací množiny se natrénoval klasifikátor RDF vždy s rozdílnými parametry (hloubka a počet stromů).

Tato část je označena modře.

Po trénování byl klasifikátor otestován na testovací množině a vypočtena jeho úspěšnost. Výsledky byly ukládány pro každý klasifikátor pro klasifikované pixely před i po post processingu. Tato část je označena zeleně.

Po ukončení testování může být jeden klasifikátor aplikován na hloubkové obrazy v reálném čase. Tato část je označena růžově.

V této kapitole bude podrobně popsána každá část postupu. Všechny programy byly napsány buď v C++ nebo Pythonu za podpory knihovny OpenCV.

OpenCV (Open Source Computer Vision) je multiplatformní knihovna pro práci s obrazem dnes vyvíjená a spravovaná společností Itseez. Vznikla v roce 1999 jako projekt ruské pobočky firmy Intel. Poprvé byla představena v roce 2000 na konferenci IEEE Conference on Computer Vision and Pattern Recognition. Verze 1.0 byla vydána v roce 2006, v roce 2009 pak verze 2.0 a v roce 2015 verze 3.0. V této práci byla použita verze 2.4.10. Knihovna obsahuje nejen funkce počítačového vidění, ale i funkce strojového učení.

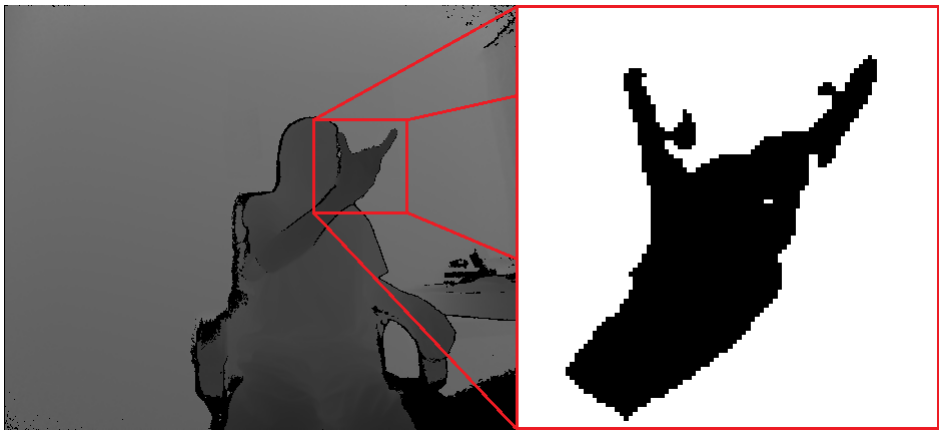
5.1 Vytvoření datasetu

Jak už bylo zmíněno v úvodu kapitoly, pro trénování náhodného rozhodovacího lesa bylo potřeba vytvořit dataset hloubkových obrazů. Jelikož se jedná o učení s učitelem, bylo též nutné v těchto obrazech najít pixely, které byly součástí hledaného objektu, ruky.

5.1.1 Získávání hloubkových obrazů a prvotních labelů ruky

Focení trénovacích obrazů probíhalo následovně. Nejprve jsem si z důvodu barevného odlišení nabarvil levou ruku na červeno lihovým fixem. Poté jsem spustil program, který četl hloubkové a RGB obrazy z Kinectu. Prvním krokem programu byla konverze barevného obrazu do HSV (viz Podkapitola 3.1.2). V něm jsem pro algoritmus Cam Shift (viz Podkapitola 3.2.1) obdélníkem označil svou obarvenou ruku. Pro ni byl stvořen barevný model v podobě histogramu. Algoritmus Cam Shift takto ruku sledoval na základě barevného odlišení a vracel jednak vypočtené prohledávací okénko a jednak vypočtenou pravděpodobnostní mapu. Prahováním (viz Podkapitola 3.2.2) pravděpodobnostní mapy jsem získal nejpravděpodobnější oblasti, kde by se ruka v barevném obraze mohla nacházet. Pomocí hledání kontur (viz Podkapitola 3.2.3) a výběru největší oblasti jsem vyfiltroval šum. Prvotními labely se tedy staly právě tyto nejpravděpodobnější oblasti. Díky registraci barevných a hloubkových obrazů (viz Podkapitola 2.2) jsem získával přibližné označení pixelů ruky i v hloubkovém obraze. Přibližné zejména kvůli tomu, že registrace není 100procentně přesná nebo že se nějaký šum připojil k oblasti ruky. Na Obrázku 5.2 je ukázka hloubkového obrazu spolu s výsečí odpovídajícího binárního obrazu labelů ruky. Je zřejmé, že ruka v tomto konkrétním obraze je označena nepřesně. K oblasti označující ruku se připojilo trochu šumu a uvnitř je díra. Takové chyby byly velmi častým jevem při prvotním označování ruky, nejčastěji způsobené rychlým pohybem.

Takto bylo vyfoceno celkem 8792 obrazů. Při snímání jsem se snažil pokrýt co nejvíce možných pozic ruky vsedě i vestoje. Přibližně polovina obrazů byla otočena



Obrázek 5.2: Výseč prvotní podoby označení ruky.

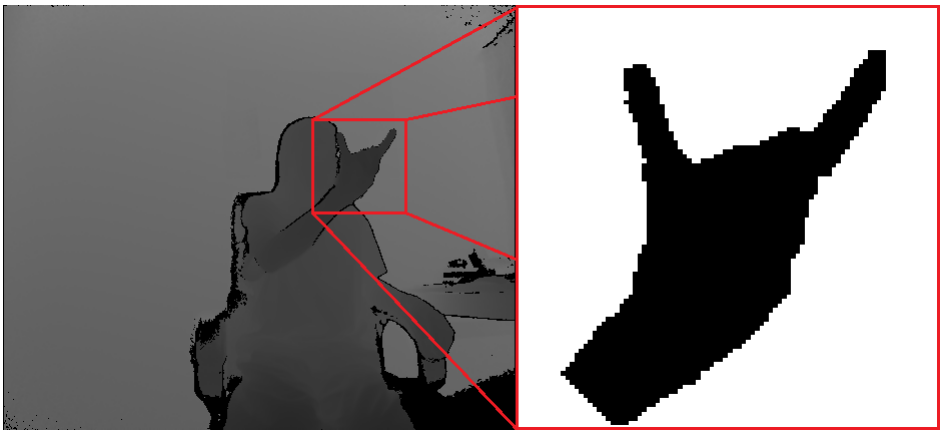
tak, aby existovala data i pro pravou ruku.

