Mendelova univerzita v Brně
Provozně ekonomická fakulta

Optimalizace algoritmů pro opakované vyhledání objektu

Diplomová práce

Vedoucí práce:
Ing. Jan Kolomazník, Ph.D.

Bc. Luboš Juránek

Brno 2016

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Janu Kolomazníkovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval. Mé poděkováni patři též Bc. Markétě Puchalové za pomoc při gramatické kontrole práce a poskytnutí zázemí, které mi pomohlo tuto práci zkompletovat.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Optimalizace algoritmů pro opakované vyhledání objektu**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně, dne 23. května 2016 .....

Abstract

Juránek, L. *Optimization of algorithms for repeatedly object detection.* Diploma thesis. Brno, 2016

The goal of this thesis is to develop a method for repeatedly object detection. This method uses a sequence of images, allowing faster object detection than similar methods working with only one image. Selected problem is solved by creating a demonstration method for object detection by a robot K4 which was created by robotics team PEF MENDELU AiStorm. This demonstration method is later improved to be able to work with sequence of images. This improvement accelerates method by 43 %. The main benefits of the thesis are the possibilities of image processing to object detection, example of the use of a sequence of images to get the same results faster and also a possible solution of computer vision for demonstration system, the robot K4 intended for Ketchup House contest.

Keywords: computer vision, image processing, object detection, sequence of images

Abstrakt

Juránek, L. *Optimalizace algoritmů pro opakované vyhledání objektu.* Diplomová práce. Brno, 2016

Cílem této práce je vytvořit metodu pro opakované vyhledání objektů v obraze. Tato metoda využívá posloupnosti snímků, čímž dosahuje vyšší rychlosti detekce objektu, než obdobné metody pracující pouze s jedním obrazem. Zvolený problém je vyřešen vytvořením demonstrační metody pro rozpoznávání objektů robotem K4, který byl vytvořen robotickým týmem PEF MENDELU AiStorm. Tato demonstrační metoda je následně vylepšena o zpracování následujících snímků z posloupnosti. Tímto vylepšením se podařilo dosáhnout zrychlení o 43 %. Hlavními přínosy práce jsou návrh možností zpracování obrazu pro detekci objektů, ukázka využití posloupnosti snímků k získání stejných výsledků rychleji, a také možné řešení počítačového vidění demonstračního systému, robota K4 určeného k soutěži Ketchup House.

Klíčová slova: počítačové vidění, zpracování obrazu, detekce objektů, posloupnost snímků

Obsah

1 Úvod a cíl práce

Tato diplomová práce se věnuje odvětví výpočetní techniky, které nazýváme strojové vidění (*machine vision*, zejména v oblasti průmyslové automatizace) nebo častěji počítačové vidění (*computer vision*). Toto odvětví úzce souvisí s počítačovou grafikou a hlavně s digitálním zpracováním obrazu (*image processing*), které dle Lima (1990) zpracovává rastrovou grafickou informaci a pomocí matematických operací (převážně metody diskrétní matematiky) a zpracování signálu (*signal processing*) pracuje s obrazem jako se signálem (většinou dvourozměrným).

Historie počítačového vidění se začala odvíjet na počátku sedmdesátých let dvacátého století, kdy počítače umožnily zpracovávat dostatečně velké množství dat k použití digitálního zpracování obrazu (v té době zejména s použitím Fourierovi transformace) (Granlund a Knutsson, 1995). Od té doby nachází uplatnění v mnoha různých oblastech a jak uvádí Šťastný (2006), již před deseti lety patřilo k nejprogresivněji se rozvíjejícím oblastem informačních technologií, což můžeme předpokládat i v současné době.

Cílem počítačového vidění je získat obraz (většinou obrazovým snímačem ve fotoaparátu či kameře), vhodně jej zpracovat (právě za použití digitálního zpracování obrazu), analyzovat takto získaná data a pochopit je, čímž získáme informace o tom, co se v obraze děje a jaké objekty se v něm vyskytují. Zjednodušeně řečeno se počítačové vidění snaží duplikovat schopnost lidského vidění a chápání dějů a pozic objektů v okolním 3D prostoru.

Oblastí souvisejících s počítačovým viděním je mnoho. Na straně jedné jsou oblasti, které nám pomáhají splnit jeho cíle, z nichž můžeme napříč různými obory jako jsou matematika, fyzika, matematická informatika či umělá inteligence jmenovat zejména statistiku, geometrii, optimalizaci, optiku, strojové učení, neuronové sítě, zpracování signálu a neurobiologii. Na straně druhé jsou naopak oblasti, které počítačového vidění využívají jako jsou medicína, automatizace v průmyslu, armádní technika, robotika a další.

1.1 Cíl a metodika práce

Cílem této práce je optimalizovat vybrané algoritmy pro opakované vyhledání objektu v posloupnosti snímků a tuto optimalizaci následně demonstrovat na vybraném reálném systému, konkrétně na mobilním robotovi K4. Pro splnění tohoto cíle je nutné provést následující kroky:

Nejprve je nutné sestavit teoretický základ pro klasifikaci (rozpoznávání) objektů v obraze, tedy rozebrat kroky digitálního zpracování obrazu a na něm postaveného počítačového vidění, které k výsledné klasifikaci vedou. Tato teorie, na níž je postavena praktická část diplomové práce, je popsána v kapitole 2.

Následujícím problémem, který je nutno rozebrat a vyřešit v rámci této práce, je vhodný výběr prostředků k praktické implementaci rozpoznávání objektů v obraze v reálném systému. Součástí řešení tohoto výběru je stručný popis nejpoužívanějších

knihoven určených pro zpracování obrazu a počítačové vidění a programovacích jazyků, které tyto knihovny podporují. Tomuto popisu a výsledku výběru prostředků, které jsou dále použity v praktické části diplomové práce, se věnuje kapitola 3.

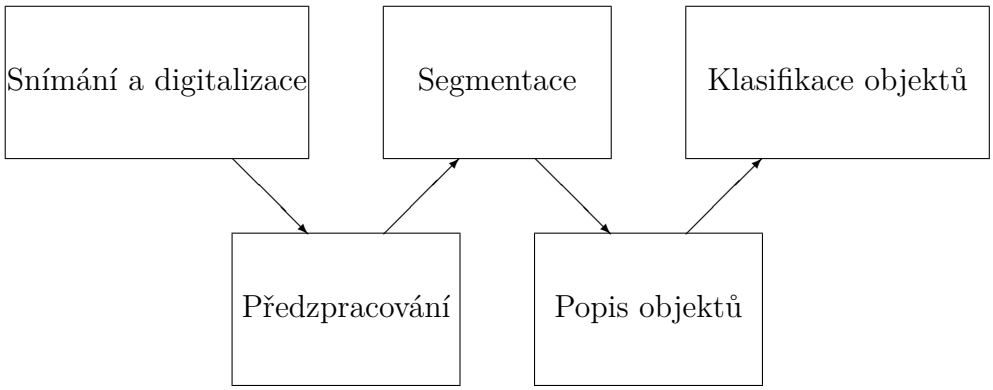
Další část práce se již zaměřuje na praktickou realizaci optimalizace algoritmů pro opakované vyhledání objektu v posloupnosti snímků, konkrétně na popis vybraného reálného systému, mobilního robota K4, který rozpoznávání objektů v obraze využívá a jehož algoritmy se v rámci této práce pokusíme optimalizovat. Tento popis systému je obsažen v kapitole 4.

Cílová optimalizace, popsána v kapitole 5, využívá poznatků získaných ze splnění kroků popsaných výše, na základě kterých provedeme návrh optimalizace algoritmů vybraného reálného systému z předchozí kapitoly a následnou implementaci pomocí zvoleného programovacího jazyka a knihovny určenou pro praktické nasazení.

Poslední část práce se věnuje testování implementovaného řešení, kdy na základě nasazení a výsledků tohoto testování je potřeba vypracovat závěrečné zhodnocení splnění cíle této diplomové práce. Tuto závěrečnou část můžeme najít v kapitole 6.

2 Zpracování obrazu

V této kapitole se budeme věnovat hlavnímu prostředku k dosažení počítačového vidění, kterým je zpracování obrazu. Toto zpracování obrazu pak omezíme pouze na rozpoznávání, jinak řečeno také klasifikaci, objektů v digitalizovaném obraze pomocí jeho zpracování. Jak píše Šťastný (2006) ve své habilitační práci, tento proces se obvykle skládá z následujících základních kroků: předzpracování, segmentace obrazu, popis objektů a klasifikace objektů. Než však můžeme přistoupit k těmto krokům, je potřeba samotný obraz získat snímáním a jeho následnou digitalizací (viz schéma na obrázku 1).



Obrázek 1: Schéma kroků ke klasifikaci objektů.

Předzpracování obrazu (*image preprocessing*) jsou metody, které nám vylepší obraz pro další zpracování. Může se jednat o odstranění vad čočky snímacího zařízení jako například zkreslení obrazu, potlačení digitálního šumu nebo jiné úpravy, které nám později pomohou a zvýrazní důležité části obrazu. Prince (2012) předzpracování obrazu obecně definuje jako jakoukoliv transformaci pixelů před stavěním modelu týkajícího se dat reálného světa, tedy odstranění co nejvíce nechtěných variací aspektů reálného světa, které nesouvisí se zadaným úkolem (v našem případě rozpoznávání objektů), jako jsou například rozlišení fotoaparátu nebo nasvícení, a zároveň ponechání všech aspektů obrazu, které jsou důležité pro další rozhodování.

Segmentace obrazu je jedním z nejdůležitějších kroků vedoucích k analýze zpracovávaných obrazových dat. Jejím hlavním cílem je rozdělit obraz na jednotlivé části, které jsou spjaty s objekty nebo oblastmi reálného světa obsaženými v obraze (Šonka, Hlaváč a Boyle, 2007). Tímto procesem odlišíme rozpoznávané objekty od zbytku obrazových dat, který nás pro další zpracovávání již nezajímá.

Předposledním krokem je popis nalezených objektů, tedy oblastí segmentovaných z předzpracovaného obrazu. Tento krok můžeme provést dvěma základními způsoby (Holota a Fiřt, 2009). Prvním způsobem je kvantitativní přístup, který spočívá v popisu objektů pomocí souboru číselných charakteristik jako je například velikost objektu. Druhým způsobem je kvalitativní přístup, což je popis relací mezi objekty a jejich tvarových vlastností. Výběr vhodného způsobu je většinou ovlivněn

použitým rozpoznávacím algoritmem v posledním kroku, který využívá tento popis jako vstupní informaci.

Nakonec můžeme provést samotnou klasifikaci, tedy rozpoznávání objektů v obrazu pomocí klasifikátoru. Ta spočívá v zařazení v obraze nalezených objektů do skupiny předem známých tříd, což je podmnožina prvků, jejichž atributy mají z hlediska klasifikace společné rysy (Šťastný, 2006).

2.1 Snímání a digitalizace

Pro naše účely není potřeba znát a podrobně rozepisovat technologie snímání obrazu reálného světa, je však dobré vědět, že snímaný obraz je vlastně analogový elektrický signál, který je spojitý v čase i úrovni, a proto nevhodný pro další zpracování na počítači. Z tohoto důvodu je nutné spojitý analogový signál digitalizovat na digitální signál, který je diskrétní, pomocí vzorkování (*sampling*) v matici $M \times N$ bodů za dodržení Shannonova teorému[1] a následnému kvantování (*quantization*) spojité jasové úrovně každého vzorku do K intervalů.

Protože vzorkování obrazu probíhá ve dvojrozměrném prostoru, je nutné také určit vhodnou vzorkovací mřížku, což je plošné uspořádání bodů při vzorkování. Většinou se používá pravidelná čtvercová mřížka, kde všechny vzorkovací body mají v digitalizovaném obraze odpovídající obrazové elementy, které se nazývají pixely, jenž pokrývají celý digitalizovaný obraz.

Dalším nutným krokem procesu digitalizace obrazu je kvantování obrazové funkce. Zatímco vzorkování se postará o zobrazení spojitého reálného obrazu v konečném množství pixelů, určuje tedy rozlišení obrazu a věrnost jeho detailů, tak kvantování určí konečné množství barev použitých v obraze. Tento počet kvantovacích úrovní musí být dostatečně veliký, aby lidské oko nepoznalo rozdíl mezi spojitou množinou barev v reálném světě a diskrétní množinou použitých barev v obraze, a tak nevznikaly falešné obrysy. Například v monochromatickém obraze je člověk schopen rozlišit přibližně padesát jasových úrovní (Hlaváč a Sedláček, 2009).

Vzhledem k tomu, že jsme místo jasových úrovní z definice kvantování obrazu použili pojmu barva, bylo by dobré si definovat, jakým způsobem je vytvořena. Způsobů, jak vytvářet barvy, je více a jsou dány barevnými modely, které většinou využívají vlastností míchání barev. Nejčastěji se jedná o trichromatické (tříbarevné) modely, a to buď aditivní (podklad je černý a barva se tvoří přidáváním barev, například RGB) nebo subtraktivní (podklad je bílý a barva se tvoří odečítáním barev od bílé, například CMY). Existuje však i mnoho barevných modelů, ve kterých barvy nevznikají pomocí míchání. Nejpoužívanějšími barevnými modely ve zpracování obrazu jsou dle Martiška (2002) RGB a HSV či jemu podobné.

Barevný model RGB se hodí zejména v případě, kdy chceme pracovat s jednotlivými barevnými komponentami, zatímco HSV barevný model nám dovoluje mnohem přesněji pracovat s jasem a sytostí. Hlavně přesné úpravy jasu mohou být

[1] Dle Shannonova teorému je ideální frekvence pro vzorkování rovna dvojnásobku maximální frekvence vyskytující se ve spojité funkci obrazu.

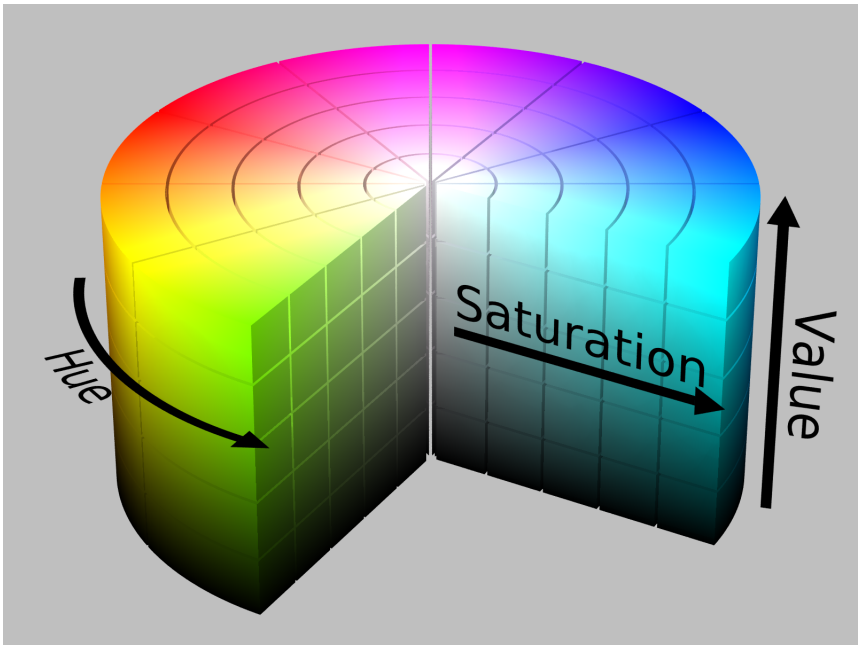
výhodné v mnoha metodách předzpracování obrazu. Pro využití výhod obou barevných modelů si dále ukážeme způsob jejich vzájemného převádění.

RGB (Red, Green, Blue)

Aditivní trichromatický a pravděpodobně nejrozšířenější barevný model, kde každý pixel je reprezentován trojicí barev, červenou, zelenou a modrou. V nejznámější, 24-bitové, reprezentaci, kde každá ze tří základních barev je popsána na osmi bitech je $(0, 0, 0)$ barvou černou a $(255, 255, 255)$ barvou bílou.

HSV (Hue, Saturation, Value)

Odstín (*hue*) je barvou tak, jak ji běžně chápeme (růžová, žlutá, atp.) a vyjadřuje se většinou ve stupních, určujících úhel na kole barevných odstínů, tedy od 0° do 360°. Sytost (*saturation*) měří jak blízko je barva ke stupňům šedi, kde 0 znamená bílou, šedou či černou barvu a 1 je čistá barva dle odstínu. A nakonec jas (*value*) udává hodnotu pixelu, kolik světla barva odráží, světelnost. Nejvyšší světelnost 1 znamená barvu zářivou, intenzivní. Nejmenší světelnost 0 pak znamená barvu nejtmavší, konkrétně černou. Ukázku grafického zobrazení barevného modelu HSV mapovaného do podoby válce můžeme vidět na obrázku 2.



Obrázek 2: Barevný model HSV v podobě válce (Isometric Land, 2016).

