Mendelova univerzita v Brně
Provozně ekonomická fakulta

Využití GPU výpočtů pro rozpoznání dopravních značek

Diplomová práce

Vedoucí práce:
Ing. David Procházka, Ph.D.

Bc. Karel Zídek

Brno 2015

Zadání práce

Děkuji vedoucímu mé práce, panu Ing. Davidu Procházkovi, Ph.D., za odborné vedení, trpělivost a ochotný přístup. Poděkování patří rovněž přítelkyni Svetlaně za její inspiraci a trpělivost a mé rodině za podporu během celého studia a při psaní této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Využití GPU výpočtů pro rozpoznání dopravních značek**
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 15. 5. 2015 .....

Abstract

Zídek, Karel Bc. *Usage of GPU acceleration in traffic sing recognition* Diploma thesis. Brno, 2014.

The thesis deals with the problem of GPU acceleration of algorithms for traffic sign recognition. Theoretical part of the thesis outlines methods for object detection with emphasis on the traffic sign detection problem. Further, it provides comparison of two well known tools for programming on the GPU: CUDA and OpenCL. On the basis of the review, an architecture of own solution is proposed. Finally, the thesis contains description of the implementation as well as evaluation of the results.

Keywords

GPGPU, object detection, traffic sign, OpenCV, CUDA, OpenCL

Abstrakt

Zídek, Karel Bc. *Využití GPU výpočtů pro rozpoznání dopravních značek.* Diplomová práce. Brno, 2015.

Tato práce se zabývá problémem GPU akcelerace algoritmů pro rozpoznávání dopravních značek. Teoretická část popisuje metody pro detekci objektů s důrazem na detekci značek. Dále poskytuje porovnání dvou nástrojů pro programování na GPU: CUDA a OpenCL. Na základě těchto zkoumání je navržena vlastní architektura detekce značek. Práce dále obsahuje popis implementace a porovnání výsledků.

Klíčová slova

GPGPU, detekce objektů, dopravní značka, OpenCV, CUDA, OpenCL

Obsah

Seznam obrázků

1 Úvod a cíl práce

1.1 Úvod

Potřeba dopravního značení se proplétá lidskou historií stejně, jako cesty, které značení obklopuje. I známé úsloví *„Všechny cesty vedou do Říma“* je odvozeno od Římských milníků, které začaly být využívány přibližně sto let před naším letopočtem a udávaly vzdálenost od jmenovaného města. Značky moderního charakteru spatřily světlo světa mnohem později. Teprve s rozvojem automobilismu byly v Anglii, začátkem minulého století, zákonem stanoveny značky v podobě tak, jak je známě dnes, tedy zvláště kruhy a trojúhelníky. Tyto tvary se staly natolik univerzální, že ač provedení značek se v jednotlivých zemích může lišit, tento aspekt zůstává celosvětově univerzální.

Potřeba rozpoznat dopravní značky se stala klíčovou nejen pro řidiče, ale postupně se dostala do spektra zájmu automatických strojů a nástrojů. Nejviditelněji se s tímto problémem začali zabývat výrobci automobilů, kteří ve snaze nabízet modernější vybavení svých luxusních vozů vytvořily první implementace rozpoznávání značek. Jejich postupy vycházejí sice z vědeckých poznatků a postupů dnešní doby v oblastech počítačového vidění, ale jejich konkrétní řešení jsou bohužel střeženy, jako průmyslová tajemství, a nejspíše tomu tak ještě nějaký čas bude.

Mnoho firem, které se zabývají kartografií a mapováním narazilo následně na stejný problém. Netížila je sice rychlost zpracování, které je u vozu třeba stíhat v reálném čase, díky tomu bylo možné problém řešit *„manuálně“*. Technik tedy strávil většinu své pracovní doby procházením obrázků a označováním nalezených značek a jejich popisem. Z tohoto důvodu se ukázalo jako přínosné navrhnout systém který by tuto činnost zjednodušil a co nejvíce z této stereotypní práce přenesl na stroj, který může úkol řešit během noci a poskytne technikovi pouze jednoduchý vhled pro kontrolu.

Jelikož se jedná o mnoho výpočtů nad RGB daty, které stále dokola počítá procesor, ukázalo se jako jedna z možných cest, obohatit navržené řešení a jeho implementaci o zpracování založené na principu GPGPU (General-purpose computing on graphics processing units). Ačkoliv není primárně usilováno o řešení poskytovaná v reálném čase je jistě vhodné průběh řešení zkrátit pomocí běžně dostupných technologií, protože data získané během mapování silnic jsou velmi objemná, často v řádech stovek GiB.

1.2 Cíl práce

V rámci této práce, bude navržena architektura aplikace pro detekci dopravního značení a vytvořena její experimentální implementace, které bude dále rozšířena o GPGPU výpočty pro zvýšení výkonu. Tento modul bude rozlišovat a detekovat zvolené dopravní značky v RGB podkladech. Pro splnění tohoto úkolu je třeba splnit několik dílčích cílů:

- Prostudovat metody obecných výpočtů na GPU, prostudovat kritéria a zvolit vhodnou metodu pro použití v rámci implementace řešení.
- Prostudovat algoritmy počítačového vidění a strojového učení, seznámit se s nimi.
- Stanovit kritéria pro posouzení a užitečnosti algoritmů počítačového vidění pro dosažení cíle a zvolit vhodné metody k začlenění do vlastního implementovaného řešení.
- Vytvořit vhodnou testovací množinu dat, na které bude možné hotové řešení otestovat a porovnat.
- Navrhnout vlastní postup řešení, provést jeho implementaci a ověření výsledků.

Pro realizaci hlavního cíle musí být dílčí cíle splněny.

2 Rozpoznání dopravních značek

Stručná verze této rešerše byla publikovaná v článku *Architecture of Assistance System for Traffic Signs Inventory* (30).

V současné době již bylo publikováno množství článků, které se zabývají detekcí dopravních značek. Bohužel velké množství těchto řešení ukazuje postupy vhodné pro ideální atmosferické podmínky. Mnoho takových řešení narazí na problémy, pokud se dostanou do reálných podmínek na silnicích, kde dochází k rapidním změnám počasí a faktory těchto změn – např. do deště, sněhu či mlhy. To v kombinaci s dalšími technickými faktory, jako např. vyvážení bílé či optické zkreslení dále snižuje účinnost algoritmů. Příklad těchto problémů ilustruje obrázek 1.

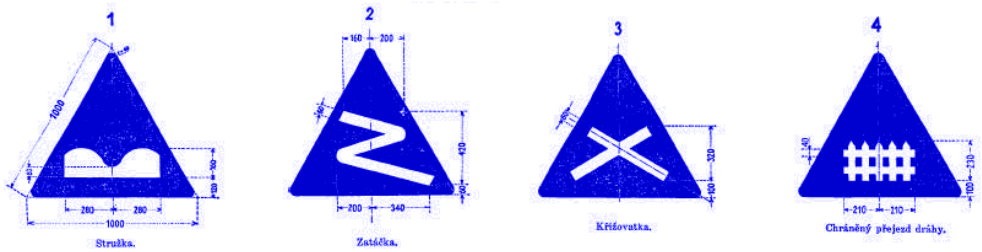
Obrázek 1: Snímky značek narušené šumem, špatnými světelnými podmínkami a deštěm. Zdroj: Karel Zídek, Data: Geodis

Firmy pracující v oblasti mapování, jako např. *Google*, *TomTom* či *Mapy.cz* se právě při inventarizaci potýkají s uvedenými problémy. Při zpracování nevyžadují výsledky dostupné v reálném čase, ale dostatečně přesné řešení, které korektně identifikuje značky i při kombinaci nepříznivých faktorů je klíčové pro využití takovýchto řešení.

2.1 Vývoj podoby dopravního značení

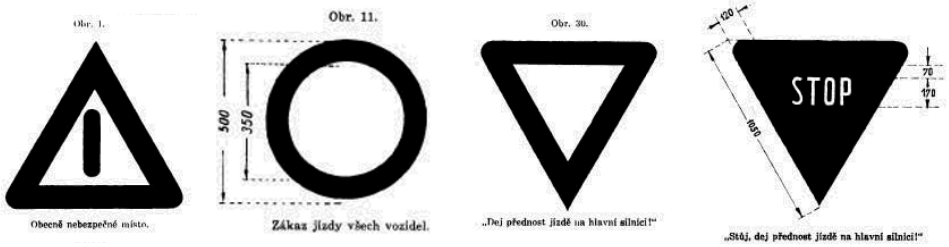
Dopravní značení moderního charakteru začalo být předpokládáno a požadováno skrze tzv. Pařížskou konvenci z roku 1909. V Československu byly zákonem zavedeny výstražné značky od 1. listopadu roku 1935. Značky se tvarově prakticky nelišily od současné podoby výstražných značek – jednalo se šest značek ve tvaru rovnostranného trojúhelníku. Jejich barevné provedení však bylo v modré barvě, jak ilustruje obrázek 2. Symbolika se v obecné rovině blížila symbolice, která je používaná i dnes.

V roce 1938 byl počet výstražných značek rozšířen na jedenáct a dále přibyly kategorie zákazových značek a značek informačních. Zákon dále zaváděl i světelné značky pro řízení dopravy a značkování silnic. Podoba zákazových značek byla stanovena kruhovým tvarem s červeným orámováním.



Obrázek 2: Podoba dopravních značek z roku 1935. Zdroj: Vládní nařízení 203/1935 Sb. (31)

O rok později roku 1939, v rámci Protektorátu Čechy a Morava, byla provedena další úprava dopravního značení. Asi nejvýraznější změnou byla změna provedení výstražných značek z původního modro-bílého do současné podoby s červeným orámováním. Obrázek 3 ilustruje podobu značek z tohoto období.



Obrázek 3: Podoba dopravních značek z roku 1939. Zdroj: Vládní nařízení 242/1939 Sb. (32)

Poválečné období přineslo další vývoj a zvláště rozšíření počtu značek. Za zajímavou změnu lze považovat změnu značky „Stop" z trojúhelníkového tvaru na kruhový jak ilustruje obrázek 4. Tato podoba vydržela až do roku 1976, kdy byla změněna na dnes stále používaný osmiúhelníkový tvar.

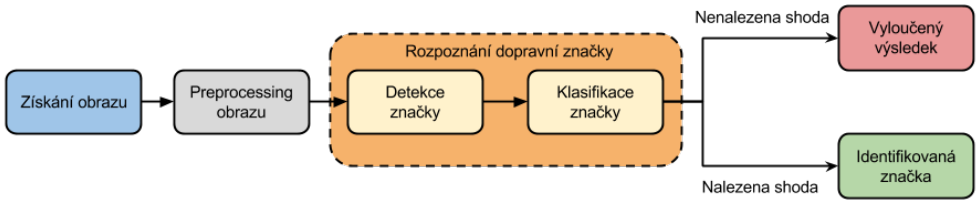
Obrázek 4: Podoba dopravních značek z roku 1961. Zdroj: Vyhláška 141/1960 Sb. (33)

V letech 1961, 1967, 1971 a 1976 docházelo k aktualizacím a úpravám, které již neměly zásadní vliv na vizuální podobu značek. Báze značek byla rozšiřována, zvláště v oblasti informačních a doplňkových značek.

Současná podoba značení je dána *Vyhláškou ministerstva vnitra č. 30/2001 Sb.*, která nahradila předchozí československou podobu z roku 1989. I tato úprava je průběžně novelizována – např. v roce 2009 bylo zavedeno značení pro měření rychlosti a v roce 2010 rozšířeno značení týkající se cyklistického provozu.

2.2 Proces rozpoznání a identifikace

Proces rozpoznání značky je obecně ilustrovaný na obrázku 5. V prvním kroku je získán RGB obrázek, který může potencionálně obsahovat značku. V dalším kroku je obrázek připraven pro zpracování rozpoznávacími algoritmy. To obnáší zbavení se co největšího množství nepotřebných informací (barva) a napravení opravitelných chyb (zmírnění šumu, zkreslení). V dalším kroku, tedy při detekci značky je následně vyhledávána oblast zájmu, ve které je pravděpodobný výskyt dopravní značky. V této části je možné hledat konkrétní barvu, tvar či kombinaci apod. – tedy vlastnosti typické pro značky na základě kterých lze z celého obrazu určit pouze oblasti pravděpodobného výskytu značky. Zvolené oblasti jsou poté zpracovány klasifikačním modulem, který se pokusí určit konkrétní typ značky. V případě úspěchu je značka identifikována v případě neúspěchu je možné výstup zamítnout či přenechán k přehodnocení technikem.



Obrázek 5: Diagram procesu identifikace dopravní značky. Zdroj: Karel Zídek (30)

Způsoby preprocesingu, detekce a klasifikace se odvíjejí od účelu rozpoznávacího systému. V případě systému určeného např. pro dopravní prostředky je třeba uvažovat detekci v reálném čase a tedy přizpůsobit jednotlivé kroky tomuto účelu. V případě např. offline systému pro inventarizaci sesbíraných dat je možné zvolit řešení nepracující v reálném čase, ale přinášející předpokládané výhody v podobě přesnější detekce.

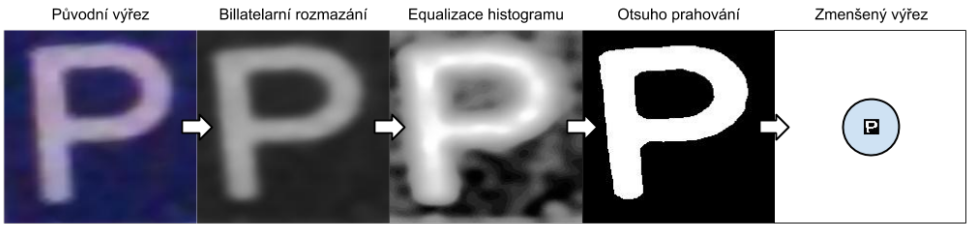
Pro nalezení oblasti, ve které je předpokládaný výskyt dopravní je možné využít rozdílné přístupy. Každý přístup zohledňuje některou z vlastností hledané značky (tvar, barva). Každý z těchto přístupů má své slabá a silná místa proto je pro dosažení optimálního výsledku vhodné tyto přístupy vhodně kombinovat či používat souběžně, tak aby se výsledky jednotlivých cest navzájem podporovaly. Je zřejmé, že takovýto přístup nemusí být vhodný pro některé typy systémů. V případě systému pracujících v reálném čase může být takovýto přístup nepoužitelný. Pro identifikaci značky taktéž existuje množství přístupů. U některých je nutné v předcházející fázi nalézt vhodné oblasti zájmu a tedy značku nejdříve detekovat, jiné mohou přímo

na vstupním snímku značku identifikovat a proces nalezení zcela přeskočit. Některé přístupy je možné používat samostatně i kombinovat.

2.3 Zpracování obrazu

Existuje široká škála algoritmů zpracování obrazu, které se používají jako podpůrné prostředky pro různé metody detekce objektů v obraze. Tyto algoritmy lze zařadit do kategorie *předzpracování obrazu* (*preprocessing*). S jejich pomocí lze upravit vstupní obrazová data do podoby, která vede ke zlepšení kvality výsledků zvoleného detekčního algoritmu.

Využití *preprocessingu* pro podporu detekce objektů je třeba vždy zvážit v kontextu použití zvolené detekční metody. Existují metody detekce, při kterých lze považovat některé druhy *preprocessingu* za nedílnou součást, u jiných je jejich použití zbytečné, případně nevhodné.



