Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Rozpoznávání struktury křižovatky z veřejně dostupných zdrojů (map)

Plzeň, 2014 Tomáš Pašek

Prohlášení

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

Plzeň, 2014

Tomáš Pašek

Poděkování

Rád bych poděkoval vedoucímu této diplomové práce, Ing. Tomášovi Potužákovi Ph.D., za cenné rady a konzultace v průběhu jejího zpracování. Dále bych chtěl poděkovat Ing. Kamilu Ekšteinovi Ph.D. za konzultace algoritmů z oblasti rozpoznávání obrazu.

Abstrakt

Cílem této diplomové práce je rozpoznání křižovatky ze satelitních snímků, které může být použito pro simulování dopravní sítě. Tato práce prozkoumává různé algoritmy rozpoznávání obrazu, vysvětluje jejich matematické principy a jejich vhodnost pro oblast rozpoznávání křižovatky. Vybrané algoritmy jsou použity v navržené a vytvořené aplikaci, jejíž výstupem je strukturovaně popsaná křižovatka. Nakonec je testována a zhodnocena účinnost rozpoznávání aplikace.

Klíčová slova: rozpoznání obrazu, křižovatka, Houghova transformace, SIFT, SURF, detekce kontur, OpenCV, Java

Abstract

The aim of this thesis is crossroads recognition based on satellite pictures which may be used for traffic network simulations. This work studies different algorithms for image recognition, explains their mathematical principles and their suitability for domain of crossroads recognition. Selected algorithms are used in designed and created application which produces structured crossroad description. Finally, recognition success of the application is tested and evaluated.

Keywords: image recognition, crossroad, Hough transform, SIFT, SURF, contours detection, OpenCV, Java

Obsah

1 Úvod

Úkolem této práce je implementovat aplikaci, která ze satelitního snímku (dostupného z některého z mapových serverů) zachycujícího běžnou dopravní křižovatku automaticky vygeneruje strukturovaným popisem rozpoznanou křižovatku. Program by měl určit kolik se v křižovatce vyskytuje ramen, do jakého směru vedou, kolik obsahují vstupních (řadicích) a výstupních jízdních pruhů. V případě vstupních pruhů je nutné určit vliv pruhu na křižovatku (tzn. zda-li se jedná o pruh rovný, odbočovací, nebo jejich kombinaci). Výstup této práce by měl posloužit pro urychlení vytvoření strukturovaného zápisu křižovatek využitelného v modelování simulace dopravní sítě.

Tato práce se nejprve v Kapitole 2 zaměří na definici jednotlivých prvků křižovatky a jejich zajímavých vlastností. Dále prozkoumáme některé ze známých metod pro zpracování obrazu a rozpoznávání a budeme uvažovat použití těchto metod pro detekci jednotlivých prvků křižovatky (Kapitoly 3, 4 a 5). Na základě této úvahy se v Kapitole 7 pokusíme vybrat množinu metod, jejichž implementace by mohla k úspěšnému rozpoznání křižovatky. Podíváme se také na komerční a volně dostupné nástroje, které nám mohou usnadnit vývoj aplikace (Kapitola 6).

Před samotnou implementací aplikace provedeme v Kapitole 8 analýzu jednotlivých funkčních celků programu a pokusíme se vytvořit objektově orientovaný návrh aplikace bez vazby na konkrétní programovací jazyk. Aplikaci následně implementujeme dle objektového návrhu a dříve vybrané množiny metod vedoucích k řešení. Implementovanou aplikaci nakonec v Kapitole 9 otestujeme a z výsledků vyvodíme závěr.

2 Popis křižovatky

V této kapitole budou podrobně popsány způsoby detekce různých prvků křižovatky. Nejprve si představíme jednotlivé prvky, které můžeme v křižovatce nalézt a u některých z nich vyzdvihneme jejich vlastnosti, které později použijeme pro detekci. Po představení všech prvků křižovatky a jejich vlastností probereme strukturovanou podobu křižovatky. Následně se zaměříme na možnosti detekce důležitých prvků křižovatky. Nejprve se blíže podíváme na rozpoznání okrajových a dělících čar vyskytujících se na vozovce, dále pak na rozpoznání směrových šipek, kterých využijeme pro detekci řadicích pruhů a ramen křižovatky.

2.1 Prvky křižovatky

Pro pochopení problematiky je vhodné nejprve si uvědomit, jaké všechny elementy se na pozemní komunikaci a křižovatce mohou vyskytnout. Na vozovce můžeme bohužel nalézt celou řadu rušivých objektů, které budou rozpoznávání ztěžovat. V této podkapitole bude provedena dekompozice prvků křižovatky od největších po nejmenší s definováním zajímavých vlastností daných prvků z hlediska budoucího rozpoznávání.