Barvu určenou barevným modelem RGB můžeme jednoduše převést do modelu HSV pomocí následujících matematických vzorců (platí pro 24-bitovou reprezentaci)[2]:

Nejdříve si vypočítáme pomocné hodnoty M a m:

$$M = max\{R, G, B\}$$

$$m = min\{R, G, B\}$$

Jas je poté definován jako:

$$V = M/255$$

Sytost je 0, v případě, že $M = 0$, jinak:

$$S = 1 - m/M$$

A pokud $G \geq B$, tak odstín se vypočítá:

$$H = cos^{-1}(\frac{R - \frac{1}{2}G - \frac{1}{2}B}{\sqrt{R^2 + G^2 + B^2 - RG - RB - GB}})$$

jinak:

$$H = 360 - cos^{-1}(\frac{R - \frac{1}{2}G - \frac{1}{2}B}{\sqrt{R^2 + G^2 + B^2 - RG - RB - GB}})$$

Z hodnot HSV můžeme také zpětně vypočítat hodnoty barevných komponent RGB barevného modelu, pomocí následujících vzorců:

Nejdříve si vypočítáme pomocné hodnoty M, m a z:

$$M = 255V$$

$$m = M(1 - S)$$

$$z = (M - m)[1 - |(H/60°)\, mod\, 2 - 1|]$$

Jednotlivé barevné komponenty R, G a B jsou pak definovány jako:

$$(R, G, B) = \begin{cases} (M, z\ + m, m) & \text{pokud } 0° \leq H < 60° \\ (z\ + m, M, m) & \text{pokud } 60° \leq H < 120° \\ (m, M, z\ + m) & \text{pokud } 120° \leq H < 180° \\ (m, z\ + m, M) & \text{pokud } 180° \leq H < 240° \\ (z\ + m, m, M) & \text{pokud } 240° \leq H < 300° \\ (M, m, z\ + m) & \text{pokud } 300° \leq H < 360° \end{cases}$$

[2]Při sestavování vzorců pro převod bylo čerpáno zejména z online převodní kalkulačky Had2Know dostupné z:
http://www.had2know.com/technology/hsv-rgb-conversion-formula-calculator.html

2.2 Předzpracování obrazu

Jak bylo zmíněno výše, metod předzpracování obrazu je celá řada, dají se libovolně kombinovat a často se používají spíše jako ad hoc heuristiky na základě zkušeností nežli jako výsledek exaktního postupu aplikovaného na poznatky získané z trénovacích dat. Mezi základní a často používané metody předzpracování obrazu však můžeme zahrnout převedení na stupně šedi (*grey scale transformation*), změnu jasu a kontrastu (*brightness correction*), ekvalizaci histogramu (*histogram equalization*), ostření obrazu (*image sharpening*) a filtraci (*filtration*).

Převedení na stupně šedi

Tato metoda převede všechny pixely barevného obrazu na ekvivalent ve stupních šedi odstraněním informací o odstínu a sytosti a ponecháním pouze jasu, což v RGB barevném modelu odpovídá nahrazení hodnot všech barevných komponent jedinou hodnotou, kterou získáme spočítáním vážené sumy R, G a B komponent (pseudokód 1). Váhy ve vážené sumaci odpovídají citlivosti lidského oka na jednotlivé barvy a mohou se v různých implementacích mírně lišit. Knihovny skriptovacího programovacího jazyka MATLAB (MathWorks, 2016) používají následující vzorec: $0,2989 \times R + 0,5870 \times G + 0,1140 \times B$.

```
int x, y;
RGB color;
unsigned char intensity;
for (y = 0; y < frameHeight; y++) do
    for (x = 0; x < frameWidth; x++) do
        getPixel(x, y, color);
        intensity = round(color.red * 0.2989 + color.green * 0.5870 +
        color.blue * 0.1140);
        color.red = intensity;
        color.green = intensity;
        color.blue = intensity;
        putPixel(x, y, color);
    end
end
```

Algorithm 1: grey scale transformation

Změna jasu a kontrastu

Zvýšení či snížení jasu provádíme zesvětlováním, respektive ztmavováním barvy jednotlivých pixelů. To zajistíme změnou hodnoty pixelu, v RGB barevném modelu nejjednodušeji zvýšením či snížením hodnot všech barevných komponent o stejný koeficient změny jasu. Následující algoritmus 2 zvýší jas celého obrazu o hodnotu

10, v praxi je samozřejmě nutné také ošetřit přetečení či podtečení hodnot barevných komponent. Pokud bychom chtěli provést změnu jasu složitějším, ale přesnějším způsobem, musíme převést obraz do barevného modelu HSV, změnit samotný jas dle požadavků a provést zpětný převod do RGB barevného modelu (viz výše v podsekci HSV sekce snímání a digitalizace).

```
int x, y;
RGB color;
unsigned char brightnessCoefficient = 10;
for (y = 0; y < frameHeight; y++) do
    for (x = 0; x < frameWidth; x++) do
        getPixel(x, y, color);
        color.red += brightnessCoefficient;
        color.green += brightnessCoefficient;
        color.blue += brightnessCoefficient;
        putPixel(x, y, color);
    end
end
```

Algorithm 2: brightness correction

Obrázek 3: Ukázka zvýšení jasu a kontrastu; vlevo původní obrázek, uprostřed zvýšení jasu a vpravo zvýšení kontrastu.

Pokud chceme dosáhnout vyššího kontrastu v obraze, je nutné zvětšit rozdíly jasu mezi jednotlivými pixely. Toho dle Martiška (2002) nejjednodušeji dosáhneme vynásobením všech hodnot pixelů koeficientem změny kontrastu. Použijeme tedy stejný algoritmus jako pro změnu jasu, ale místo operace sčítání použijeme operaci násobení. Pokud bude koeficient změny kontrastu větší než jedna, kontrast se zvýší, koeficient změny kontrastu menší než jedna kontrast snižuje. Musíme mít na paměti, že změnou kontrastu se mění hodnoty všech pixelů v obraze, čímž se také mění celkový jas. Proto po změnách kontrastu často následují i úpravy jasu, který celkový

jas opětovně srovnají do přibližně původní hodnoty (pokud jsme kontrast zvyšovali, jas snížíme a naopak). Názornou ukázku změny jasu a kontrastu si můžeme prohlédnout na obrázku 3.

Ekvalizace histogramu

Histogram obrazu je grafické znázornění četností hodnot jednotlivých pixelů pomocí sloupcového grafu, který nám umožňuje získat představu o zastoupení světlých a tmavých pixelů v obrazu. Pomocí něj můžeme vylepšit kontrast obrazu, který se skládá z pixelů buď příliš světlých a/nebo naopak příliš tmavých, přičemž barevné odstíny mezi těmito extrémy téměř zcela chybí, a to zajištěním rovnoměrného zastoupení světlých, středních a tmavých barevných odstínů. Tento problém se často objevuje například u fotografií, které jsou pořizované proti světlu a nedá se řešit klasickými způsoby změn jasu a kontrastu.

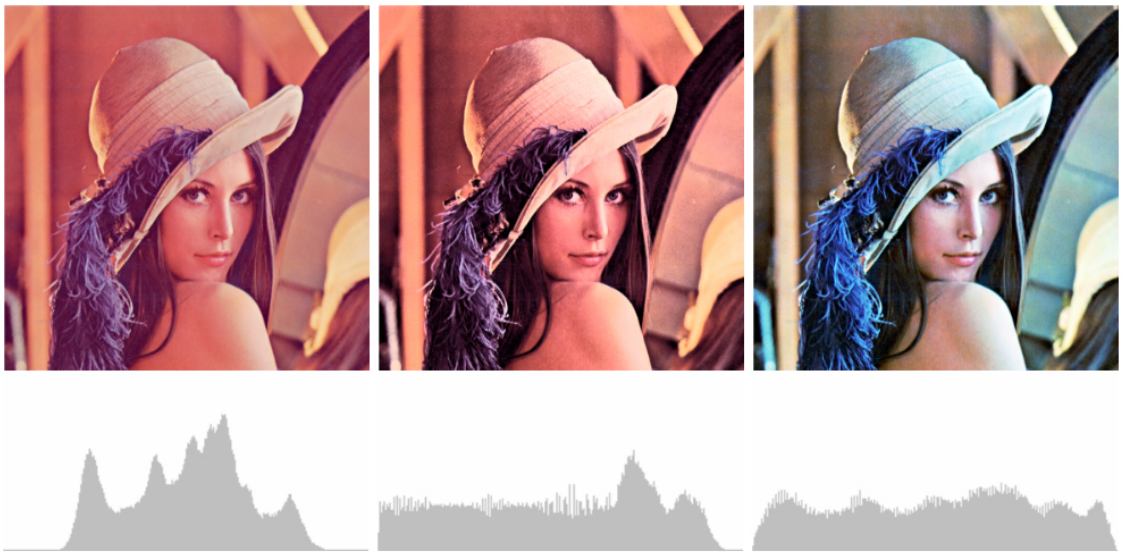
```
int x, y;
unsigned char value;
for (y = 0; y < frameHeight; y++) do
    for (x = 0; x < frameWidth; x++) do
        getPixel(x, y, value);
        if (value != 0) then
            value = round((255 * sumHistogram[value]) / (frameHeight *
            frameWidth));
        end
        putPixel(x, y, value);
    end
end
```

Algorithm 3: histogram equalization

Důvodem je to, že obraz jako celek má sice kontrast velmi vysoký, ale samostatně mají světlé i tmavé odstíny kontrast naopak velmi nízký a zvyšováním kontrastu klasickým způsobem (viz předchozí podsekce) ho však paradoxně ještě více snižujeme (Martišek, 2002). Jeho zvyšováním totiž zesilujeme světlé tóny do ještě světlejších a tmavé tóny do ještě tmavších. Nepomůže ani změna jasu, protože jeho zvyšování ořezává světlé barvy a jeho snižování ořezává tmavé barvy. Je tedy nutné změnit jas jednotlivých pixelů tak, aby histogram byl blízký rovnoměrnému rozložení a obraz byl tak téměř zcela jasově vyvážen (úplného dosažení rovnoměrného rozložení většinou v praxi není možné kvůli diskrétnímu charakteru obrazu po kvantování).

Pseudoalgoritmus 3 ekvalizace histogramu představuje jasové vyvážení a využívá předem připravený kumulativní histogram jasových hodnot k nahrazení každého pixelu za pixel o hodnotě $f_i = round(max \times c_i)$, kde c_i je relativní kumulativní četnost pixelu a max je jeho nejvyšší možná hodnota. Takto spočítanou hodnotu poté

musíme vydělit celkovým počtem pixelů v obrazu. Ukázka ekvalizace histogramu je pak na obrázku 4.



Obrázek 4: Ukázka ekvalizace histogramu s histogramy jasu; vlevo původní obrázek, uprostřed ekvalizace jasu a vpravo ekvalizace všech barevných komponent.

Ostření obrazu

Kontrastnějšího obrazu můžeme dosáhnout i ostřením obrazu, které se dá vnímat jako zdůraznění vysokých frekvencí ve frekvenčním spektru (Hlaváč a Sedláček, 2009). Zaostření obrazu dosáhneme tím, že zvýrazníme hrany, tedy místa, kde se obrazová funkce strmě mění. Takto upravený obraz, který obsahuje strmější hrany, se dá získat tak, že od hodnoty každého pixelu odečteme hodnoty pixelů sousedních. Pro získání tohoto výsledku můžeme použít matematického operátoru zpracovávajícího dvě funkce zvaného konvoluce, kde jednou funkcí je původní obraz a druhou nějaký filtr (více v následující podsekci filtrace).

Pomocí konvoluce chceme vypočítat novou hodnotu pixelu ve výstupním obraze $g(x, y)$ jako lineární kombinaci hodnot vstupního obrazu f v malém okolí O právě zpracovávaného pixelu (x, y). Příspěvku jednotlivých pixelů v okolí O, který je vážen koeficienty h odpovídá rovnice diskrétní konvoluce s jádrem h, kterému se říká konvoluční maska (v některé odborné literatuře také kernel nebo konvoluční matice) (Hlaváč a Sedláček, 2009):

$$f(x, y) = \sum_{(m,n)} \sum_{\in O} h(x - m, y - n)g(m, n)$$

Většinou se používá obdélníkové okolí O v podobě 3×3 matice, kde právě zpracovávaný pixel leží uprostřed konvoluční masky. V případě ostření obrazu se používají

následující konvoluční masky (jedna používá jako sousední pixely 4-okolí, druhá 8-okolí):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Filtrace

Filtrací myslíme obecný název pro metody transformace obrazu, které jej procházejí pixel po pixelu a postupně mění jejich hodnotu (jas) na základě hodnot pixelů v okolí. Ostření obrazu popsané v minulé podsekci je jednou z metod filtrace a stejně tak i v ostatních se požadované transformace dosahuje pomocí konvoluce, jejímž vstupem je obraz a filtr v podobě konvoluční masky. Dle účelu se metody filtrace mohou rozdělit do dvou skupin (ukázky na obrázku 5).

Obrázek 5: Ukázka vybraných lineárních filtrací; vlevo identita $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, uprostřed Laplaceův operátor pro detekci hran $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ a vpravo Gaussovo vyhlazování $\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

První skupinou je vyhlazování, které se snaží o potlačení šumu. Druhou skupinou je detekce hran (někdy zvané též gradientní operátory), které se snaží o odhad derivace obrazové funkce z hodnot v okolí právě zpracovávaného pixelu. Principiálně by se metody vyhlazování daly považovat za dolnofrekvenční propust[3] a metody

[3]Filtr přenáší nízké frekvence a potlačuje vysoké frekvence. Šum většinou mívá široké frekvenční pásmo, a tak je omezením vysokých frekvencí dolní propustí potlačen. Nicméně spolu se šumem mizí všechny jevy odpovídající vysokým frekvencím jako jsou hrany či tenké čáry a obraz je celkově rozmazán.

detekce hran (mezi které patří také ostření obrazu) za hornofrekvenční propust[4].

Tyto metody vypočítávají novou hodnotu zpracovávaného pixelu pomocí lineárních kombinací hodnot, jsou tedy nazývány lineární filtrace. U ní si můžeme všimnout, že vyhlazování a detekce hran fungují opačně, proto existují i nelineární metody, pomocí kterých můžeme dosáhnout například vyhlazení šumu a přitom zachovat hrany a detaily v obraze. Nelineární metody filtrace také nahrazují hodnotu zpracovávaného pixelu dle hodnot pixelů v okolí, avšak nikoliv jejich lineární kombinací, ale jinou funkcí. Nejčastěji používanými funkcemi jsou minimální hodnota pixelů v okolí, která zmenšuje (tzv. eroduje) světlé objekty a zvětšuje (tzv. dilatuje) tmavé objekty, a maximální hodnota pixelů v okolí, která naopak eroduje tmavé objekty v obraze a dilatuje světlé (Martišek, 2002).

2.3 Segmentace obrazu

Při procesu segmentace obrazu se dá zaměřit na dva její typy, kompletní segmentaci a částečnou segmentaci. Výstupem kompletní segmentace je množina oddělených oblastí přesně odpovídajících objektům ve vstupním obraze, čehož však často nelze dosáhnout bez použití dalších metod zpracování obrazu na výstup částečné segmentace. Výstupem částečné segmentace jsou oddělené homogenní oblasti s ohledem na jas, barvu, odrazivost, kontext, atp.

Základní metody segmentace se dají rozdělit do tří skupin, globální znalosti o obrazu nebo jeho částech (většinou na základě histogramu, například prahování (*thresholding*)), segmentaci na bázi hran (například detekce hran (*edge detection*)) a segmentaci na bázi oblastí (například segmentace narůstáním oblastí (*region-growing methods*)).

Prahování

Prahování je jednoduchý segmentační postup, který konvertuje obraz ze stupňů šedi (případně barev) na monochromatický obraz (obsahující pouze bílé a černé pixely) na základě předpokladu, že pixely s podobným jasem v blízkém okolí obvykle patří ke stejnému objektu, protože povrch objektu má konstantní odrazivost či pohltivost světla. Výsledný obraz by měl obsahovat všechny důležité informace o počtu, pozici a tvaru objektů, ale nikoliv další nedůležité informace, a tak by měl zjednodušovat další kroky vedoucí k rozpoznávání a klasifikaci objektů.

Nejběžnějším postupem prahování je volba jedné jasové konstanty (prahu). Všechny jasové úrovně pod touto konstantou se změní na černou, tedy hodnotu 0, a všechny nad ní na bílou, tedy hodnotu 1. Problém nastává v určení přesné hodnoty této konstanty, a to zejména pokud ji chceme určit algoritmicky. Jednoduchou metodou je tzv. *p-tile* metoda, u které je nutné určit, kolik procent obrazu by po

[4]Filtr potlačuje nízké frekvence, a tak relativně zesiluje vysoké frekvence. Tím se v obrazu zvýrazní místa, kde se náhle mění obrazová funkce, jako jsou hrany.

prahování mělo být zaplněno bílými a kolik černými pixely. To můžeme udělat v případě, že dokážeme odhadnout, jak velkou plochu obrazu objekty zabírají (například pokud rozpoznáváme tištěný text na papíře). Dle histogramu obrazu pak můžeme určit práh.

Obecně je používání histogramu k určení práhu běžné a v některých případech se z něj samotného dá vhodný práh určit. Jeden z takových případů nastává, pokud se histogram skládá ze dvou vrcholů a nízkého bodu mezi nimi. Určení prvního vrcholu je snadné, stačí najít nejvyšší hodnotu v histogramu, najít druhý vrchol však není tak triviální (druhá nejvyšší hodnota pravděpodobně stále patří prvnímu vrcholu) a podle Parkera (2011) pomáhá vynásobit hodnoty histogramu druhou mocninou vzdálenosti od prvního vrcholu, takže upřednostníme vrcholy, které jsou vzdálenější.

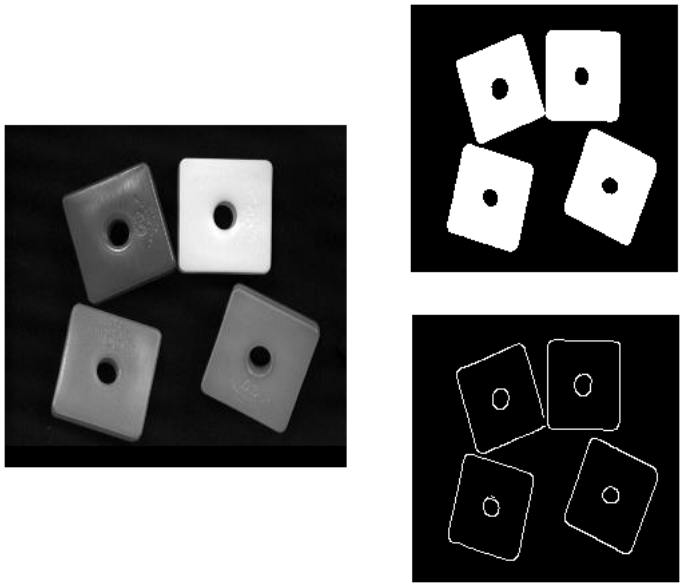
Detekce hran

Můžeme předpokládat, že hranice rozpoznávaných objektů, které oddělují objekt od pozadí, se skládají z hran, kde je výrazná změna obrazové funkce (od černé po bílou), tedy vysoká frekvence ve frekvenčním spektru obrazu (získaného například Fourierovou transformací). Již v sekci předzpracování obrazu jsme si popsali filtry, které principiálně fungují jako hornofrekvenční propust a v podsekci filtrace jsme je pojmenovali jako metody pro detekci hran (používané také pro ostření obrazu). Konvoluční masky používané v těchto metodách filtrace se nazývají také gradientní operátory a jejich aplikace je základem pro většinu metod segmentace na základě detekce hran. Právě gradientem se dá popsat výše zmíněná změna obrazové funkce, jelikož gradient je v matematice diferenciální operátor, který je směrem růstu. Hrany jsou pak kolmice na směr gradientu (Šťastný, 2006).