Obrázek 6: Schéma jednotlivých kroků zpracování obrazu. Zdroj: Karel Zídek

V rámci *preprocessingu* vstupních dat je cílem poskytnout do následující fáze detekce data zbavená a vyčištěná od co největšího množství chyb. Dalším cílem je zbavit se nepodstatných či nadbytečných informací. Nejčastěji používané metody lze zahrnout do těchto kategorií:

2.3.1 Redukce šumu

Šum v digitálním obraze je produktem nedokonalosti senzorů a obvodů snímací techniky. Do obrazu snímané reality je při zpracování snímacím zařízením přimíchána nadbytečná informace – náhodný signál, který zhoršuje kvalitu snímaného obrazu. Míru šumu v obraze lze odvozovat od kvality samotných elektronických prvků, podmínek při kterých pracují (teplota, čas dne) či zvolené citlivosti snímacího zařízení.

Cílem odstranění šumu je nahrazení barvy pixelu průměrnou hodnotou okolních pixelů. Ze statistického pohledu je možné vypočítat, že při zprůměrování devíti pixelů standardní odchylka hodnoty šumu v obraze klesne na třetinu své hodnoty (4). Vedlejším efektem tohoto postupu je, že dojde i ke zprůměrování ostrých přechodů v obraze a tedy k celkovému poklesu ostrosti obrazu. Přesto pro detekci objektů v obraze i přes ztrátu ostrosti obrazu má snížení hodnoty šumu obecně větší vliv na kvalitu výsledku detekce a proto je použití odstranění šumu pro mnoho detekčních algoritmů doporučenou součástí.

Mezi nejčastější operace odstranění šumu patří:

- **Průměrování** je základním *lineárním filtrem*, který lze použít k odstranění šumu. Na obraz je aplikovaný jednoduchý *konvoluční filtr*, jehož výsledkem je průměrná hodnota okolních pixelů. Příklad filtru s maticí 3×3 ukazuje vzorec 1.

$$pixel_{avg} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{1}$$

- **Gaussův filtr** patří do skupiny *lineárních filtrů*, který pro definici *konvolučního filtru*, který je použit pro výpočet hodnoty každého pixelu jsou použity hodnoty Gaussovy *distribuční funkce*, jak definuje vzorec 2 (21). Proměnné x a y reprezentují vzdálenosti na osách (tedy šířku filtru) a σ standardní odchylku Gaussovského rozložení.

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

- **Medián** je *nelineární filtr*, který využívá výpočtu střední hodnoty okolí pixelu k nahrazení jeho původní hodnoty. Tento přístup je efektivní zvláště pro eliminaci šumu typu *pepř a sůl*. Příklad výpočtu median filtru ilustruje vzorec 3, kde matice ilustruje okolí vypočítávaného pixelu.

$$pixel_m = median \begin{pmatrix} 88 & 58 & 99 \\ 60 & 2 & 76 \\ 92 & 69 & 92 \end{pmatrix} \tag{3}$$

- **Bilaterální filtr** Patří mezi *nelineární filtry*, které zachovávají ostrost hran. Filtr je definovaný vzorcem 4, což je výsledkem práce týmu Smithse a Bradyho (22) a znovuobjevitelů Tomasiho a Manduchiho (25). Filtr nahrazuje každý pixel váženou hodnotou sousedících bodů. Váha přiřazená každému sousedu se snižuje jak na základě vzdálenosti v obraze (doména prostoru S) tak i na základě vzdálenosti na ose intenzity (doména rozsahu R). Hodnotu funkce tak lze uvažovat jako vzdálenost definovanou na $S \times R$. Parametry σ_S a σ_r udávají míru filtrace obrázku I. G_{σ_S} je gaussovský kernel snižující vliv vzdálených pixelů, G_{σ_r} je gaussovský kernel snižující intenzitu pixelů q rozdílnou od I_p (19).

$$\begin{aligned} BF[I]_p &= \frac{1}{W_p} \sum_{q \in S} G_{\sigma_S}(\|p - q\|) G_{\sigma_r}(\|I_p - I_q\|) I_q \\ \text{kde } W_p &= \sum_{q \in S} G_{\sigma_S}(\|p - q\|) G_{\sigma_r}(\|I_p - I_q\|) \end{aligned} \tag{4}$$

2.3.2 Úprava kontrastu

Mnoho obrazových dat je kvalitativně narušeno nevhodnou expozicí, zvláště v situacích s neoptimálními světelnými či atmosferickými podmínkami. Tento problém lze částečně opravit pomocí analýzy a následné korekce *histogramu*. *Histogram* reprezentuje rozložení jasu ve zpracovávaném obraze. Můžeme z něj tedy získat informace o dynamickém rozsahu obrazu a na základě toho určit, zda je obrázek v pořádku, nebo zda došlo k snížení dynamického rozsahu, *přeexpozici* či *podexpozici*. Mnohdy se jedná o obrázky, které jsou příliš světle, nebo příliš tmavé.

Problém chybné expozice je možné částečně opravit pomocí tzv. *vyrovnání histogramu* (*histogram equalization*). Skrze vyrovnání histogramu dojde k vylepšení kontrastu obrazu – metoda nalezne nejnižší či nejvyšší hodnoty jasu v obraze a ty, pokud se nerovnají maximu či minimu, posune na hodnotu maxima či minima. Ostatní hodnoty v obraze jsou přepočítány na základě tohoto posunu, který vizuálně „roztáhne“ celý graf. Nevýhodou může být potencionální zvýšení šumu v obraze, který tato operace může zvýraznit.

2.3.3 Změna rozlišení

Převzorkování vstupního obrázku do nižšího rozlišení může být v některých případech výrazným faktorem pro zrychlení průběhu navazujících algoritmů. Záznamová zařízení disponují stále větším počtem megapixelů, které poskytují detailní zachycení reality, ale jejich zpracování může být časově extrémně náročné. V algoritmech detekce objektů skrze počítačové učení je mnohdy vhodné používat vzorky, které mají řádově pouze stovky pixelů, což je pro dosažení výsledku dostačující.

Další uplatnění změny rozlišení může být v případě výskytu detekovaného objektu pouze ve vybrané části obrazu. Je nutné zvážit úhel snímání a umístění kamery. V případě detekce značek lze obraz ořezat o části, ve kterých je výskyt značek nepravděpodobný, čímž dojde ke zrychlení samotného procesu detekce, protože objem zpracovávaných dat se sníží.

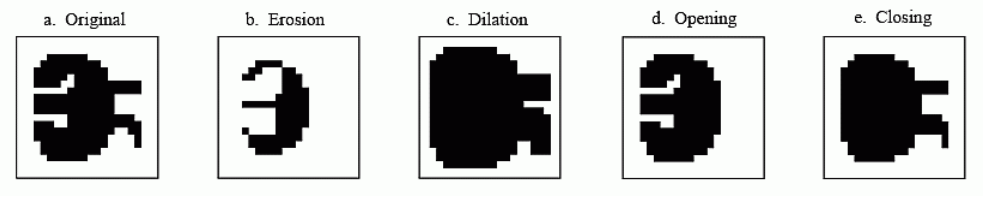
2.3.4 Morphologické operace

Morpohologické operace jsou kernelové operace, které slouží k transformacím binárních obrazů. V případě detekce objektů je možné využít tyto operace pro zlepšení výsledků segmentace vyprahovaného obrazu. Účelem těchto funkcí je snížení šumu, zvýšení odstupu či sloučení elementů v obraze a snížení členitosti okrajů elementů.

Mezi základní operace, jejichž vizuální reprezentaci ilustruje obrázek 7, patří:

- **Eroze** (*erosion*) je operace při které kernel spolu se stanoveným kotevním bodem (většinou střed kernelu) posouván skrze obrázek. Pro každý pixel je spočítána minimální hodnota v oblasti překryvu kernelu a pixel kotevního bodu je ve výsledném obrázku nahrazen touto hodnotou.

- **Dilatace** (*dilatation*) je opačná operace k erozi při které kernel spolu se stanoveným kotevním bodem (většinou střed kernelu) posouván skrze obrázek. Pro každý pixel je spočítána maximální hodnota v oblasti překryvu kernelu a pixel kotevního bodu je ve výsledném obrázku nahrazen touto hodnotou.
- **Otevírání** (*opening*) je operací při které je použita eroze a následně po ní dilatace. Tento postup se používá v binárním obraze k odstranění drobného šumu.
- **Zavírání** (*closing*) je operací při které je použita dilatace a po ní eroze. Tento postup se používá v binárním obraze k odstranění děr v souvislých plochách.



Obrázek 7: Příklad morphologických operací. Zdroj: Steven W. Smith (23)

2.4 Metody založené na segmentaci

Segmentace obrazu zahrnuje metody, které mají za cíl rozčlenit obraz na oblastmi které se vyznačují některou ze společných vlastností. Tím je možné obraz snáze analyzovat a využít k detekci objektů, rozpoznávání objektů či medicínské vizualizaci.

2.4.1 Prahování

Prahování patří mezi základní segmentační metody. V běžném případě se provádí prahování na obrázku, který je převedený do stupňů šedé. Pro zpracování algoritmu je nutné stanovit hranici H_t, která udává hodnotu v rozsahu od 0 do 255. Hranice udává zda bude novému pixelu P_t přiřazena hodnota černé barvy či barvy bílé na základě hodnoty původního pixelu p, jak je zobecněno ve vzorci 5. Výsledek procesu prahování ilustruje obrázek 8.

$$P_t = \begin{cases} 255 & : \text{pro } p \geq H_t \\ 0 & : \text{pro } p < H_t \end{cases} \tag{5}$$

Otsuho metoda

Otsuho metoda prahování umožňuje automatické stanovení hranice H_t pro obrázky z rozdílně exponované obrázky. Stanovení fixního prahu může vést při zpracovávání příliš tmavých případně světlých obrázků k získání černé či bílé plochy bez

Obrázek 8: Ukázka prahování. Zdroj: Karel Zídek

analyzovatelných oblastí. Otsuho metoda řeší tento problém skrze procházení všech možných hodnot prahu H_t a hledáním takové hodnoty, kdy je rozdíl mezi třídami černých a bílých pixelů minimální (18). Získáváme tak výstup, který obsahuje analyzovatelné oblasti i při špatně exponovaném vstupním obrázku.

Prahování založené na barvě

Prahování založené na barvě může být poměrně silným segmentačním nástrojem. Zvláště v případě specificky barevných objektů je možné rychle vymezit pravděpodobnou oblast výskytu objektu a značně tak omezit objem prohledávaných dat a získat tak ROI (*regions of interest*). Před započetím prahování je převést vstupní obrázek z běžného RGB prostoru do prostoru HSL či HSV. Tyto barevné prostory nabízejí efektivnější možnost filtrování na úrovni barvy, protože hodnota barvy je zaznamenána jako samostatný kanál. To vede k přesnější detekci barvy, menšímu zašumění.

Obdobně jako u klasického prahování vytváříme binární výstupní obrázek, kde parametrem jsou hodnoty prahů H_{lower} a H_{upper}, které udávají hodnotu *odstínu* (*hue*). Vlastnosti odstínu jsou definovány v CIECAM02 modelu publikovaném komisí CIE (15). Hodnota odstínu vstupního bodu p je porovnána s hodnotami prahů (vzorec 6). Pokud hodnota spadá do vymezené hranice získává výstupní bod P_t bílou barvu, pokud ne je nastaven jako černý.

$$P_t = \begin{cases} 255 & : \text{pro } H_{lower} \leq p \leq H_{upper} \\ 0 & : \text{jinak} \end{cases} \tag{6}$$

Je zřejmé, že tato metoda je funkční pouze v případě, že barevnost objektu v obrázku spadá do vymezených hranic. V případně barevně zkreslených vstupů vlivem špatné vyvážení bíle barvy, nevhodné barevná teploty, posunutím expozice či zkreslení barevnosti vlivem počasí nebo denní doby, může tento přístup selhávat. Jako řešení je možné použít několik sad nastavení pro různé světelné podmínky (13).

Porovnání barevných a bezbarvých metod bylo provedeno týmem Igora Bonačiho (3) s výsledkem, že barevná segmentace poskytuje obecně lepší výsledky.

Přesto šance na úspěch závisí na vzdálenosti značky od pozorovatele. V případě vzdálených značek byla metoda segmentace méně přesná, protože metoda očekává alespoň nějaký objem barevných informací, který je u vzdálené značky nedostačující.

2.4.2 Regionální metody

Regionové metody mají za cíl segmentovat skupiny pixelů, které mají stanovené společné vlastnosti. Princip vychází z metody semínkového vyplňování. Na počátku jsou stanoveny *semínkové pixely* (*seed pixels*). Následně dochází z těchto počátků k rozrůstání vybraných segmentů dle stanovených vlastností (např. jas). Tím vzniká segmentovaná oblast. Metoda je citlivá na výběr výchozích bodů a stanovení podmínek při kterých se oblast rozrůstá.

2.4.3 Hranové metody

Detekce hran je jádrovým (kernelovým) algoritmem, který z analyzovaného obrazu získává pozice hran, tedy míst kde dochází k prudkým přechodům mezi hodnotami jasu. Detekci hran je tedy možné provést skrze analýzu derivace hranové funkce, která bude nabývat vyšších hodnot právě v místech změn jasu.

Sobelův filtr

Nejčastějším způsobem je detekce provedena skrze *konvoluční jádro*. Správně zvolené konvoluční jádro umožňuje získat výstupní obraz ve kterém jsou zvýrazněny pouze hrany. Nejznámější variantou pro detekci hran je tzv. Sobelův operátor (vzorec 7).

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \mathbf{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{7}$$

Cannyho detektor hran

Cannyho detektor je postup navržený v roce 1986 Johnem F. Cannym, který je v současné době považovaný za optimální detektor pro detekci hran. Cannyho detektor má za cíl nízkou míru chybovosti, korektní a přesné nalezení hrany a minimální odezvu – tedy neznásobování hran (17). Cannyho detektor používá tyto kroky k nalezení hran:

- **Filtrace šumu**, která je provedena pomocí Gaussova filtru s volitelným rozměrem jádra. Princip Gaussova filtru je podrobněji popsán v kapitole 2.3.1.
- **Nalezení gradientu intenzity (*intensity gradient*)** v obraze. Tato část je prakticky realizovaná pomocí Sobelova filtru. Jako první je aplikovaná konvoluční maska s jádry uvedenými ve vzorci 7. Síla gradientu a směr je poté

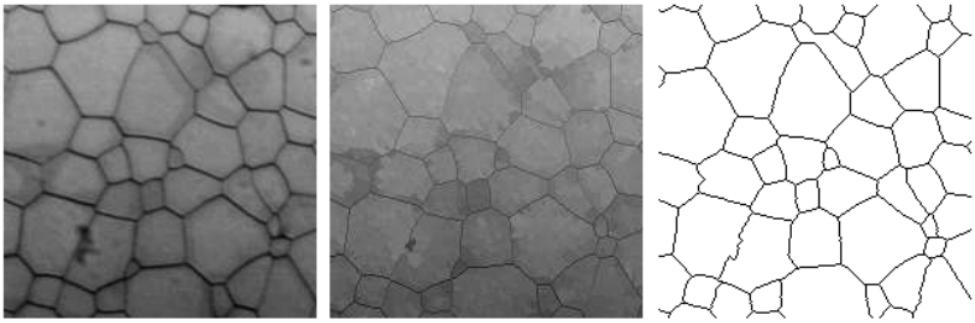
vypočítán za pomoci vzorce 8. Směr gradientu θ je zaokrouhlen na jednu z hodnot 0, 45, 90 nebo 135 (17).

$$\begin{aligned} S &= \sqrt{S_x^2 + S_y^2} \\ \theta &= \arctan(\frac{S_y}{S_x}) \end{aligned} \tag{8}$$

- **Potlačení ne-maxim** vede k odstranění pixelů, které nejsou součástí hran. Tím získáme pouze tenké linie (17).
- **Hystereze** ke které je použito hodnot dvou prahů *upper* a *lower*. Doporučený poměr prahů je 2:1, případně 3:1 (17).
  1. Pokud je hodnota pixelu větší než práh *upper* pixel je prohlášen za součást hrany.
  2. Pokud je hodnota pixelu nižší, než práh *lower* pak je pixel zahozen.
  3. Pokud je hodnota pixelu mezi prahy, pak je přijata pouze pokud je spojena s pixelem, který má hodnotu vyšší než *upper*.