5.1.2 Zpřesňování labelů ruky

Ve snaze mít dataset co nejpřesnější bylo potřeba chyby labelů odstranit. Toho bylo dosaženo ručním procházením jednotlivých obrazů a aplikací algoritmů Graph Cut (viz Podkapitola 3.2.6) nebo Flood Fill (viz Podkapitola 3.2.7). Díky implementaci algoritmu Graph Cut ve své interaktivní verzi Grab Cut v knihovně OpenCV bylo možné a ve většině případů velmi vhodné přitom využít již hotové prvotní labely. V první řadě byly vytvořeny obrazy popisující jisté popředí (labely ruky) a jisté pozadí (invertované labely ruky). Výseč těchto obrazů je vidět vlevo na Obrázku 5.3. Algoritmus Grab Cut by v takovém případě ale nepomohl, o všech pixelech by bylo rozhodnuto již před samotným průběhem. Bylo tedy potřeba vytvořit jakýsi prostor nejistoty mezi těmito dvěma oblastmi. Toho bylo dosaženo morfologickou erozí (viz Podkapitola 3.2.5) se strukturním elementem 3×3 s individuálním počtem iterací podle konkrétních obrazů. Po erozi se zmenší oblast označující jisté popředí a oblast označující jisté pozadí ustoupí. Tak mezi nimi vznikne pás pixelů, o nichž se bude rozhodovat. Výseče obrazů popisující jisté popředí a jisté pozadí po erozi jsou vpravo na Obrázku 5.3. Tyto obrazy poté tvoří masku pro samotný algoritmus, který rozhodne o pixelech z onoho neznámého pásu. Výsledkem je zpřesněný finální obraz labelů. Příklad je znázorněn na Obrázku 5.4.

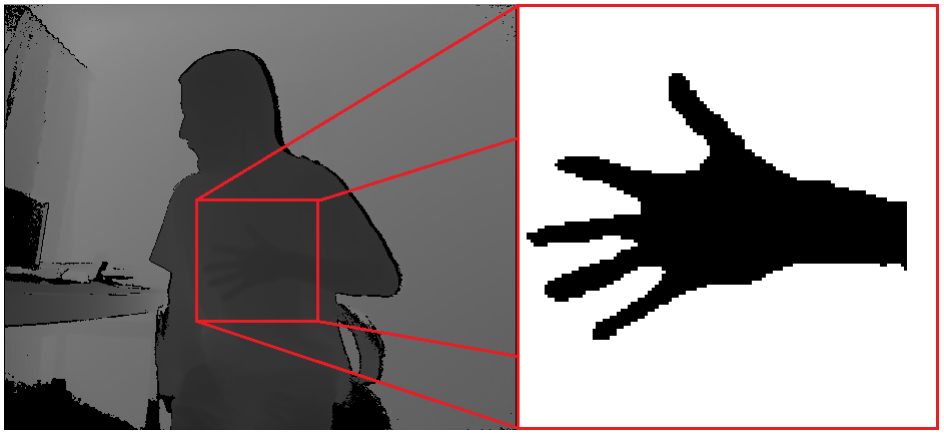
Obrázek 5.3: Výseče jistého popředí a jistého pozadí před a po erozi.

Ne vždy však tento postup vracel uspokojivé výsledky. Zejména v případech, kdy se ruka nacházela příliš blízko těla. Grab Cut pak měl problém rozpoznat hranice mezi rukou a pozadím a neoznačoval správně například mezery mezi prsty. V takových případech byl použit algoritmus Flood Fill s nastavením minimálních prahů.



Obrázek 5.4: Výseč finální podoby označení ruky dosažené algoritmem Grab Cut.

Po malých částech byla ruka takto označena. I přesto však bylo často potřeba výsledky upravovat ručně. Příklad obrazu řešeného tímto způsobem je na Obrázku 5.5. Některé obrazy nešly řešit ani přes Grab Cut ani přes Flood Fill a byly proto smazány. Z celkového počtu 8792 obrazů jich tak zbylo 8740.



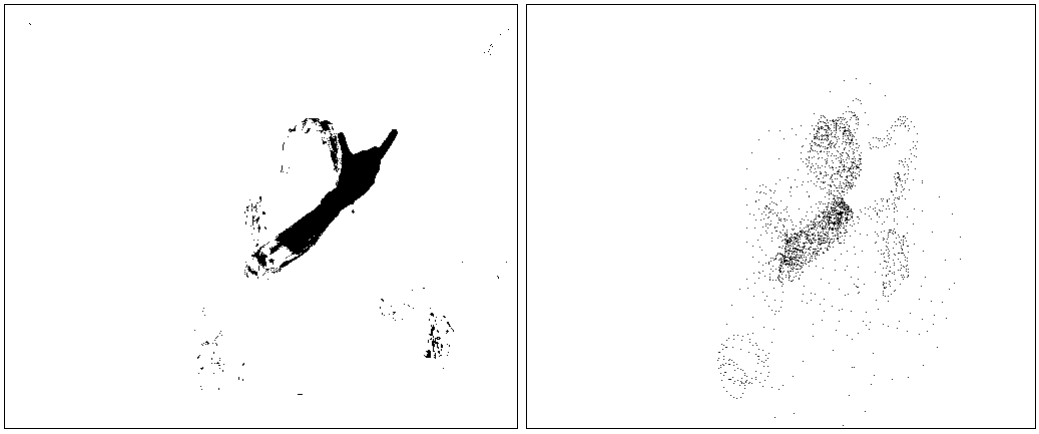
Obrázek 5.5: Výseč finální podoby označení ruky dosažené algoritmem Flood Fill.

5.1.3 Výběr labelů pozadí

V této chvíli jsem měl připravené hloubkové obrazy a labely rukou v nich. Pro trénování však bylo nutné vybrat pixely zastupující třídu pozadí. V prvních experimentech byly tyto pixely vybírány náhodně, což vedlo ke špatným výsledkům. Co se týče pixelů ruky, většinou je klasifikátor dokázal takto rozpoznat uspokojivě. Problémem bylo, že nedokázal poznat, kde končí ruka a začíná předloktí. Po tomto zjištění jsem k výběru zástupců pozadí přistoupil tak, že jsem vybíral vždy pás pixelů okolo ruky a zbytek náhodně. V některých výsledcích se ukázalo, že klasifikátor se snaží ruku a předloktí oddělit. Nicméně pořád klasifikoval loket, hlavu či opěradla židle jako ruku.

To jsem vyřešil tak, že jsem opět prošel všechny obrazy a ručně označil tato problematická místa. Vlevo na Obrázku 5.6 je vidět příklad špatné klasifikace předloktí, hlavy a kolem opěradel židle, vpravo pak odpovídající ruční označení pixelů pozadí.

Podobně byly zpracovány všechny ostatní obrazy.



Obrázek 5.6: Výsledek prvních experimentů a označení zástupců třídy pozadí.

Dataset nyní obsahoval 8740 hloubkových obrazů, labely ruky a vybrané labely pozadí, přičemž zastoupení vzorků třídy ruky se početně rovnalo zastoupení vzorku třídy pozadí. První část postupu řešení byla tímto dokončena.