2.1.1 Rameno křižovatky

Každá křižovatka se skládá alespoň ze dvou ulic (případně silnic), které se kříží. Budeme-li se na tyto cesty dívat ze středu křižovatky, můžeme o těchto cestách mluvit jako o ramenech křižovatky. Rameno křižovatky představuje množinu vstupních a výstupních jízdních pruhů, které spolu utváří jasně oddělitelnou část křižovatky. U každého ramena křižovatky tedy lze určit jeho počátek (zřejmě střed křižovatky) a jeho směr. Směr můžeme udávat několika různými způsoby. Můžeme ho chápat jako světovou stranu (pak ho lze reprezentovat slovní popisem světové strany nebo číselným popisem, kdy dvanáct světových směrů budeme reprezentovat čísly na hodinách), nebo ho můžeme vyjádřit jako úhel, který svírá s předem definovanou přímkou (například s osou x nebo y). Směr budeme vždy chápat jako světovou stranu (případně úhel), na kterou bychom se vydali, kdybychom opouštěli křižovatku z jejího středu po daném rameni.

Rameno obsahuje dva druhy pruhů. První typ označujeme jako výstupní a jedná se o pruhy, kterými křižovatku opouštíme. Druhým typem jsou pruhy vstupní (řadicí), tedy pruhy po nichž do křižovatky vjíždíme a jsou určené pro řazení automobilů před křižovatkou, nebo místem odbočení [1]. Řadicí pruhy je dále možné dělit do pěti skupin, dle jejich směřování, čili:

- rovný pruh
- rovný a odbočovací pruh (vpravo, vlevo, nebo vpravo i vlevo zároveň)
- odbočovací pruh (vpravo, vlevo, nebo vpravo i vlevo zároveň)

2.1.2 Vozovka

Povrch samotné vozovky je tvořený asfaltovou vrstvou. Nepříjemností pro rozpoznávání je, že asfalt na vozovkách z technických důvodů nemůže podléhat normám zákona, které by určovaly

jeho barevnost. Nelze tedy jasně určit rozmezí, ve kterém se bude pohybovat rozsah barev vozovky a bude velmi těžké odlišit vozovku od jiných prvků okolí (například chodníky, tramvajové pásy kolejí), které budou narušovat rozpoznávání.

Pokud vozovka obsahuje alespoň dva jízdní pruhy, jejichž šířka je větší než 3,00 m, pak musí být označena vodorovným značením s dělící čarou mezi jednotlivými pruhy. Z historických důvodů, kdy stavění budov a komunikací nepodléhalo dnešním normám, tato podmínka není vždy splněna. Pokud tedy vozovka obsahuje dva pruhy a její šířka je menší než 6,00 m, vyznačují se pouze okrajové čáry vozovky [1].

2.1.3 Okrajové a dělící čáry na vozovce

Okraj vozovek I. třídy se ze zákona označuje vodicí čárou, která musí být bílá a musí mít šířku 0,125 m. Jednotlivé jízdní pruhy musí být také odděleny bílou čarou (pokud to šířka vozovky umožňuje, viz výše), která je rovnoběžná s okrajovou čarou a musí být ve stanovené vzdálenosti 3,00 m. Šířka jízdního pruhu v obci, rychlostní silnice nebo rychlostí místní komunikace by neměla překročit 3,50 m [1]. Křižovatky ve městech a na hlavních tazích z pravidla mívají okrajové značení a je proto možné se pokusit toto značení detekovat. Situaci ale může ztížit fakt, že se můžeme setkat i s šířkou okrajové čáry 0,25 m. Tento případ může nastat při oddělení parkovacího pruhu, nebo na směrově rozdělené pozemní komunikaci.

V některých nebezpečných místech, nebo na křižovatkách, na nichž se protisměrné pruhy rozevírají ke středu křižovatky, můžeme nalézt dvojitou dělící čáru. Šířka mezi jednotlivými čárami dvojité dělící čáry je zpravidla 0,125m, ale může být i větší. Pokud šířka mezery překročí hodnotu 0,5 m, je nutno plochu mezi čárami vyplnit šikmými rovnoběžnými čárami, viz Obrázek 1.

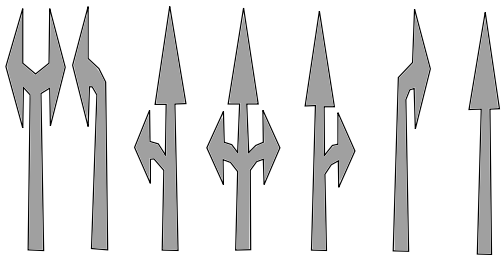
Obrázek 1: Šikmé rovnoběžné čáry

Dalším typem čáry, kterou velice často nalezneme na křižovatce, je přerušovaná čára. Tu nejčastěji nalezneme jako oddělující čáru pruhů klasické vozovky, ale i u připojovacích pruhů, řadicích pruhů, parkovacích pruhů, nebo například zastávkového pruhu pro městkou hromadnou dopravu.

Řadicí pruh je v většinou zakončen příčnou čarou souvislou. Tato čára vyznačuje, kde je nutno zastavit vozidlo za účelem dát přednost v jízdě [1].

2.1.4 Šipky na vozovce

Aby bylo možné snadno a jasně oddělit a označit řadicí pruhy vozovky, vyznačují se do nich směrové šipky. První šipky jsou malovány v dostatečné vzdálenosti, aby se řidič stihl zařadit do správného pruhu. Poslední šipky jsou malovány ve vzdálenosti 5 m od příčné čáry souvislé (vyskytuje-li se na konci řadicího pruhu). Celkový počet šipek v jednom řadicím pruhu je 3 až 5 a opakují se ve vzdálenosti 5 – 20 m, přičemž každá ze šipek má délku 5 m [1]. Nejvíce používané šipky, se kterými se můžeme na křižovatkách setkat jsou k vidění na Obrázku 2.