Použití lineárních metod filtrace pro detekci hran (tedy aplikace gradientních operátorů) však v praxi pro segmentaci nestačí. Používají se složitější algoritmy, které obsahují další kroky, jako například Cannyho hranový detektor (obrázek 6), který před určením gradientu provádí také eliminaci šumu Gaussovým filtrem (lineární filtr s konvoluční maskou vycházející z podoby Gaussova rozdělení pravděpodobnosti) a po něm pak nalezení lokálních maxim pro nalezení největšího gradientu a nakonec prahování a eliminaci nevýznamných hran (Moeslund, 2009). Využití gradientních operátorů však není jedinou možností jak detekovat hrany. V případech, kdy známe tvar a velikost hledaného objektu, můžeme použít Houghovu transformaci (metoda pro nalezení parametrického popisu objektů v obraze, zejména přímek, kružnic nebo elips).

Narůstání oblastí

Tyto metody si kladou za cíl rozčlenit obraz do co největších souvislých homogenních oblastí z hlediska zvoleného kritéria homogenity, například jasu nebo barvy (Umbaugh, 2010). Jednou z konkrétních metod tohoto typu segmentace je spojování oblastí, kde se na počátku rozdělí obraz do množiny malých oblastí (nejpřesnější



Obrázek 6: Ukázky způsobů segmentace, vlevo původní obraz, vpravo nahoře vyprahován, vpravo dole použití Cannyho hranového detektoru.

je rozdělit obraz po pixelech), následně se definuje kritérium homogenity, podle kterého se budou oblasti spojovat (například rozdíl jasů dvou sousedních oblastí musí být menší než 5), a nakonec se postupně projde celý obraz a dokud to lze, tak se spojují sousední oblasti vyhovující kritériu (polohu sousedů určuje maska, většinou 4-okolí nebo 8-okolí). Je dobré zmínit, že tato metoda segmentace poskytuje různé výsledky nejen dle počátečního nastavení kritéria homogenity a volby počátečních oblastí, ale také dle způsobu procházení obrazem.

Druhou metodou segmentace narůstáním oblastí je štěpení oblastí, která má opačný základní princip než spojování oblastí. Na počátku se začíná s celým obrazem, který se následně dělí na menší oblasti pokud nesplňují zvolené kritérium homogenity. Tato metoda poskytuje jiné výsledky než spojování oblastí a v praxi se používají metody využívající oba principy zároveň.

2.4 Popis objektů

Nalezené segmenty, které již považujeme za objekty, musíme vhodným způsobem popsat. Tímto popisem máme na mysli získání příznaků z objektu, které musí přesně odpovídat jeho charakteristickým rysům, a zároveň splňovat praktické požadavky, jako jsou například invariantnost vůči posunu, změně jasu, rotaci a případně i změně měřítka. U kvantitativního přístupu k popisu objektů se k tomuto účelu používá tzv. příznakový vektor (*feature vector*). Ten obsahuje příznaky dle metody jeho výpočtu ze segmentovaných dat a pro příklad jimi mohou být velikost, obvod či průměrná jasová úroveň.

Metod pro výpočet příznakového vektoru je mnoho, jedním z nich je popis objektů dle ramen objektu. V této metodě popisu, jak píše Šťastný (2006), odpovídají jednotlivé složky příznakového vektoru délkám ramen vektorů v polárním souřadnicovém systému s počátkem v těžišti právě prohledávaného objektu. Takto získaný vektor dostatečně dobře charakterizuje tvar objektu a umožňuje popis i složitějších tvarů. U kvalitativního přístupu popisu je místo toho objekt rozdělen na primitiva, což mohou být např. přímky, úsečky, oblouky, křivky a podobně. Těmto primitivům jsou přiřazeny terminální symboly, vztahy mezi nimi jsou pak popsány jednorozměrnými relacemi, tedy řetězci terminálních symbolů, což je slovo. Množina slov popisujících objekt pak tvoří jazyk třídy objektu, který je generován gramatikou.

Detekce příznaků

Dalšími způsoby jak získat příznaky objektu je využití příznakových detektorů (*feature detectors*). Takto jsou získávány například příznaky hran (pro tento účel můžeme využít již zmíněný Cannyho hranový detektor), rohů (*corners*) nebo blobů[5]. Roh je místo v hraně, kde se prudce změní její směr, a je podmnožinou tzv. významných bodů (*interest points*), což mohou být například izolované body s jinou hodnotou jasu než má okolní oblast, tedy body, které se výrazně liší od svého okolí. Celé oblasti významných bodů jsou pak označovány jako bloby. Pro detekci rohů můžeme použít například Moravcův algoritmus, který sleduje podobnosti zkoumaného pixelu s okolními pixely pomocí malých čtvercových oblastí v okolí. Jako blob detektor pak je možno použít například hessián, jak je uvedeno v disertační práci aplikace moderních metod pro identifikaci obrazových dat[6] (Kolomazník, 2014).

Kolomazník (2014) dále uvádí, že v dnešní době nejznámější a nejúspěšnější metody tohoto typu jsou algoritmy SURF a SIFT, které nejenže vyhledávají v obraze významné body a další příznaky (ilustrační ukázka na obrázku 7), ale také z jejich okolí následně sestavují jednoznačný deskriptor. Díky tomu je možné pomocí těchto algoritmů identifikovat objekty v obraze bez předchozí segmentace.

Scale-invariant feature transform (SIFT)

Tento algoritmus se snaží rozpoznat objekt tím, že hledá v obraze velké množství příznakových vektorů, které jsou jednoznačně identifikovány. Z nich je poté vytvořen deskriptor, který je pro identifikaci objektu srovnáván s již vytvořenými deskriptory v databázi tříd pomocí nalezení jejich nejmenší euklidovské vzdálenosti (více o klasifikaci objektů v následující sekci). Algoritmus SIFT hledá příznakové vektory ve čtyřech fázích (Lowe, 2004).

[5]Pro tento termín neexistuje v odborné literatuře ustálený český překlad, uvádím tedy anglický termín.

[6]Autor uvádí, že Hessova čtvercová matice se skládá z druhých parciálních derivací skalární funkce a popisuje lokální zakřivení funkce, čehož se využívá pro analýzu lokálních extrémů obrazové funkce.

Obrázek 7: Modelový příklad srovnání nalezených příznaků na dvou stejných obrazech pomocí metody SURF.

První fází (*Scale-space extrema detection*) je prohledání celého obrazu ve snaze nalézt všechny vůči velikosti a rotaci invariantní potenciální významné body, což jsou maxima v derivačním obraze, který je získán pomocí metody DoG[7]. Druhou fází (*Keypoint localization*) je odstranění nestabilních kandidátů, jako jsou málo kontrastní či na hranách se vyskytující body, pomocí prozkoumání okolí každého příznaku. Třetí fází (*Orientation assignment*) je přiřazení směru zbylým příznakům pomocí gradientů a čtvrtou fází (*Keypoint descriptor*) je výpočet výsledného deskriptoru ze získaných gradientů.

Speeded up robust features (SURF)

Algoritmus SURF je následníkem algoritmu SIFT, má stejný cíl, nalézt vůči velikosti a rotaci invariantní příznakové vektory, z kterých bude vypočítán deskriptor pro klasifikaci objektů, ale dosahuje rychlejších výsledků. Principiálně funguje podobně jako SIFT. Z jeho popisu vyplývá (Bay, Ess, Tuytelaars a Gool, 2006), že největším rozdílem je první fáze (*Scale-space extrema detection*), kde se vypočítá integrální obraz[8] a následně se detekují významné oblasti, bloby, pomocí hessiánu. Dalším velkým rozdílem je pak určení směrů příznaků pro výpočet výsledného deskriptoru

[7] *Difference of Gaussian*, jedná se o rozdíl dvou obrazů vyhlazených Gaussovým vyhlazováním s různou standardní odchylkou Gaussovy distribuční funkce.

[8] Způsob digitální reprezentace obrazu tak, že každý pixel x představuje součet hodnot předchozích pixelů doleva a nahoru. Tedy pravý spodní pixel obsahuje součet všech pixelů obrázku.

v závěrečných fázích, kde jsou místo směrů určených gradienty použity odezvy na Haarovu vlnku[9] (*Haar-wavelet responses*).

2.5 Klasifikace objektů

Posledním krokem je klasifikace (rozpoznávání). Dle Holoty a Fiřta (2009) se ve většině případů jedná o zařazení objektů nalezených v obraze do skupiny předem známých tříd. Stejně jako se popis objektů dělí do dvou základních způsobů, kvantitativního a kvalitativního, tak i klasifikace se odpovídajícím způsobem dělí do dvou skupin, rozpoznávání příznakově popsaných objektů a strukturálního rozpoznávání. Příznakově popsané objekty většinou popisuje příznakový vektor a rozpoznávání může probíhat například na principu hledání minimální vzdálenosti dvou vektorů. Mezi základní a nejčastější (Šťastný, 2006) metody patří Hammingova vzdálenost, která hledá rozdíly mezi jednotlivými vektory.

Strukturální rozpoznávání klasifikuje objekty popsané kvalitativním popisem (někdy také nazývány syntakticky popsané objekty). Toto rozpoznávání je založeno na principu rozboru slova (řetězce jazyka) a kontroly správnosti syntaxe pro všechny třídy. Jeho úkolem je určit, zda gramatika může generovat objekt, a tedy analyzovaný objekt odpovídá objektům dané gramatiky. Šťastný (2006) dále píše, že nejjednodušším způsobem tohoto rozpoznávání je tzv. porovnávání se vzorem, kdy řetězec reprezentující objekt je porovnáván s prvky množiny řetězců, představující jednotlivé vzorové objekty. Je-li však pro rozpoznání potřeba úplný popis objektu, je nutná syntaktická analýza, a to buď shora dolů nebo zdola nahoru. Jinými způsoby klasifikace objektů jsou pak například učící algoritmy typu neuronových sítí.

[9] Vlnka (*wavelet*) je funkce používaná k rozkladu signálu vlnkovou transformací k získání jeho časově-frekvenčního popisu.

3 Knihovny pro zpracování obrazu

K implementaci aplikace, která obsahuje počítačové vidění nebo zpracování obrazu je vhodné využít knihovny, které k tomuto účelu vznikly. V této práci se omezíme pouze na otevřený software (*open-source software*) popřípadě volná díla (*public domain*), a to zejména knihovnu OpenCV ve spojení s programovacím jazykem Python, která bude použita v praktické části. Stejně tak je vhodné popsat knihovnu BoofCV a program ImageJ, protože je využívá (v programovacím jazyce Java) reálný systém, který je optimalizován v rámci praktické části této práce.

3.1 OpenCV

Multiplatformní knihovna pro manipulaci s obrazem OpenCV (*Open-source Computer Vision*) je vydána pod BSD licencí a tudíž je zdarma pro akademické i komerční použití. Má rozhraní pro programovací jazyky C++, C, Python a Javu a podporuje operační systémy Windows, Linux, Mac OS, iOS a Android. Knihovna OpenCV je zaměřena především na počítačové vidění a zpracování obrazu v reálném čase. Napsána je v optimalizovaném C/C++ a může také využívat výhod paralelního zpracování na více-jádrových procesorech. V kombinaci s OpenCL navíc může využívat i výhod hardwarové akcelerace dalších výpočetních platforem jako jsou například grafické karty[10].

OpenCV je označováno za univerzální nástroj počítačového vidění. Má širokou škálu modulů, které mohou pomoci s mnoha problémy počítačového vidění. Jednou z nejužitečnějších částí OpenCV je však podle (Brahmbhatt, 2013) jeho architektura a správa paměti. Ta poskytuje framework, v kterém můžeme pracovat s obrazy a videem jakýmkoliv způsobem, ať už s využitím algoritmů OpenCV nebo vlastních, bez dalších starostí s alokací a dealokací paměti pro obrazy.

3.2 BoofCV

BoofCV je Javová knihovna s otevřeným kódem určená pro počítačové vidění v reálném čase a robotické aplikace. Je napsaná od nuly pro snadné použití a vysoký výkon. Její funkcionalita pokrývá širokou škálu disciplín včetně optimalizovaných nízkoúrovňových funkcí pro zpracování obrazu, kalibrace kamery, detekce/sledování příznaků, SFM (*structure-from-motion*) a rozpoznávání. BoofCV je vydáno pod Apache 2.0 licencí pro akademické i komerční použití.

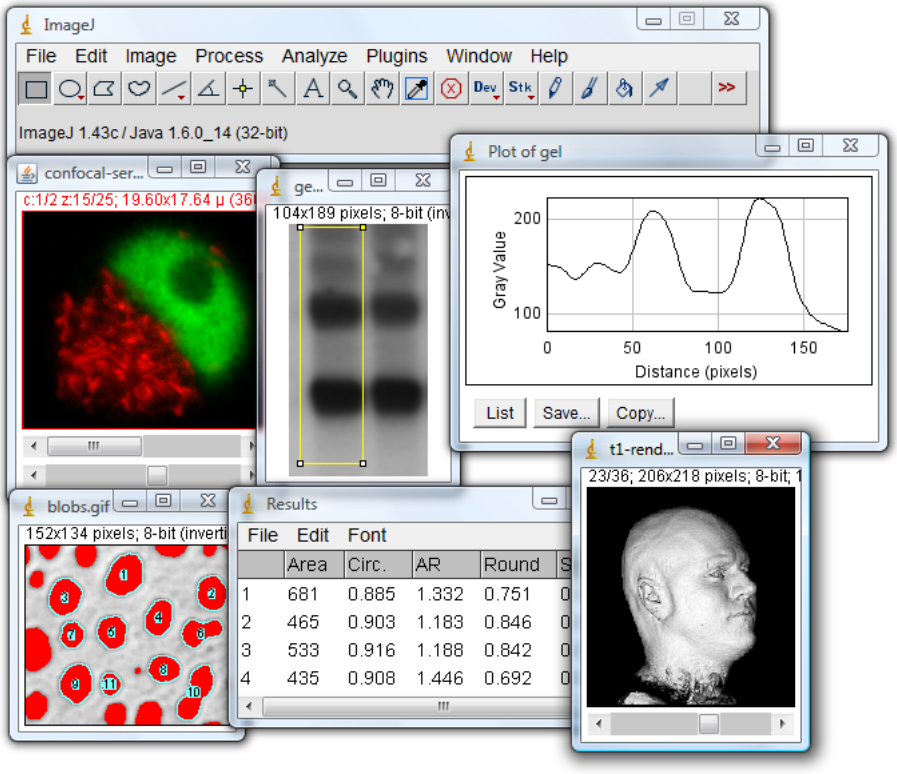
BoofCV je organizováno do několika softwarových balíčků: zpracování obrazu (*image processing*), příznaky (*features*), geometrické vidění (*geometric vision*), ka-

[10]Informace vychází z dokumentace knihovny OpenCV dostupné na http://opencv.org/

librace (*calibration*), rozpoznávání (*recognition*), vizualizace (*vizualize*) a IO. Balíček s názvem zpracování obrazu obsahuje běžně používané funkce pro zpracování obrazu, které pracují přímo s pixely. Příznaky obsahují algoritmy pro získávání příznaků pro použití ve vysokoúrovňových operacích. Kalibrace má funkce pro určení vnitřních a vnějších parametrů kamery. Rozpoznávání je pro rozpoznávání a sledování komplexních objektů v obrazu. Geometrické vidění je složeno z funkcí pro zpracování získaných příznaků obrazu za pomoci využití 2D a 3D geometrie. Softwarový balíček vizualizace má funkce pro renderování a zobrazení získaných příznaků obrazu. IO má vstupní a výstupní funkce pro různé datové struktury[11].

3.3 ImageJ

ImageJ je public domain Javový program pro zpracování obrazu, který je inspirován programem NIH Image pro Macintosh. Běží buď jako online applet nebo jako stáhnutelná aplikace (ukázka uživatelského rozhraní na obrázku 8) na jakémkoliv počítači s nainstalovanou Javou verze 1.4 a novější. Stáhnutelná distribuce je dostupná pro operační systémy Windows, Mac OS, Mac OS X a Linux.



Obrázek 8: Ukázka uživatelského rozhraní programu ImageJ (ImageJ, 2016).

[11]Informace vychází z dokumentace knihovny BoofCV dostupné na http://boofcv.org/

Tento program umí zobrazit, upravit, analyzovat, zpracovat, uložit a vytisknout 8-bitové, 16-bitové a 32-bitové obrazy, a to v mnoha různých obrazových formátech včetně TIFF, GIF, JPEG, BMP, DICOM, FITS a RAW. Podporuje tzv. zásobníky, sérii obrazů, které sdílejí jedno okno. ImageJ je napsaný multivláknově, takže časově náročné operace jako je čtení souboru s obrazem mohou být prováděny paralelně s jinými operacemi.

ImageJ umí vypočítat plochu a statistiku hodnot pixelů v uživatelem definovaných výběrech z obrazu. Také umí měřit vzdálenosti a úhly či vytvářet histogramy a grafy funkcí. Podporuje také standardní funkce pro zpracování obrazu jako jsou úpravy kontrastu, ostření, vyhlazování, detekce hran či mediánové filtrování. Pomocí ImageJ můžeme provádět geometrické transformace s obrazem jako jsou změna velikosti, rotace či překlápění. K dispozici máme i lupu, která nám dovoluje obraz až 32-krát zvětšit či zmenšit, nebo podporu více oken (obrazů) současně, která je limitována pouze dostupnou pamětí.