5.2 Extrakce příznakových vektorů

Druhou částí postupu řešení byla extrakce informativních příznaků o vybraných a označených pixelech z minulého kroku. V článcích [1] a [2] se mluví o tom, že v každém uzlu stromů se porovnává rozdíl hloubek v aktuálně zpracovávaném pixelu (u, v) a pixelu vzdáleném o vektor posunu $(\Delta u, \Delta v)$ s natrénovaným prahem d_t. Pro praktické účely jsem zavedl okénko, z něhož byly vybírány ony vzdálené pixely. Jestliže I je hloubkový obraz, pak jednotlivé dimenze příznakového vektoru jsou tvořeny podle

$$I\left(u\ + \frac{\Delta u}{I(u,v)}, v\ + \frac{\Delta v}{I(u,v)}\right) - I(u,v). \tag{5.1}$$

Dělení posunů Δu a Δv hloubkou $I(u, v)$ podle [2] zajišťuje nezávislost na hloubce. To znamenalo nekonstantní velikost zmíněného okénka. Čím větší hloubka aktuálně zpracovávaného pixelu, tím menší okénko. Konkrétní řešení tohoto problému spočívalo v zjištění minimální nenulové a maximální hloubky zachycené ve všech hloubkových obrazech a stanovení minimální velikosti okénka pro maximální hloubku. Minimální zjištěná naměřená hloubka I_{min} byla 7282 (velikosti okének pro pixely o hloubce 0 jsou rovny velikostem okének pro pixely o hloubce I_{min}), maximální I_{max} pak 65104. Velikost okénka a_{min} pro pixely o maximální hloubce jsem experimentálně zvolil 15 pixelů. Velikosti okénka a se pak pro jiné hloubky v aktuálních pixelech (u, v) spočtou podle

$$a(u,v) = a_{min} \cdot \frac{I_{max}}{I(u,v)}. \tag{5.2}$$

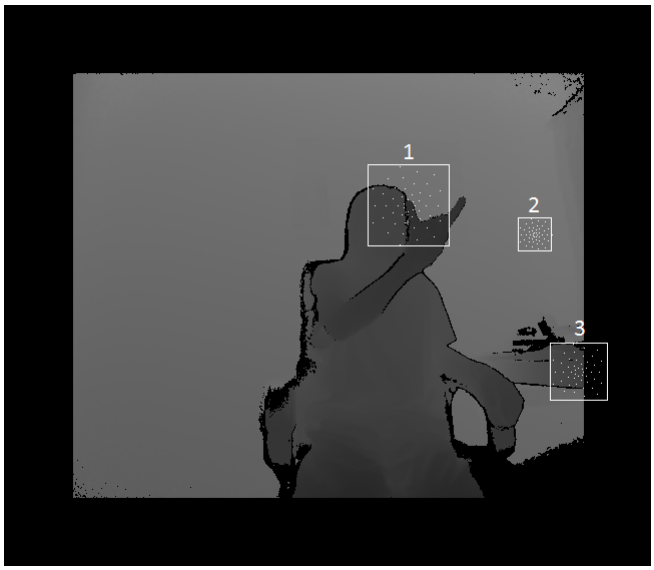
Na Obrázku 5.7 jsou znázorněny 3 příklady velikosti okénka. V případě označeným číslem 1 jde o pixel prstu v hloubce 11909. To podle uvedeného vztahu po zaokrouhlení odpovídá velikosti okénka 82×82. Další 2 okénka pro hloubky 29592 a 16837 mají velikost 33×33 a 58×58 pixelů.

Důležitou otázku, již bylo nutno vyřešit, byla volba dimenze příznakového vektoru. Tento problém byl vyřešen s ohledem na paměťové nároky při trénování RDF v OpenCV, kde je potřeba mít všechna trénovací data současně v paměti. Dimenze byla nastavena na 49. Z okének bylo tedy vybíráno 49 pixelů. Jejich výběr probíhal následovně. Nejprve byly předpočítány polární souřadnice. Byly definovány množiny pro úhel Φ a pro poloměr r o velikosti 49. Množina úhlů obsahovala 49 čísel vždy po 7 stejných od 0 do 360 s krokem 360/7. Přitom byla ke každé druhé hodnotě přičtena polovina kroku, tedy 360/14. Množina poloměrů obsahovala 7krát opakující se sekvenci od 1 do 8. Množiny tak popořadě společně představovaly polární souřadnice vybraných bodů v počátečním stavu. Poté byly přepočítány na kartézské souřadnice podle vzorců

$$\Delta u_i = r_i \cdot \cos(\Phi_i), \tag{5.3}$$

$$\Delta v_i = r_i \cdot \sin(\Phi_i), \tag{5.4}$$

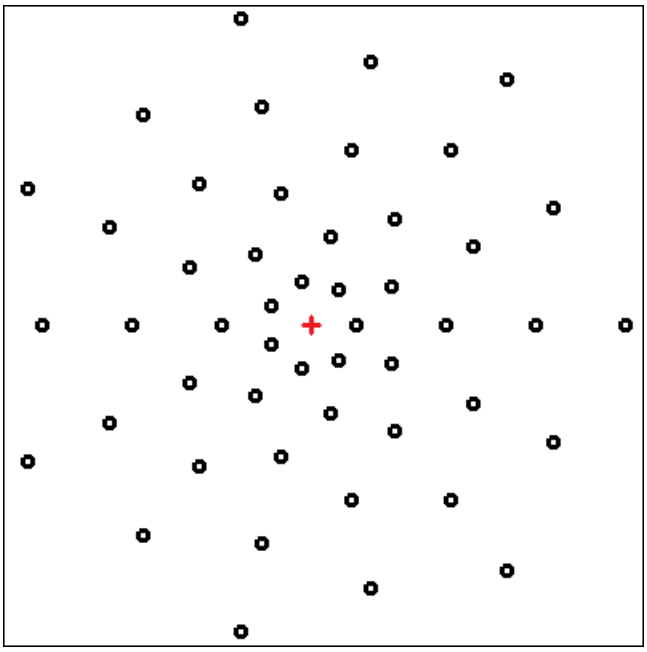
kde $i = 1, 2, ..., 49$. Pro aplikaci výběru pixelů takto definovanými souřadnicemi je bylo už jen nutné vždy přenásobit podle velikosti konkrétního okénka. Až poté byly souřadnice zaokrouhleny. Výběry jsou zobrazeny na Obrázku 5.7 jako bílé body uvnitř jednotlivých okének. Na Obrázku 5.8 je zvětšený příklad rozvržení bodů v okénku. Červeně je vyznačen aktuálně zpracovávaný pixel a černě zmíněné body.



Obrázek 5.7: Aspekty extrakce příznakových vektorů.

Na Obrázku 5.7 si dále lze všimnout, že u okénka označeného číslem 3 některé bílé body přesahují hranice obrazu. Kvůli tomu byl při extrakci příznakových vektorů každý obraz zvětšen o polovinu maximální velikosti okénka do všech stran. Nové hodnoty černého rámečku kolem obrazů jsou všechny nastaveny na 0.

Z takto vybraných pixelů byly přečteny hloubkové hodnoty, od nichž se odečetla hloubková hodnota zpracovávaného pixelu (u, v). Výsledkem extrakce je vektor o 49 hodnotách pro každý pixel označený labelem součástí ruky či pozadí.