Obrázek 2: Nejčastější směrové šipky

2.1.5 Rušivé prvky

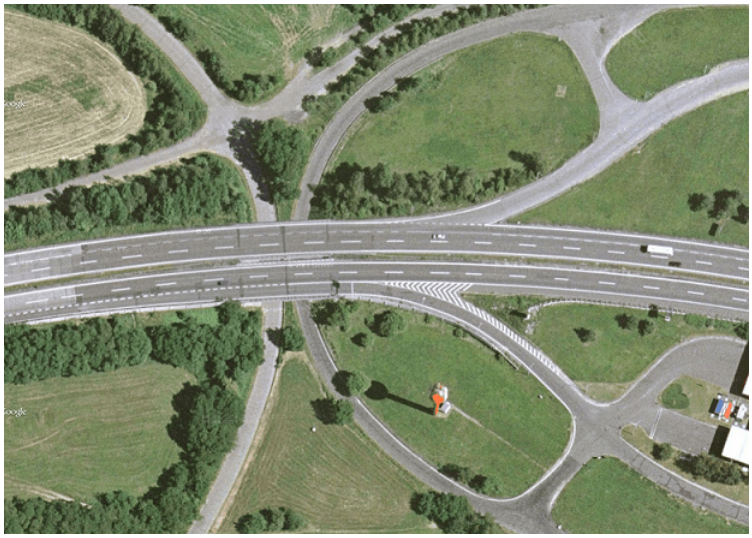
Z výše popsaných prvků křižovatky je zřejmé, že samotné značení křižovatek je velice různorodé a závislé na mnohých aspektech. Situaci nám nadále bude komplikovat velké množství rušivých elementů. Jelikož se tato práce zabývá rozpoznáváním křižovatek z reálných leteckých fotografií, musíme čelit nepříjemnostem, jako jsou stíny okolních budov a stromů, auta jedoucí v jízdních pruzích, lampy veřejného osvětlení zasahující do vozovky, ale i přechody pro chodce, nebo autobusové zastávky. Všechny tyto prvky budou negativně ovlivňovat použité algoritmy a bude proto potřeba na ně nějak reagovat.

2.1.6 Problémové křižovatky

Situaci navíc komplikuje mnoho dalších faktorů, které nám mohou zcela zamezit úspěšné rozpoznání křižovatky. Stojí-li například vedle křižovatky most, pravděpodobně nám skryje celé rameno křižovatky a dojde ke špatnému rozpoznání. Dalšími nevhodnými křižovatkami jsou víceúrovňové křižovatky a dálniční napojovací pruhy, které lze vidět na Obrázku 3. Tento druh křižovatek je pro úlohu nevhodný a rozpoznávání s největší pravděpodobností nebude úspěšné.

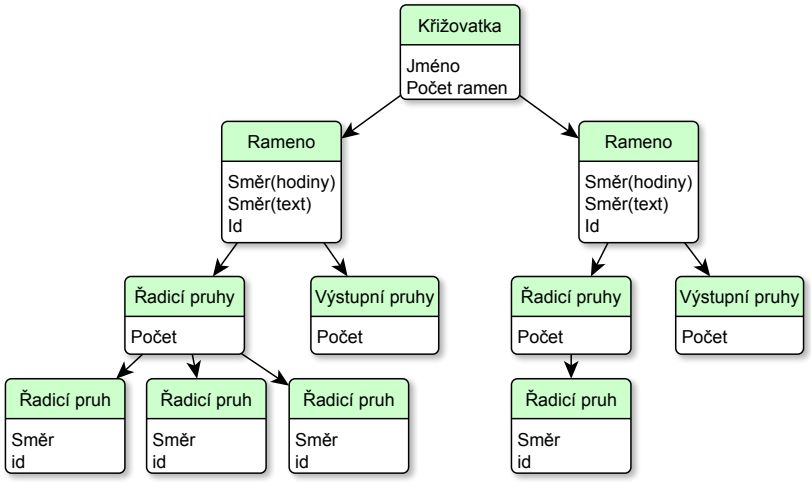
2.2 Struktura křižovatky

Z výše popsaných prvků křižovatky můžeme definovat základní dělení křižovatky pro její strukturní popis. Kořenový prvek v dekompozičním stromu tvoří samotná křižovatka, u níž potřebujeme znát její jméno a počet ramen, které ji tvoří. Dalším prvkem budou samotná ramena, u kterých budeme uchovávat jejich směr (jak v textové podobě tak ve vyjádření pomocí pozice směru na hodinách) a unikátní identifikační číslo. Každé rameno obsahuje řadicí a výstupní pruhy. Prvek výstupních pruhů představuje list dekompozičního stromu, jelikož nás u něj zajímá



Obrázek 3: Příklad problémové křižovatky

pouze počet výstupních pruhů. U řadicích pruhů potřebujeme znát jejich počet. Pro každý řadicí pruh pak vytvoříme vlastní prvek, který bude udávat směr pruhu a unikátní identifikační číslo pruhu. Příklad dekompozičního stromu křižovatky je nahlédnutí na Obrázku 4.