2.4.4 Segmentace rozvodím

Algoritmus *segmentace rozvodím* (*watershed segmentation*) byl poprvé navržen v roce 1979 a je kombinací regionové a hranové metody. Principem postupu bylo umístění „zdroje vody“ do lokálních minim reliéfu a zaplavení celého obrazu. Při zaplavování dochází k tvorbě bariér, pokud se potkají různé zdroje (2). Tento postup byl později rozšířen o myšlenku *topologické segmentace* v roce 1997. Tato segmentace přistupuje k obrazu jako k topologické mapě a snaží se lépe zachovávat kontrast mezi lokálními minimy původního obrazu (5) což ilustruje obrázek 9.



Obrázek 9: Topologická segmentace rozvodím Zdroj: M. Couprie, G. Bertrand (5)

2.5 Metody založené na vyhledávání vlastnosti

Při detekci objektů v obraze je možné jako kritérium stanovit některou z jeho *vlastností* (*feature*). Co lze považovat za *vlastnost* či nikoliv se mezi autory liší. *Vlastnost*

je tedy třeba posuzovat dle použité metody. Obecně lze za *vlastnost* považovat něco, co je z pohledu zvoleného postupu dostatečně významné pro schopnost rozlišování a opakovatelné při dalších pokusech. Mezi *vlastnosti* lze zařadit např. tvar, nebo gradient.

2.5.1 Využití detektorů tvaru

Metody zaměřené na vyhledávání tvaru se snaží zhodnotit vlastnosti nalezených objektů. V případě detekce značky je třeba nejdříve informaci o tvaru z obrazu získat, poté se pokusit nalezené hodnoty porovnat s očekávaným tvarem. Při tomto procesu je třeba počítat s výskytem deformací a nerovností, takže např. značka z části zakrytá listem může ve výsledku nabízet po nalezení zcela jiný tvar, než je očekáváno.

Detekce tvaru v obraze vychází ve své podstatě z vhodné kombinace morphologických operací (viz sekce 2.3.4) a segmentačních metod (viz sekce 2.4). Obraz je upraven do podoby, kdy je v obraze možné detekovat tvary (prahování, detekce hran) a dle potřeby vyčištěn či opraven skrze morphologické metody.

Obrázek 10: Aplikovaný Sobelův filtr při Cannyho detekci hran Zdroj: Karel Zídek

Cannyho detektor hran

Jeden z běžných postupů, který zle zařadit do této kategorie je Cannyho detektor hran (viz sekce 2.4.3). Použitím tohoto algoritmu dostaneme výstup, který zobrazuje hrany nalezené v obraze a v případě implementace v rámci knihovny OpenCV i morphologický strom tvarů, který lze dále analyzovat. Vizualizaci hran získaných Sobelovým filtrem v rámci Cannyho detekce ilustruje obrázek 10.

Houghova transformace

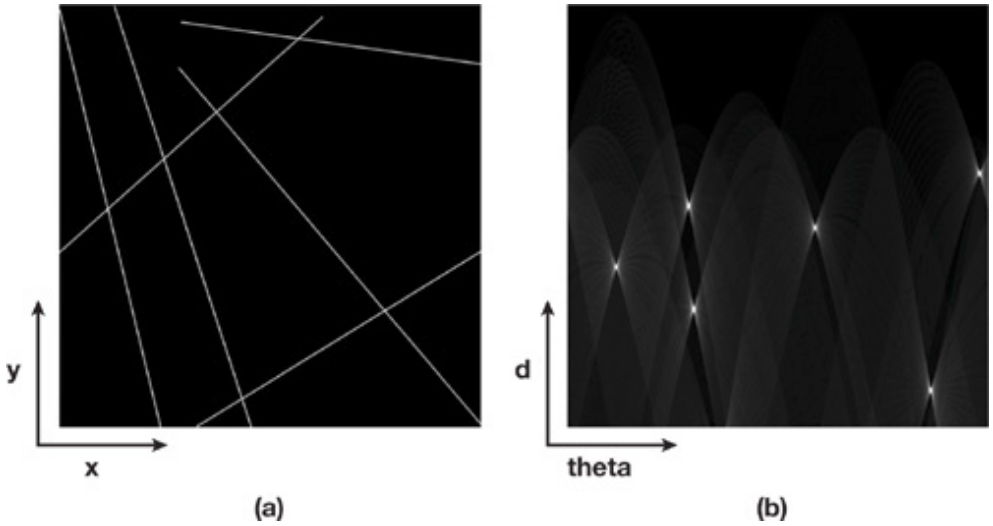
Pro identifikaci tvarů je možné následně použít např. *Houghovu transformaci*. Tato metoda, původně určená k detekci přímek v obraze, dnes umožňuje detekovat běžné geometrické tvary (kruhy, elipsy). Problémem těchto metod je jejich invariantnost

na perspektivní zkreslení a velký počet nalezených čar může mít podstatný vliv na výkon.

Pro provedení Houghovy transforamce je nutné na vstupním obraze provést vyhledávání hran. Pro tento účel je běžně používán Sobelův filtr (viz sekce 2.4.3). Postup lze odvodit z parametrického zápisu přímky uvedeném ve vzorci 9 (1), kde r udává vzdálenost normály na přímku od počátku, θ udává úhel, který tato normála svírá s osou x. Pokud máme již existující přímku, pak pro každé (x, y) ležící na přímce jsou hodnoty r a θ konstantní (9).

$$x \cos\theta + y \sin\theta = r \tag{9}$$

Pokud budeme na obraz obsahující křivku nahlížet z pohledu *Houghova prostoru* tedy prostoru (r, θ), pak lze pro všechny body ležící na křivce pohlížet jako na sinusoidní křivku. Pro body ležící v prostoru (x, y) na přímce se jejich sinusoidní křivky protnou v kolineárním bodě, což ilustruje obrázek 11. Skrze nalezení těchto bodů a zhodnocením jejich intenzity zjistíme přítomnost křivek v obraze (20).



Obrázek 11: Reprezentace přímek v Houghově prostoru Zdroj: Franck Diard (7)

2.5.2 Využití detektorů gradientů

Gradient je možné chápat, jako vektor který, vyjadřuje směr, případně růst či velikost změny. Tuto vlastnost je možné používat pro detekci objektů v obraze, protože je gradient, jako veličina, je invariantní vzhledem ke zvolenému souřadného systému.

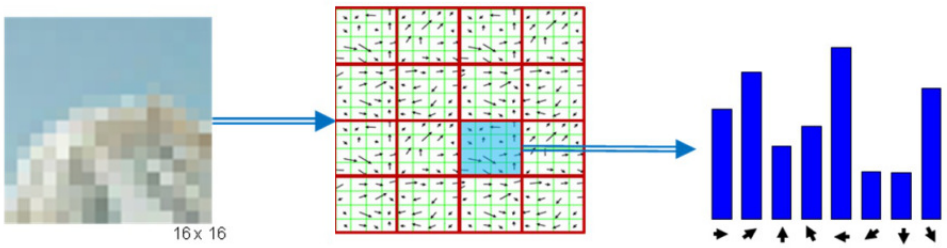
Histogram orientovaných gradientů

Za další z vlastnost při detekci objektů je možné považovat tzv. *Histogram orientovaných gradientů* (*Histogram of Oriented Gradients – HOG*). Tuto vlastnost poprvé definovali v roce 2005 Navneet Dalal a Bill Triggs (6).

Metoda je založena na výpočtu výskytu orientovaných gradientů v částech obrazu. Obraz je rozdělen na malé čtvercové regiony (buňky) – typická velikost buňky je 8×8 pixelů. V každé buňce je pro každý pixel vypočítaný gradient. Pro výpočet gradientu se používají kernely (vzorec 10) ve vertikálním a horizontálním směru (6).

$$[-1, 0, 1] \text{ a } [-1, 0, 1]^T. \tag{10}$$

Z takto vypočítaných gradientů je sestaven tzv. *histogram orientovaných gradientů*. Histogram buňky je normalizován na základě svého okolí. Tento postup je podobný postupu, který využívá algoritmus *SIFT* a je invariantní na změny jasu a otočení, protože histogram neobsahuje žádné geometrické informace (26). Rozdělení na buňky a postup výpočtu *histogramu orientovaných gradientů* ilustruje obrázek 12.



Obrázek 12: Výpočet Histogramu orientovaných gradientů. Zdroj: Gil Levi (11)

Histogramy pro celé buňky jsou sestaveny do *popisného vektoru* (*feature vector*), který je možné následně porovnávat a identifikovat tak podobné či totožné objekty.

Obrázek 13: Vizualizace výpočtu HOG deskriptoru. Zdroj: Karel Zídek, Data: Geodis, Vizualizace: HOGgles (29)

Velikost *popisného vektoru* je dána velikostí vstupního obrazu a nastavením rozdělení do buněk. V ilustrovaném příkladu (obrázek 12) je vstupem obrázek o velikosti 16×16 pixelů. Ten je rozdělen do buněk po 4×4 pixelů. Pro každou buňku ve vypočítán histogram gradientů, který tvoří 8 hodnot. Pro popis zvoleného obrázku je tedy vytvořen popisný vektor o velikosti 128 (16×8) hodnot (11).

Ukázka vizualizace *HOG deskriptoru* je ilustrovaná na obrázku 13. První část obrázku ukazuje vstupní obraz – značku. Značka je ve druhé části zobrazena upravená do podoby vhodné ke zpracování v počítači. Ve třetím části je vizualizován spočítaný *HOG deskriptor*.

2.6 Metody založené na počítačovém učení

Metody počítačového jsou pro oblast počítačového vidění velmi významné. Návrh algoritmu, který na základě stanovených parametrů dokáže analyzovat a rozhodovat se dle dodaných dat je klíčový v mnoha vědeckých oblastech. Již systémy, které pouze mechanicky reagují na nastalou situaci (např. automobilový systém ABS), se ukázaly jako velmi usnadňující lidský život.

Není divu, že dnes již existuje několik společností, které se snaží tuto původní myšlenky co nejdále a vznikají tak prototypy vozů, které nepotřebují řidiče a dokáží se v provozu pohybovat sami. Zde již není možné použít jednoduchý mechanický systém, ale je třeba „naučit“ vůz rozpoznávat svoje okolí a naučit jej reagovat na nastalé situace. Množina scénářů již není konečná (jako pro ABS – prokluzuje či neprokluzuje kolo), ale naopak nekonečná. Situace, provoz i podmínky se mění a tvůrce systému musí zaručit, že se vůz rozhodne v každé situaci správně.

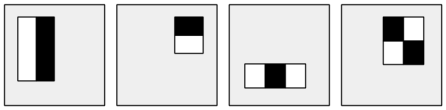
Mezi používané metody, které jsou vhodné k využití při rozpoznávání značek patří neuronová síť využívající Viola-Jonesových deskriptorů.

2.6.1 Viola-Jones

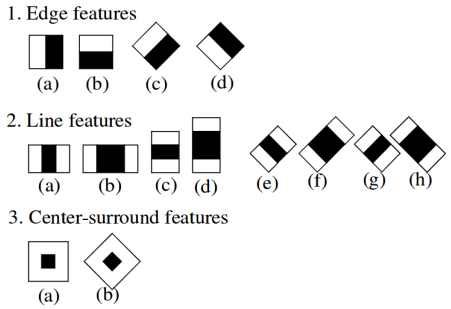
Algoritmus Viola-Jones byl navržený autory Paulem Violou a Michaelem J. Jonesem, kteří se snažili vyřešit problém rozpoznávání obličejů. To je také oblast, kde se algoritmus úspěšně používá. Přesto jeho implementace není s obličeji pevně svázána, ale lze jej využít k vyhledávání dalších objektů. Algoritmus má za cíl detekci předdefinovaného objektu, ne jeho rozpoznání. Je zřejmé, že cílem implementace byla robustnost, tedy získat co největší počet objektů korektně detekovaných. Těchto výsledků je možné dosahovat i v reálném čase.

Pro detekci objektů algoritmus používá tzv. *Haar features*, neboli *Haarových příznaků*. *Haaruv příznak* je plošný detektor charakteristických rysů daného objektu (14). Jedná se o využití vlastností tzv. *Haarových waveletů* (*Haarova vlnka* – funkce používaná k vlnkové transformaci). Hodnota příznaku, např. definovaném, jako dvě obdélníkové oblasti, je spočítána jako rozdíl součtu hodnot jasu mezi těmito regiony. V obrázku o rozměrech 24×24 pixelů je možné definovat sadu až 160 000 různých filtrů (28).

Původní sada Haarových příznaků, které definoval Viola a Jones, zobrazená na obrázku 14, byla později rozšířena Linehartem a Maydtem na sadu příznaků, které jsou pootočené o 45° a mají a cíl rozšířit sadu použitelných vlastností a zlepšit detekci některých typů objektů. Vzhled pootočených příznaků ilustruje obrázek 15.



Obrázek 14: Vizuální reprezentace Haarových příznaků Zdroj: Karel Zídek



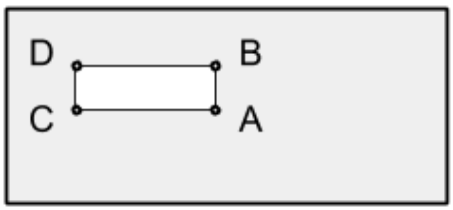
Obrázek 15: Vizuální reprezentace pootočených Haarovy příznaků Zdroj: R. Lienhart and J. Maydt (12)

Haarovy příznaky je možné velmi rychle a jednoduše spočítat v obraze pomocí tzv. *integrálního obrázku*. Definici *integrálního obrázku* lze vyjádřit pomocí vzorce 11, kde $ii(x, y)$ je integrální obrázek a $i(x', y')$ je původní obrázek (28).