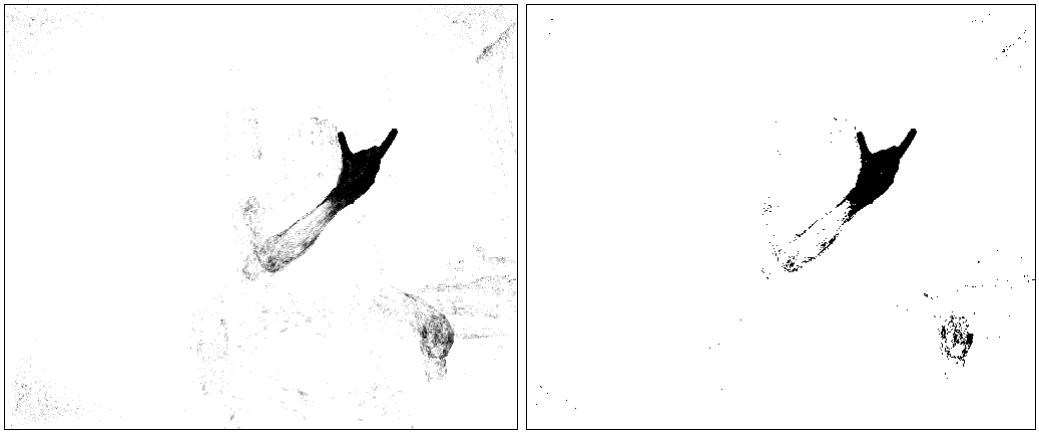
Obrázek 5.8: Výběr vzdálených pixelů.

5.3 Trénování a testování klasifikátoru

Po dokončení extrakce příznakových vektorů ze všech označených pixelů byly vektory použity pro trénování a testování náhodných rozhodovacích lesů. Pro trénování bylo potřeba zvolit poměr rozdělení datasetu na trénovací a testovací množinu, hloubku stromů a jejich počet.

Poměr rozdělení jsem zvolil následovně. Z celkového počtu 8740 obrazů jsem zvolil vždy 7000 náhodných pro trénování a zbytek pro testování. To odpovídá přibližně poměru 80:20. Pro zjištění vhodného nastavení hloubky a počtu stromů jsem postupoval podobně jako v [1]. To znamenalo postupné testování lesů se 4 až 10 stromy s krokem 2 o hloubce 10 až 30 s krokem 5. Vstupem trénovacího algoritmu tedy byly příznakové vektory z oněch 7000 trénovacích obrazů spolu s informacemi učitele pro každý vektor zjištěné z obrazů označujících odpovídající pixely a parametry lesa. Veškeré trénování probíhalo na počítačích Metacentra. Natrénování jednoho klasifikátoru trvalo závisle na jeho parametrech od 4 do 14 hodin.

Testování lesa znamenalo aplikaci klasifikátoru na příznakové vektory všech pixelů všech obrazů z testovací množiny. Díky implementaci RDF v OpenCV je možné pro každý pixel zobrazit finální zařazení do třídy, ale i pravděpodobnost tohoto zařazení. Vlevo na Obrázku 5.9 je výsledek klasifikace s použitím pravděpodobnostního přístupu, každý pixel popisuje kolik stromů lesa ho klasifikovalo jako součást ruky. Vpravo je pak příklad jednoduchého přiřazení do třídy ruky. Pokud většina stromů klasifikuje pixel do třídy ruky, pak ho tak klasifikuje celý les. Tento příklad byl klasifikován lesem se 6 stromy o hloubce 30.



Obrázek 5.9: Výsledek klasifikace.

Jako metodu hodnocení kvality klasifikátoru jsem zvolil precision, recall a F1-score (viz Podkapitola 4.2.1). True positives v tomto kontextu znamenají pixely, které byly klasifikovány jako součást ruky, přičemž tak byly označeny při vytváření datasetu. False positives znamenají pixely, které byly klasifikovány jako součást ruky, přičemž tak nebyly označeny. True negatives znamenají pixely, které byly klasifikovány jako součást pozadí, přičemž tak byly označeny. False negatives znamenají pixely, které byly klasifikovány jakou součást pozadí, přičemž tak nebyly označeny. Pro pravou

část Obrázku 5.9 bylo dosaženo výsledků

$$\begin{aligned} \text{Precision} &= 70{,}35\ \%, \\ \text{Recall} &= 98{,}93\ \%, \\ \text{F1-score} &= 82{,}23\ \%. \end{aligned}$$

Precision, recall a F1-score byly spočítány vždy pro celou testovací množinu. V Tabulce 5.1 jsou zapsány výsledky v procentech v závislosti na nastavení parametrů lesa (H - hloubka stromů, P - počet stromů).

	Precision				Recall				F1-score			
H/P	4	6	8	10	4	6	8	10	4	6	8	10
10	26,48	27,85	28,45	29,57	87,83	88,70	88,62	89,78	40,69	42,39	43,07	44,49
15	40,09	41,86	41,54	41,90	88,60	90,05	91,18	91,7	55,20	57,15	57,08	57,52
20	47,73	54,40	57,00	59,26	90,87	91,92	91,96	93,04	62,59	68,35	70,38	72,41
25	57,90	62,74	66,74	68,06	91,08	92,76	93,30	93,80	70,80	74,85	77,81	78,88
30	57,43	63,83	66,63	**68,30**	91,59	92,90	93,10	**93,95**	70,59	75,67	77,67	**79,10**

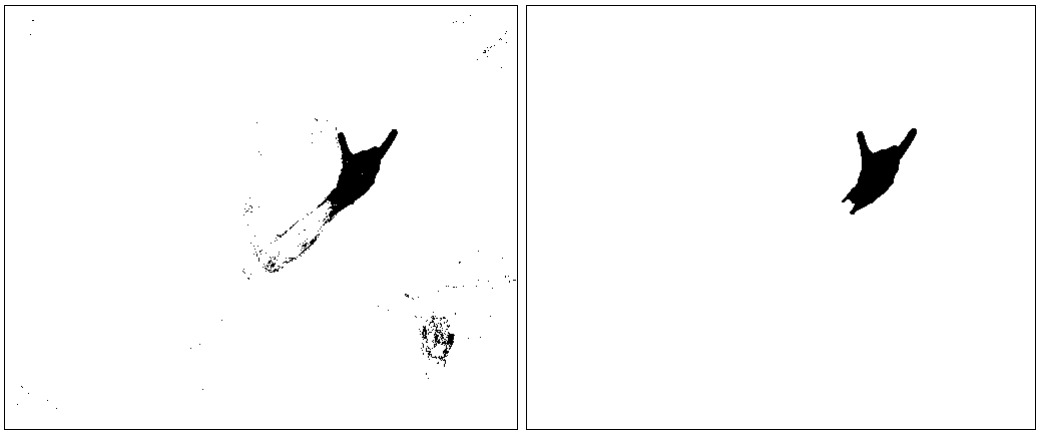
Tabulka 5.1: Výsledky testování s různým nastavením parametrů lesa.

Je vidět, že precision roste jak s hloubkou, tak s počtem stromů. Čím méně je ve výsledných obrazech false positives, tím větší je precision. False positives například vpravo na Obrázku 5.9 tvoří osamocené bílé body či malé oblasti. Tyto chyby se nalézají opět především v oblastech předloktí, hlavy a opěradel židle. Rozhodně se ale výběr zástupců pozadí pro trénování (viz Obrázek 5.6) projevil jako přínosný. Dále je z tabulky vidět, že též recall roste s rostoucími parametry lesa, nicméně ne tak markantně jako v případě precision. Celkově je tedy vidět, že vetší problém je s false positives než s false negatives, které by se projevily například jako díry v oblasti ruky či chybějící prsty.