Obrázek 4: Příklad dekompozičního stromu křižovatky

3 Detekce okrajových čar vozovky

Již víme, že silnice I. třídy musí být ohraničeny okrajovými čárami o šířce 0,125 m (a větší) viz Kapitola 2.1.3. Podíváme-li se na okrajovou čáru vozovky na reálné fotografii, uvidíme souvislou linii bílých pixelů o definované šířce, která je určena přiblížením a rozlišením dané fotografie. K detekci těchto bílých pixelů se nabízí naivní přístup, tedy určení hranice pro barevné kanály R (červený), G (zelený) a B (modrý) každého pixelu. Pokud hodnoty jednotlivých barevných složek překročí určenou hranici, označíme bod za bílý. Pro lidské oko je ale nemožné rozdělit barvu pixelu do tří barevných složek. Museli bychom použít barevný filtr (ať už fyzikální či softwarový) pro každý z barevných kanálů a určit tak hranici, od které je pro daný kanál bod bílý. Filtrací ale zároveň bohužel ztrácíme představu o ostatních barevných složkách a daný pixel tak musíme zkoumat najednou s filtrem i bez filtru, abychom měli jistotu, že určená hranice odpovídá realitě. Na fotografiích, které jsou dostupné, je bohužel velice časté, že postranní čáry jsou staré a neúplné, nebo je překrývá stín okolních budov a automobilů. Z těchto důvodů se nabízí použít algoritmus, který nalezne hranice barevných kanálů za nás [2].

Jedná se tedy v podstatě o klasifikační problém, tedy úlohu ve které daný vstup na základě rozhodnutí klasifikátoru zařadíme do některé z alespoň dvou tříd. V našem případě se budeme snažit klasifikovat zkoumané pixely do dvou tříd: bílý a nebílý bod. Jednotlivé barevné složky pixelů, tedy data, se kterými budeme v algoritmu pracovat, nazýváme příznaky. Ty ve spojení vytváří příznakový prostor, který je trojrozměrný a každý rozměr tvoří právě jeden z barevných kanálů pixelu. Trénovací množinu prvků z příznakového prostoru lze snadno generovat uživatelem z reálných fotografií, například vytvořením jednoduchého programu, ve kterém bude uživatel určovat, zda je daný pixel součástí okrajové čáry nebo ne [2].

Ze základní analýzy úlohy víme, že podstatou řešení je nalezení hranice pro každý z barevných kanálů. Hledáme tedy funkci, která pro body s hodnotami barevných složek menšími než příslušné hranice bude klasifikovat tyto body jako nebílé a naopak většími jako bílé. Takových funkcí se dá nalézt mnoho. Pro jednoduchost řešení budeme navíc požadovat, aby funkce byla spojitá a šla snadno spočítat. Funkci potřebujeme definovat v třírozměrném příznakovém prostoru. Optimální volbou bude jedna z logistických funkcí, jelikož splňuje všechny zmíněné požadavky. Výběr takovéto funkce nás vede na logistickou regresi, která přímo vychází z lineární regrese [2].

3.1 Lineární regrese

Lineární regrese je pro velkou část problémů umělé inteligence špatně aplikovatelná a pro rozpoznávání většinou nevhodná. Díky své jednoduchosti založené na predikci hodnot ale představuje základní stavební prvek dalších regresních postupů [2]. Aby bylo pro čtenáře snadnější porozumět postupům používaným v logistické regresi, která z lineární regrese přímo vychází, budou na lineární regresi vysvětleny jednotlivé fundamentální techniky regresních metod.

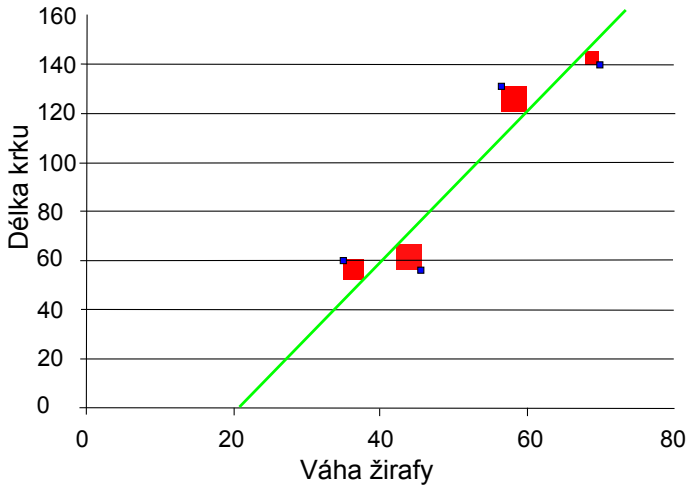
Lineární regrese je algoritmus umělé inteligence pro strojové učení se s učitelem, který slouží k předvídání hodnot, které nebyly obsaženy v trénovací množině (viz níže). Jelikož se jedná o

regresní úlohu, bude jejím výstupem spojitá funkce, která bude predikovat vývoj příznaků, které se nenacházely v trénovací množině (tzn. množina příznaků, kterou měl algoritmus k dispozici na začátku svého učení). Učitel musí pro každý příznak z množiny příznaků dodat „správnou" odpověď [2]. Příklad trénovací množiny je k nahlédnutí v Tabulce 1.