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{11}$$

Pozice $ii(x, y)$ v obraze tedy reprezentuje součet hodnoty všech pixelů, které leží směrem vlevo a nahoře. Pro výpočet jasu libovolné části obrazu jsou potřebné pouze tři matematické operace, jak ukazuje obrázek 16. Suma S_Σ jednotlivých pozic v integrálním obraze A, B, C, D je dána vzorcem 12.

$$S_\Sigma = A - B - C + D \tag{12}$$



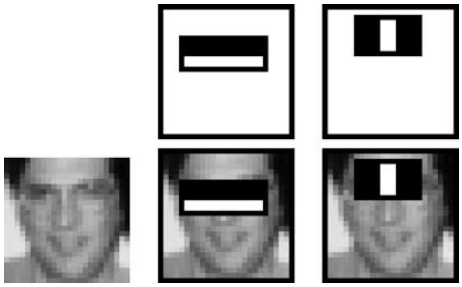
Obrázek 16: Součet v integrálním obrázku Zdroj: Karel Zídek

Mezi *Haarovy příznaky* jsou nejpoužívanější dva typy. Tyto příznaky je možné používat samostatně, případně jako jejich kombinaci:

- *Local binary patterns* (LBP), které se využívají zvláště na klasifikace textur v obraze.

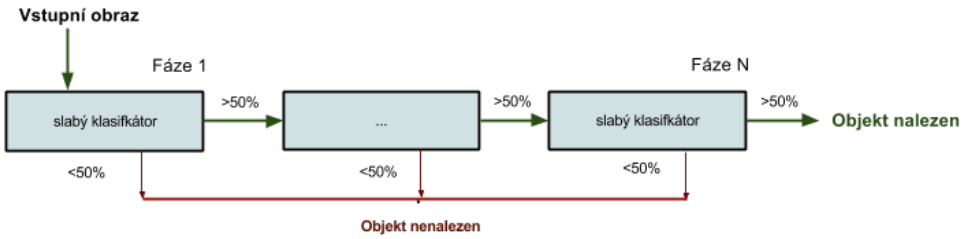
- *Histogram of oriented gradients* (HOG), které využívá techniky *hranově orientovaných gradientů*.

Viola (28) předpokládá, že pouze zlomek všech možných *Haarových příznaků* je možné zkombinovat do efektivního klasifikátoru. Z toho důvodu je pro nalezení těchto příznaků použita varianta učícího se algoritmu *AdaBoost*, která vybírá příznaky a zároveň učí klasifikátor. Obrázek 17 ukazuje příklad dvou nalezených příznaků zvolených metodou *AdaBoost*.



Obrázek 17: Nalezené Harrovy příznaky. Zdroj: Paul Viola a Michael J. Jones (28)

Metoda *AdaBoost* využívá pro úspěšné zhodnocení více *Haarových příznaků* tzv. *kaskádový klasifikátor*. Jedná se o sadu *slabých klasifikátorů* (tj. takové, u kterých neočekáváme nejlepší výsledek a je postačující pokud rozliší objekt ve více než 50% případů). Sada *slabých klasifikátorů* seřazená v sekvenci je tzv. *boostovaná* a může tak ve výsledku poskytnout velkou klasifikační schopnost, získáváme tak *silný klasifikátor* (*perceptron*) (28). Jednoduché schéma takovéhoto klasifikátoru ilustruje obrázek 18. Vstupem do klasifikátoru jsou hodnoty jednotlivých *Haarových příznaků*. Výstupem je třída reprezentující obsahuje hledaný objekt, nebo třída reprezentující nepřítomnost objektu.



Obrázek 18: Kaskáda slabých klasifikátorů. Zdroj: Karel Zídek

Slabý klasifikátor $h(x, f, p, \theta)$, který obsahuje *Haarovu vlastnost* f, práh θ a polaritu p udávající směr nerovnosti je definovaný vzorcem 13 (14).

$$h(x, f, p, \theta) = \begin{cases} 1 & : \text{pro } pf(x) < p\theta \\ 0 & : \text{jinak} \end{cases} \tag{13}$$

Sloučením slabých klasifikátorů získáváme klasifikátor silný. Každý slabý klasifikátor má při procesu trénování stanovenu svoji váhu α_t, kterou se podílí na celkovém výsledku. Pokud budeme uvažovat, že P udává práh silného klasifikátoru vypočtený při trénování a T je celkový počet slabých klasifikátorů, pak můžeme nadefinovat silný klasifikátor C pomocí vzorce 14 (14).

$$C = sign(\sum_{t=1}^{T} \alpha_t \cdot h(x, f, p, \theta) - P) \tag{14}$$

Pro korektní natrénování je potřeba trénovací databáze, která ve vhodném případě obsahuje tisíce obrázků daného objektu a obdobný počet obrázků negativních, tedy takových, který daný objekt neobsahují. *AdaBoost* v průběhu trénování stanoví jak váhy pro jednotlivé slabé klasifikátory θ, tak i celkovou váhu P.

2.6.2 Algoritmy podpůrných vektorů

Algoritmy podpůrných vektorů (*Support vector machines*) je metoda strojového učení vynalezená V. Vapnikem v roce 1970. Je zařazená do skupiny tzv. *jádrových algoritmů* (*kernel machines*). Jaderné funkce slouží k tzv. *nelineární separaci*. Cílem těchto metod je snaha využít efektivity algoritmů pro nalezení *lineární hranice* (*hyperplane*) mezi dvěma třídami (pozitivní a negativní). V této podobě jsou SVM schopné reprezentovat složité nelineární funkce (35). V rámci řešení je tak možné upravit data, které běžně nelze oddělit lineární hranicí, přidáním dalšího rozměru převést do podoby, kdy lze lineární řešení nalézt. V praxi se tyto metody uplatňují zvláště pro klasifikaci textu.

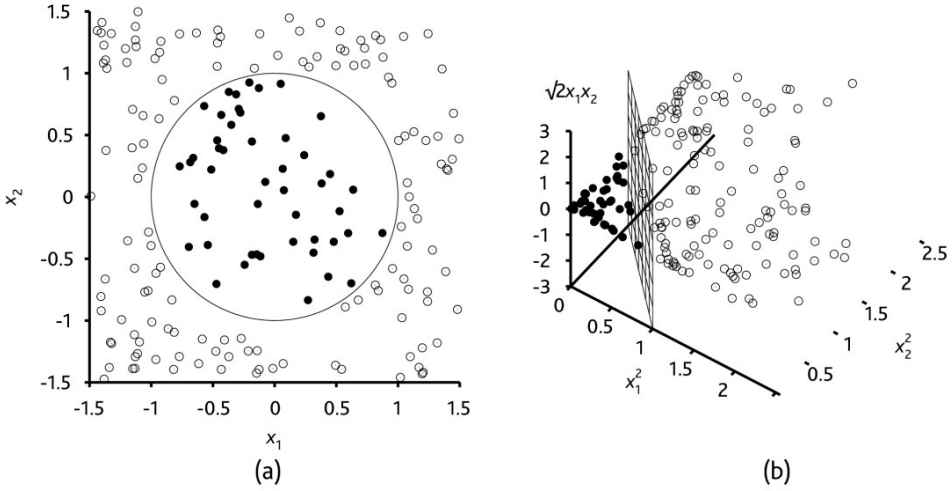
Pokud posuzujeme data pouze dvourozměrně, jak definuje Verplancke (27) a ilustruje na první části (a) obrázku 19, pak zjistíme, že lineární oddělení je v ilustrovaném případě nemožné, neboť hranice tříd je tvořena funkcí $x_1^2 + x_2^2 \leq 1$. Pokud přidáme původním dvěma atributům (x_1, x_2) třetí založený na prvních dvou získáme lineárně separovatelná data. Atributy můžeme nadefinovat funkcemi $f_1 = x_1^2$, $f_2 = x_2^2$, $f_3 = \sqrt{2}x_1x_2$. V případě ilustrovaného příkladu pak můžeme v prostoru $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$ třídy rozlišit pomocí roviny, jak ilustruje druhá část (b) obrázku 19.

Skrze mapovaní do prostoru bylo možné nalézt řešení pomocí lineárního oddělení, tedy *nadrovinu* (*hyperplane*).

Pro trénování SVN, jak uvádí Mrázová (16), se definuje vstupní vektor $((x_1, y_1), (x_2, y_2), \ldots (x_m, y_m))$, kde hodnoty $x_i = (x_1 \ldots x_n)$ udávají vstupní data v podobě vektoru a hodnota $y_n \in \{-1, 1\}$ udává označení pozitivní či negativní třídy, do které zvolený vektor spadá. SVM poté hledá váhovou funkci:

$$f(x) = \langle w \cdot x_i \rangle + b, \tag{15}$$

kde w udává váhový vektor. Pak lze stanovit, že:



Obrázek 19: Data v dvojrozměrném a trojrozměrném prostoru. Zdroj: Verplancke (27)

$$y_i = \begin{cases} 1 & : \text{pokud } \langle w \cdot x_i \rangle + b \geq 0 \\ -1 & : \text{pokud } \langle w \cdot x_i \rangle + b < 0 \end{cases} \tag{16}$$

a tedy nadrovina, která separuje pozitivní a negativní trénovací data má tvar:

$$\langle w \cdot x_i \rangle + b = 0 \tag{17}$$

Těchto nadrovin je v oblasti rozhodovací hranice (plochy) existuje nekonečně mnoho a je tedy třeba stanovit jeden nejvhodnější. SVM vyhledá nadrovinu, která má poskytuje největší okraj a tedy usiluje o minimalizaci chybovosti (16).

3 Analýza a metodika

Pro návrh vlastní architektury pro detekci značek bude třeba porovnat v první fázi používané algoritmy pro detekci objektů. Po prostudování dostupných algoritmů bude třeba navrhnou vlastní experimentální řešení, které vhodnou kombinaci zvolených algoritmů využívá.

Následně budou porovnány knihovny, které umožňují využíti GPGPU výpočtů pro implementaci algoritmů využitelných k řešení problematiky detekce značek. V rámci výběru knihovny bude zhodnoceno využití možností frameworků pro přímou práci s GPU (Nvidia CUDA, OpenCL). Cílem zhodnocení bude nalezení řešení, které umožní implementovat detekci značek v efektivní podobě prostřednictvím knihovny která využívá obecných výpočtů na GPU a zhodnotit možnosti využití knihovny v rámci aplikace s ohledem na efektivitu a kvalitu řešení a zvláště s cílem zrychlení prováděných výpočtů.

Zvolená knihovna musí nejenom umožňovat implementaci zvolených algoritmů na GPU, ale umožňovat realizovat implementaci zvolených algoritmů v ekvivalentní podobě k CPU kódu, aby bylo možné provést měření a porovnání výsledků. Není tedy nutné, aby knihovna nutně umožňovala low-level programování na GPU, ale je postačující, aby umožňovala GPU efektivně využít ke GPGPU výpočtům (prostřednictvím Nvidia CUDA nebo OpenCL), které jsou využitelné pro porovnaní přínosu CPU a GPU implementace detekce značek.

Pro zvolené algoritmy je třeba připravit vstupní data (trénovací sadu) a zvolit jejich parametry a nastavení tak, aby aplikace byla schopna navržené detekce značek. Pro ověření funkčnosti navrženého řešení bude třeba hotovou aplikace otestovat skrze reálná data a změřit a porovnat úspěšnost implementace ve verzi počítané na CPU spolu s verzí počítanou na GPU. V poslední fázi bude zhodnocena efektivita CPU a GPU řešení, jeho přínos a diskutované potenciální místa pro další rozvoj a vylepšení aplikace.

3.1 Návrh experimentálního detektoru značek

V rámci literární rešerše bylo prozkoumáno široké spektrum algoritmů, které umožňují řešit problematiku detekce značek jak čistě skrze analýzu obrazu (segmentační metody, vyhledávání tvaru), tak čistě metodami počítačového učení. Každá z kategorií má své přednosti a slabá místa.

Algoritmy segmentace a vyhledávání tvaru

Algoritmy z této kategorie je obecně možné jednoduše implementovat. V obecném případě je cílem vhodně aplikovat segmentační filtry tak, aby z obrazu byly odstraněny všechny nežádoucí elementy a detekované oblasti byly hledanými objekty. Tyto postupy jsou obecně náchylné na kvalitu zaznamenaných dat. Např. při vhodných podmínkách je barevná segmentace velmi úspěšnou metodou, ale při změně

počasí je třeba potýkat se se změnou barevného spektra. Navržení robustních řešení je i přesto možné (např. práce Hasana Fleyeha (10)).

Výhodou těchto metod je, že nevyžadují předchozí přípravu dat či trénování, výzvou je ovšem zvolení vhodných parametrů zvolených filtrů. Oblast implementace většiny filtrů, kterou lze zlepšit v rámci GPU akcelerace, spočívá v maticových operacích a aplikaci různých druhů konvolučních filtrů. Přínos takového řešení tak lze rámcově odvodit z implementace provedené v rámci článku *Image Processing Methods Optimization by Means of GPU Computing* (34). Pro řešenou problematiku se vlastnosti těchto algoritmů hodí pro rámcové zpřesnění identifikace značky.

Algoritmy počítačového učení

Algoritmy počítačového učení nabízejí postupy založené na identifikaci podle naučených vzorků. Výhodou je, že dobře naučený algoritmus dokáže být spolehlivý i za různých podmínek a je dostatečně robustní. Nevýhodou, že kvalita výstupu je závislá na kvalitě trénovacích dat. Trénování podle typu algoritmu může trvat od několika minut až po hodiny případně dny, přičemž doba trénování nutně nezaručuje lepší výsledky algoritmu.

Výhodou těchto algoritmů pro inventarizační systém, skrze který proudí neustále nová a nová data, může být schopnost učit se z nově dostupných vzorků a adaptovat se tak na nové situace, zvláště pokud se očekává revize korektních a nekorektních vzorků. Pro takový systému může tato vlastnost býti klíčovou. Díky této vlastnosti byla zvolena skupina těchto algoritmů jako více vhodná pro implementaci do experimentální aplikace.

3.1.1 Volba algoritmů a návrh řešení

Po provedeném zhodnocení byl zvolen jako řešený experimentální postup pro řešení aplikace kombinace algoritmů Viola-Jones a SVM s případným zpřesněním skrze algoritmy segmentace a vyhledávání tvaru. Řešení je uvažováno pro potřeby inventarizačního systému a proto po výstupu není nutně potřebné zpracování dat v reálném čase, což umožňuje využití vícestupňové detekce, která by mohla být pro systém pracující v reálném čase přítěží. Přesto je cílem, v rámci stanovených omezení, algoritmus (zvláště GPGPU akcelerovaná verze), který by poskytl co nejefektivnější řešení upravitelný, v ideálním případě, do podoby zvládající i detekci v reálném čase.

Dvoufázová architektura

Algoritmus Viola-Jones je v počítačovém učení považován za dostatečně robustní pro detekci objektů v rámci počítačového vidění. Jeho hlavní nevýhodou je nutnost předem připraveného soubor vstupních dat (který nemusí být vždy dostupný) a velká časová náročnost na trénování, případně přetrénování, detekční kaskády. Pro klasický přístup k identifikaci značek by byla nutnost provést natrénování algoritmu Viola-Jones pro každou značku kterých je nezanedbatelné množství.

V rámci navrženého řešení je tak identifikace pomocí Viola-Jones zobecněna na danou třídu značek. To by mělo usnadnit trénování, protože bude možné použít širší množství vzorků, které jsou v hlavních vlastnostech velmi podobné. Množství detekovaných tříd se tak zmenší na několik hlavních (trojúhelníkové, kruhové, atd.). Stejně tak počet výpočetních úkonů k nalezení oblasti s předpokládanou oblastí daného typu značky. V rámci robustnosti algoritmu je možné předpokládat, že tato fáze poskytne dostatečný počet korektně nalezeného typu značky s minimálním množstvím chybných detekcí.