5.4 Post processing

Kvůli vysokému počtu false positives ve výsledných obrazech jsem se dále rozhodl zkusit tyto odstranit a zvýšit tak precision, respektive F1-score. Vyšel jsem z pravděpodobnostních obrazů. Z nich byly vybrány pixely s nenulovou pravděpodobností. Z hodnot těchto pixelů byla vypočtena střední hodnota. Pravděpodobnostní obraz je pak prahován prahem, který je roven 175 procentům střední hodnoty. Tento postup byl zvolen na základě experimentování s prahem konstantním pro všechny obrazy. Například jsem zkoušel uvažovat pouze ty pixely, které měly alespoň 80 procent. Nicméně ukázalo se, že pravděpodobnosti pixelů ruky v některých obrazech se na tyto hodnoty nedostaly, zatímco chybné pixely předloktí ano. Ruka tím pádem byla ztracena. Bylo tedy potřeba volit práh v závilosti na konkrétní podobě výsledných obrazů. Po prahování byl aplikován mediánový filtr, který měl za úkol odstranit nejmenší šum - samostatné body a malé oblasti. Po této operaci ve většině obrazů zbyla v ideálních případech pouze ruka a některé větší chybné oblasti. Dalším krokem bylo hledání kontury s maximálním obsahem, přitom předpokládaje, že tato bude odpovídat oblasti ruky. Na Obrázku 5.10 je vidět příklad odstranění bodů a malých oblastí z výsledného obrazu.

Následkem post processingu v tomto konkrétním případě je snížení počtu false



Obrázek 5.10: Výsledek klasifikace po post processingu.

positives a tím pádem navýšením precision. Mediánový filtr se postaral o zaplnění děr, nicméně trochu též pozměnil okraje oblasti ruky, se kterými problém nebyl. Proto došlo ke snižení recall. Celkově se post processing v tomto konkrétním případě vyplatil, neboť vzrostlo F1-score. Bylo dosaženo výsledků

$$\begin{aligned} \text{Precision} &= 94{,}57\ \%, \\ \text{Recall} &= 97{,}56\ \%, \\ \text{F1-score} &= 96{,}04\ \%. \end{aligned}$$

Precision, recall a F1-score byly spočítány opět vždy pro celou testovací množinu, na kterou byl aplikován popsaný postup post processingu. V Tabulce 5.2 jsou opět zapsány výsledky v procentech v závislosti na nastavení parametrů lesa (H - hloubka stromů, P - počet stromů).

	Precision				Recall				F1-score			
H/P	4	6	8	10	4	6	8	10	4	6	8	10
10	70,96	72,72	66,90	67,32	24,04	59,26	68,25	75,00	35,92	65,33	67,57	70,95
15	87,55	84,72	80,85	78,90	53,90	69,87	80,21	82,12	66,73	76,58	80,53	80,48
20	93,93	90,35	88,44	85,40	61,07	77,59	83,20	87,88	74,01	83,48	85,74	86,62
25	**95,71**	92,55	90,92	88,17	64,39	81,73	86,96	**90,01**	76,99	86,80	88,89	89,08
30	95,21	92,91	90,45	89,10	63,38	81,81	88,29	89,48	76,10	86,63	88,29	**89,29**

Tabulka 5.2: Výsledky testování s různým nastavením parametrů lesa po post processingu.

Na první pohled je vidět, že post processing přinesl zlepšení precision. Nicméně pro některá nastavení hloubky a počtu stromů se dost zmenšil recall. Zlepšení se tedy projevilo spíše pro větší a složitější lesy.

5.5 Real-time

Posledním krokem byla aplikace klasifikátoru na hloubkové obrazy plynoucí z Kinectu v reálném čase. S tím se pojilo několik problémů. Prvním problémem byla skutečnost, že program nestíhal extrahovat příznakové vektory pro všechny pixely

jednotlivých obrazů a tyto klasifikovat tak, aby bylo dosaženo alespoň 25 FPS (frames per second), což se považuje za rychlost, kterou lidské oko vnímá plynule. Omezil jsem se tedy pouze na extrakci příznaků a klasifikaci pixelů vymezených čtvercovým okénkem. Tím se snížil jejich počet a došlo ke zrychlení. Dalším problémem byla volba jednoho z natrénovaných klasifikátorů. Při této volbě rozhoduje, zda uživateli jde spíše o rychlost pohybu či přesnost označení klasifikovaných pixelů. Jestliže je důležitější rychlost, je vhodné zvolit co nejmenší počet stromů s malou hloubkou. V tomto případě ale hrozí, že se projeví již zmiňovaný problém špatné klasifikace předloktí a sledovací okénko může tím pádem „utéct“ až na loket. Jestliže rychlost pohybu nehraje tak důležitou roli, je možné použít les s větším počtem stromů o větší hloubce. V takovém případě je sice program pomalejší a neběží rychlostí 25 FPS (některé obrazy nejsou zpracovávány), ale nedochází zde tak často ke špatné klasifikaci předloktí.

Samotné sledování probíhá následovně. Uživatel stiskne tlačítko a proběhne automatická detekce ruky klasifikací všech pixelů v 16krát zmenšeném obraze. Ta odhadne počáteční polohu a velikost sledovacího okénka. Pro všechny pixely z tohoto okénka se dále extrahují příznakové vektory, které jsou klasifikovány. Výsledkem klasifikace je opět pravděpodobnostní obraz. Ten je zpracován obdobným způsobem popsaným v Podkapitole 5.4. Výsledkem zpracování je v ideálním případě oblast, která odpovídá ruce. Z této oblasti jsou vypočteny momenty (stejně jako v algoritmu Cam Shift). Z momentů je spočten střed nového okénka jako těžiště oblasti (u_c, v_c)

$$u_c = \frac{M_{10}}{M_{00}}, \tag{5.5}$$

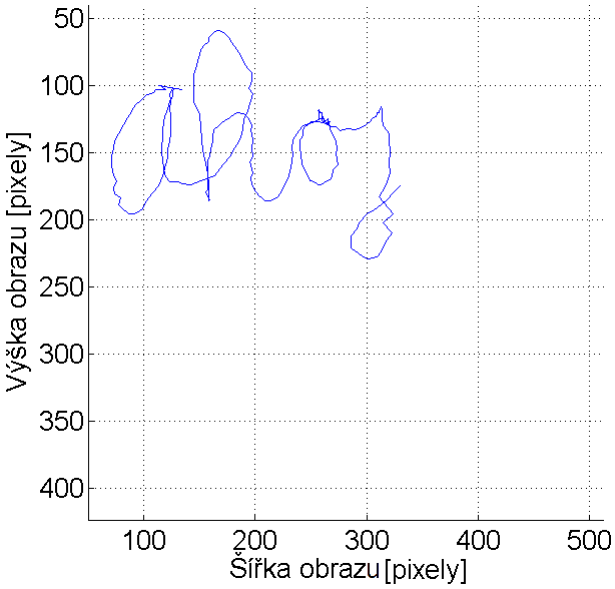
$$v_c = \frac{M_{01}}{M_{00}}. \tag{5.6}$$

Pro každé spuštění programu jsou tyto hodnoty ukládány na disk pro další zpracování nebo vizualizaci. Příklad takové vizualizace je na Obrázku 5.11, kde jsem rukou napsal slovo „ahoj“.