Váha žirafy	Délka krku
56	132
68	140
35	60
45	57

Tabulka 1: Příklad trénovací množiny lineární regrese

Tabulka zachycuje množinu příznaků s jednorozměrným příznakovým prostorem, ve kterém jediný příznak představuje váhu žirafy. Informací učitele je délka krku žirafy. Na tato data se vlastně můžeme dívat jako na souřadnice bodů v 2D prostoru, kdy váha žirafy bude představovat souřadnici x a délka krku bude představovat souřadnici y. Úkolem lineární regrese je tyto data aproximovat nejvhodnější (tzn. s minimální kvadratickou chybou) lineární funkcí (viz Obrázek 5) tak, abychom pro každou další váhu žirafy dokázali předpovědět délku jejího krku porovnáním s funkcí.



Obrázek 5: Proložení příznakového prostoru funkcí lineární regrese

V případě jednorozměrného příznakového prostoru je tedy hledaná funkce přímka s obecným předpisem $f(x) = C_0 + C_1 x$. Tuto funkci nazýváme hypotézou, jelikož se jedná o reprezentaci predikce nových hodnot. Hypotéza představuje rozhodovací funkci lineární regrese a formálně ji zapíšeme jako [2]:

$$h_\Theta(x) = \Theta_0 + \Theta_1 x, \tag{1}$$

kde Θ_0, Θ_1jsou parametry hypotézy a x představuje lineární regresi jedné proměnné. Naší další snahou tedy bude nastavit parametry hypotézy tak, aby hodnoty predikované hypotézou pro příznaky $x^{(i)}$(tedy příznak na pozici i v příznakové množině) byly co nejblíže hodnotám učitele

$y^{(i)}$(tedy odpověď učitele na příznak na pozici i v množině příznaků). Hledáme vlastně metriku, která bude vyjadřovat, jak dobré je aktuální řešení. Nabízí se metoda nejmenších čtverců, tedy metoda, která počítá sumu kvadrátu vzdálenosti všech bodů z trénovací množiny od aktuálního tvaru hypotézy. Grafické znázornění metody můžeme vidět na Obrázku 5, kde jsou modře vyznačeny jednotlivé příznaky, červeně velikost kvadratické chyby a zeleně aproximační funkce lineární regrese. Metriku budeme nazývat cenová funkce a definujeme ji jako [2]:

$$J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 \tag{2}$$

Cenovou funkci $J(\Theta_0, \Theta_1)$ se snažíme minimalizovat. Jak je známo, minimum funkce lze spočítat z derivace funkce. Optimalizaci cenové funkce můžeme provést numericky pomocí techniky gradientního sestupu, k čemuž je třeba vyjádřit parciální derivace pro jednotlivé parametry hypotézy [2]:

$$\frac{\partial J(\Theta_0, \Theta_1)}{\partial \Theta_{0..1}} = \frac{\partial}{\partial \Theta_{0..1}}\frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \Theta_0}\frac{1}{2m}\sum_{i=1}^{m}(\Theta_0 + \Theta_1 x^{(i)} - y^{(i)})^2, \tag{3}$$

tedy pro jednotlivé parametry:

$$\frac{\partial J(\Theta_0, \Theta_1)}{\partial \Theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)}) \tag{4}$$

$$\frac{\partial J(\Theta_0, \Theta_1)}{\partial \Theta_1} = \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \tag{5}$$

Doposud jsme se zabývali pouze popisem jednoduchého případu lineární regrese s jednorozměrným příznakovým prostorem. V případě vícerozměrného příznakového prostoru se naštěstí situace příliš nemění. Tvar hypotézy se rozšiřuje o další parametr Θ_i pro každou novou proměnnou x_i. Předchozí příklad trénovací množiny příznaků zobrazený v Tabulce 1 by ve vícerozměrném příznakovém prostoru mohl vypadat například jako v Tabulce 2.

Váha žirafy = x_1	Věk žirafy = x_2	...	Počet žiraf ve stádě = x_n	Délka krku
56	8	...	8	132
68	12	...	6	140
35	2	...	7	60
45	4	...	5	57

Tabulka 2: Příklad trénovací množiny lineární regrese ve vícerozměrném příznakovém prostoru

Hypotézu lineární regrese vícerozměrného příznakového prostoru můžeme pak snadno vyjádříme úpravou Vzorce 1 přidáním dalších parametrů Θ_i a příslušných proměnných x_i. Hypotézu pak vyjádříme jako:

$$h_\Theta(x) = \Theta_0 + \Theta_1 x_1 + ... + \Theta_n x_n \tag{6}$$

Dodefinováním prvku $x_0 = 1$ pro všechny příznaky z příznakové množiny pak můžeme tvar hypotézy zapsat jako skalární součin vektorů:

$$h_\Theta(\boldsymbol{x}) = \boldsymbol{\Theta}^T \boldsymbol{x} \tag{7}$$

Cenová funkce se pro lineární regresi vícerozměrného příznakového prostoru v podstatě nemění, jen ji upravíme pro vektorový zápis:

$$J(\boldsymbol{\Theta}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\Theta(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \tag{8}$$