V rámci druhé fáze byl zvolen algoritmus SVM, který bude sloužit pro určení konkrétního typu značky. Vzhledem k velmi pozitivním výsledkům, které tento algoritmus poskytuje např. při rozpoznávání znaků (viz práce Antonia Carlose Gaye Thomého (24)) je předpokládáno, že by identifikace značek mohla být touto cestou efektivně řešitelná. Výhodou je, že výsledné řešení není cíleno na systémy, které se potřebují z detekovaného značení rozhodovat. Je tedy možné v rámci algoritmu tolerovat větší množství chybných identifikací. V rámci inventarizačního systému se předpokládá, že tento proces pouze usnadňuje práci technikovy, který po provedení detekce provede finální potvrzení výsledků detekce. Potvrzené pozitivní a zamítnuté chybně pozitivní výsledky je tak možné recyklovat a využít k přetrénování celého SVM modulu (trénování je podstatně méně časově náročné, než u Viola-Jones) a tedy dosáhnou progresivního zlepšování hotového systému s množstvím analyzovaných dat.

Pro ověření zvoleného řešení byla v prvním případě zvolena značka „Dej přednost v jízdě“. Důvodem pro volbu této značky se stal fakt, že se v dostupných datech vyskytovala velmi často, čímž je možné ověřit schopnost postupu opakovaně tuto značku nalézt a identifikovat. Výhodou je, že je poměrně výrazná a nesplývá s pozadím. Nevýhodou je, že kategorie trojúhelníkových značek orientovaných směrem dolu obsahuje pouze jednu značku.

V druhém případě byla zvolena modrá čtvercová značka „Parkoviště“. Ve vlastních datech se vyskytuje nejčastěji. Vzhledem ke kvalitě dat je předpokládaná, jako nejhůře detekovatelnou značkou, protože je nejméně výrazná a mnohdy splývá s pozadím.

3.2 Srovnání knihoven pro práci s GPU

Mezi nejčačtěji používané frameworky pro programování na GPU a výpočty GPGPU jsou Nvidia CUDA a OpenCL. Jedná se o nástroje, které umožňují nízkoúrovňovou implementaci řešení, ale vyžadují komplexní znalost implementovaných algoritmů. Oproti nim stojí balík OpenCV, který nabízí komplexní sadu funkcí a optimalizovaných algoritmů používaných v počítačovém vidění, které byly a stále jsou rozšiřovány o jejich GPU implementace.

Zvláště pro knihovnu OpenCV byly hledány alternativy, ale dostupná řešení byla buď velmi základní a pro účely projektu zcela funkčně nedostatečná (knihovna

MinGPU[1]), případně slibné projekty, ale bez podpory GPU akcelerace (knihovna SimpleCV[2]).

3.2.1 Nvidia CUDA

Jedná se o framework, který vznikl ze spolupráce Jamese Funga (University of Toronto) s formou Nvidia. Zčlánků vydaných v roce 2002 postupně vznikl balík, který vedl k vydání balíku Nvidia CUDA v roce 2006. Jednalo se tedy o první dostupné řešení na trhu. Tento balík umožňuje pomocí téměř standardní verze jazyka C implementovat aplikace využívající GPGPU výpočtů na grafických kartách firmy Nvidia.

Mezi výhody tohoto balíku patří:

- Pokročilá a přehledná implementace. Množství snadných funkcí.
- Ucelené vývojové multiplatformní prostředí.
- Proprietární licence – freeware.

Mezi nevýhody tohoto balíku patří:

- Svázanost s grafickým hardwarem firmy Nvidia.
- Pouze framework – neobsahuje hotová využitelná řešení.

3.2.2 OpenCL

Je frameworkem, který byl, zvláště v posledních fázích vývoje, inspirován praktičností Nvidia CUDA. Jedná se tedy o obdobné řešení, které nabízí balík CUDA. OpenCL je vyvíjeno v rámci Khronos group, která sdružuje firmy, které pracují v dané oblasti a zaštituje další „Open-“ projekty. Řešení bylo uvedeno na trh o něco poději,než balík CUDA, v roce 2008.

Mezi výhody tohoto balíku patří:

- Dostupnost pro širokou škálu platforem – Nvidia, Amd, Intel…
- Ucelené vývojové multiplatformní prostředí.
- Licence – Royalty Free Open Standard

Mezi nevýhody tohoto balíku patří:

- Složitější a méně přehledná implementace
- Pouze framework – neobsahuje hotová využitelná řešení.

[1]MinGPU: A minimum GPU library for Computer Vision. Knihovna je dostupná na adrese: http://vision.eecs.ucf.edu/MinGPU/

[2]SimpleCV. Knihovna je dostupná na adrese http://simplecv.org/

3.2.3 Knihovna OpenCV

OpenCV (*Open Source Computer Vision*) je multiplatformní knihovna, která sdružuje funkce cílené na oblast počítačového vidění. Původně vyvinuta Ruskou pobočkou firmy Intel, je dnes vyvíjena výzkumnou laboratoří Willow Garage a společnosti Itseez pod open-source BSD licencí. Původní implementace dnes nabízí široké spektrum funkcí jak z oblastí zpracování obrazu, počítačového viděni, detekce objektů, počítačového vidění, clusterování včetně implementace vybraných funkcí s podporou GPU.

Balík OpenCV se pro účely aplikace hodí nejvíce. Nabízí podpůrné metody pro detekce, rozpoznávání objektů či gest a počítačového učení včetně popsaných a zvolených algoritmů. Knihovna implementuje většinu známých a používaných metod pro počítačové vidění.

Pro účely aplikace se hodí zvláště z těchto důvodů:

- Knihovna je rozšířená, vyvíjená a má širokou uživatelskou základnu.
- Implementuje široké spektrum algoritmů počítačového vidění.
- Implementace algoritmů nejsou naivní či experimentální, ale jedná se o optimalizované produkční verze.
- Podstatná část algoritmů je implementovaná v GPU verzi prostřednictvím frameworku Nvidia CUDA, měnší část skrze OpenCL.
- Knihovna poskytuje kvalitní dokumentaci a podporu.
- Knihovna je multiplatformní.

Za nevýhody lze považovat:

- Malé praktické možnosti ovlivnění implementace algoritmů.
- Problematická identifikace chyb v rámci knihoven.

Na základě těchto výhod a při uvažování možnosti převodu aplikace z experimentální podoby do podoby reálné, případně transformace do modulu již hotového systému, se jako vhodná volba pro použití v rámci experimentální aplikace jeví použití knihovny OpenCV. Dílčí implementace zvolených algoritmů pouze pomocí samotném frameworku dosáhnou stejného výkonu, jako implementace skupiny odborníků v rámci udržované knihovny, pouze ve velmi optimistickém případě. Stejně tak další rozvoj aplikace spolu s rozšířením funkcionality je skrze OpenCV daleko efektivnější, než v případě low-lewel implementace.

3.3 Funkcionalita řešení

Navržené řešení má za cíl nalézt a určit vybrané typy dopravních značek na sérii snímků, které byly pořízeny během reálného snímání ulic. Nalezené značky bu-

dou rozpoznány a identifikovány. Průběh detekce bude zobrazen skrze uživatelské rozhraní, které zobrazí jak průchod analyzovanými snímky, tak výsledné nalezené značky s jejich výřezy.

Vlastní experimentální implementace detekce bude probíhat ve dvou fázích. V první fázi bude určen obecný typ značky (trojúhelníková, kulatá). V druhé fázi bude určena konkrétní varianta značky z nadefinovaných typů, případně dojde k zamítnutí chybně nalezené značky z první fáze. Řešení je možné doplnit o třetí fázi, která využívá dalších detekčních postupů pro případné zpřesnění detekce značek.

Pro splnění navržené funkcionality je třeba provést:

1. Přípravu testovacích a trénovacích dat
2. Natrénovat z dostupných dat dílčí moduly pro detekci značek
3. Provést průchod sérií snímků a jejich analýzu
4. Detekce pozice kategorie značky pomocí algoritmu Viola-Jones
5. Určení typu či zamítnutí značky pomocí klasifikace SVN
6. Aplikace metod dále zpřesňující detekci
7. Zobrazení nalezených značek a statistických dat

4 Realizace aplikace

Aplikace byla realizována na základě provedené rešerše a s cílem praktického ověření návrhu zvolené funkcionality. Nejedná se tedy o hotový nástroj pro praktické využití, ale spíše o experimentální implementaci, jejímž cílem je potvrdit, či vyvrátit, funkčnost navrženého řešení. Přesto je v rámci implementace bráno v potaz potenciální praktické využití a vlastní návrh aplikace tak tento fakt zohledňuje.

4.1 Použité nástroje

Implementace aplikace byla provedena pomocí jazyka C++11 s překladačem `g++` verze 4.8.2 pod operačním systémem Ubuntu 14.04 LTS s nainstalovanými ovladači s podporou Nvidia CUDA. K GPGPU výpočtům byla použita grafická karta Nvidia Geforce GT 755M.

Pro umožnění programování na GPU bylo nutné nainstalovat balík Nvidia CUDA 6.5. Jako součást balíku Nvidia CUDA bylo nainstalováno i vývojové prostředí Nvidia Nsight založené na editoru Eclipse, které umožňuje spravovat nejenom projekty určené pro Nvidia CUDA, ale i klasické projekty psané v C++11 v kombinaci s knihovnou OpenCV.

Pro účely vlastní implementace byla použita poslední dostupná stabilní verze balíku OpenCV, konkrétně verze 2.4.11. Instalační balík OpenCV je v základních verzích, i pro operační systém Ubuntu, distribuovaný v kompatibilním sestavení, které má vypnutou podporu výpočtů na grafických kartách. Pro jejich zapnutí je třeba provést vlastní sestavení ze zdrojových kódu s modifikací parametrů sestavení a zapnutí podpory výpočtů na GPU.

Aplikaci bylo vytvořeno, za účelem efektivní prezentace výstupů, uživatelské rozhraní, které zobrazuje zpracovávaný obraz, detekované značky a statistické informace o době trvání detekce, jak pro CPU, tak i pro GPU verzi implementace současně.

4.2 Příprava testovacích a trénovacích dat

V rámci realizace aplikace bylo jednou z klíčových částí příprava trénovacích dat detekčních algoritmů. Jednalo se v první fázi o přípravu dat pro trénování detektoru využívajícího algoritmu Viola-Jones, po níž následovala příprava dat a trénování klasifikátoru SVM. Pro ověření funkcionality navržených postupů byly využity fotografie z reálného pozemního snímání.

Použitá data

Data, která byla pořízena jako součást pozemního snímání silnic byla poskytnuta firmou Geodis. Jedná se přibližně o 800 snímků pořízených ze snímacího vozidla o rozlišení 2448×2050 pixelů. Snímky byly pořízeny dvěma kamerami, první z nich

orientovaná po směru jízdy, druhá přibližně ve směru opačném. Zachycený úsek reprezentuje část ulice Úvoz a oblast Mendlova náměstí v Brně. Data byla pro využití v rámci aplikace zmenšena, pro zlepšení celkového výkonu, na poloviční rozlišení.

Snímky byly pořízeny během počínajícího lehkého deště – vyznačují se tedy zhoršenou kvalitou v oblasti světelných podmínek, zvýšením hladiny šumu a zkreslením obrazu vlivem dešťových kapek. Tyto negativní vlastnosti nasnímaných dat umožňují ověření funkčnosti řešení na datech, která se typicky mohou vyskytovat za běžného provozu.

4.2.1 Trénování algoritmu Viola-Jones

Algoritmus Viola-Jones pro vygenerování potřebuje dostatečně velký počet pozitivních a negativních vzorků. Negativní vzorky byly vygenerovány z vybraných obrázků ze série které by měly reprezentovat většinu vizuálních prvků a prostředí a velkou variabilitu.

Splitter tool

Pro získání co největšího množství vhodných negativních vzorků byl vytvořen jednoduchý nástroj pro rozřezání obrázků, které neobsahují dopravní značky, do vzorků o velikosti 100×100 pixelů. Tím vzniklo v základní variantě 5760 vzorků používaných jako negativní pozadí.

Generování pozitivních vzorků

Jelikož data z nasnímaných ulic neobsahovaly dostatečné množství značek jednotlivých typů bylo třeba uměle zvýšit počet vstupních vzorků. K tomuto účelu slouží nástroj knihovny OpenCV `opencv_createsamples` jehož použití je zobrazeno v kódu 1.

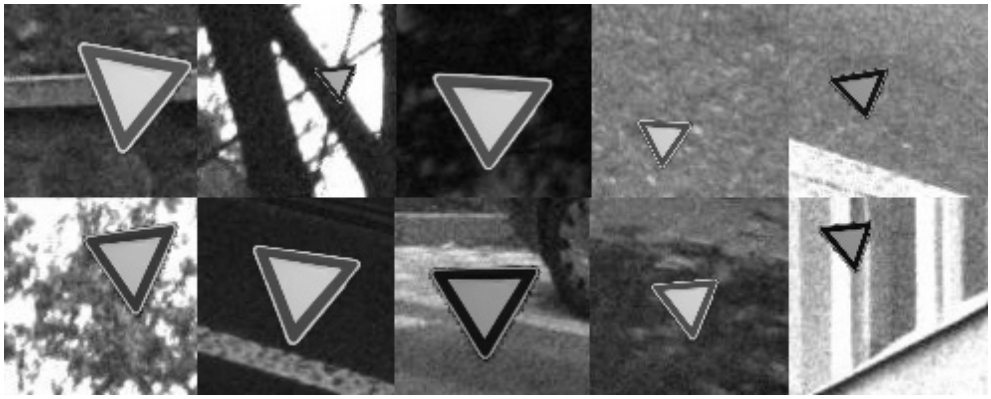
Kód 1: Generování pozitivních vzorků

```
opencv_createsamples -img positive/positive0.jpg -bg negative.txt
  -bgcolor 0 -bgthresh 0 -num 200
  -maxxangle 0.2 -maxyangle 0.3 -maxzangle 0.4  -w 48 -h 48
  -info positive_generated/0/annotations.lst
```

Vstupem se staly vybrané vzorky dopravní značky, u kterých bylo odstraněno pozadí. Každý jednotlivý pozitivní vzorek je předáván generátoru parametrem `-img` a cestou k souboru. Soubor obsahující seznam negativních vzorků je generátoru předáván pomocí parametru `-bg`. Skrze parametry `-bgcolor 0` a `-bgthresh 0` je nastaveno, že průhledné pozadí je ve vstupním obrázku definované černou barvou. Počet vygenerovaných výstupů je definovaný skrze parametr `-num`. V základní variantě bylo, pro každý z pěti pozitivních vzorků nastaven požadavek vygenerování 200

variant, čímž vzniklo celkem 1000 pozitivních vzorků Velikost výstupu byla v první fázi stanovena na základní hodnotu na 24×24 pixelů. Jelikož se tato velikost ukázala jako nevhodná, protože trénování bylo ukončeno předčasně po několika fázích, byla zvětšena velkost vzorku na dvojnásobek – 48×48 pixelů.