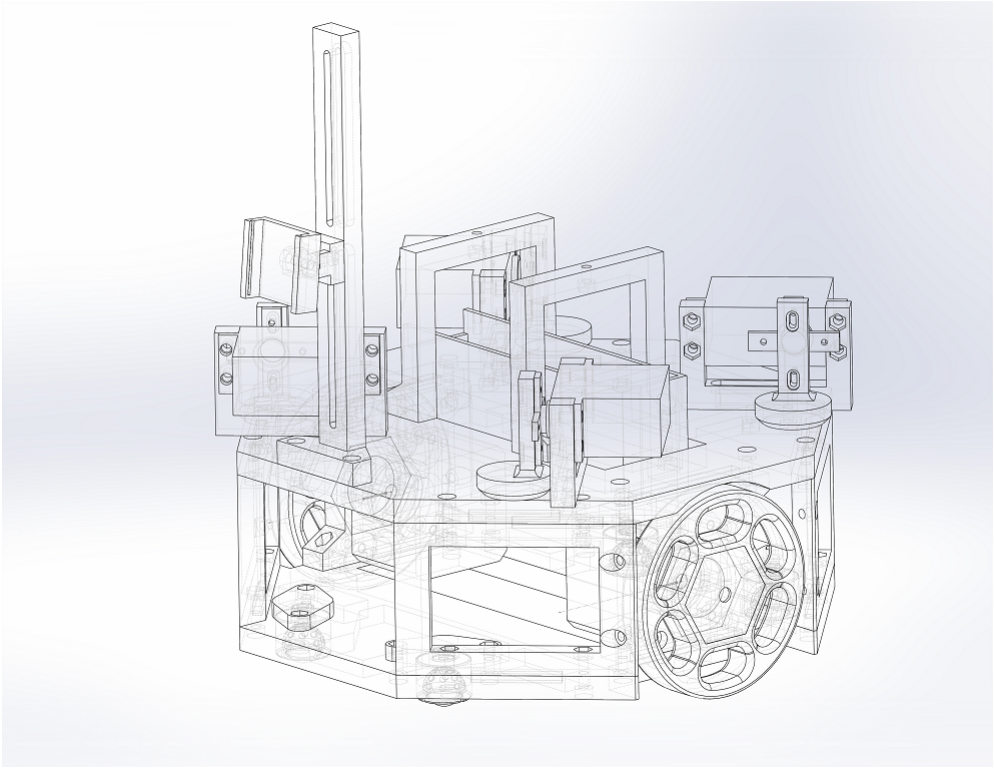
ImageJ byl navržen s otevřenou architekturou, která poskytuje rozšiřitelnost pomocí Javových pluginů. Přídavné pluginy mohou být vyvíjeny pomocí editoru a kompilátoru Javy, který je přímo zabudován v programu. Pomocí uživatelem napsaných pluginů je tak možné vyřešit téměř jakýkoliv problém zpracování obrazu nebo analýzy obrazu[12].

[12]Informace vychází z dokumentace programu ImageJ dostupné na http://imagej.nih.gov/ij/

4 Popis systému použitého jako demonstrace optimalizace

Optimalizaci rozpoznávání objektů, která je cílem této práce, demonstrujeme na mobilním robotovi s označením K4, což je indoorový robot, který byl sestaven za účelem soutěžení na každoročně pořádané soutěži Ketchup House v rámci akce Robotický den v Praze. Tento robot byl vytvořen týmem AiStorm, který spadá pod ústav informatiky na provozně ekonomické fakultě Mendelovy univerzity v Brně.

Tým AiStorm se snaží na robotu K4 testovat nové postupy a technologie. Jeho konstrukce, jejíž model je na obrázku 9, je kompletně provedena pomocí 3D tisku na průmyslové 3D tiskárně, má tvar osmiúhelníku s koly umístěnými v hlavní ose, čímž se dosáhlo optimálních jízdních vlastností, rovnoměrného rozložení váhy a schopnosti otáčet se kolem středu pro minimalizaci nebezpečí kolize s jiným robotem během soutěžního klání.



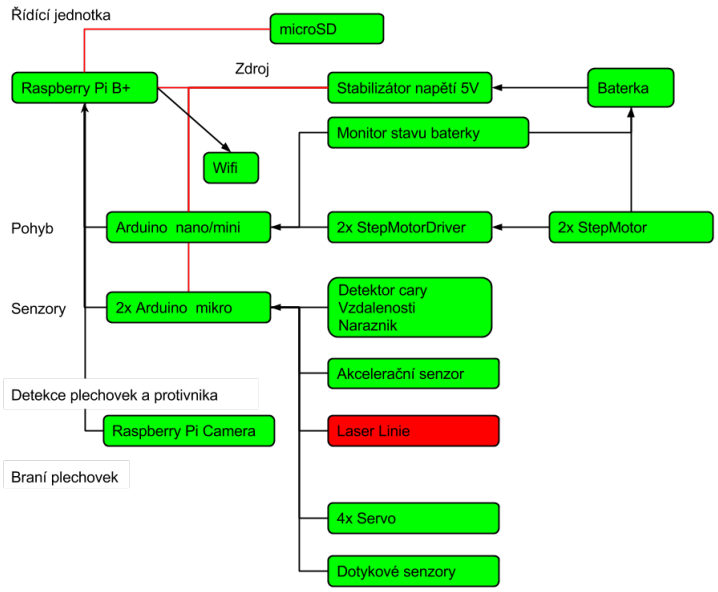
Obrázek 9: Ukázka konstrukčního modelu robota K4 (AiStorm, 2016).

Robot K4 se skládá z pěti úrovní, které jsou tvořeny dvěma základními deskami, na kterých je z obou stran umístěna elektronika. Spodní deska je zespodu osazena senzory pro detekci čar na podložce a motory hnacích kol. Dvou kolům pomáhají v pohybu čtyři ložiskové kuličky. Pomyslnou druhou úrovní jsou místa po obvodu spodní základní desky, které jsou určeny pro senzory umožňující detekci překážek

v bezprostředním okolí robota. Tyto senzory však nejsou v současné době na robotu K4 implementovány.

Třetím patrem je elektronika namontována na spodní straně horní základní desky. Toto patro obsahuje stabilizátor napětí, rozvody do zbylých částí robota, dvojice ovladačů krokových motorů a malý jednodeskový počítač Arduino, který řídí pohyby robota. Na vrchní části horní základní desky jsou umístěny čtyři magnetické uchopovače plechovek a držák na modelářskou baterii. Uchopovače jsou zkonstruovány tak, aby byly schopné přitáhnout plechovku i z větší vzdálenosti a zajistit ji, aby nebránila pohybu robota.

Poslední patro obsahuje další malý jednodeskový počítač, tentokrát Raspberry Pi 2 Model B, který funguje jako řídící jednotka, a držák na kameru s Raspberry Pi kamerou. Toto patro nás z hlediska rozpoznávání objektů a jeho optimalizace pro jejich opakované vyhledávání v posloupnosti snímků zajímá nejvíce.

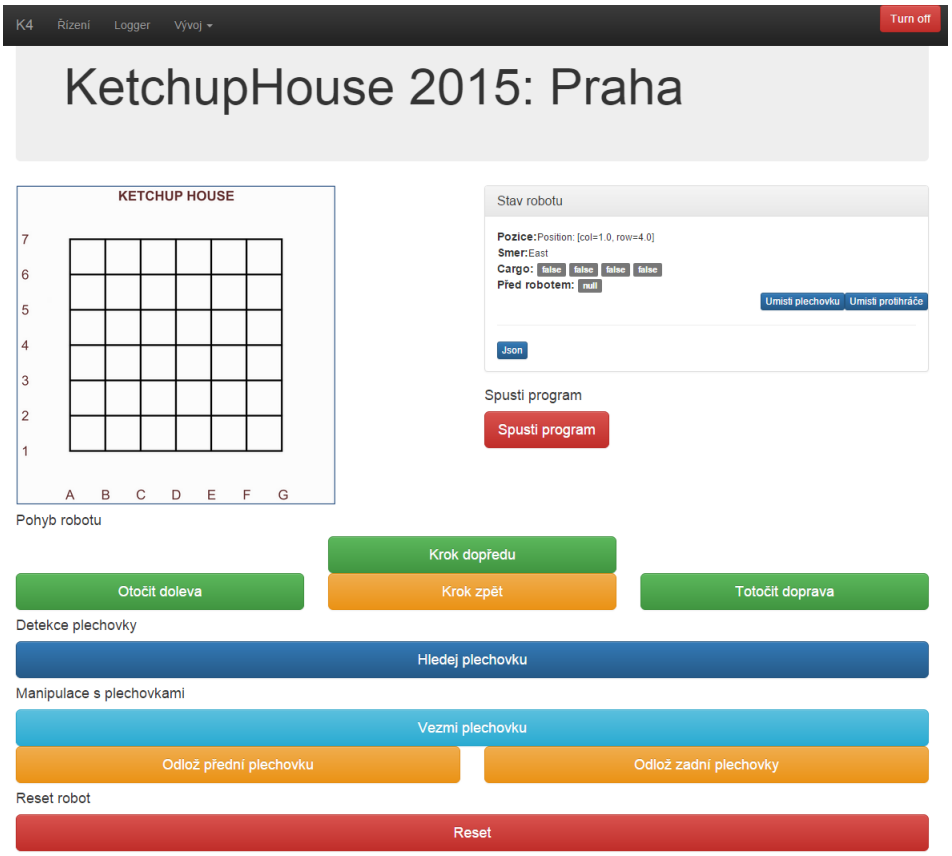


Obrázek 10: Diagram zapojení jednotlivých částí robota K4 (AiStorm, 2016).

Při návrhu robota se myslelo zejména na jeho modularitu, proto jsou jednotlivé části hardwaru robota K4 propojeny do stromové struktury, jejímž kořenem je jednodeskový počítač Raspberry Pi 2 Model B. Druhou úroveň tvoří k němu připojené jednodeskové počítače Arduina, které již přímo pracují se senzory a motory v úrovni třetí. Díky tomuto návrhu se podařilo vytvořit izolované jednotlivé funkční bloky robota, kde každý má přidělený právě jedno Arduino, čímž vznikly funkčně nezávislé bloky s jasně definovanou odpovědností a funkcionalitou, což je výhodné například

pro testování nebo budoucí rozšiřování. Finální zapojení je znázorněno na obrázku 10.

Komunikace mezi Raspberry Pi a Arduiny probíhá pomocí I2C sběrnice, kde hlavní řídící jednotka, Raspberry Pi, vystupuje v roli master a je napájena přes stabilizátor z centrální baterie. Operačním systémem na Raspberry Pi je Raspbian (Debian Wheezy) a pro komunikaci s hardwarem byla použita knihovna rpi4j, která umožňuje práci s GPIO porty a tedy i komunikaci s Arduiny prostřednictvím I2C. Každý hardwarový modul (Arduino) je reprezentován jedním rozhraním. Součástí hlavního softwaru je také Embeded Jetty server, který vytváří kontrolní rozhraní prostřednictvím HTTP protokolu. Robota tedy lze online monitorovat a ovládat vzdáleně pomocí běžného webového prohlížeče (viz obrázek 11). Responzivní design stránek umožňuje pohodlné řízení i z mobilního telefonu (AiStorm, 2016).

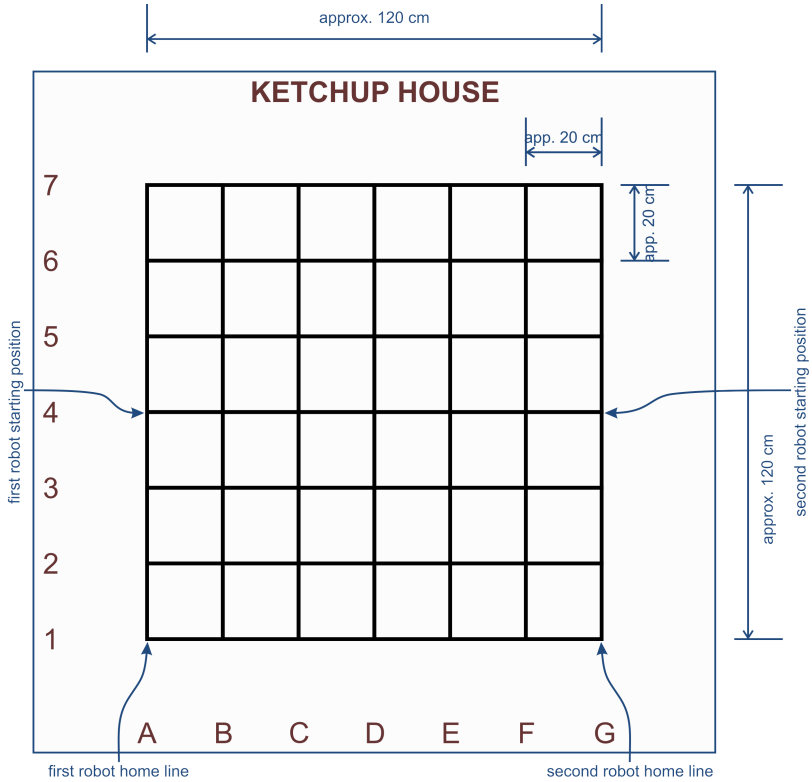


Obrázek 11: Webové kontrolní rozhraní robota K4 (AiStorm, 2016).

4.1 Ketchup House

Dvojice robotů na hřišti (náhled na obrázku 12) soupeří o to, kdo bude mít na konci zápasu více plechovek s kečupem. Během každého tříminutového zápasu se plně autonomní (samostatný) robot snaží na hřišti vyhledávat a převážet plechovky

s kečupem tak, aby jich na konci zápasu měl co nejvíce na své domácí čáře. Plechovky jsou před zápasem umístěny na hřišti, které je reprezentováno sítí 7x7 černých čar s rozestupem cca 20 cm. Čáry jsou široké cca 1,5 cm, podlaha je bílá. Okolo hřiště je na každou stranu cca 30 cm volná plocha (mohou na ní ale být popisky). Na hřišti je umístěno 7 kečupů: dva na pozicích D3 a D5 a pět náhodně, a to tak, že čtyři z nich středově symetricky ve sloupcích C, D, E a pátý na jednu z dosud volných pozic ve sloupcích C, D a E. Pokud při hře robot odebere některý z náhodně umístěných kečupů a vzdálí se s ním na více než jeden čtvereček a soupeř bude také alespoň 1 čtvereček daleko, bude na toto místo doplněn další kečup, a to až do celkového počtu 12 kečupů ve hře. Roboti startují na pozicích A4 resp. G4. Během hry se smějí pohybovat po celé ploše hřiště, nejen po čarách. Kečup je v plechovkách běžně prodávaných v obchodech, jejich rozměry jsou přibližně: průměr 5,5 cm, výška 7,5 cm, hmotnost 165 g (Robotic Day, 2016).



Obrázek 12: Schématický plán hřiště hry Ketchup House (Robotic Day, 2016).

Samotné zpracování obrazu v současné době zajišťuje program pro zpracování obrazu ImageJ rozšířen o funkcionalitu knihovnou BoofCV pomocí implementovaného wrapperu, který získává z kamery obraz a dále má na starosti rozpoznávání objektů a scény v okolí robota a jeho relativní pozicování. Robot K4 se v rámci soutěže Ketchup House snaží sbírat plechovky z pole pomocí magnetických držáků. Pro tento účel jezdí po vodících čárách, kalibruje se prostřednictvím kamery a momentálně sleduje pouze nejbližší křižovatku a rozhoduje se, zda na ní je plechovka,

kterou se pokusí sebrat, nebo nikoliv. Toto zjištění probíhá na vyprahovaném obrazu pomocí detekce hran, a to konkrétně aplikací dvojice konvolučních masek, po které se pak na základě množství černých pixelů rozhoduje, zda obraz obsahoval pouze křižovatku nebo křižovatku s plechovkou:

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

Tato metoda sice při optimálních podmínkách dosahuje vysoké úspěšnosti rozpoznání plechovky, ale vyžaduje, aby křižovatka byla nasnímána v podobě vodorovné a svislé černé čáry jen s minimálními odchylkami (ukázka na obrázku 13). Větší pootočení obrazu značně snižuje efektivitu metody. Tento nedostatek lze vyřešit buď přesným postavením robota K4 na křižovatce nebo lépe předzpracováním snímaného obrazu pootočením tak, aby kolmé čáry křižovatky byly dokonale svislé a vodorovné.

Druhým problémem této metody je nedostatečná detekce nepřátelského robota, kde se ukázalo jako chybné předpokládat, že zmíněnými konvolučními maskami vyfiltrovaný binární obraz obsahující robota bude ve většině případů obsahovat více černých pixelů než obraz pouze s plechovkou. Můžeme však předpokládat, že velikost nepřátelského robota bude vždy větší než velikost křižovatky, zatímco plechovky se vyskytují uprostřed křižovatek. Proto můžeme kontrolovat kraje snímaného obrazu a v případě, že budou obsahovat cokoliv jiného než černé čáry křižovatky, vyhodnotit tento objekt jako nepřátelského robota.

Obrázek 13: Ukázka rozpoznávání plechovky na křižovatce; zleva ilustrační foto prázdné křižovatky před a po aplikaci dvojice filtrů, vpravo pak dvojice s plechovkou.

5 Návrh a implementace detekce objektů

V rámci praktické části této práce nebudeme vylepšovat ani navrhovat metody rozpoznávání objektů, konkrétně tedy plechovek, v omezeném pohledu mobilního robota K4 na jedinou křižovatku. Místo toho rozšíříme pohled Raspberry Pi kamery umístěné na robotovi, abychom v jednom snímku mohli sledovat více křižovatek současně. Jakmile toto vylepšení provedeme, zároveň musíme navrhnout zcela odlišný přístup k řešení celé problematiky detekce. Není již dále možné sledovat pouze jediné místo ve vstupním obraze, a tak si celý problém můžeme dekomponovat na tři základní kroky jeho řešení.