Konečně parciální derivace jednotlivých parametrů budou vypadat následovně [2]:

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \Theta_j} = \frac{\partial}{\partial \Theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\Theta_0 x_0^{(i)} + \Theta_1 x_1^{(i)} + ... + \Theta_n x_n^{(i)} - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^{m} (h_\Theta(\boldsymbol{x}^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \tag{9}$$

3.2 Logistická regrese

Logistická regrese je algoritmus umělé inteligence pro strojové učení se s učitelem. Na rozdíl od lineární regrese není smyslem logistická regrese predikce hodnot pro neznámé prvky, ale binární klasifikace. Jejím výstupem je tedy odpověď Ano/Ne. Na výstup logistické regrese můžeme také pohlížet, jako na pravděpodobnost, s kterou patří nový prvek do dané třídy. Hypotéza tedy neslouží k odhadu nové hodnoty, ale k vytyčení hranice v příznakovém prostoru a tím jeho rozdělení na dva podprostory.

3.2.1 Rozhodovací hranice

Jako rozhodovací hranice nám poslouží logistická funkce. Obecnou logistickou funkci můžeme zapsat jako [2]:

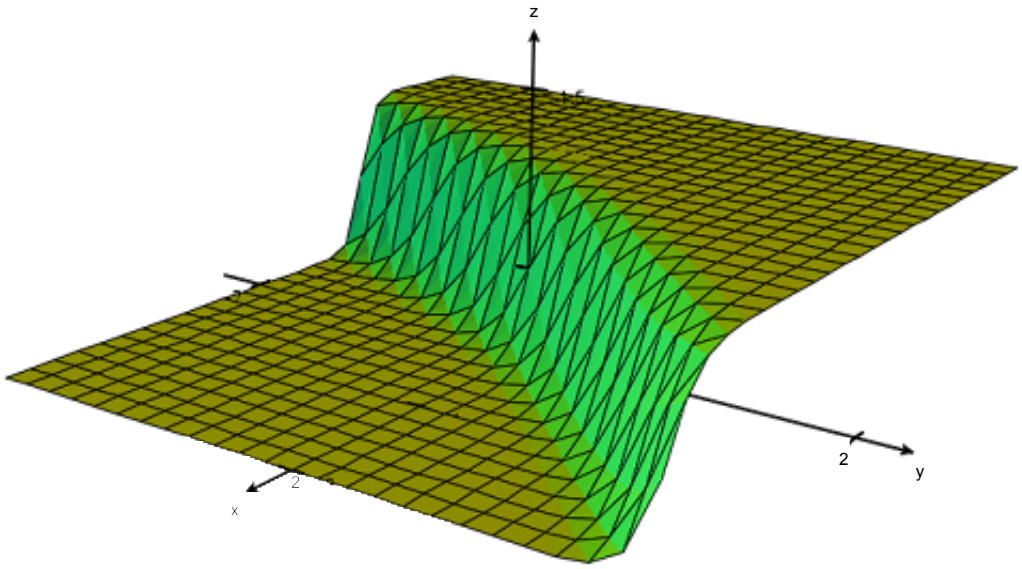
$$f(t, a, m, n, T) = \frac{a * (1 + m * e^{-\frac{t}{T}})}{1 + n * e^{-\frac{t}{T}}} \tag{10}$$

Takovýchto funkcí existuje nekonečně mnoho. Aby naše řešení bylo jednoduché a přitom efektivní, zvolíme si vhodně jednotlivé parametry tak, aby výsledná logistická funkce byla sigmoida, kterou posléze aplikujeme na příznakový prostor. Volíme tedy parametry: a = 1, m = 0, n = 1, T = 1 a $t = \boldsymbol{\Theta}^{\mathrm{T}} \boldsymbol{x}$ a získáváme předpis:

$$f(\boldsymbol{\Theta}^{\mathrm{T}} \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\Theta}^{\mathrm{T}} \boldsymbol{x}}} \tag{11}$$

Tato funkce bude základním prvkem algoritmu logistické regrese a budeme se snažit její úpravou na základě trénovacích dat získat co nejlepší rozhodovací hranici v příznakovém prostoru.

Podobu možné rozhodovací hranice v příznakovém prostoru můžeme vidět na Obrázku 6.



Obrázek 6: Sigmoida v 3D prostrou.

Úlohu logistické regrese můžeme zapsat jako:

$$y = h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \epsilon \langle 0, 1 \rangle \tag{12}$$

kde y je třída, do které klasifikujeme a $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$ je hypotéza, tedy funkce, jejíž tvar rozhoduje o klasifikaci prvku. Tato funkce je výše popsaná sigmoida. Platí tedy vztah [2]:

$$h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = f(\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x}}} \tag{13}$$