Generátor použije vstupní obrázek značky a vloží jej do negativních obrázků což ilustruje obrázek 20. V rámci vložení je pozitivnímu obrázku náhodně změněna velikost a je náhodně orotován dle parametrů `-maxxangle 0.2`, `-maxyangle 0.3` a `-maxzangle 0.4` ve zvolené ose o úhel jehož maximum (v radiánech) parametr deklaruje. Hodnoty byly zvoleny experimentálně a zohledňují předpokládané zkreslení značky v jednotlivých osách. Hodnoty úhlů byly stanovenu v rozsahu od cca 10° do cca 25°. K vygenerovaným výstupům je vytvořen popisný soubor (definovaný parametrm `-info` a cestou), ve kterém je ke každému obrázku uložena cesta a pozice pozitivního vzorku.



Obrázek 20: Ukázka generovaných pozitivních vzorků. Zdroj: Karel Zídek

Popisy pozitivních vzorků je nutné sloučit do jednoho souboru. Aby se při tréningu, který probíhá sekvenčně, předešlo generovaných vzorků pouze z jednoho pozitivního vzorku jsou záznamy v popisném souboru skriptem náhodně zpřeházeny. V dalším kroku je z těchto vzorků vygenerován soubor, který slouží jako vstup pro trénování algoritmu. Tvorba vstupního souboru je generovaná skrze stejný nástroj (`opencv_createsamples` – kód 2), jiné funkcionality je docíleno změnou parametrů nástroje.

Kód 2: Generování souboru s pozitivním vstupními vzorky

```
opencv_createsamples -info positives.txt -bg negative.txt
  -bgcolor 0 -bgthresh 0 -num 1000
  -maxxangle 0.2 -maxyangle 0.3 -maxzangle 0.4 -w 48 -h 48
  -vec positive.vec
```

Parametrem `-info` předává nástroji seznam pozitivních vzorků. Parametr `-vec` s parametrem souboru udává výstupní soubor s pozitivními vzorky . Parametr `-num` v tomto případě udává počet vzorků ve výstupním souboru.

Trénování Viola-Jones

Pro trénování algoritmu Viola-Jones je určen nástroj `opencv_traincascade`. Tento nástroj z předdefinovaných vzorků vytvoří XML soubor, který obsahuje kaskádu jednotlivých Haarových příznaků. Tento popis je následně používán v rámci aplikací pro detekci natrénovaného objektu.

Trénování probíhá po jednotlivých fázích (*stages*) kaskády a je ukončeno, pokud je dosažen nastavený limit počtu fází, nebo kaskáda dosáhne nastavené minimální úspěšnosti (*minimum hit rate*) – v základu nastavené na hodnotu 0.995. Podoba celého příkazu, který nastavuje všechny potřebné parametry pro trénovaní ukazuje kód 3.

Kód 3: Trénování modulu Viola-Jones

```
opencv_traincascade -data classifier
  -vec positive.vec -bg negative.txt
  -baseFormatSave
  -precalcValBufSize 2048 -precalcIdxBufSize 2048
  -numPos 500 -numNeg 2000
  -numStages 16
  -featureType HAAR
  -w 48 -h 48
```

Aplikaci je předána výstupní složka skrze parametr `-data`, vygenerovaný vektor s pozitivními daty skrze parametr `-vec`, jejich velikost (`-w 48 -h 48`) a negativní vzorky skrze parametr `-bg`. Parametr `-baseFormatSave` nastavuje generovaní zpětně kompatibilního formátu výstupu. Pro urychlení samotného zpracování bylo parametry `-precalcValBufSize 2048 -precalcIdxBufSize 2048` zvětšeno množství přidělené paměti pro předpočítané hodnoty na 2048Mb.

Počet pozitivních vzorků o rozměrech 48×48 pixelů (parametry `-w 48 -h 48`) byl nastaven na 500 a počet negativních na 2000 při maximálním počtu fází nastaveném na hodnotu 16 (parametry `-numPos 500 -numNeg 2000 -numStages 16`). Typ příznaků byl nastaven na Haarovy příznaky (parametr `-featureType HAAR`).

Trénování je v mnoha případech velmi časově náročné (v řádu až desítek hodin) a proto je vhodné zvážit hodnoty parametrů a počty vzorků pro proces trénování.

Problémy OpenCV

Při řešení skrze knihovnu OpenCV se z počátku ukázalo porovnání CPU a GPU verze Viola-Jonesova algoritmu problematické skrze chyby v balíku OpenCV. Balík OpenCV přešel, v předchozích verzích, u CPU verze Viola-Jonesova algoritmu na jiné formátování XML souboru s kaskádou Haarových příznaků. GPU implementace zůstala u původního formátování XML souboru. Trénování, i se stejnými daty, do různých formátů souboru by znehodnotilo porovnání výsledků.

Nástroj `opencv_traincascade` byl tedy programátory doplněn o parametr `-baseFormatSave`, který by měl umožňovat natrénovaná data uložit do původního formátu souboru kompatibilním s CPU i GPU implementací. Bohužel skrze chybu je XML soubor v kompatibilním režimu generován s chybou (chybějícím parametrem), který způsobuje odmítání souboru GPU verzí algoritmu. Tento problém bylo nutné identifikovat a následně nalézt řešení, které se sestává z automatizované úpravy vygenerovaného souboru po ukončení trénování. Upravený soubor je akceptovaný CPU i GPU verzí algoritmu se stejnými výsledky.

Dalším problémem se ukázalo použití širší (pootočené) sady Haarových příznaků. Tato sada po natrénování sice poskytovala v CPU verzi implementace lepší výsledky, ale GPU implementace detektoru odmítala s takto vytvořený trénovací soubor akceptovat. Aby bylo možné v rámci práce porovnávat CPU a GPU verzi byla pro další práci použita pouze základní sada Haarových příznaků.

Předpokladem je, že v novějších verzích OpenCV bude možné používat pro obě implementace nový formát XML natrénovaných dat a bude možné využít i na GPU rozšířené sady Haarových příznaků, což by mělo poskytnout spolehlivější detekci.

4.2.2 Trénování algoritmu SVM

Pro vhodné natrénování algoritmu SVM bylo třeba připravit, obdobně jako pro algoritmus Viola-Jones sadu pozitivních a negativních vzorků. Vzhledem k nedostatku reálných podkladů, bylo opět využito generátoru primárně určeného pro Viola-Jones.

Nevýhodou tohoto přístupu je nutnost přizpůsobit se výstupu generátoru pozitivních vzorků a zpracovávat tak vzorky v SVM pouze ve stupních šedé.

SVM training tool

Jelikož trénování modulu SVM je časově méně náročné, než Viola-Jones není jako součást OpenCV distribuovaný nástroj pro trénování algoritmu z příkazové řádky, ale je nutné třídu reprezentující modul SVM natrénovat v rámci vlastní implementace. Jelikož je možné stav natrénovaného modulu uložit do samostatného souboru byl vytvořen nástroj, který vygeneruje soubor s natrénovanými hodnotami. Vlastní řešení detekce tedy pouze načítá připravená data.

Nástroj provede sekvenční načtení pozitivních a negativních vzorků z předem definovaných souborů, které obsahují seznam obrázků. Pro každý obrázek je provedena normalizace (převzorkování na velikost 64×64 pixelů) a je pro něj spočítán vektor *histogramů orientovaných gradientů*. K tomuto vektoru je přiřazen příznak, který udává, zda se jedná o pozitivní vzorek či negativní. Po zpracování všech dostupných souborů je provedeno natrénování modulu SVM.

Nastavení SVM je popsáno v kódu 4. Typ SVM je nastavený na `CvSVM::C_SVC`, což je n-třídový klasifikátor. Kernel je nastavený jako `CvSVM::LINEAR`, tedy lineární regrese, která je zároveň nejrychlejší.

Kód 4: Trénování SVM

```
CvSVMParams params;
params.svm_type     = CvSVM::C_SVC;
params.kernel_type  = CvSVM::LINEAR;
params.term_crit    = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);

CvSVM svm;
svm.train(training_data, labels, cv::Mat(), cv::Mat(), params);
```

Testování SVM

Pro natrénování SVM bylo použito 4000 vzorků, které obsahovaly 1000 pozitivních vzorků a 3000 negativních vzorků, rozdělených na dvě skupiny (testovací a trénovací). Algoritmus byl skrze první skupinu natrénován a jeho úspěšnost byla následně ověřena na testovací skupině což je uvedeno v tabulce 1. Natrénovaný SVM modul byl následně ručně ověřen na testovacím úseku silnice.

Tabulka 1: Výsledek testování SVM

Počet vzorků	**Korektně určené**	**Chybně určené**	**Úspěšnost**
2000	1436	564	71 %

Trénování nad generovanými daty ukázalo, že algoritmus je schopen dosáhnout, i na takovýchto datech, velmi dobrých výsledků a úspěšnosti přes 90%. Generovaná data, ačkoliv se reálným ve velké míře přibližují, však nedokážou nahradit data reálná. Sady, které vykazovaly po natrénování vysokou úspěšnost na testovacích datech se ukázaly na testovacím úseku reálné silnice nepoužitelné a některé značky algoritmus minul. Proto, s ohledem na použitelnost celé aplikace, byla vybrána sada, která sice nedosahovala v trénovací fázi nejlepších výsledků, ale dokázala na úseku reálné silnice, i přes zvýšený výskyt chybně pozitivních identifikací, žádnou ze značek neminout.

Je předpokladem, že natrénování SVM s dostatkem reálných dat by mělo tento krok zcela odstranit a poskytnout dostatečnou přesnost identifikace. Stejně tak kvalitnější a rozsáhlejší sada nasnímaných dat by lépe umožnila ověřit funkčnost druhé fáze při rozlišování různých typů značek patřící do stejné kategorie.

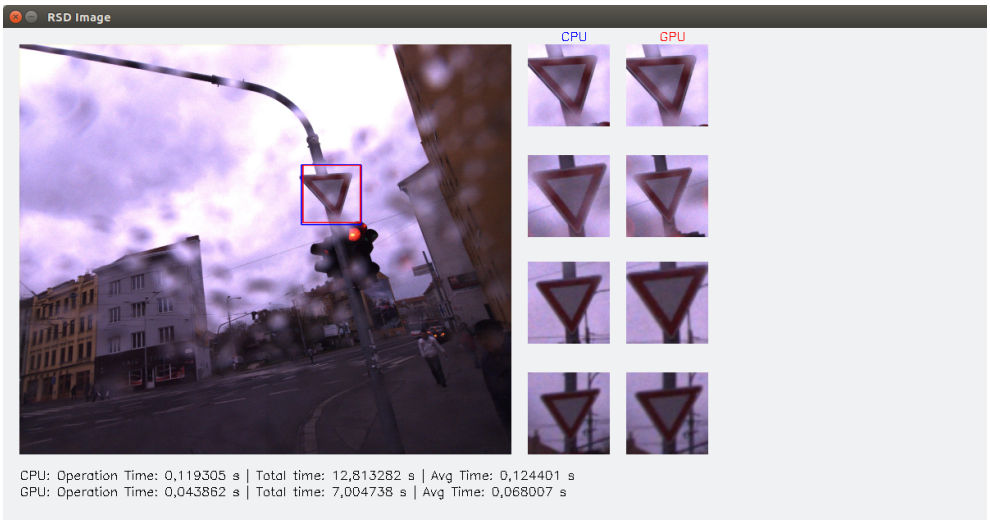
4.3 Implementace

Aplikace pro detekci a identifikaci značek je implementovaná prostřednictvím jazyka C++11 s překladačem g++ verze 4.8.2 pod operačním systémem Ubuntu 14.04 LTS s využitím knihovny OpenCV v podobě, která umožňuje využívat GPU implementace vybraných algoritmů prostřednictvím Nvidia CUDA. Aplikace poskytuje jed-

noduché grafické rozhraní, které zobrazuje průjezd nasnímanou trasou a informuje o proběhu detekce, statistických údajích (časy výpočtů) a nalezených značkách.

4.3.1 Uživatelské rozhraní aplikace

Základem aplikace je hlavní okno, které zobrazuje detekované ROI, které poskytuje kombinace detektoru Viola-Jones a SVN. Ty jsou zvýrazněny vykreslením čtverců – modrou barvou pro CPU verzi, červenou barvou pro GPU verzi. V dolní části aplikace je jsou vypsány průběžné výsledky detekce. Jedná se zvláště o hodnoty aktuální rychlosti výpočtu CPU verze a GPU verze algoritmu nad zobrazeným obrazem, celková doba výpočtu pro obě varianty a průměrná doba výpočtu obou variant.



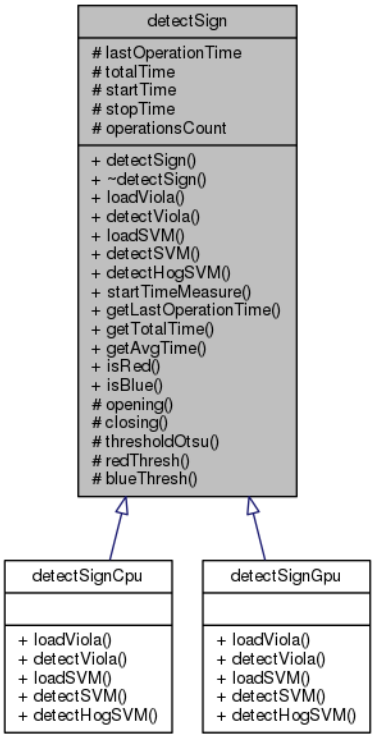
Obrázek 21: Vzhled experimentální aplikace RSID. Zdroj: Karel Zídek

V pravé části okna jsou připravené dva sloupce pro výřezy u kterých algoritmus z dostupných vzorků zobrazí ty, které považuje za identifikovanou značku. Je tedy možné porovnat jak rychlost CPU a GPU verze, tak nalezené detekce obou algoritmů. Okno je je tvořeno s využitím funkcionality OpenCV a jeho vzhled je ilustrovaný na obrázku 21.

4.3.2 Modul Viola-Jones

Modul Viola-Jones byl implementovaný v rámci svou tříd – `detectSignCpu` a `detectSignGPU`. Tyto třídy jsou potomky třídy `detectSign`. Třída `detectSign` deklaruje virtuální metody pro obecnou detekci skrze algoritmus Viola-Jones a tedy vlastní implementace detekčního algoritmu je řešena až v rámci potomků (obrázek 22). Toto řešení umožňuje volit typ detekce až za běhu aplikace vytvořením instance zvoleného typu třídy. Tento přístup byl zvolen, protože grafická karta podporující akceleraci skrze Nvidia CUDA nemusí být nutně na sestavě, která aplikaci provozuje

přítomna, takže je možné v rámci implementace jednoduchou detekcí tuto situaci ošetřit a volit způsob druh implementace podle dostupného hardware.



Obrázek 22: Diagram dědičnosti třídy detectSign Zdroj: Karel Zídek

CPU implementace

CPU implementace, která je zobrazená v kódu 5, spočívá ve vytvoření třídy `cv::CascadeClassifier` do které je načtena natrénovaná konfigurace z XML souboru. Tento proces se v rámci implementace provádí pouze při inicializaci, pro detekci je následně třída využívána opakovaně. Pro detekci třída poskytuje metodu `detectMultiScale`, která detekuje objekty s různými rozměry ve vstupním obrázku.