Prvním krokem je zúžení celého obrazu na jednotlivé oblasti zájmu (*regions of interest*), což provedeme detekcí křižovatek herního plánu, které mohou být buď prázdné nebo zakryté plechovkou. Toto rozhodnutí je krokem druhým a pro účely soutěže Ketchup House také krokem dostačujícím. Tato práce však počítá ještě s krokem třetím, ve kterém se pokusíme využít některých znalostí získaných v předchozích krocích a použít je k opakovanému vyhledání cílových objektů v následujícím snímku či snímcích, které v posloupnosti snímá Raspberry Pi kamera. Aby toto opakované vyhledání bylo smysluplné, musí přinášet přidanou hodnotu oproti oddělenému zpracovávání jednotlivých snímků. Co se týče této přidané hodnoty v našem konkrétním případě, tato práce je zaměřena na vyšší rychlost detekce plechovek na křižovatkách, což přinese požadovanou optimalizaci.

Pro tento účel byly navrhnuty dvě metody. První metoda využívá pro detekování křižovatek vhodně nastaveného Cannyho hranového detektoru a následně provedené Houghově transformace pro detekci přímek. Z průsečíků přímek jsou za pomoci principů minimálního ohraničujícího obdélníku a lineární perspektivy sestaveny oblasti zájmu. V těchto oblastech je poté prováděna detekce plechovky, a to jak v prvním snímku pomocí příznaků plechovky a křižovatky, tak ve snímcích následujících pomocí rozdílu obrazů. Druhá metoda je principiálně velmi podobná, má však za cíl nahradit běžné metody zpracování obrazu méně používanými či novými ekvivalenty a otestovat jejich efektivitu.

5.1 První metoda

Při návrhu a implementaci této metody se ukázalo, že nejzávažnějšími problémy k řešení je správné nastavení Cannyho hranového detektoru, zejména použitého prahování, a vhodný výběr přímek určených Houghovou transformací. Na takto postaveném základu pak zbývalo navrhnout rychlý způsob rozdělení průsečíků přímek do oblastí zájmu a detekce objektů. Z hlediska zpracování série snímků pak bylo klíčové minimalizovat rizika špatné detekce způsobené vlivem drobného pootočení kamery.

Cannyho hranový detektor s využitím Otsuovy metody

Tento několika krokový algoritmus pro detekci hran byl již několikrát zmíněn v kapitole Zpracování obrazu. Zde si jej popíšeme podrobněji s důrazem na jeho přesné nastavení v této metodě. Prvním krokem je redukce šumu, která je provedena aplikací Gaussova filtru pomocí konvoluce a konvoluční masky o velikosti 5×5:

$$\frac{1}{159} \times \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Dalším krokem je určení gradientů (první derivace) hran, což je provedeno opět filtrací, kde jako konvoluční maska je použit gradientní Sobelův operátor 3×3 používaný pro detekci hran, a to v horizontálním (pro zjištění G_x) a vertikálním (pro zjištění G_y) směru:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

K určení celkového hranového gradientu je pak použit vzorec $G = |G_x| + |G_y|$, který poskytuje dostatečnou přesnost a vysokou rychlost výpočtu, přestože přesnější by bylo použít odmocninu součtu druhých mocnin. Směr gradientu je pak vypočítán jako $\Theta = \arctan(G_y/G_x)$. Předposledním krokem je průchod celým obrazem po pixelech a ponechání pouze těch hodnot pixelů, které jsou lokálním maximem ve směru gradientu ve svém okolí (v obraze zůstanou pouze tenké linie hran). Posledním krokem je tzv. prahování s histerezí (*hysteresis thresholding*), kde pomocí dvou prahových hodnot je posuzováno, zda je hrana mezi nimi a je tedy významná.

Protože vstupní obrazy této metody mohou být snímány za různých světelných podmínek, není vhodné mít hodnotu obou prahů prahování s histerezí konstantní. Testování ukázalo, že vyhovující je určit vyšší prahovou hodnotu pomocí Otsuova algoritmu pro prahování[13], který se snaží najít vhodnou prahovou hodnotu mezi pozadím a popředím obrazu, použitého na vstupní obraz převedený do stupňů šedi. Nižší hodnota práhu pro prahování s histerezí je pak v této metodě určena jako polovina vyšší.

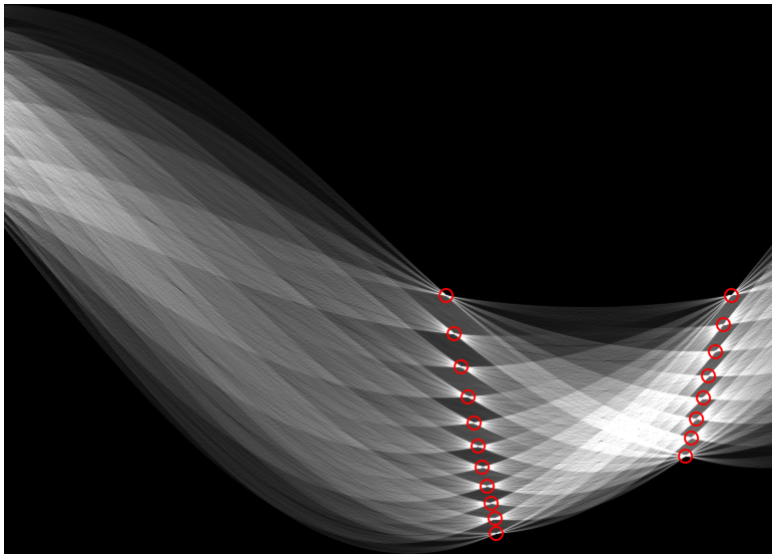
Detekce přímek s využitím Houghovy transformace

Jakmile jsou v obraze pouze hrany nalezené Cannyho hranovým detektorem a obraz je tedy binární, můžeme použít Houghovu transformaci pro identifikaci přímek v obraze. Tato transformace vychází z polární soustavy souřadnic a předpokladu, že

[13] Algoritmus Otsuho metody pro automatické prahování je i s ukázkami a výpočty vysvětlen na http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

každým bodem (x, y) mohou procházet přímky dle rovnice $r = x \cos\theta + y \sin\theta$, kde r je vzdálenost tohoto bodu od počátku souřadnic a θ je úhel, který se dá v kartézské soustavě souřadnic vyjádřit jako úhel mezi tímto bodem a osou x.

Pokud pro každý bod v obraze ponechaný Cannyho hranovým detektorem určíme křivku popisující graf funkce vycházející z této rovnice s definičním oborem $r > 0$ a oborem hodnot $0 < \theta < 2\pi$, pak se tyto křivky protnou v místech, kde více bodů může tvořit jednu přímku (názorná ukázka tohoto Houghova prostoru s vyznačenými průsečíky na obrázku 14). Použitá Houghova transformace v této metodě pracuje s diskrétními hodnotami této funkce s přesností jeden pixel pro proměnnou r a jeden stupeň pro proměnnou θ a body proložené přímkou skutečně označí za přímku pouze v případě, že jich je více než 100 [14].



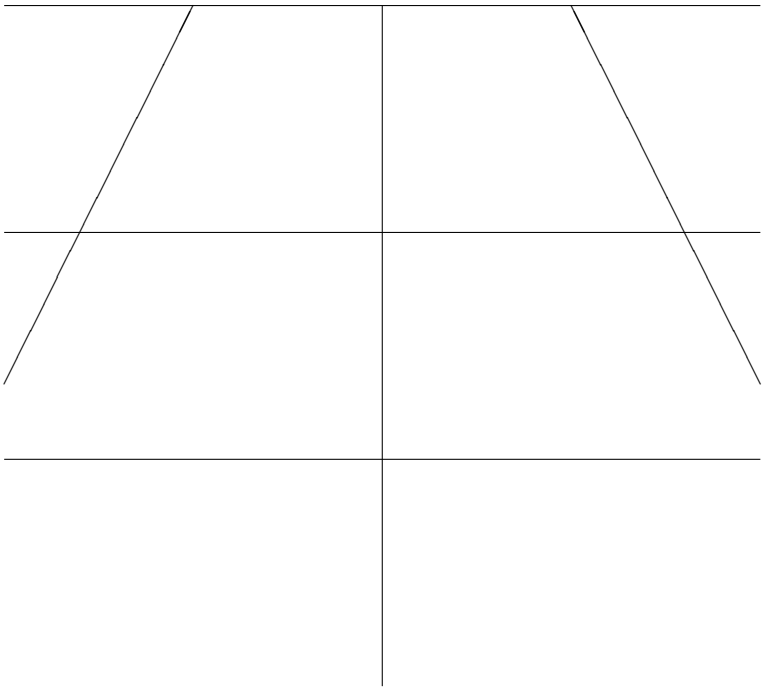
Obrázek 14: Ukázka Houghova prostoru s vyznačenými průsečíky sinusoid. Autor: Brian Moore, CC BY-SA 4.0

V této metodě nejsou potřeba všechny přímky nalezené Houghovou transformací, ale pouze ty, které odpovídají čárám na hrací podložce. Za předpokladu, že mobilní robot K4 stojí přímo na křižovatce, na kterou po čáře přijel, pak další čáry hracího pole jsou k vektoru pohledu robota buď kolmé nebo rovnoběžné. Na robotem snímaném obrazu však bude skutečně rovnoběžná pouze čára, na které stojí. Další rovnoběžky se vlivem lineární perspektivy promítnou jako sbíhající se čáry svírající úhel 80° [15]. Situaci názorně zobrazuje obrázek 15 a díky této znalosti můžeme ze všech přímek vybrat pouze ty, které svou souřadnicí θ s odchylkami odpovídají stupňům 180°(π radiánů), 90°(1,5708 radiánů), 40°(0,698132 radiánů) a 140°(2,44346 radiánů). Poslední dvě zmíněné můžeme ještě dále omezit nejen dle

[14] Prahová hodnota, která se nejlépe osvědčila při jejím testování na 120 testovacích snímcích.

[15] Tento úhel byl experimentálně naměřen a je závislý na přesném nastavení Raspberry Pi kamery mobilního robota K4.

úhlu natočení, ale také dle umístění na pravé či levé straně obrazu pomocí souřadnice r. Tyto hodnoty můžeme odpovídajícím způsobem měnit dle natočení robota.



Obrázek 15: Schéma čar herního plánu z pohledu robota.

Posledním krokem je zjistit průsečíky vybraných přímek k přibližné detekci křižovatek. Polární souřadnice si můžeme převést na kartézské pomocí vzorců $x = r \times \cos\theta$ a $y = r \times \sin\theta$. Libovolný bod $(x2, y2)$ pak získáme rovnicemi $x2 = x + c \times (-\sin\theta)$ a $y2 = y + c \times \cos\theta$, kde c je libovolná konstanta. Průsečík mezi přímkami $a : [A1(x, y), A2(x, y)]$ a $b : [B1(x, y), B2(x, y)]$ pak existuje, pokud se následující vzorec nerovná nule:

$$(A1(x) - A2(x)) \times (B1(y) - B2(y)) - (B1(x) - B2(x)) \times (A1(y) - A2(y))$$

Pokud existuje průsečík dvou přímek, můžeme zjistit jeho souřadnice (x, y), což je ukázáno prostřednictvím pseudoalgoritmu 4.

Detekce objektů s využitím Union Find

Po detekci křižovatek (obrazové ukázky na obrázku 16) zbývá rozpoznat, zda na křižovatkách je či není plechovka. Křižovatky máme identifikované množinami průsečíků v jejich okolí, které musíme rozdělit dle konkrétní křižovatky a následně z nich

```
delta x = (A1(x) - A2(x), B1(x) - B2(x));
delta y = (A1(y) - A2(y), B1(y) - B2(y));
Function det(a, b)
    return a[0] * b[1] - a[1] * b[0];
div = det(delta x, delta y);
if (div != 0) then
    d = (det(A1(x, y), A2(x, y)), det(B1(x, y), B2(x, y));
    x = det(d, delta x) / div;
    y = det(d, delta y) / div;
    return x, y;
end
```

Algorithm 4: line intersection

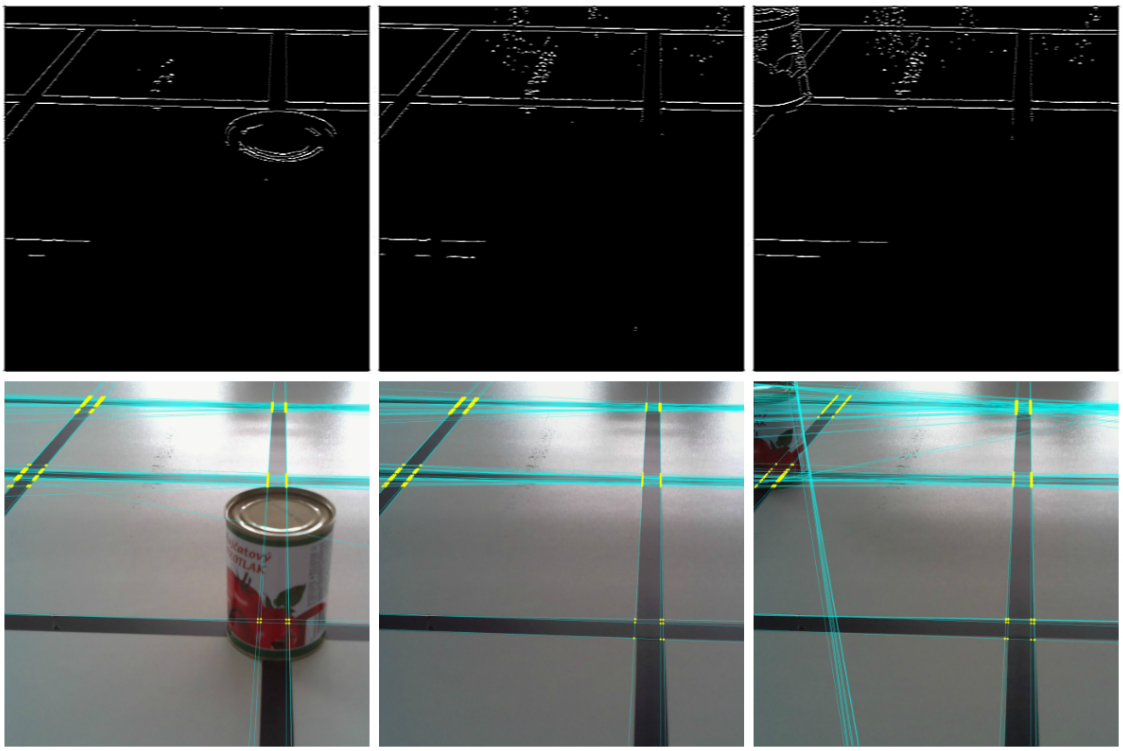
aproximovat spojitou oblast. Pro jejich rozdělení můžeme použít datovou strukturu disjoint-set a na ní postavený algoritmus pro řešení Union Find problému[16].

Jako kritérium rovnocennosti průsečíků (tedy jejich příslušnost ke stejné křižovatce) můžeme použít vzdálenost mezi nimi. Pro určení vzdálenosti dvou bodů ve dvourozměrném prostoru zadaných kartézskými souřadnicemi se v tomto případě hodí například Chebyshevova vzdálenost $D(x, y) = max(|x_2 - x_1|, |y_2 - y_1|)$. Mnohem lepší výsledky však má mírná úprava této rovnice, kdy budeme nahlížet na rozdíly ve vzdálenostech na ose x a ose y s různou váhou. Měřením testovacích snímků v rozlišení 5 Mpx (2592 px : 1944 px) jsme zjistili, že nejmenší vzdálenost mezi křižovatkami na ose x je 925 px, zatímco nejmenší vzdálenost na ose y je 355 px. Polovinu těchto vzdáleností můžeme použít jako povolenou odchylku průsečíků k vyhodnocení jejich rovnocennosti, tedy průsečíky patří ke stejné křižovatce pokud:

$$|x_2 - x_1| \leq 462 \wedge |y_2 - y_1| \leq 127$$

Z rozdělených průsečíků můžeme jednoduše vytvořit ohraničující obdélník zjištěním minimálních a maximálních souřadnic skupin bodů na obou osách. My však potřebujeme vytvořit plochu, kterou případně může zabírat detekovaný objekt (plechovka). Tento objekt je pochopitelně zobrazen na obraze tím větší, čím blíže je kameře, proto i požadovaná plocha musí odpovídajícím způsobem měnit svoji velikost. Testováním bylo naměřeno a spočítáno, že velikost objektu roste lineárně dle funkcí $f : y = 0,5x + 318$, kde y je výška objektu a x je vzdálenost v pixelech a $f : y = 0,24x + 194$, kde y je šířka objektu a x je vzdálenost v pixelech, přičemž

[16]Pro termíny *disjoint-set data structure* a *Union Find problem* neexistují v odborné literatuře ustálené české překlady, uvádím tedy anglické termíny. Datová struktura disjoint-set slouží k vytvoření systému navzájem se nepřekrývajících podmnožin. Problém Union Find je následující: „Řekněme, že máme určitý počet položek. Můžeme sloučit jakékoliv dvě položky a považovat je za rovnocenné. Kdykoliv se můžeme zeptat na jakékoliv dvě položky, zda jsou považovány za rovnocenné nebo ne.“ Řešení tohoto problému se uplatňuje také například v Kruskalově algoritmu pro hledání minimální kostry v grafu.