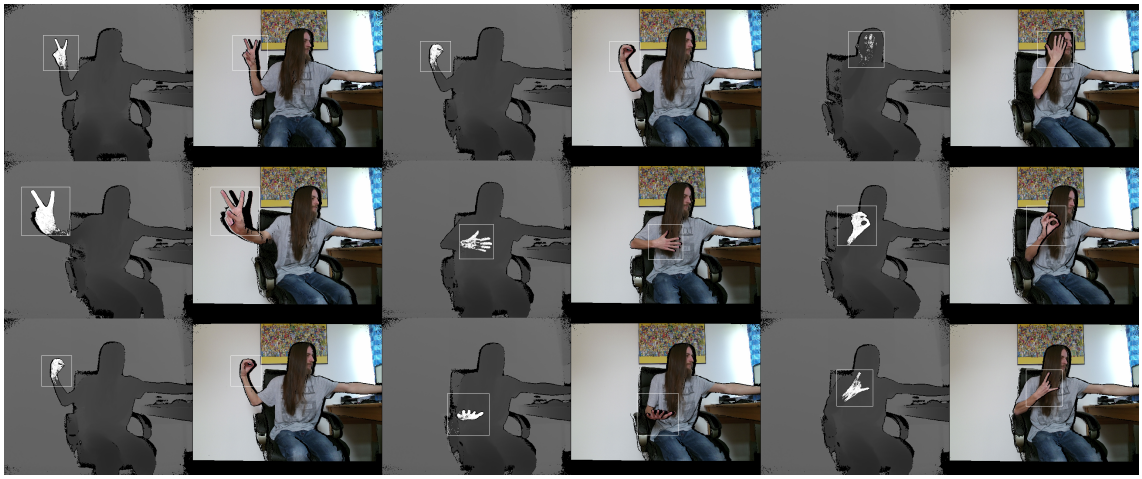
Velikost sledovacího okénka se mění spolu s tím, jak se sledovaná ruka pohybuje v hloubce. Blíží-li se ruka kameře, je potřeba okénko zvětšovat a naopak. To jsem vyřešil experimentálním určením velikosti okének pro několik hloubek pro maximální rozpětí prstů. Z naměřených hodnot jsem sestavil lineární funkci. Velikost sledovacího okénka b se pak pro hloubku $I(u_c, v_c)$ vypočte jako

$$b = c_1 \cdot I(u_c, v_c) + c_2, \tag{5.7}$$

kde $c_1 = -0{,}0068$ a $c_2 = 192{,}9257$. Nové okénko je použito vždy v následujícím obrázku. Na Obrázku 5.12 jsou znázorněny příklady funkce real-time aplikace. Tyto konkrétní obrazy byly klasifikovány lesem s 10 stromy o hloubce 25. Pro názornou ukázku jsou v hloubkových obrazech bíle vybarveny ty pixely, které měly alespoň 80 % pravděpodobnost, že patří do třídy ruky.



Obrázek 5.11: Vizualizace pohybu ruky ve videu.



Obrázek 5.12: Ukázka různých pozic ruky při provozu real-time aplikace.

Kapitola 6

Problémy při řešení

Při práci se vyskytly jisté problémy, které budou v této kapitole popsány.

6.1 Složitost příznakových vektorů

Při extrakci příznakových vektorů byla vyzkoušena celá řada přístupů, nicméně žádná nepřinesla lepší výsledky a přitom by byla rychlejší než popsaná metoda. Původně byla například navržena metoda rozložení bodů v klasické čtvercové mřížce (jeden z důvodů proč je stále dimenze vektorů 49). Z každé buňky mřížky by se pak uvažoval medián nebo například průměr či vážený průměr. Vzhledem k tomu, že velikost strany okénka nebyla vždy dělitelná 7 a velikosti jednotlivých buněk vznikaly neceločíselné, bylo potřeba okénko z obrazu speciálně vyčlenit a změnit jeho velikost na nejbližší násobek 7. Poté ho rozdělit na buňky a z nich spočítat medián. To se projevilo jako příliš zdlouhavá operace nevhodná pro real-time aplikaci.

6.2 Přítomnost druhé ruky v hloubkových obrazech

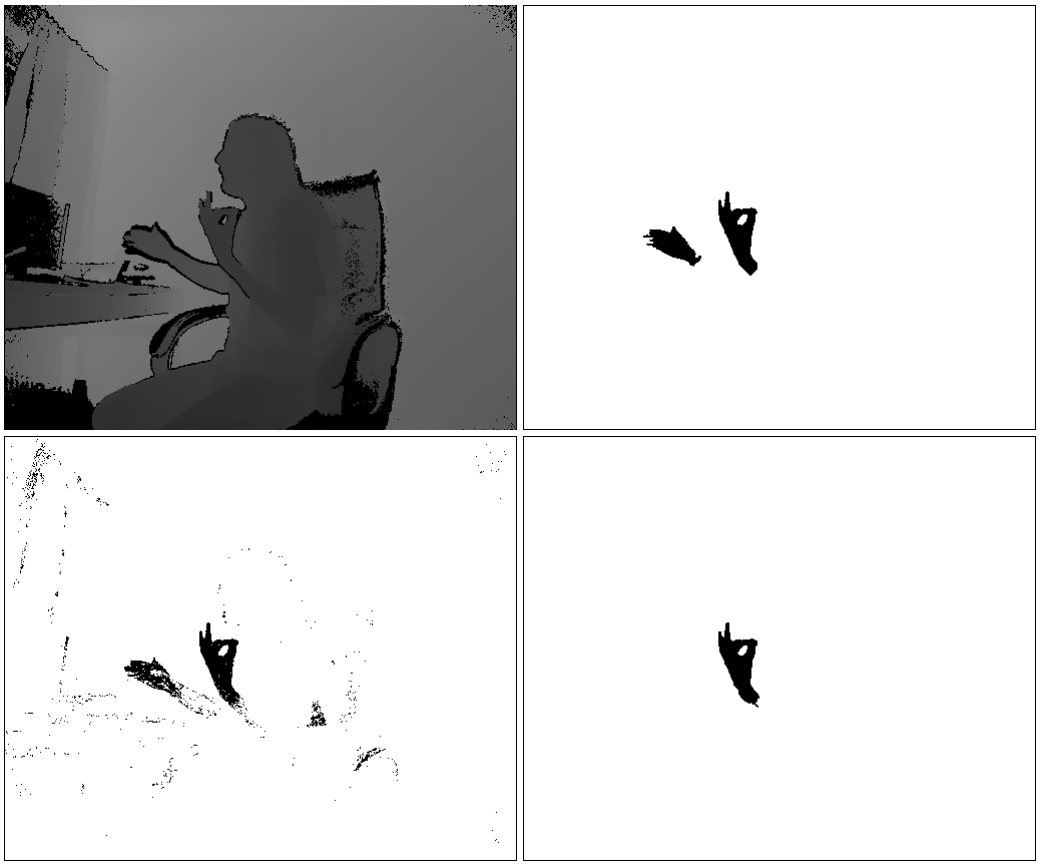
Velkým problémem a důvodem snížené míry kvality klasifikátorů byla přítomnost mé druhé ruky v trénovacích i testovacích obrazech. Rozhodl jsem se k tomuto problému přistoupit tak, že pro účely trénování byla tato označena stejným postupem jako v případě první ruky. V případě testování s post processingem, kde jsem vybíral pouze jednu oblast, jsem však znalost o pozici druhé ruky ztrácel a tím docházelo k poklesu precision i recall. Příklad takového problému je na Obrázku 6.1. Vlevo nahoře je původní hloubkový obraz, vpravo nahoře trénovací labely rukou, vlevo dole výsledek klasifikace bez post processingu a vpravo dole výsledek klasifikace s post processingem. Výsledná hodnocení výsledku bez post processingu pro tento obraz byla

$$\begin{aligned} \text{Precision} &= 65{,}55\ \%, \\ \text{Recall} &= 90{,}6\ \%, \\ \text{F1-score} &= 76{,}07\ \%. \end{aligned}$$