Kód 5: Implementace CPU detekce pomocí algoritmu Viola-Jones

```
cv::CascadeClassifier cascade;
cascade.load(xmlFile);

std::vector<cv::Rect> objects;
cascade.detectMultiScale(image, objects, 1.1, 4,
    0 , cv::Size(24,24), cv::Size(400,400));
```

Parametry metody jsou vstupní obrázek (`image`) a výstupní vektor čtvercových oblastí detekovaných objektů (`objects`) a omezení na minimální a maximální

velikost detekovaných oblastí. Mezi klíčové parametry patří parametr škálování a minimální počet sousedů.

Detekce různě velikých objektů se provádí postupným škálováním vstupního obrázku. Parametrem škálování lze tedy ovlivnit nejenom počet jednotlivých velikostních vzorků (a tedy časovou náročnost), ale i přeneseně přesnost detekce, protože velmi volné nastavení může způsobit minutí některých objektů. Parametr byl nastaven na hodnotu 1.1×.

Počet sousedů udává kolik nalezených čtverců musí být v oblasti detekce určeno, aby byl vzorek přijat. Hodnota byla experimentálně zvolena na 3 čtverce, což se ukázalo jako nastavení poskytující přijatelný výstup.

GPU implementace

GPU implementace založena na třídě `cv::gpu::CascadeClassifier_GPU`, zobrazená v kódu 6, je třída obdobná CPU verzi. Metodou `detectMultiScale` určenou pro detekci je obdobou metody CPU verze s rozdílem zaměněného pořadí parametrů. Jejich význam je totožný s CPU verzí.

Kód 6: Implementace GPU detekce pomocí algoritmu Viola-Jones

```
cv::gpu::CascadeClassifier_GPU cascade;
cascade.load(xmlFile);

cv::gpu::GpuMat gpuImage(image);
cv::gpu::GpuMat gpuObjects;
cv::Mat         gpuObjectsDownload;

std::vector<cv::Rect> objects;

int objectCount = cascade.detectMultiScale(gpuImage, gpuObjects,
    cv::Size(400,400), cv::Size(24,24), 1.1, 4 );

if (objectCount > 0) {
  gpuObjects.colRange(0, objectCount).download(gpuObjectsDownload);
  cv::Rect* hostObjects = gpuObjectsDownload.ptr<cv::Rect>();

  for(int i~= 0; i~< objectCount; ++i)
      objects.push_back(hostObjects[i]);
}
```

Pro účely srovnání byly parametry škálování a minimálního počtu sousedů nastaveny na totožné hodnoty jako u CPU verze.

Zásadním rozdílem oproti CPU verzi je přístup ke zpracovávaným datům. Jelikož implementace nemůže obejít principy používané při GPGPU výpočtech je nutné na grafické kartě nejdříve paměť alokovat, nahrát do ní potřebná data a poté zkopírovat výsledek zpět z GPU, jak ilustruje kód 6.

4.3.3 Modul SVM

Modul SVM je implementovaný obdobně jako v případě Viola-Jones ve třídách `detectSignCpu` a `detectSignGPU`. Toto řešení vychází z předpokladu výběru vhodné třídy za běhu aplikace.

CPU implementace

Pro druhou fázi porovnání prostřednictvím SVM je nejdříve nutné pro výsek obrázku poskytnutý modulem Viola-Jones spočítat *popisný vektor HOG* (*HOG feature vector*). Výpočet je proveden metodou `getHog()`, která jako parametry přijímá vstupní výřez a vektor pro uložení výsledku výpočtu.

Metoda `getHog()`, jejíž funkčnost je ilustrovaná v kódu 7, převede obrázek do černobílých barev a změní velikost na 64×64 pixelů.

Kód 7: Implementace metody getHog()

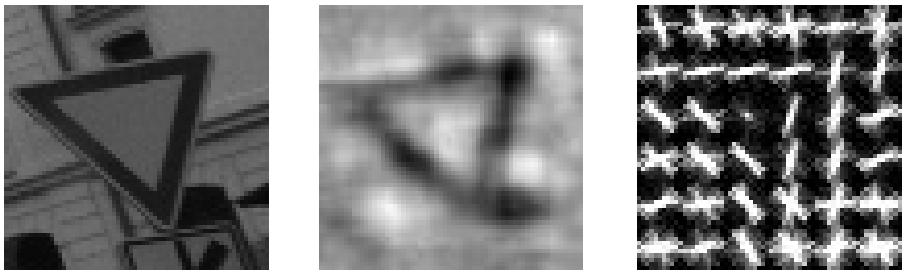
```
cv::HOGDescriptor hog;
hog.winSize = cv::Size(64,64);

std::vector<float> feature;

cv::Mat image;
cv::cvtColor(inputImage, image, CV_BGR2GRAY);
resize(image,image, cv::Size(64,64));

std::vector<cv::Point> locations;
hog.compute(image, feature, cv::Size(8, 8), cv::Size(0, 0),
    locations);
```

Tento krok je ústupek použitému způsobu trénování. Jelikož trénovací data byla vytvořena generátorem pro Viola-Jones jsou ve stupních šedé, je třeba do takto naučeného SVM posílat co nejshodnější vstup. V případě, že by bylo možné natrénovat SVM pomocí množství reálných (bavených) vzorků měl by tento krok být zcela vypuštěn, protože SVM samotné dokáže barevné obrázky zpracovat.

Metoda spočítá výsledný *vektor HOG vlastností* do proměnné `feature`, k výpočtu použije předem nastavenou velikost okna (`winSize` – 64×64 pixelů) a parametrem předanou velikost kroku (8×8 pixelů). Zbylé parametry nemají na požadovaný výpočet vliv a je možné jejich hodnoty zanedbat.

Jakým způsobem vypadá vizualizace vypočítaného *HOG vektoru* pro zvolenou značku v analyzovaných datech ilustruje obrázek 23. První část obrázku ukazuje výřez, který poskytuje SVM modulu detekce skrze Viola-Jones. Druhá část ukazuje obraz, který je upraven do podoby připravené k výpočtu *HOG vektoru*. V třetí části je vizualizovaný vypočtený *HOG vektor* zkoumané značky.

V rámci samotné detekce (kód 8) je po výpočtu *HOG vektoru* provedena konverze z `std::vector` do `cv::Mat` a výsledná matice pojmenovaná `data` je předána



Obrázek 23: Výpočet HOG vektoru. Zdroj: Karel Zídek, Data: Geodis, Vizualizace: HOGgles (29)

jako parametr metody `predict` třídy `svm`. V případě binární klasifikace je výstupem hodnota `result`, která nabývá hodnot $+1$ a -1, což značí zařazení do třídy pozitivních či negativních vzorků.

Kód 8: Implementace SVM predikce (CPU varianta)

```
CvSVM svm;

svm.load(datFile.c_str());

std::vector<float> hogFeature;
cv::Mat data(1,hogFeaturesCount,CV_32FC1);

getHog(image, hogFeature);

for (int i=0; i~< hogFeaturesCount; i++) {
    data.at<float>(0,i) = hogFeature.at(i);
}

float result = svm.predict(data);
```

GPU implementace

GPU implementace využívá třídy `cv::gpu::HOGDescriptor` (kód 9), která umožňuje počítání a detekci objektů za pomoci HOG. Implementace této třídy je primárně využívaná pro detekci lidí a tedy v rámci OpenCV jsou nabízeny přepočítané modely SVM, které lze v rámci aplikace nastavovat pomocí metody `setSVMDetector`.

Pokud chce uživatel použít vlastní definici SVM detektoru dostává se do drobných obtíží, protože samotný `cv::gpu::HOGDescriptor` neumožňuje načítat natrénované hodnoty SVM z datového souboru a naopak třída `CvSVM`, určená pro vlastní detekci SVM (bez podpory GPU), která načítání umožňuje, sice interně pracuje s potřebnými natrénovanými daty, ale proměnná, která je ukládá je typu `private`

a tedy nepřístupná spolu s neexistující metoda na jejich získání. řešením se nakonec ukázal kód uživatele DXM podůrného programátorského fóra *StackowerFlow*, který poskytl uživatelům s podobným problémem kód potomka třídy `CvSVM` s impelemntovanou metodou `getSupportVector()`, která převádí interní reprezentaci vektoru do formátu potřebného pro načtení ve třídě `cv::gpu::HOGDescriptor` (8). Toto řešení bylo v rámci aplikace využito pro získání zmíněného vektoru.

Kód 9: Implementace SVM predikce (GPU varianta)

```
CvSVM svm;
svm.load(datFile.c_str());

std::vector<float> supportVector;
svm.getSupportVector(supportVector);

cv::gpu::HOGDescriptor svmHog;
svmHog = cv::HOGDescriptor(cv::Size(64,64), cv::Size(16, 16),
    cv::Size(8, 8), cv::Size(8, 8), 9);
svmHog.setSVMDetector(supportVector);

cv::cvtColor(image , image ,  CV_BGR2GRAY);
resize(image,image, cv::Size(64,64));

cv::gpu::GpuMat gpuImage;
gpuImage.upload(image);

std::vector<cv::Point> objHog;
svmHog.detect(gpuImage,objHog);
```

V rámci implementace je tak při inicializaci detekce vytvořena instance modifikované třídy `CvSVM`. Z této třídy je získán natrénovaný vektor pro detekci značky (`supportVector`). Následně je vytvořena třída `cv::gpu::HOGDescriptor` s parametry velikosti celého vstupního obrázku (zde 64×64 pixelů), velikostí bloku (podporované pouze 16×16 pixelů) a velikostí kroku a buňky (podporované pouze hodnoty 8×8 pixelů) a počtu vzorků (*bins*), který je taktéž fixně stanoven na 9. Po vytvoření třídy je HOG detektoru nastaven SVM detektor pomocí získaného vektoru `supportVector`. Poté je provedena úprava obrázku do potřebné podoby načež je nahrán na GPU. Metodou `detect` následně do proměnné `objHog` získáme seznam bodu, které reprezentují pozitivní nálezy SVM detektoru.

4.4 Porovnání výsledků

Vlastní implementace byla spolu s natrénovanými detekčními algoritmy podrobena testu na reálných datech. Výsledky z této analýzy byly počty správných a chybně pozitivních detekcí v jednotlivých fázích, ale i časy za které si CPU a GPU verze dokázala s úlohou poradit. Lze tedy porovnat nejenom funkcionalitu vlastní apli-

kace, ale i zhodnotit, zda je při zpracování totožného úkolu vhodné používat GPU akcelerované varianty algoritmů.

Vzorek z reálného snímání silnic obsahuje jak průměrně kvalitní fotografie za běžného osvětlení, tak fotky méně kvalitní, kde postupně se zhoršující počasí přecházející v déšť, což se podepisuje na kvalitě pořízených snímků.

Snímaný úsek zahrnuje část ulice Úvoz a okolí Náměstí Míru v Brně a je složený ze 418 snímků, které byly poskytnuty firmou Geodis, s pracovním rozlišením 2448 × 2050 pixelů a s výskytem různých typů značek.

4.4.1 Výsledky modulu Viola-Jones

V první fázi zpracování vstupních dat bylo cílem modulu Viola-Jones detekovat místa předpokládaného výskytu obecného typu značky.

Typ značky: Dolu orientovaný trojúhelník

Modul Viola-Jones při detekci obecné, dolu orientované, trojúhelníkové značky dosáhl výsledků uvedených v tabulce 2. Z testu vyplynulo, že v rámci OpenCV je implementace CPU a GPU verze je do jisté míry odlišná – tato odlišnost se projevila, při stejně nastavených parametrech, natrénovaných datech a vstupech, v různých počtech detekcí. Experimentování s nastavením parametrů škálování a minimálním počtem sousedů dokázalo počty detekcí k sobě více přiblížit, ale i v těchto případech byly oběma verzemi algoritmu detekovány odlišné regiony ve zdrojovém obrázku.

Tabulka 2: Výsledek modulu Viola-Jones na 418 snímcích – dolu orientovaný trojúhelník

Varianta algoritmu	**Počet pozitivních detekcí**	**Počet chybně pozitivních detekcí**	**Počet minutých značek**
CPU	19	1574	0
GPU	20	144	0

Důležitým výsledkem je, že oba algoritmy dokázaly najít v rámci jízdy vozidla všechny průjezdy okolo zvoleného typu značky, kterou lze minout 7×. Předpokladem je, že vzhledem k neideálnímu vstupu do trénování této fáze, skrze generovaná data, dochází k ovlivnění vlastního výsledku a trénování s dostatečným množstvím reálných dat by poskytlo kvalitnější výsledek.

U průjezdu okolo značky je důležité, aby značka byla zachycena alespoň jednou v sekvenci po sobě následujících obrázků a v rámci průjezdu dané trasy nebyla minuta.

Na obrázku 24 je zobrazen příklad pozitivní detekce značky a tři typické příklady chybně pozitivních detekcí. Je vidět, že v případě chybně pozitivních detekcí se typicky jedná o objekty, které ostrými hranami trojúhelník značky výrazně při-

Obrázek 24: Příklad pozitivní a třech typických chybně pozitivních detekcí. Zdroj: Karel Zídek, Data: Geodis

pomínají. Např. většina chybně pozitivních vzorků CPU detekce obsahuje různě překřížené vedení, tyto vzorky jsou ve většině případu ignorovány GPU verzí.

Typ značky: Modrý obdélník

Vyhledávání modré značky znamenalo pro navržený model závažnější výzvou. Nejenom, že její tvar se podobá mnoha dalším objektům ve scéně. Navíc je ze značek ve snímaných datech poměrně málo výrazná. Výsledek měření provedený stejným přístupem, jako v případě trojúhelníkového typu značek, je zachycen v tabulce 3.

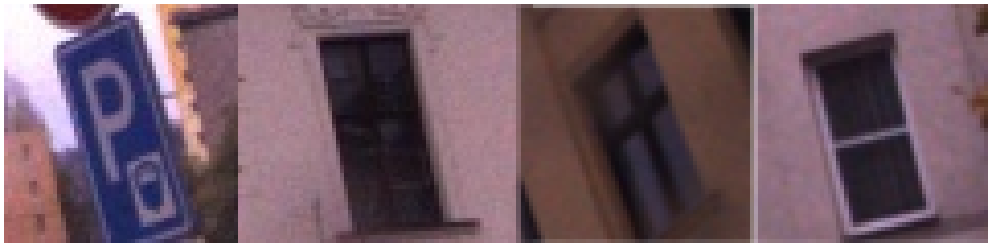
Tabulka 3: Výsledek modulu Viola-Jones na 418 snímcích – modrý obdélník

Varianta algoritmu	**Počet pozitivních detekcí**	**Počet chybně pozitivních detekcí**	**Počet minutých značek**
CPU	26	3012	0
GPU	19	1490	1

V rámci průjezdu bylo nalezeno 10 značek, které šlo detekovat. Některé z nich však nebyly po směru jízdy – jednalo se o značky v odbočkách, které byly kamerou zachyceny. GPU algoritmus minul pouze jednu značku během průjezdu, jednalo se právě o značku umístěnou ve vedlejší silnici (obrázek 25). Jak je možné z naměřených dat vyvodit, i při stejném nastavení, je GPU algoritmus schopen lepe detekovat hůře viditelné značky. Lze tedy předpokládat, že při změně parametrů a zvýšením citlivosti by značka byla zaznamenána, protože citlivější CPU verze ji bez problémů zachytila. Taktéž, jako v prvním případě, by těmto případům mělo předejít trénování s negenerovanými, reálnými vzorky.