Funkce $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$ pak představuje odhad pravděpodobnosti, že modelovaná veličina nabude hodnoty 1 (jev nastane, nebo nenastane), pokud je na vstupu vektor příznaků $\boldsymbol{x}$. Definiční obor funkce je $(-\infty, \infty)$ a obor hodnot je $\langle 0; 1 \rangle$. Po natrénování funkce můžeme nový prvek klasifikovat pouhým porovnáním součinu vektorů $\boldsymbol{\Theta}$ a $\boldsymbol{x}$, jelikož platí vztah [2]:

$$\begin{aligned} pro\ y = 1\ &:\ h_{\boldsymbol{\Theta}}(\boldsymbol{x}) \geq 0.5\ , tj.\ f(\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x}) \geq 0.5\ , tj.\ \boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x} \geq 0 \\ pro\ y = 0\ &:\ h_{\boldsymbol{\Theta}}(\boldsymbol{x}) < 0.5\ , tj.\ f(\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x}) < 0.5\ , tj.\ \boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x} < 0 \end{aligned} \tag{14}$$

Obdobně jako v případě lineární regrese je nutné doplnit pro všechny prvky z příznakové množiny příznak $x_0 = 1$, aby bylo možné provádět vektorové operace popsané výše a přitom nebyla ovlivněná podoba výsledné hypotézy.

3.2.2 Cenová funkce

Abychom byli schopni funkci $h_{\boldsymbol{\Theta}}(\boldsymbol{x})$ natrénovat tak, aby co nejlépe odpovídala datům z trénovací množiny a klasifikace nových dat proběhla s co nejmenší chybou, budeme potřebovat definovat iterační algoritmus, který na základě vstupních dat upraví parametry vektoru $\boldsymbol{\Theta}$. Zároveň budeme potřebovat nějakou metriku, která bude určovat, jak dobré je aktuální nastavení. Je tedy potřeba najít cenovou funkci $Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), \boldsymbol{y})$. Pro určení této funkce vyjdeme ze známého tvaru cenové funkce lineární regrese vysvětleného v Kapitole 3.1:

$$J(\boldsymbol{\Theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_{\Theta}(\boldsymbol{x}^{(i)}) - y^{(i)})^2, \tag{15}$$

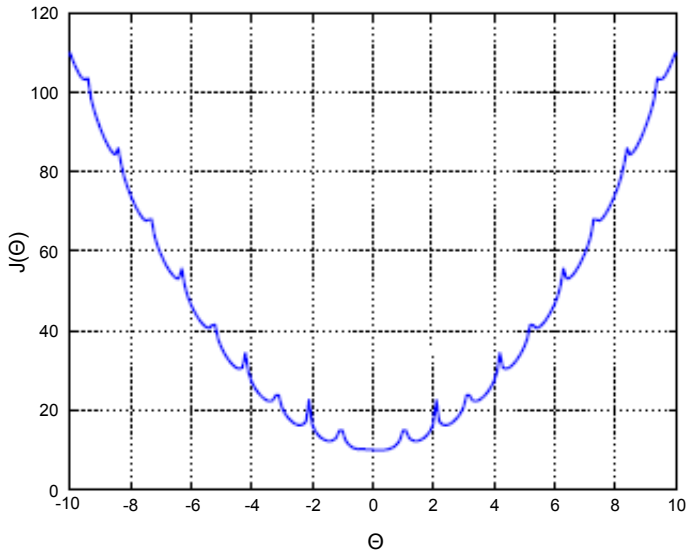
použijeme substituci [2]:

$$Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y) = \frac{1}{2}(h_{\boldsymbol{\Theta}}(\boldsymbol{x}) - y)^2, \tag{16}$$

a získáváme:

$$J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=0}^{m} Cost(h_{\boldsymbol{\Theta}}(x_i), y_i) \tag{17}$$

Kdybychom ale tento tvar chtěli rovnou použít, narazíme na problém. Následný gradientní sestup by v tomto případě nefungoval, jelikož funkce $J(\boldsymbol{\Theta})$ není konvexní a šance nalezení globálního optima je minimální [2]. Průběh cenové funkce lze vidět na Obrázku 7.



Obrázek 7: Průběh cenové funkce logistické regrese. Zdroj [2]

Je tedy potřeba definovat cenovou funkci jinak. Cenovou funkci na Obrázku 7 se můžeme pokusit nahradit jinou funkcí, která by měla lepší průběh. Nejvíce vyhovující cenovou funkcí se jeví logaritmické funkce – jedna pro každou klasifikaci. Výsledná cenová funkce tedy vznikne spojením dvou logaritmických funkcí s tímto zápisem [2]:

$$\begin{aligned} pro\ y = 1\ :\ Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y) &= -log(h_{\boldsymbol{\Theta}}(\boldsymbol{x})) \\ pro\ y = 0\ :\ Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y) &= -log(1 - h_{\boldsymbol{\Theta}}(\boldsymbol{x})) \end{aligned} \tag{18}$$