S post processingem pak

$$\begin{aligned} \text{Precision} &= 96{,}55\ \%, \\ \text{Recall} &= 60{,}31\ \%, \\ \text{F1-score} &= 74{,}24\ \%. \end{aligned}$$

Zejména vlivem takovýchto případů jsou výsledky recall v Podkapitole 5.4 sníženy.



Obrázek 6.1: Problém druhé ruky v trénovacích obrazech.

V real-time aplikaci se tento problém projeví jedině v případech, kdy by obě ruce byly přítomny ve sledovacím okénku.

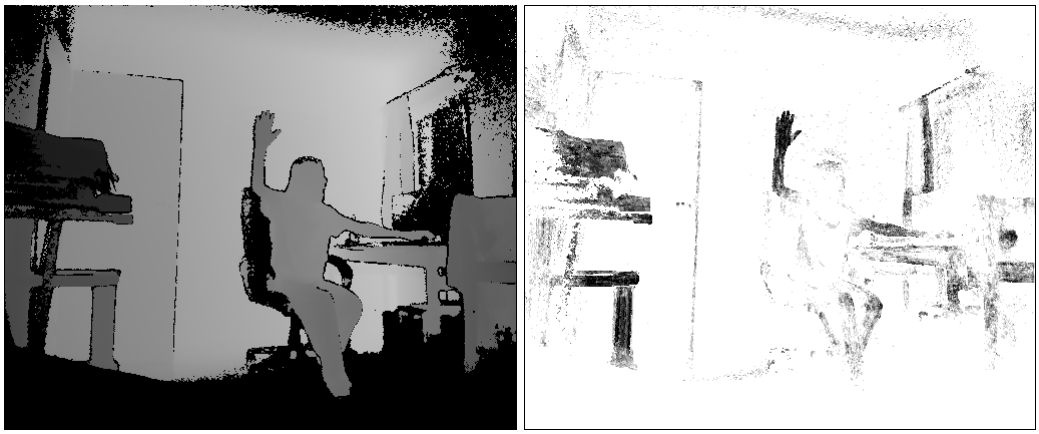
6.3 Časové a paměťové nároky

Ruční označování rukou spolu s pixely zastupujícími třídu pozadí byla časově náročná úloha, která dohromady zabrala měsíc. Trénování jednoho klasifikátoru zabralo na počítačích Metacentra až 14 hodin při použití 20 GB RAM. Celkově se tedy jednalo o velmi časově náročnou úlohu.

6.4 Scéna sledování

Vzhledem k tomu, že trénování probíhalo výhradně z hloubkových obrazů, na nichž se nachází pouze člověk, židle a za nimi rovná stěna, lze předpokládat, že real-time aplikace by nemusela uspokojivě fungovat v jiných podmínkách. Problémem

může též být větší vzdálenost ruky od Kinectu než byla v trénovacích obrazech. Tyto problémy znázorňuje Obrázek 6.2, ve kterém došlo nejen k problému špatně klasifikovaného předloktí, ale i k mnoha problémům v okolí.



Obrázek 6.2: Nevhodná scéna sledování.

6.5 FPS real-time aplikace

Rychlost sledování ruky v reálném čase je omezena především složitostí použitého klasifikátoru. Doba klasifikace jednoho příznakového vektoru roste závisle na počtu stromů a jejich hloubce. Například při použití nejjednoduššího uvažovaného nastavení se FPS pohybují v intervalu 25 až 30, což se dá považovat za real-time, ovšem hrozí zvýšené riziko špatné klasifikace předloktí. Může se tak stát, že prohledávací okénko „sjede“ na loket. Při použití nejsložitějšího uvažovaného nastavení rychlost běhu klesne na 10 až 15 FPS. Nicméně riziko špatné klasifikace předloktí je zde menší. Vhodným kompromisem mezi rychlostí a přesností se ukázal klasifikátor se 4 stromy o hloubce 25, jehož FPS se pohybují v intervalu 20 až 25 a přesnost je ve většině případů dostačující.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat metodu sledování ruky v hloubkových obrazech získaných senzorem Kinect v2. Na základě článku [1] byla zvolena metoda klasifikace jednotlivých pixelů hloubkových obrazů do tříd ruky a pozadí.

Postup řešení zpočátku spočíval ve vytvoření datasetu, který nakonec čítal 8740 hloubkových obrazů spolu s obrazy labelů ruky a pozadí. Z těchto obrazů pak byly navrženou metodou extrahovány příznakové vektory. Ty byly použity k trénování náhodných rozhodovacích lesů. Těchto klasifikátorů bylo natrénováno více vždy s rozdílným nastavením počtu stromů a jejich hloubky. Každý z nich byl ohodnocen mírou kvality klasifikace F1-score. Ukázalo se, že čím větší hloubku stromy měly, tím byla klasifikace na testovacích množinách přesnější. Nejsložitější uvažovaný klasifikátor s 10 stromy o hloubce 30 dosahoval skoro 80 %. Po použití metod post processingu na výsledné obrazy s klasifikovanými pixely dosahovaly složitější lesy až na 90 % F1-score.

Dále byly klasifikátory použity na hloubkové obrazy plynoucí z Kinectu v reálném čase. Zde se ukázalo, že záleží na volbě konkrétního nastavení použitého lesa. Zejména počet stromů lesa má velký vliv na rychlost zpracování pixelů sledovacího okénka. K dosažení ideální rychlosti běhu programu 25 FPS je kvůli tomu nutné použít klasifikátor nejlépe s co nejméně stromy. S použitím lesa o méně stromech nebo s menší hloubkou stromů se ovšem pojí zmíněný problém nižší přesnosti klasifikace. Dobrým kompromisem v této úloze se ukázal být klasifikátor se 4 stromy o hloubce 25.

Při provozu sledovacího programu není nutné, aby uživatel ruku v prvním obraze označil ručně. Za podmínky, že je dobře viditelná, ji program detekuje automaticky. Sledovací program byl otestován na více lidech. Ukázalo se, že sledování jiné ruky než té z trénovacích obrazů je také možné.