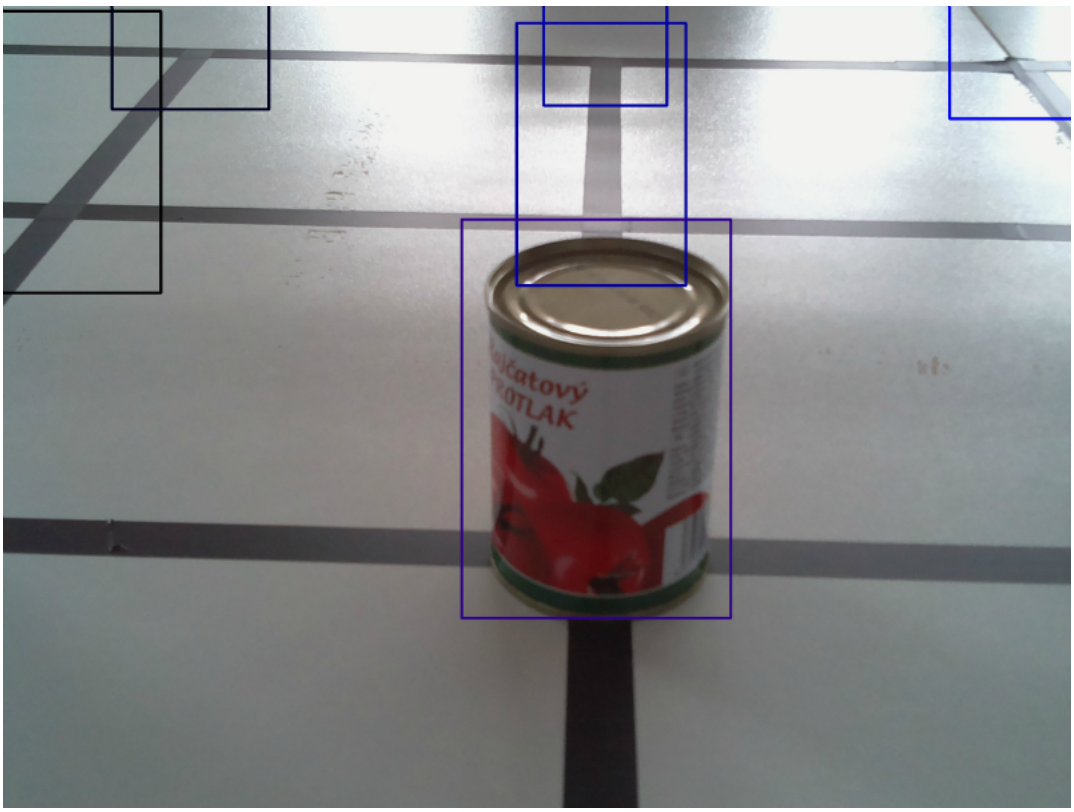


Obrázek 16: Ukázka obrazů po provedení Cannyho hranového detektoru (nahoře) a následně Houghově transformaci s průsečíky přímek (dole, průsečíky jsou vyznačeny žlutě). Můžeme si všimnout, že při snímání byla hrací plocha velmi špatně osvětlena, spodní křižovatku hranový detektor vůbec neodhalil, přesto byla detekována.

vzdálenost je měřena na kolmici od počátku, za který je považován horní okraj obrazu.

Posledním faktem, který musíme vzít v potaz, je, že detekovaný objekt stojící na křižovatce, díky úhlu kamery, má 50 % své šířky na obou stranách křižovatky, zatímco na ose y je přibližně 20 % jeho výšky pod křižovatkou a zbytek nad ní. Ukázka vypočítaných ploch kolem křižovatek, tedy oblastí zájmu, je na obrázku 17.

Po tomto kroku již můžeme přistoupit k samotnému popisu a klasifikaci objektů, protože oblasti zájmu představují potřebnou segmentaci obrazu, která nám zajistí, že veškeré rušivé objekty mimo křižovatky, jako jsou například pravidly Ketchup House povolené popisky a další objekty na okrajích hracího plánu, nám neovlivní rozpoznávání. Pro popis objektů můžeme nejjednodušeji využít rozdílných příznaků obou možností, prázdné křižovatky a plechovky na křižovatce. Jednou z mnoha možností je detekce bodů s vysokou saturací, protože jakkoliv přirozeným světlem osvětlená křižovatka z bílých ploch a černých čar bude na obrazu v odstínech šedi, zatímco můžeme předpokládat, že běžně zakoupená plechovka s jakýmkoliv obsahem bude na svém obalu obsahovat barvy (v našem případě většinou červenou, ale saturace bude vyšší v případě jakékoliv barvy). Klasifikace pak spočívá v porovnání těchto



Obrázek 17: Ukázka vypočítaných oblastí zájmu na reálném snímku.

hodnot, které jsou pro prázdnou křižovatku většinou nula či velmi blízké nule.

Rozdíl snímků s využitím Gaussova vyhlazování

Také pro následující obrazy z posloupnosti snímků můžeme použít způsob popisu a klasifikace objektů z minulého kroku. V tomto případě se jedná o rychlou detekci kontur s vysokou saturací, kterou je však možno nahradit za složitější, přesnější, ale většinou také mnohem pomalejší metody. Naším cílem je ale využít posloupnosti snímků, kdykoliv je to možné tak, aby nebyly potřeba tyto jednosnímkové metody, a dosáhli jsme ještě vyšší rychlosti při zachování přesnosti detekce.

Proto tato metoda pracuje s absolutním rozdílem snímků, kterého se využívá také v metodách odčítání pozadí (*background subtraction*) používaných u kamerových záznamů. Omezeně jej můžeme použít také v našem problému na dvojici stejných křižovatek ze dvou po sobě jdoucích snímků v případě, že původně byla křižovatka prázdná, a tedy kontrolujeme, zda se tento fakt nezměnil.

Jedinou další nutností je, aby byl snímek pořízen ze stejného místa, v našem případě křižovatky, na které stojí mobilní robot K4, a při stejném natočení kamery. Díky tomu, že robot K4 ví o své pozici a natočení, tak problémem mohou být pouze drobné posuny kamery, objektů či hrací plochy nebo světelné podmínky mezi jed-

notlivými snímky. Pro minimalizaci pravděpodobnosti rizika špatné detekce, kvůli měnícím se světelným podmínkám, můžeme oba porovnávané snímky převést do stupňů šedi a dále ekvalizovat jejich jasový histogram. S drobnými posuny kamery atd. se zase můžeme vyrovnat pomocí principu metody *difference of Gaussians*, kdy oba obrazy jsou rozmazány různě silným filtrem Gaussova vyhlazování, v našem konkrétním případě s konvolučními maticemi 3×3 a 5×5. Výsledný rozdíl je nakonec vyprahován, jak můžeme vidět na obrázku 18.

Obrázek 18: Ukázka detekce vysoké saturace (vlevo) a vyprahovaného rozdílu křižovatek (vpravo).

5.2 Druhá metoda

Tato metoda je v podstatě složena z alternativ algoritmů v metodě předchozí a slouží tak k nalezení nejlepších možných variant, z kterých se následně může složit nejefektivnější metoda. Konkrétně se jedná o nahrazení Otsuovi metody pro určení prahů v Cannyho hranovém detektoru za medián jasu jednotlivých pixelů. Poté je vyzkoušena metoda PCLines místo Houghovi transformace. Další je nahrazen Union Find algoritmus použitím k-dimensionálního stromu (*k-d tree*). A nakonec je v kroku rozdílu snímků použito vyhlazování mediánem (*median blur*) místo Gaussova vyhlazování.

Cannyho hranový detektor s využitím mediánu jasu

Automatické určení prahů pro prahování v Cannyho hranovém detektoru se může provést mnohými způsoby. My se však potřebujeme pro zajištění vysoké rychlosti držet těch nejjednodušších, nejlépe takových, které se obejdou bez dalšího předzpracování obrazu, jako je například ekvalizace histogramu. Ačkoliv takovým velmi rychlým a jednoduchým způsobem je použít průměrnou hodnotu jasu obrazu, Kerry Wong (2009) píše, že použít medián hodnoty jasu vykazuje lepší výsledky.

Na základě jeho testování se ukázalo, že na obrazech s přibližně vyrovnaným jasovým histogramem má průměrná hodnota i medián jasu, použitý jako práh, v Cannyho hranovém detektoru srovnatelné výsledky. V případě jiných obrazů však

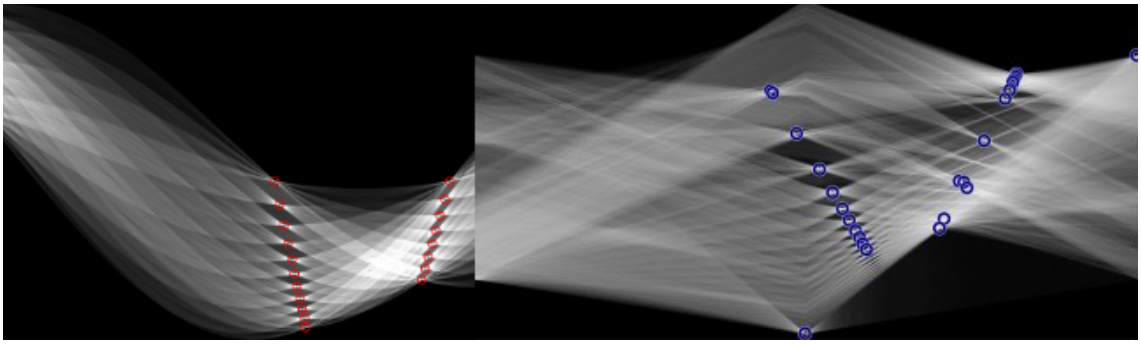
medián dosahuje mnohem lepších výsledků, srovnatelných s výsledky na obrazech s ekvalizovaným histogramem[17].

Detekce přímek s využitím PCLines - paralelních souřadnic

Metoda PCLines byla vyvinuta na Fakultě informačních technologií Vysokého učení technického v Brně, a to konkrétně na ústavu počítačové grafiky a multimédií v roce 2011. Díky tomu je v dnešní době poměrně neznámá a zmínky o ní lze nalézt jen v malém množství odborné literatury novější datace. Přesto se jedná o jednu z mála možností, jak nahradit Houghovu transformaci pro detekci přímek v obraze.

Algoritmus PCLines nepoužívá žádné operace s pohyblivou řádovou čárkou ani goniometrické funkce, proto má oproti Houghově transformaci nespornou výhodu ve snadnější akceleraci různými hardwarovými architekturami. Zejména počítačové systémy s malou, ale velmi rychlou, operační pamětí, jako jsou dnešní grafické procesory, umožňují používat PCLines v real-time aplikacích (Havel, Herout, Dubská a Jošth, 2014).

V našem případě GPU akceleraci nevyužijeme s ohledem na cílovou architekturu, můžeme však implementaci metody PCLines porovnat v Houghovou transformací ve snaze zvýšit rychlost detekce přímek díky absenci goniometrických funkcí pro rasterizaci sinusoid v Houghově prostoru (ukázka srovnání Houghova prostoru a prostoru PCLines je na obrázku 19).

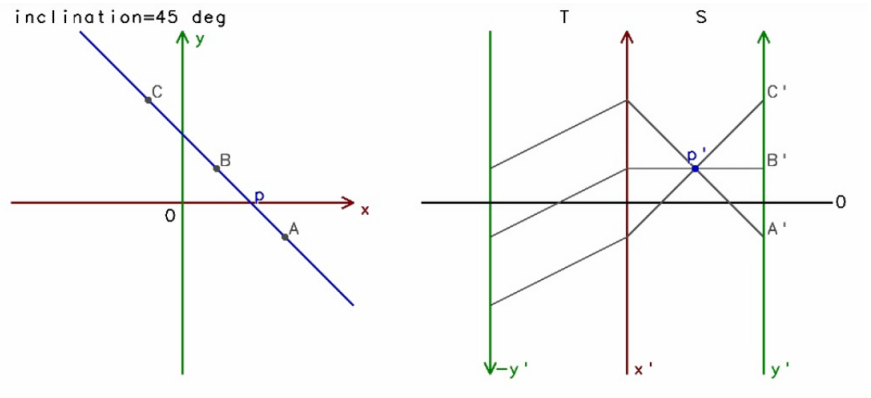


Obrázek 19: Ukázka Houghova prostoru (vlevo, Autor: Brian Moore, CC BY-SA 4.0) a prostoru rovnoběžných souřadnic (vpravo, (Dubská, Herout a Havel, 2011)).

Dle autorů PCLines, Markéty Dubské, Jiřího Havla a Adama Herouta (2011), je tento algoritmus obměnou Houghovy transformace, kde místo Houghova prostoru se jednotlivé body promítnou do prostoru rovnoběžných souřadnic (*parallel coordinate space*). Tento prostor se v našem případě skládá ze tří rovnoběžných os, jak názorně ukazuje obrázek 20. Na první ose jsou vyneseny záporné y souřadnice bodů v kartézském souřadnicovém systému, na prostřední ose x souřadnice a na poslední

[17]Ukázky jsou dostupné na: http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/

ose y souřadnice bodů. Odpovídající souřadnice všech bodů jsou v prostoru rovnoběžných souřadnic mezi všemi třemi osami propojeny úsečkami, a pokud se úsečky bodů protínají, pak tyto body mohou tvořit přímku v reálném prostoru.



Obrázek 20: Ukázka převodu bodů z kartézské soustavy souřadnic (vlevo) do rovnoběžné soustavy souřadnic (vpravo) (Dubská, Havel a Herout, 2011).

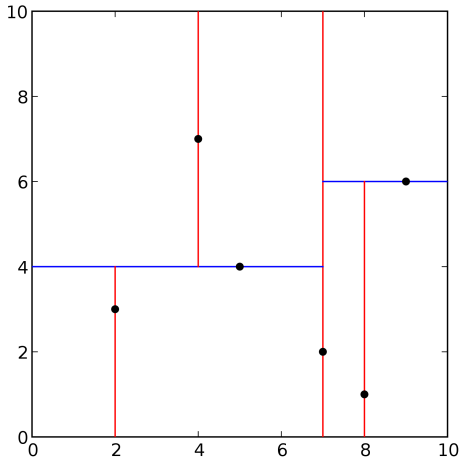
Prostor mezi osami x' a y' je přímý (*straight*) prostor rovnoběžných souřadnic S pro dvourozměrný reálný prostor a prostor mezi osami $-y'$ a x' je ohnutý (*twisted*) prostor rovnoběžných souřadnic T, který znemožňuje, aby se průsečík v prostoru paralelních souřadnic, pro přímky pod vysokým úhlem sklonu (nad $\frac{\pi}{2}$ radiánů) v reálném prostoru, zobrazil v nekonečnu. Z průsečíku ve výsledném TS prostoru lze snadno vypočítat směrnici přímky a hodnotu funkce přímky pro $x = 0$ (*y-intercept*) v reálném prostoru (Larson, 2015): Směrnice přímky $k = -1 + \frac{d}{u}$ a průsečík s osou y $f(0) = d \times \frac{v}{u}$, kde u je vzdálenost průsečíku od osy x, v je vertikální vzdálenost od 0 a d je šířka prostoru T nebo S, dle polohy průsečíku.

Detekce objektů s využitím k-d stromu

V první metodě je pro detekci křižovatek použit algoritmus Union Find, který umožňuje, ze všech průsečíků v obraze nalezených přímek, vytvořit skupiny bodů (dvourozměrná data) patřících k jednotlivým křižovatkám. Pro ukládání vícerozměrných dat a rychlé vyhledávání blízkých hodnot slouží také datová struktura k-dimensionální strom (dále v textu k-d strom (v našem případě dvou-dimensionální)).

K-d strom očekává na vstupu množinu bodů v k-dimensionálním prostoru, ze kterých je pak zkonstruován vyvážený strom, který tento prostor rozděluje polorovinami tak, aby každý bod měl svůj vlastní kvádr. Snahou je rozdělit prostor tak, aby každá rozdělená část obsahovala malý počet bodů, ke kterým je pak snadný přístup na základě jejich souřadnic. V našem případě dvourozměrných bodů (jak ukazuje obrázek 21), je tak plocha rozdělena k-d stromem na obdélníky, kde každý bod má svůj obdélník.

Pro práci s datovou strukturou k-d strom je ideální, pokud předem známe souřadnice bodu, ke kterému budeme hledat další blízké body do určité vzdálenosti.



Obrázek 21: Vizualizace prostoru dvou-dimensionálního k-d stromu s body [(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)]. Autor: KiwiSunset (Wikipedia), CC BY-SA 4.0

V našem případě však žádné body předem neznáme, proto musíme využít libovolný nalezený průsečík a pomocí k-d stromu k němu najít blízké body, které spolu s původním průsečíkem tvoří skupinu bodů jedné křižovatky. Pro zbytek bodů pak tento algoritmus 5 opakovat.

```
groups = [];
while (intersections != []) do
    tree = kdtree(intersections);
    group = tree.similar(intersections[0]);
    groups.append(group);
    intersections -= group;
end
```

Algorithm 5: Pseudoalgoritmus hledání skupin průsečíků pomocí k-d stromu.

Rozdíl snímků s využitím vyhlazování mediánem

Zde místo lineárního filtru Gaussova vyhlazování použijeme nelineární filtr vyhlazování mediánem, který každý pixel nahradí mediánem hodnoty jeho okolí. Jednou z vlastností této metody je, že na rozdíl od jiných způsobů vyhlazování, nová hodnota zpracovávaného pixelu není nově vypočítána, ale je použita hodnota bodu již existujícího v původním obraze. Rozdíl stejných snímků, jen mírně posunutých, by tedy měl vést k vyššímu množství nulových pixelů, než u jiných metod vyhlazování.

6 Testování a vyhodnocení řešení

Pro rekapitulaci můžeme znovu ve zkratce uvést společný postup obou metod, které ve snímcích detekují křižovatky a určí, zda se na nich nachází objekt nebo ne. Tento postup názorně ukazuje pseudoalgoritmus 6. Nasnímaný obraz pomocí funkce *getImage()* binarizujeme pomocí Cannyho hranového detektoru. Takto získaný binární obraz použijeme pro detekci přímek, z nichž funkcí *linesSelection()* vybereme pouze přímky odpovídající čárám herního pole. Z těchto přímek získáme průsečíky funkcí *linesIntersections()*, které následně seskupíme do skupin dle jejich pozice. Pomocí těchto skupin bodů, odpovídajících jedné křižovatce, vytvoříme souřadnice ohraničujících obdélníků funkcí *relativeBoundingRectangles()*.

```
image = getImage();
binaryImage = cannyEdgeDetector(image);
lines = detectLines(binaryImage);
selectedLines = linesSelection(lines);
points = linesIntersections(selectedLines);
crosses = groupPoints(points);
rectangles = relativeBoundingRectangles(crosses);
while not move do
    for rectangle in rectangles do
        if object[rectangle] == false then
            object[rectangle] = detectionBySaturation(image[rectangle]);
        end
        else
            object[rectangle] = detectionByDifference(image[rectangle],
            oldImage[rectangle]);
        end
    end
    oldImage = image;
    image = getImage();
end
```

Algorithm 6: Pseudoalgoritmus kompletní detekce.