Je nutné brát v potaz, že na vývojovém hardwaru (dvoujádrový procesor Intel Core i5) trvalo natrénování fáze Viola-Jones cca 10 hodin. Je tedy obtížné a extrémně časově náročné experimentálně v této oblasti hledat nejvhodnější nastavení trénování.

Zbylé chybně pozitivní detekce spadají zvláště do kategorie oken (obrázek 25), které tvarem, zkreslením (zkosením) i jasem do značné míry sdílí společné charakteristiky s hledaným typem.



Obrázek 25: Příklad minuté značky a třech typických chybně pozitivních detekcí. Zdroj: Karel Zídek, Data: Geodis

4.4.2 Výsledky modulu SVM

Modul SVM měl za cíl z předvybraných vzorků, u kterých se předpokládá, že obsahují detekovaný obecný typ značky, detekovat konkrétní variantu značky.

Značka: Dej přednost v jízdě

V prvním případě (dolu orientovaná trojúhelníková značka) se jednalo o určení značky „Dej přednost v jízdě“. Výsledné počty dobře a špatně identifikovaný vzorků jsou uvedeny v tabulce 4. Okolo značky bylo v rámci celé jízdy projeto 7×.

Tabulka 4: Výsledek modulu SVM při identifikaci – „Dej přednost v jízdě“

Varianta algoritmu	**Počet vstupních vzorků**	**Počet pozitivních detekcí**	**Počet chybně pozitivních detekcí**	**Počet chybně negativních detekcí**
CPU	1593	19	281	0
GPU	164	20	94	0

Bylo by jistě žádoucí, aby hodnoty chybně pozitivních identifikací nabývaly co nejnižších hodnot. V rámci experimentální aplikace je nejdůležitějším poznatkem, že žádný vzorek, který obsahoval hledanou značku, nebyl minut. V případě implementace do systému inventarizace značek by i tyto hodnoty umožnili technikovi jednoduše chybně pozitivní vzorky označit ručně. Na základě schválení výsledků technikem by následně bylo možné rozšířit sady pozitivních i negativních vzorků a přeučením obou fází výrazně zpřesnit detekci. Předpokladem je, že takováto aplikace by po několika detekčních průchodech měla výrazně zlepšit svoji přesnost.

Značka: Parkování

V druhém případě se jednalo o určení obdélníkové modré značky „Parkování“. Výsledné počty pozitivních a chybně pozitivních vzorků jsou uvedeny v tabulce 5. Okolo značky bylo v rámci celé jízdy projeto 10×.

Tabulka 5: Výsledek modulu SVM při identifikaci – „Parkování“

Varianta algoritmu	Počet vstupních vzorků	Počet pozitivních detekcí	Počet chybně pozitivních detekcí	Počet chybně negativních detekcí
CPU	3038	26	792	0
GPU	1509	19	685	0

Náročnější varianta značky přinesla, v porovnání s prvním případem, horší výsledky. Opět lze předpokládat, že kvalitu detekce lze ovlivnit v rámci obou fází přetrénováním s kvalitnějšími vzorky značek.

Dodatečné zpřesnění detekce

V rámci aplikace byl proveden experiment, kdy navržené řešení skrze moduly umělé inteligence bylo experimentálně podpořeno volně nastaveným algoritmem z oblasti segmentace.

Vzorky nalezené pomocí modulu Viola-Jones byly filtrovány na základě předpokladu barvy značky. Filtr byl nastaven velmi volně a bylo předpokládáno, že stačí pouze $1/10$ pixelů nacházejících se ve stanoveném barevném rozsahu. Stejně tak byly zahozeny vzorky příliš malé. Tyto kroky efektivně odfiltrovaly všechny chybně pozitivní vzorky z budov, oblohy či oken a ukázaly tak, že vhodnou kombinací dostupných algoritmů lze výsledky podstatně zlepšit. V rámci toho filtru nebyla minuta žádná ze značek, během průjezdy trasou, dostupná v sadě detekovaných vzorků modulem Viola-Jones.

Tabulka 6: Výsledky dodatečného zpřesnění detekce

Varianta	Počet vstupních vzorků	Počet pozitivních detekcí	Počet chybně pozitivních detekcí	Počet chybně negativních detekcí
Červená CPU	300	15	4	4
Červená GPU	114	16	1	5
Modrá CPU	818	24	45	0
Modrá GPU	704	19	17	1

4.4.3 Výsledky doby výpočtu

V rámci běhu aplikace byly měřený jednotlivé výpočetní části pro obě varianty algoritmů. Aplikace v rámci svého průběhu zobrazuje informace o době výpočtu aktuálního snímku, celkovém času výpočtů a průměrné hodnotě doby výpočtu. Tyto

hodnoty jsou vypočítány průměrem z deseti průjezdů zvolenou trasou na použité konfiguraci.

V rámci časového měření byla provedena detekce a identifikace prvního (trojúhelníkového) typu značky. Změřené hodnoty času jsou uvedeny v tabulce 7. GPU algoritmus při zvolených hodnotách a velikosti vstupních dat dosahuje 29 % zlepšení, při aplikačně používaném rozlišení 1224 × 1025, oproti CPU variantě.

Tabulka 7: Časy výpočtů CPU a GPU detekce na 418 snímcích trojúhelníkové značky (průměr z 10 průjezdů trasy)

Varianta algoritmu	**Rozlišení**	**Celkový čas výpočtu**	**Průměrný čas výpočtu**	**Zrychlení**
CPU	2448 × 2050	225,08 s	0,5463 s	
GPU	2448 × 2050	161,50 s	0,3920 s	28 %
CPU	1224 × 1025	45,53 s	0,1105 s	
GPU	1224 × 1025	32,43 s	0,0787 s	29 %
CPU	816 × 683	17,17 s	0,0417 s	
GPU	816 × 683	12,91 s	0,0313 s	25 %

GPU implementace tak poskytuje dostatečné zrychlení oproti CPU variantě a je možné uvažovat implementaci a provoz právě na GPU. Pro porovnání byl průchod aplikací a detekce změřena při plném, polovičním a třetinovém rozlišení. Výsledky tohoto porovnání jsou zobrazeny v tabulce 7 a ukazují přínos GPU verze při všech testovaných rozlišeních.

5 Závěr

Hlavním cílem práce bylo vytvoření experimentální aplikace detekující dopravní značky a na základě této aplikace provést zhodnocení vlivu GPU akcelerace pro řešení tohoto úkolu. Vytvořená aplikace ukázala, že akcelerace prostřednictvím GPU není pro použití v systému, který neběží v reálném čase, kritická, ale její využití a přínos je znatelný a je tedy možné řešení založené na GPU verzi detektoru doporučit.

Původní zadání bylo obohaceno o návrh vlastního řešení pro detekci značek pomocí dvoufázového přístupu, který byl uvažován zvláště s ohledem na praktické požadavky geografických firem a jejich inventarizačních systémů pro správu silnic.

Experimentální aplikace pro detekci značek potvrdila, že navržený dvoufázový přístup zvolený pro detekci značek je použitelný a že přináší předpokládané výhody. V první fázi dojde k detekci obecného typu značky. V druhé fázi je možné, v rámci obecného typu značky, dále rozlišit konkrétní druh značky s možností dalšího trénování na základě dat, které systém zpracovává.

Výkon a výsledky aplikace ukázaly, že dosažené řešení je zcela akceptovatelné, ale v rámci aplikace je mnoho míst, skrze které lze dosáhnou výsledků daleko lepších. Stejně tak experiment ukázal, že i lehká podpora skrze další algoritmy může značně pomoci k zlepšení dosažených výsledků.

Základní oblastí pro zlepšení je množství a kvalita reálných dat – zde lze očekávat, že firma zabývající se pořizováním prostorových dat právě těmito materiály disponuje a v případě získání takových to dat lze algoritmy počítačového učení, skrze reálná data, natrénovat k výrazně lepším výsledkům.

Aplikace může být budoucnu rozšířen do podoby, která dále spravuje správně a chybně identifikované vzorky poskytovat prostředky pro získávání nových dat pro trénování a tedy zlepšovat kvalitu detekce v průběhu svého provozu.

V rámci vývoje aplikace bylo třeba překonat i několik problémů, které se vázaly, jak k formátu natrénovaných hodnot které algoritmy akceptují, tak k nemožnosti využít širší sadu Haarových příznaků algoritmu Viola-Jones. Tento problém, který se nejspíše váže k přípravě nové hlavní verze OpenCV 3, se však podařilo překonat v podobě, že nenarušuje běh aplikace, ale zároveň poskytuje místo pro zlepšení, jakmile budou patřičné chyby v implementaci OpenCV opraveny.

Literatura

1. Ballard, D.; Brown, C.: *Computer Vision – The Hough method for curve detection*. Prentice-Hall, 1982, 123–131 s.
   URL http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/LIB/bandb4_3.pdf
2. Beucher, S.; Lantuéj, C.: Use of Watersheds in Contour detection. *Workshop on image processing*, 1979: str. 1–12.
   URL http://cmm.ensmp.fr/~beucher/publi/watershed.pdf
3. Bonači, I.; Kusalić, I.; Kovacek, I.; aj.: Addressing false alarms and localization inaccuracy in traffic sign detection and recognition. *16th Computer Vision Winter Workshop*, 2011: s. 1–8.
4. Buades, A.; Coll, B.; Morel, J.-M.: Non-Local Means Denoising. [online], 2011.
   URL http://www.ipol.im/pub/art/2011/bcm_nlm/
5. Couprie, M.; Bertrand, G.: Topological gray-scale watershed transform. *In Proc. of SPIE Vision Geometry*, ročník V, č. 3168, 1997: str. 136–146.
6. Dalal, N.; Triggs, B.: Histogram sof Oriented Gradients for Human Detection. *Conference on Computer Vision and Pattern Recognition*, 2005.
7. Diard, F.: GPU Gems 3. [online], 2008, [cit. 12.04.2015].
   URL http://http.developer.nvidia.com/GPUGems3/gpugems3_ch41.html
8. DXM: Training custom SVM to use with HOG Descriptor in OpenCV. [online], 2014, [cit. 6.05.2015].
   URL http://stackoverflow.com/questions/15339657/training-custom-svm-to-use-with-hogdescriptor-in-opencv
9. Fisher, R.; Perkins, S.; Walker, A.; aj.: Hough transform. [online], 2003, [cit. 29.04.2015].
   URL http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm
10. Fleyeh, H.: Color detection and segmentation for road and traffic signs. *IEEE Conference on Cybernetics and Intelligent Systems*, 2004.
11. Levi, G.: A Short introduction to descriptors. [online], 2015, [cit. 26.02.2015].
    URL https://gilscvblog.wordpress.com/2013/08/18/a-short-introduction-to-descriptors/
12. Lienhart, R.; Maydt, J.: An Extended Set of Haar-like Features for Rapid Object Detection. *ICIP02*, ročník I, 2002: s. 900–903.
13. Linderoth, M.; Robertsson, A.; Johansson, R.: Color-based detection robust to varying illumination spectrum. *Robot Vision (WORV)*, 2013: s. 120–125.
14. Malach, T.; Bambuch, P.; Malach, J.: Detekce obličeje v obraze s využitím prostředí MATLAB. *Mezinárodní konference Technical Computing Prague*, ročník 13, 2011: str. 78.

15. Moroney, N.; Fairchild, M.; Hunt, R.; aj.: The CIECAM02 Color Appearance Model. *International Commission on Illumination (CIE) Technical Committee 8-01*, 2002: s. 1–6.

16. Mrázová, I.: Aplikace teorie neuronových sítí. [online], 2011, [cit. 04.05.2015]. URL http://ksvi.mff.cuni.cz/~mraz/atns/ATNS_Prednaska_SVM.pdf

17. OpenCV: Canny Edge Detector. [online], 2015, [cit. 25.02.2015]. URL http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

18. Otsu, N.: A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, ročník 9, č. 1, 1979: s. 62–66.

19. Paris, S.; Durand, F.: A Fast Approximation of the Bilateral Filter using a Signal Processing Approach. *European Conference on Computer Vision*, ročník 9, č. 33954, 2006: s. 568–580.

20. Pinard, P. T.: Hough transform. [online], 2011, [cit. 20.04.2015]. URL http://www.ebsd-image.org/documentation/reference/ops/hough/op/houghtransform.html

21. Shapiro, L.; Stockman, G.: *Computer Vision*. Prentice Hall, 2001.

22. Smith, S.; Brady, J.: SUSAN – a new approach to low level image processing. *International Journal of Computer Vision*, ročník 23, č. 3, 1997: s. 45–78.

23. Smith, S. W.: The Scientist and Engineer's Guide to Digital Signal Processing. [online], 2011. URL http://www.dspguide.com/

24. Thomé, A. C. G.: *SVM Classifiers – Concepts and Applications to Character Recognition*. 2012, ISBN 978-953-51-0823-8. URL http://dx.doi.org/10.5772/52009

25. Tomasi, C.; Manduchi, R.: Bilateral filtering for gray and color images. *International Conference on Computer Vision, IEEE*, 1998: s. 839–846.

26. Vedaldi, A.: Histogram of Oriented Gradients (HOG) features. [online], 2015, [cit. 15.03.2015]. URL http://www.vlfeat.org/api/hog.html

27. Verplancke, T.; Looy, S.; Benoi, D.; aj.: Support vector machine versus logistic regression modeling for prediction of hospital mortality in critically ill patients with haematological malignancies. *BMC Medical Informatics and Decision Making*, ročník 56, č. 8, 2008.

28. Viola, P.; Jones, M. J.: Robust Real-Time Face Detection. *International Journal of Computer Vision*, ročník 57, č. 2, 2004: s. 137–154.

29. Vondrick, C.; Khosla, A.; Pirsiavash, H.; aj.: HOGgles: Visualizing Object Detection Features. [online], 2013, [cit. 10.03.2015].
URL http://web.mit.edu/vondrick/ihog/

30. Zídek, K.; Koubek, T.; Procházka, D.; aj.: Architecture of Assistance System for Traffic Signs Inventory. *Enterprise and the Competitive Environment*, 2015.

31. Československo: Vládní nařízení 203/1935 Sb., kterým se provádí zákon ze dne 26. března 1935, č. 81 Sb. z. a n., o jízdě motorovými vozidly. 1935, příloha D.

32. Československo: Vládní nařízení 242/1939 Sb. o chování v silniční dopravě (dopravní řád silniční). 1939, příloha č.1.

33. Československo: Vyhláška 141/1960 Sb. ministerstva vnitra ze dne 3. září 1960, kterou se vydávají pravidla silničního provozu. 1960, příloha.

34. Škropil, V.; Zídek, K.; Koubek, T.; aj.: Image Processing Methods Optimization by Means of GPU Computing. *In Proceedings of the 16th WSEAS International Conference on Communications (part of CSCC '12)*, 2012: s. 95–101, iSBN 978-1-61804-109-8.

35. Žižka, J.: Support vector machines (SVM). [online], 2005, [cit. 12.04.2015].
URL https://is.muni.cz/el/1433/podzim2005/PA034/09_SVM.pdf

Přílohy

A Obsah CD

- **Zdrojové kódy** experimentální aplikace
- **Natrénovaná data** algoritmů Viola-Jones a SVM
- **Skripty a nástroje**
- **Vlastní práce** v elektronické podobě