Vzhledem k tomu, že se jedná o úlohu klasifikace, lže říci, že mnohé záleží na množství a kvalitě trénovacích dat. Je tedy možné, že například problém popsaný v Podkapitole 6.4 by se dal řešit zvětšením datasetu o podobné obrazy. Problém s rychlostí běhu programu by se dal vyřešit paralelizací klasifikátoru. To by mohlo být vyřešeno použitím knihovny OpenCV s podporou knihovny TBB (Threading Building Block). Se zrychlením klasifikace by se pak uvolnil čas například pro použití více stromů nebo pro výpočet složitějších příznaků. Ty by mohly problém lépe generalizovat a sledování by tak v konečném důsledku mohlo být rychlejší a přesnější. Tímto a jiným aspektům této úlohy by se dalo věnovat v budoucnu.

Literatura

[1] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks," *ACM Trans. Graph.*, vol. 33, no. 5, pp. 169:1–169:10, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2629500

[2] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1297–1304. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995316

[3] M. Valoriani, "Programming with kinect v2," 2015. [Online]. Available: http://www.slideshare.net/MatteoValoriani/programming-with-kinect-v2

[4] L. Li, "Time-of-flight camera–an introduction," *Technical White Paper, May*, 2014.

[5] L. Xiang, F. Echtler, C. Kerl, T. Wiedemeyer, Lars, hanyazou, R. Gordon, F. Facioni, laborer2008, R. Wareham, M. Goldhoorn, alberth, gaborpapp, S. Fuchs, jmtatsch, J. Blake, Federico, H. Jungkurth, Y. Mingze, vinouz, D. Coleman, B. Burns, R. Rawat, S. Mokhov, P. Reynolds, P. Viau, M. Fraissinet-Tachet, Ludique, J. Billingham, and Alistair, "libfreenect2: Release 0.2," Apr. 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.50641

[6] M. Železný, "Zpracování digitálního obrazu," přednášky, *ZČU FAV KKY*, 2005.

[7] V. Hlaváč, "Digitální zpracování obrazu," přednášky, *ČVUT Fakulta elektrotechnická, katedra kybernetiky.*

[8] M. Sonka, V. Hlavac, R. Boyle *et al.*, *Image processing, analysis, and machine vision*. Thomson Toronto, 2008, vol. 3.

[9] P. Nishad, "Various colour spaces and colour space conversion," *Journal of Global Research in Computer Science*, vol. 4, no. 1, pp. 44–48, 2013.

[10] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, Jan 1975.

[11] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *In Workshop of Applications of Computer Vision*, pp. 214–219, 1998.

[12] N. Otsu, "A threshold selection method from gray-level histograms," pp. 62–66, Jan 1979.

[13] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[14] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: http://docs.opencv.org/2.4.10/

[15] Y. Y. Boykov and M. P. Jolly, "Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1, 2001, pp. 105–112 vol.1.

[16] G. Slabaugh and G. Unal, "Graph cuts segmentation using an elliptical shape prior," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2, Sept 2005, pp. II–1222–5.

[17] M. Hrúz and Z. Krňoul, "Metody počítačového vidění," přednášky, *ZČU FAV KKY*, 2015.

[18] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, 1956.

[19] J. Psutka, "Strojové učení, řešení úloh a rozpoznávání," přednášky, *ZČU FAV KKY*, 2014.

[20] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning." Microsoft Research, Tech. Rep. MSR-TR-2011-114, Oct 2011.

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[22] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation," School of Informatics and Engineering, Flinders University, Adelaide, Australia, Tech. Rep. SIE-07-001, 2007.

Příloha A

Dataset

Na přiloženém DVD se nachází složka dataset. Obsahuje složky depth, labelsFinal a labelsNegative. Ve složce depth se nachází všech 8792 vyfocených hloubkových obrazů datasetu. Ve složce labelsFinal se nachází všech odpovídajících 8792 binárních obrazů labelů ruky. Ve složce labelsNegative se nachází všech odpovídajících 8792 binárních obrazů labelů zástupců pozadí.

Dále složka dataset obsahuje skripty napsané v jazyce Python fe.py a train_rdf.py. Skript fe.py extrahuje příznakové vektory z datasetu a uloží je do připravené prázdné složky features. Skript train_rdf.py natrénuje klasifikátor se zvolenými parametry.

Textový soubor delete.txt obsahuje čísla nepoužitých obrazů, které skripty fe.py a train_rdf.py přeskakují. Textový soubor noFV.txt obsahuje počty trénovacích příznakových vektorů z každého obrazu a je použit skriptem train_rdf.py pro inicializaci pole trainData.

Příloha B

Sledovací program

Dále se na DVD nachází složka sledovací program. V ní je uložen soubor model3_7000_25_4_1. Jde o natrénovaný náhodný rozhodovací les ze 7000 trénovacích obrazů s 4 stromy o hloubce 25. Je zde též připraven program Tracker.

Instalace

- Nainstalujte a spusťte program Zadig, ve kterém nastavte pro Xbox NUI Sensor ovladač libusbK
- Nainstalujte OpenCV 2.4.10 a přidejte ji do PATH
- Do PATH dále přidejte Tracker/depends/freenect2/lib, Tracker/depends/libjpeg-turbo64/bin, Tracker/depends/glfw/lib-vc2013 a Tracker/depends/libusbx/MS64
- Nainstalujte a spusťte program CMake a v něm vyplňte kolonku „Where is the source code: “ cestou do své složky Tracker, kolonku „Where to build the binaries“ vyplňte například Tracker/build
- Klikněte na Configure, CMake nejspíše zahlásí chybu, že neví, kde hledat freenect2_DIR, vyplňte ho proto ručně jako Tracker/depends/freenect2, znovu klikněte na Configure a vyplňte další nenalezené složky freenect2_INCLUDE_DIR jako Tracker/depends/freenect2/include a freenect2_LIBRARY Tracker/depends/freenect2/lib/freenect2.lib a klikněte na Generate
- Ve složce Tracker/build by měl vzniknout soubor Tracker.sln, otevřete ho ve Visual Studiu a zkompilujte
- Ve složce Tracker/bin/Release by se měl objevit soubor Tracker.exe

Spuštění programu a sledování ruky

Pro spuštění programu dvojitě klikněte na Tracker.exe. Objeví se příkazový řádek. Program bude chvilku nahrávat zmíněný klasifikátor, poté se objeví 2 okna: „depth16“ s hloubkovým obrazem a „registered“ s registrovaným RGB obrazem. Ujistěte se, že ruka je dobře viditelná a stiskněte klávesu ’S’. Ruka se automaticky detekuje a bude nadále sledována. Pro ukončení sledování buď ruku schovejte, nebo

opět stiskněte klávesu 'S'. Pro ukončení celého programu stiskněte klávsu 'Esc'. Během sledování můžete v příkazovém řádku sledovat aktuální FPS.

Po ukončení programu se ve složce se souborem Tracker.exe objeví textový dokument log.txt. V něm jsou informace o poloze a hloubce sledované ruky ve všech zachycených obrazech.

Příloha C

Demonstrační videa

Poslední složka na DVD se jmenuje demonstrační videa. Obsahuje videa 1.avi, 2.avi a 3.avi. Jsou to videa, která ukazují, jak program funguje. Na videu 1.avi je sledována má ruka čili ruka z trénovacích obrazů. K ní je pak přiložen zmíněný textový soubor log1.txt. Na videu 2.avi je pak příklad funkce programu při sledování ruky menší a na videu 3.avi sledování ruky větší.