Dokud se kamera snímající obraz nepohne (podmínka *not move*), můžeme pracovat se segmenty obrazu získanými ohraničujícími obdélníky. V každém z nich se pokusíme rozpoznat objekt, a to buď detekcí vysoké saturace *detectionBySaturation()* nebo pomocí rychlejšího rozdílu snímků. Detekci vysoké saturace použijeme v prvním příchozím snímku po pohybu kamery nebo na plechovkou obsazených křižovatkách v předchozím snímku. Detekci pomocí rozdílu snímků použijeme pokud zpracovávaný segment s křižovatkou byl v předchozím snímku prázdný.

Cannyho hranový detektor (*cannyEdgeDetector()*) testujeme s dvojicí různých algoritmů k automatickému určení prahové hodnoty pro prahování s histerezí, Ot-

suovou metodou a mediánem jasu obrazu. K detekci přímek funkcí *detectLines* testujeme Houghovu transformaci a metodu PCLines. Seskupení průsečíků vybraných přímek pomocí funkce *groupPoints()* je možno provést buď algoritmem Union Find nebo k-dimensionálním stromem. A detekce objektu pomocí rozdílu snímků (*detectionByDifference()*) je aplikována na vyhlazený obraz, kde testujeme Gaussovo vyhlazování a vyhlazování pomocí mediánu.

Veškeré měření rychlostí algoritmů bylo prováděno na procesoru Intel Pentium P6100 s taktovací frekvencí 2 GHz. Dá se tedy předpokládat, že na cílovém Raspberry Pi budou algoritmy pomalejší. Například Raspberry Pi 3 Model B má takt procesoru 1,2 GHz a v současné době, na mobilním robotu K4, osazené Raspberry Pi 2 Model B má procesor s frekvencí 900 MHz. Tento rozdíl by však neměl ovlivnit závěr měření, kterým je rozhodnutí, která z dvojic poměřovaných algoritmů je rychlejší.

Měření bylo prováděno na 300 testovacích snímcích po šesti křižovatkách, které vznikly přímo snímáním Raspberry Pi kamery přidělané na mobilním robotu K4. Při vytváření této testovací sady jsme se snažili o napodobení reálných podmínek při soutěži Ketchup House, a to různými světelnými podmínkami přirozeného světla a různým počtem a polohou plechovek na křižovatkách herního plánu.

6.1 Automatické určení práhu

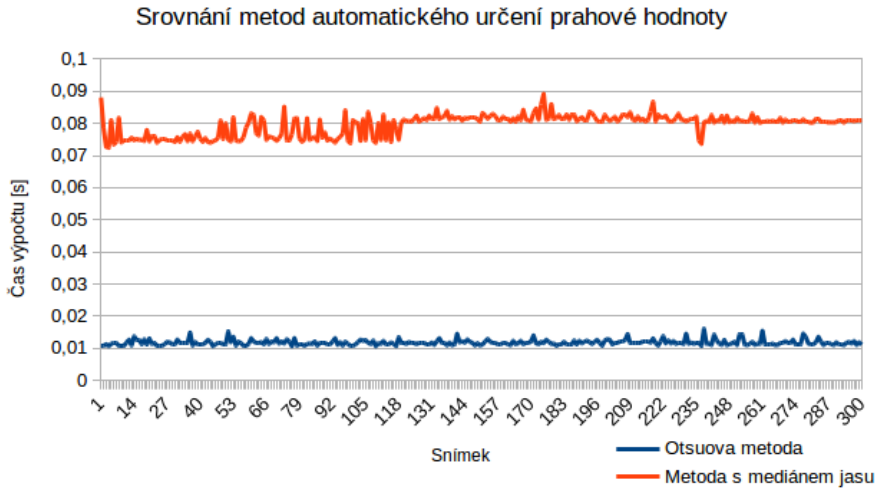
Otsuova metoda pro prahování byla implementována pomocí funkcí OpenCV. Medián jasu je zase vypočítán pomocí funkcí knihovny pro vědecké výpočty a numerické operace v Pythonu NumPy[18]. Knihovna Numpy je dle oficiální OpenCV dokumentace (2014) vysoce optimalizovaná knihovna, kterou pro některé operace využívá i OpenCV-Python. Proto implementace obou metod automatického určení práhu by měla být srovnatelná.

Měření bylo provedeno na 300 testovacích fotografiích a kritériem pro zvolení efektivnějšího algoritmu byl poměr rychlosti výpočtu a přesnosti detekce křižovatek následně provedenými algoritmy z první metody (Houghova transformace pro detekci přímek a Union Find algoritmus). Všechny testovací fotografie obsahovaly šest křižovatek. Výsledný graf měření můžeme vidět na obrázku 22 a zajímavé hodnoty pak ve srovnávací tabulce 1.

K této tabulce můžeme doplnit, že s Otsuovou metodou se nevyskytl na testovacích snímcích žádný případ chybné detekce neexistující křižovatky. Oproti tomu s prahem určeným mediánem jasu se v sedmi ze 300 testovacích snímků vyskytly případy detekce neexistující křižovatky, kterých bylo dohromady v těchto snímcích 13 (tyto chybné detekce nejsou nijak započítány v detekovaných křižovatkách ve srovnávací tabulce).

Na základě těchto hodnot můžeme rozhodnout, že Otsuova metoda pro prahování je v tomto případě vhodnější než určování práhu pomocí mediánu jasu. Nejenom, že je téměř 7× rychlejší, ale i přesnost této metody je přibližně 90 %, zatímco

[18]Knihovna je dostupná na: http://www.numpy.org/



Obrázek 22: Srovnání metod automatického určení prahové hodnoty pro Cannyho hranový detektor.

Tabulka 1: Vybrané hodnoty z měření rychlosti a přesnosti automatického určení práhu pro Cannyho hranový detektor.

Měření	Minimum	Maximum	Průměr	Medián
Otsuova Metoda:				
Určení práhu [s]	0,0106	0,0161	0,0118	0,0116
Cannyho hranový detektor [s]	0,159	0,400	0,248	0,260
Detekované křižovatky [počet]	2	6	5,4	5
Medián jasu:				
Určení práhu [s]	0,0724	0,0892	0,0795	0,0807
Cannyho hranový detektor [s]	0,233	0,919	0,351	0,352
Detekované křižovatky [počet]	1	6	4,7	5

medián jasu dosáhl pouze 78 %. V dalším měření bude tedy v Cannyho hranovém detektoru dále používána pouze Otsuova metoda.

6.2 Detekce přímek

Houghova transformace pro detekci přímek byla implementována pomocí funkcí OpenCV. Metoda PCLines je celá implementována v C/C++ s využitím některých funkcí OpenCV. Výpočet obou algoritmů je tedy zajištěn voláním C/C++ funkcí z Pythonu.

Měření detekce přímek bylo provedeno na 300 testovacích obrazech po šesti křižovatkách stejně jako měření Cannyho hranového detektoru. I zde byly kritéria výběru efektivnějšího algoritmu rychlost výpočtu a přesnost detekce křižovatek. Ani Houghova transformace ani PClines nikdy nedetekovaly neexistující křižovatku,

Tabulka 2: Vybrané hodnoty z měření rychlosti a přesnosti detekce přímek.

Měření	Minimum	Maximum	Průměr	Medián
Houghova transformace:				
Detekce přímek [s]	0,0406	0,0721	0,0497	0,0494
Detekované křižovatky [počet]	2	6	5,4	5
PCLines:				
Detekce přímek [s]	2,569	6,654	4,689	4,581
Detekované křižovatky [počet]	2	6	5,1	5

v tabulce 2 výsledků měření však vidíme, že metoda PCLines je o dva desetinné řády pomalejší než Houghova transformace.

Výhody metody PCLines jsou nezpochybnitelné v oblasti akcelerace například pomocí GPU či při použití na architekturách, kde není vhodné využívat operace s pohyblivou desetinnou čárkou, v našem případě však není vhodné ji použít. Ani v přesnosti neobstála oproti Houghově transformaci, PCLines detekovala o 5 % méně křižovatek, tedy dosáhla přesnosti 85 %.

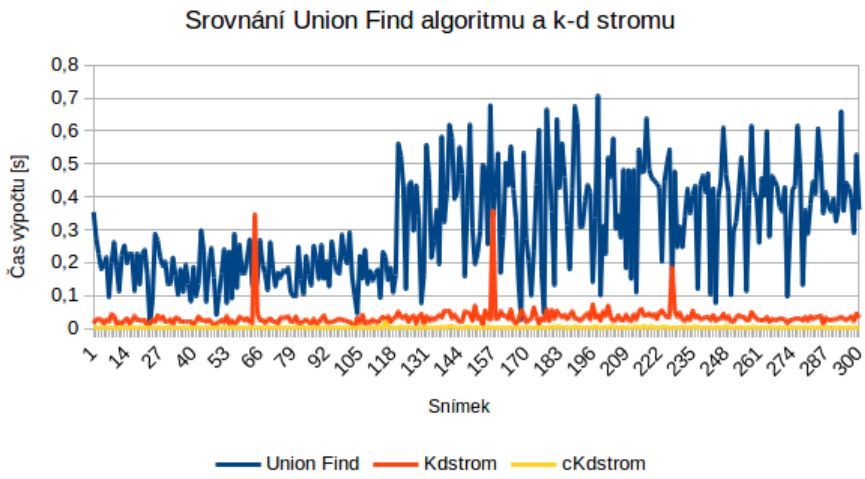
6.3 Seskupení průsečíků

Algoritmus Union Find je implementován v Pythonu s využitím objektově orientovaného programování. K-dimensionální strom je implementován pomocí funkcí Python knihovny vědeckých nástrojů SciPy[19], která je postavena na funkcích knihovny NumPy. Pro srovnání je do měření také zahrnuta implementace k-dimensionálního stromu v C/C++.

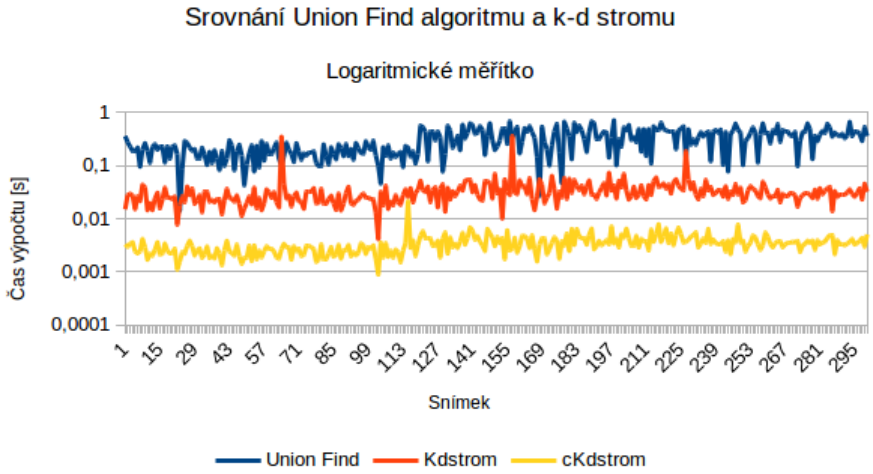
Vzhledem k tomu, že výstupy obou srovnávaných algoritmů jsou totožné a závisí pouze na zvolené maximální vzdálenosti mezi průsečíky pro seskupení, tak v tomto měření nás zajímá pouze rychlost algoritmu, která se testovala na 300 snímcích. Na grafu 23 vidíme, že k-dimensionální strom je ve většině případů přibližně o jeden desetinný řád rychlejší. K-dimensionální strom implementovaný v jazyce C/C++ (na grafu *cKDtree*) je ještě o další desetinný řád rychlejší. Přehlednější srovnání jednotlivých algoritmů pak můžeme sledovat na grafu 24 s logaritmickým měřítkem.

Na tomto grafu je také vidět, že funkce grafu obou k-dimensionálních stromů je až na výjimky velmi podobná, jen posunutá. Tyto výjimky, kdy časová náročnost algoritmu silně stoupne, však mohou být způsobeny hardwarem nebo operačním systémem. Statistické hodnoty všech tří algoritmů jsou v tabulce 3. Z grafů i tabulky jasně vyplívá, že pro námi řešený problém seskupení průsečíků je algoritmus dvoudimensionálního stromu mnohem vhodnější.

[19]Knihovna je dostupná na: https://www.scipy.org/



Obrázek 23: Srovnání Union Find algoritmu a implementací k-dimensionálních stromů v Pythonu a C.



Obrázek 24: Srovnání Union Find algoritmu a implementací k-dimensionálních stromů v Pythonu a C v logaritmickém měřítku na ose y.

Tabulka 3: Vybrané hodnoty z měření rychlosti algoritmů pro seskupení průsečíků dle podobné polohy v obraze.

Algoritmus	Minimum	Maximum	Průměr	Medián
Union Find [s]	0,0142	0,706	0,302	0,269
K-D strom [s]	0,00412	0,358	0,0334	0,0289
cK-D strom [s]	0,000892	0,0212	0,00341	0,00325

6.4 Vyhlazování segmentů

Jak Gaussovo vyhlazování, tak vyhlazování pomocí mediánu je implementováno pomocí funkcí OpenCV. Díky tomu by poměr rychlostí jednotlivých algoritmů měl

záviset pouze na jejich časové složitosti. Konkrétně nás zajímá součet doby výpočtu dvojice Gaussových vyhlazování s konvolučními maticemi 3×3 a 5×5, tedy vyhlazování zpracovávaného pixelu dle okolí do vzdálenosti jednoho a dvou pixelů. Stejné velikosti matic jsou použity i u vyhlazování mediánem. Silnější vyhlazování je aplikováno na snímek s původní prázdnou křižovatkou, slabší na nový snímek, kde detekujeme, zda se stav změnil.

Změnu stavu křižovatky detekujeme pomocí počtu bílých bodů v segmentu křižovatky po vyprahování. Dalším kritériem porovnávání tedy je přesnost detekce objektů s použitým vyhlazováním, která je udána poměrem bílých bodů na prázdných a obsazených křižovatkách. V případě, že nová křižovatka je také prázdná, počet bílých bodů by měl být co nejblíže nule, v opačném případě by měl být co nejvyšší, až do hodnoty počtu pixelů v segmentu obrazu.

Měření rychlosti vyhlazování bylo provedeno na 30 obrazech, což v tomto případě byl dostatečný počet testovacích vzorků. Měření přesnosti rozdílu snímků s daným vyhlazováním bylo provedeno na 160 vzorových křižovatkách, z nichž 42 bylo obsazených. Pro tuto metodu se ukázalo efektivním nedetekovat plechovku na celém segmentu, tvořeným 80 % výšky plechovky nad křižovatkou a 20 % výšky plechovky pod ní, ale zmenšit segment na pouhých 40 % výšky plechovky nad křižovatkou. Díky této úpravě se vyhneme většině případů, kdy plechovka zasahuje do segmentu křižovatky jiné plechovky.

Tabulka 4: Průměrná doba a maximální odchylka různých algoritmů pro vyhlazování.

Typ vyhlazování	Průměr [s]	Max. odchylka [s]
Gaussovo vyhlazování 3 ×3	0,00973	0,000608
Gaussovo vyhlazování 5 ×5	0,0159	0,00287
Vyhlazování mediánem 3 ×3	0,185	0,00409
Vyhlazování mediánem 5 ×5	6,37	0,190

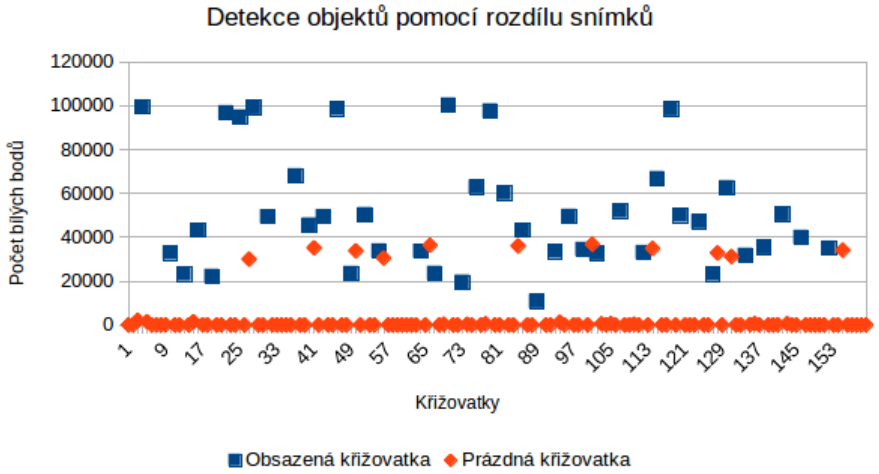
Nutný čas k provedení vyhlazování obrazu se snímek od snímku příliš neliší. Jak vidíme v tabulce 4, vyhlazování mediánem s konvoluční maticí o velikosti 5 je pro naše účely zcela nevhodný z důvodu velké časové složitosti. I slabší vyhlazování mediánem je mnohem pomalejší než Gaussovo vyhlazování a mohli bychom zvažovat jeho použití jen, pokud by to znamenalo značné zvýšení přesnosti detekce objektů metodou rozdílu snímků.

Tato podmínka pro použití vyhlazování mediánem nebyla naplněna. Tabulka 5 ukazuje celkové počty bílých bodů po provedení detekce objektů s Gaussovým vyhlazováním a rozdíly těchto hodnot v případě použití vyhlazování mediánem. Jak vidíme, tak rozdíly jsou na základě hodnot z této tabulky zanedbatelné a navíc horší. V případě prázdných křižovatek sice dosáhlo průměrného zlepšení o více než 11 bodů, ale v případě obsazených se jedná o zhoršení o více než 46 bodů. Není tedy důvod používat vyhlazování mediánem.

Výsledky detekce objektů pomocí rozdílu snímků s Gaussovým vyhlazováním jsou lépe znázorněny na grafu na obrázku 25. Počet bílých pixelů na prázdných

Tabulka 5: Vybrané hodnoty z měření přesnosti detekce objektů pomocí rozdílu snímků.

Typ křižovatky	Minimum	Maximum	Průměr	Medián
Počty bílých bodů po rozdílu snímků:				
Prázdná křižovatka	0	36781	3266,3	0
Obsazená křižovatka	10654	100257	51314	46265
Rozdíly v počtu bílých bodů při vyhlazování mediánem místo Gaussova:				
Prázdná křižovatka	-389	149	-11,4	0
Obsazená křižovatka	-433	2067	-46,1	-69



Obrázek 25: Počty bílých pixelů po detekci objektů pomocí rozdílu snímků.

křižovatkách nepřesáhne 2 500 bodů a na obsazených křižovatkách není nižší než 10 000 bodů. To představuje velký rozdíl, na základě kterého jsme schopni klasifikovat objekt. Výjimkou bylo během testování 11 případů kdy se počty bílých bodů na prázdné křižovatce pohybovaly kolem hodnoty 30 000. Tyto chybné detekce byly ve třech případech způsobeny tím, že plechovka na blízké křižovatce před kamerou zcela zakryla plechovku za ní na vzdálenější křižovatce (což pro robota K4 nepředstavuje problém). Zbylých 8 chyb bylo způsobeno kombinací změny světelných podmínek a pootočením kamery mezi dvojicí snímků. Tato metoda tak dosáhla více než 93% přesnosti.

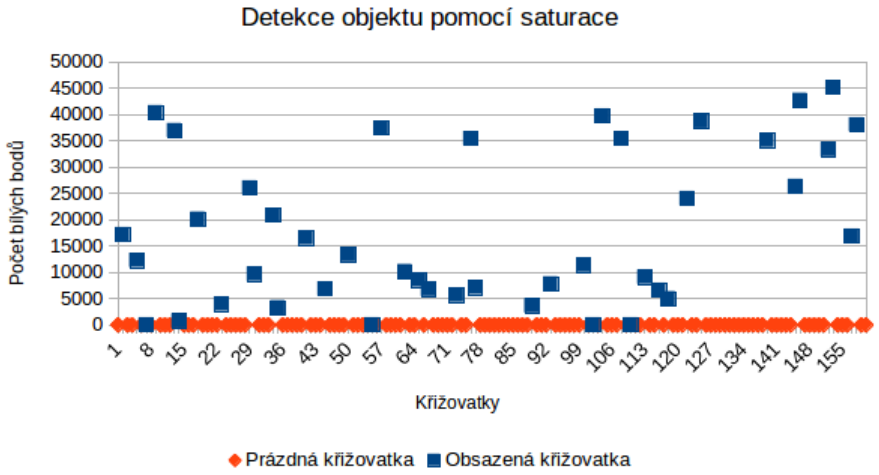
6.5 Metoda z efektivnějších algoritmů

Na základě předchozích sekcí jsme pro výslednou metodu pro opakované vyhledávání objektů vybrali algoritmus Otsuovy metody pro automatické určení práhu prahování s histerezí Cannyho hranového detektoru. Dále Houghovu transformaci pro detekci přímek, v programovacím jazyku C implementovaný dvou-dimensionální strom pro seskupení průsečíků vybraných přímek a nakonec Gaussovo vyhlazování pro mini-

malizaci rizika špatné detekce objektu rozdílem snímků z důvodu mírného posunutí kamery. Všechny algoritmy až na k-d strom jsou z první metody popsané v minulé kapitole. K-d strom je popsán v metodě druhé.

Tabulka 6: Vybrané hodnoty z měření přesnosti detekce objektů pomocí vysoké saturace.

Typ křižovatky	Minimum	Maximum	Průměr	Medián
Prázdná křižovatka	0	0	0	0
Obsazená křižovatka	0	45120	18024,6	12778

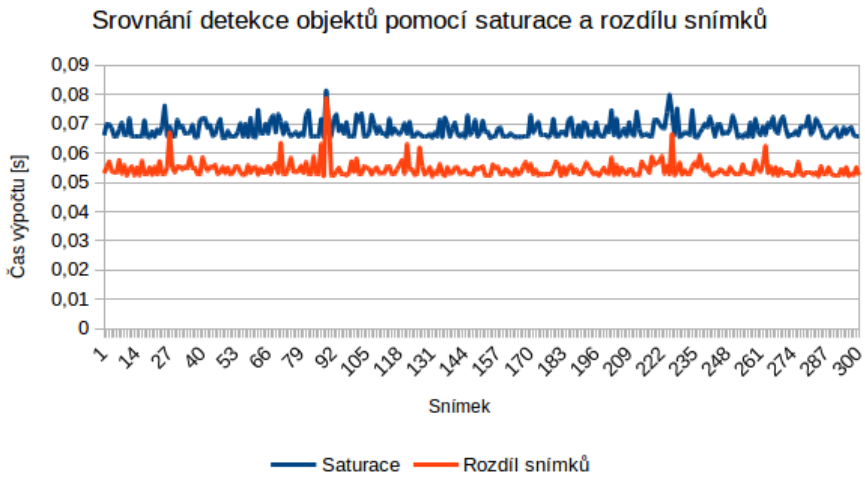


Obrázek 26: Počty bílých pixelů po detekci objektů pomocí vysoké saturace.

U této složené metody nám zbývá změřit rychlost a přesnost rozpoznávání objektů pomocí detekce vysoké saturace, a tak ověřit, že algoritmus rozdílu snímků je skutečně rychlejší, aniž by byla příliš ovlivněna spolehlivost. Přesnost detekce vysoké saturace byla testována na stejných 160 segmentech obrazu jako rozdíl snímků z předchozí sekce. Stejně jako u rozdílu snímků se zde snažíme získat co nejmenší hodnoty počtu bílých bodů v segmentech prázdných křižovatek a co nejvyšší u obsazených křižovatek. Výsledky měření můžeme vidět v tabulce 6 a na obrázku grafu 26.

Jak vidíme, prázdná křižovatka je vždy popsána nulovým počtem bílých bodů, obsazená křižovatka nenulovým, a to až do hodnoty desítek tisíc bodů. Pouze ve čtyřech případech detekce selhala. Selhání se projevilo opačným případem než u detekce rozdílem snímků, plechovka nebyla rozpoznána. Ve všech čtyřech případech k tomuto selhání došlo z důvodu zakrytí vzdálené plechovky bílým vrškem bližší plechovky, což v praxi nepředstavuje velký problém, protože bližší plechovka byla detekována správně. Tato metoda tak tedy dosáhla více než 97% přesnosti.

Rychlost algoritmu byla měřena na 300 testovacích snímcích obsahujících 1 608 detekovaných křižovatek a poměřena s rychlostí algoritmu rozdílu snímků. Výsledky



Obrázek 27: Srovnání algoritmů pro detekci objektu.

Tabulka 7: Vybrané hodnoty z měření rychlosti algoritmů pro detekci objektu.

Měření	Minimum	Maximum	Průměr	Medián
Saturace [s]	0,0651	0,0812	0,0679	0,0669
Rozdíl snímků [s]	0,0520	0,0786	0,0545	0,0535

měření jsou na obrázku 27 a v tabulce 7. Na těchto výsledcích vidíme, že rozdíl snímků je opravdu **rychlejší v průměru o téměř 25 %**. Je na uvážení, zda v konkrétní aplikaci této metody se toto zrychlení vyplatí za cenu snížení přesnosti o přibližně 4 %.

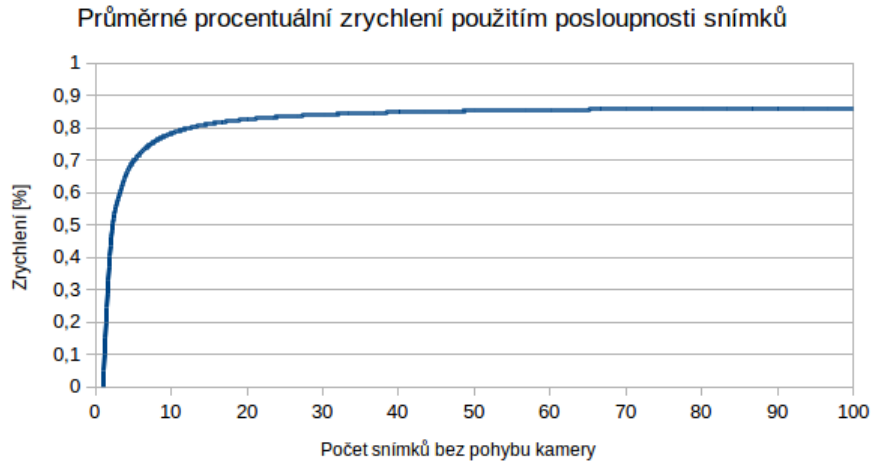
6.6 Vyhodnocení

Výsledná metoda je schopná detekovat křižovatky s 90% úspěšností a na nich pak rozpoznat plechovky s 97% či 93% úspěšností dle použitého algoritmu detekce. Rychlost zpracování jednoho obrazu vidíme na obrázku 28 a tabulce 8, kde hodnoty byly získány měřením na 300 testovacích obrazech s křižovatkami zpracovávanými pomocí detekce vysoké saturace.

Tabulka 8: Vybrané hodnoty z měření rychlosti výsledné metody pro detekci plechovek na herním hřišti.

Měření	Minimum	Maximum	Průměr	Medián
Detekce křižovatek [s]	0,270	0,844	0,402	0,408
Detekce plechovek [s]	0,0651	0,0812	0,0679	0,0669
Celkově [s]	0,335	0,917	0,470	0,475

Obrázek 28: Graf rychlostí výsledné metody pro detekci plechovek na herním hřišti v jednom snímku.



Obrázek 29: Graf průměrného zrychlení metody použitím posloupnosti snímků.

Tabulka 9: Průměrné zrychlení metody použitím posloupnosti snímků.

Počet snímků bez pohybu kamery	1	2	3	4
Po jednotlivých snímcích [s]	0,470	0,940	1,410	1,880
Využití posloupnosti snímků [s]	0,470	0,531	0,592	0,654
Procentuální zrychlení [%]	0	43	58	65

Za předpokladu, že pravděpodobnost prázdné i obsazené křižovatky je stejná,

tak výsledné zrychlení použitím posloupnosti snímků je

$$\frac{T_0 + T_1 + \frac{T_1+T_2}{2} \times (x-1)}{(T_0 + T_1) \times x}$$

, kde T_0 je čas nutný pro předzpracování obrazu a segmentaci, T_1 je čas potřebný pro detekci objektů pomocí saturace, T_2 je čas nutný pro detekci objektů pomocí rozdílu snímků a x je počet po sobě jdoucích snímků bez pohybu kamerou. Výsledné průměrné zrychlení v procentech vidíme v tabulce 9 a na grafu 29, kde si můžeme všimnout, že zrychlení se téměř ustálilo na 86 %. Už pro dvojici snímků se tedy jedná o **průměrné zrychlení 43 %**.

7 Závěr

Před začátkem této práce měl mobilní robot K4, sestavený pro soutěž Ketchup House na Robotickém dni v Praze, schopnost detekovat objekt pouze na jediné křižovatce přímo před ním. Tato práce přišla s posloupností algoritmů, které umožní tomuto robotu s dostatečnou přesností a vysokou rychlostí detekovat objekty na všech křižovatkách před ním.

Tento znatelný postup vpřed lze provést mnoha různými způsoby a metodami používanými v počítačovém vidění a zpracování obrazu. V rámci této práce jich bylo hned několik popsáno, implementováno, otestováno a vzájemně poměřeno ve snaze získat algoritmy rychlé, efektivní, optimalizované a kompatibilní se zbytkem použitých metod.

Tato snaha se povedla a výstupem práce je nejenom metoda schopná zpracovat snímek a detekovat v něm plechovky pro robota K4, ale také možnost zvýšit rychlost algoritmu použitím posloupností snímků. Práce také obsahuje již výše zmíněné srovnání několika algoritmů, které jsou obecně použitelné nejen v oblasti zpracování obrazu, a mohou tak být použity pro jiné projekty, k nimž tato práce může být inspirací. Obzvlášť přihlédneme-li k tomu, že například algoritmus PCLines je poměrně nový a informací o něm je v odborné literatuře málo.

Na základě výsledků měření bylo zjištěno, že vhodným algoritmem pro automatické určení prahu Cannyho hranového detektoru je Otsuova metoda. Dále pak, že bez využití GPU je na běžné architektuře Houghova transformace pro detekci přímek rychlejší než metoda PCLines. Pro sjednocování podobných prvků z množin dvoudimensionálních dat je velmi efektivním algoritmem k-d strom a jako rychlé vyhlazování se nejvíce osvědčilo Gaussovo vyhlazování.

Je jen na robotickém týmu AiStorm Mendelovy univerzity v Brně, zda se výslednou metodu detekce objektů či její části rozhodnout využít a nasadit na soutěžního robota K4. V případě nasazení by jako návrh dalšího zlepšení mohlo být nalezení vhodné alternativy ke Cannyho hranovému detektoru. Tento několika úrovňový algoritmus je sice v mnoha tutoriálech včetně OpenCV dokumentace doporučovaný před použitím detekce přímek, ale zároveň je také nejpomalejším článkem celé metody. Na druhou stranu jednodušší a rychlejší metody neposkytují dostatečnou přesnost.

8 Reference

AiStorm. *Mobilní robot K4* [online]. 2016.
Dostupné z: https://aistorm.mendelu.cz/cz/projekty/k4/.

Bay, A., Ess, A., Tuytelaars, T. a Gool, L. V. *Computer Vision and Image Understanding: Speeded-Up Robust Features (SURF).* Amsterdam: Elsevier, 2006.
ISSN 1077-3142.

Brahmbhatt, S. *Practical OpenCV.* New York: Apress, 2013.
ISBN 978-1-430-26079-0.

Dubská, M., Havel, J. a Herout, A. *Proceedings of SCCG 2011: Real-Time Detection of Lines using Parallel Coordinates and OpenGL.* Bratislava: Comenius University in Bratislava, 2011.
ISBN 978-80-223-3018-3.

Dubská, M., Herout, A. a Havel, J. *Proceedings of CVPR 2011: PClines - Line Detection Using Parallel Coordinates.* Colorado Springs: IEEE Computer Society, 2011.
ISBN 978-1-4577-0393-5.

Granlund, G. H. a Knutsson, H. *Signal Processing for Computer Vision.* Boston: Kluwer Academic Publishers, 1995.
ISBN 0792395301.

Havel, J., Herout, A., Dubská, M., a Jošth, R. *Journal of real-time image processing: Real-time detection of lines using parallel coordinates and CUDA.* Berlin: Springer Berlin Heidelberg, 2014.
ISSN 1861-8200.

Hlaváč, V. a Sedláček, M. *Zpracování signálu a obrazu.* Praha: Vydavatelství ČVUT, 2009.
ISBN 978-80-01-04442-1.

Holota, R. a Fiřt, J. *Digitální mikroskopie a analýza obrazu v metalografii: Digitalizace a zpracování obrazu.* Plzeň: Západočeská univerzita, 2002.
ISBN 80-7082-917-6.

ImageJ. *Basic Concepts* [online]. 2016.
Dostupné z: http://imagej.nih.gov/ij/docs/concepts.html.

Isometric Land. *Art & Design* [online]. 2016.
Dostupné z: http://http://isometricland.net/artwork/artwork.php.

Kolomazník, J. *Aplikace moderních metod pro identifikaci obrazových dat.* Disertační práce. Brno, 2014.

LARSON, B. *GPU-Accelerated Machine Vision.* [online]. 2015.
Dostupné z: https://www.objc.io/issues/21-camera-and-photos/gpu-accelerated-machine-vision/.

LIM, J. S. *Two-dimensional signal and image processing.* Englewood Cliffs, N.J.: Prentice Hall, 1990.
ISBN 0139353224.

LOWE, D. G. *International Journal of Computer Vision: Distinctive Image Features from Scale-Invariant Keypoints.* New York: Springer US, 2004.
ISSN 0920-5691.

MARTIŠEK, D. *Matematické principy grafických systémů.* Brno: Littera, 2002.
ISBN 80-85763-19-2.

MATHWORKS. *Convert RGB image or colormap to grayscale* [online]. 2016.
Dostupné z: http://www.mathworks.com/help/matlab/ref/rgb2gray.html.

MOESLUND, T. *Canny Edge Detection.* [online]. 2009.
Dostupné z: http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf.

PARKER, J. R. *Algorithms for image processing and computer vision.* New York: John Wiley & Sons, 2011.
ISBN 0-471-14056-2.

PRINCE, S. J. *Computer vision: models, learning, and inference.* New York: Cambridge University Press, 2012.
ISBN 978-1-107-01179-3.

ROBOTIC DAY. *Ketchup House* [online]. 2016.
Dostupné z: roboticday.org/2016/rules/2016-Ketchup_House-ENv1.pdf.

ŠONKA, M., HLAVÁČ, V. A BOYLE, R. *Image processing analysis and machine vision.* London: Chapman & Hall, 2007.
ISBN 978-0495082521.

ŠŤASTNÝ, J. *Netradiční metody a algoritmy pro rozpoznávání objektů technologické scény: Nontraditional methods and algorithms for object recognition of technological scene: zkrácená verze habilitační práce.* Brno: VUTIUM, 2006.
ISBN 80-214-3117-2.

UMBAUGH, S. E. *Digital Image Processing and Analysis: Human and Computer Vision Applications with CVIPtools.* Boca Raton: Taylor & Francis, 2010.
ISBN 978-1439802052.

WONG, K. D. *Canny Edge Detection Auto Thresholding* [online]. 2009.
Dostupné z: http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/.