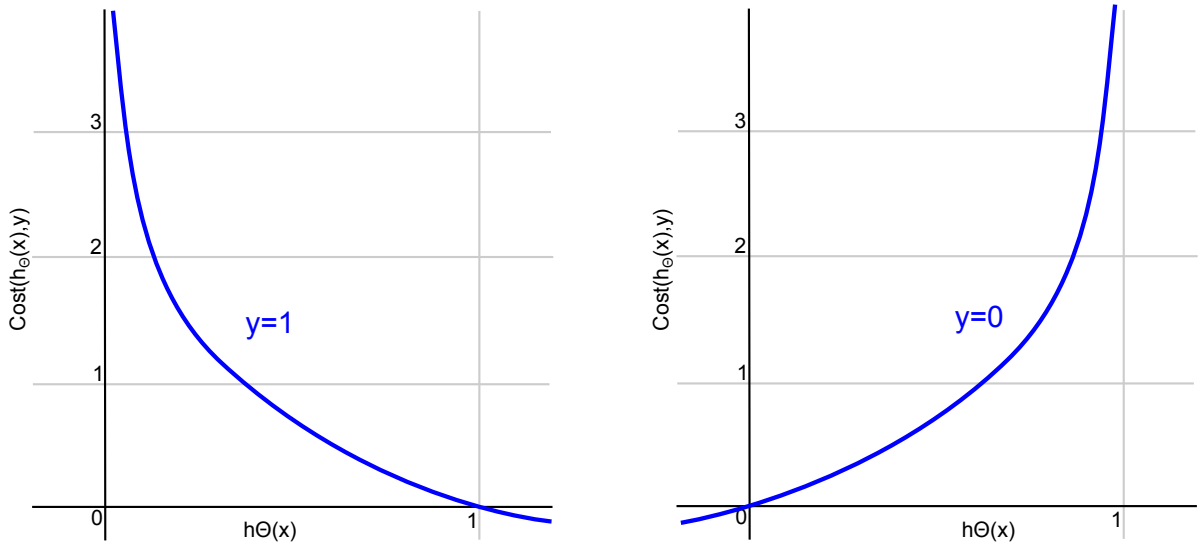
jelikož vždy platí $y\epsilon\{0;1\}$, můžeme psát zjednodušeně:

$$Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}), y) = -y \cdot log(h_{\boldsymbol{\Theta}}(\boldsymbol{x})) - (1 - y) \cdot log(1 - h_{\boldsymbol{\Theta}}(\boldsymbol{x})) \tag{19}$$

Celkovou cenu aktuálního nastavení parametrů vektoru $\boldsymbol{\Theta}$ tedy spočteme jako[2]:

$$J(\boldsymbol{\Theta}) = \frac{1}{m} \sum_{i=0}^{m} [-y_i \cdot log(h_{\boldsymbol{\Theta}}(x_i)) - (1 - y_i) \cdot log(1 - h_{\boldsymbol{\Theta}}(x_i))] \tag{20}$$

Výsledný průběh cenové funkce je zobrazen na Obrázku 8.



Obrázek 8: Průběh nové cenové funkce. Zdroj [2]

Tento tvar cenové funkce je známý z metody maximální věrohodnosti a je konvexní (můžeme tedy bez problémů použít metodu gradientního sestupu) [2].

3.2.3 Trénování algoritmu

Jelikož už známe metriku, která určuje, jak dobré je naše řešení, trénování algoritmu bude spočívat v minimalizaci této metriky. Minimalizaci cenové funkce $J(\boldsymbol{\Theta})$ provedeme metodou gradientního sestupu, k čemuž potřebujeme vyjádřit parciální derivace k výpočtu jednotlivých kroků [2]. Jejich odvození vychází z odvození cenové funkce a získáváme:

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \Theta_j} = \frac{1}{m} \sum_{i=0}^{m} Cost(h_{\boldsymbol{\Theta}}(\boldsymbol{x}^{(i)}), y^{(i)}) * x_j^{(i)} \tag{21}$$

Výsledný zápis trénování algoritmu lineární regrese znázorňuje Algoritmus 1 [2].

Algoritmus 1 Logistická regrese

```
while not_converged() do {
        for j=0..n do {
                theta[j] = theta[j] - alpha * (1/m) * sigma(j);
        }
}

...

double sigma(int j) {
        double sum = 0;
        for i = 0..m do {
                sum += (sigmoid(x[i])-y[i]))*x[i][j];
        }
        return sum;
}
```

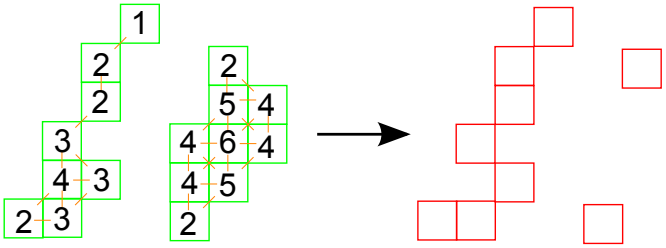
Význam proměnných:
m – počet příznaků
x – matice příznaků
y – vektor hodnocení učitele. i-té hodnocení odpovídá i-tému příznaku z matice příznaků
n – velikost vektoru theta
alpha – koeficient konvergence (viz níže)
sigmoid(x[i]) – funkce, která vypočte předem zmíněný vztah: $h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{x}}}$.

Alpha je koeficient konvergence a určuje, o jak velkou vzdálenost se při gradientním sestupu posunujeme [2]. Pokud se koeficient zvolí příliš malý, konvergence bude probíhat velice pomalu. Pokud se naopak zvolí příliš velký, cenová funkce bude divergovat, nebo budeme oscilovat poblíž globálního optima, ale nikdy ho nedosáhneme. Je proto důležité zvolit tento parametr opatrně. Jakmile ale najdeme vhodné nastavení koeficientu alfa, není již potřeba jej měnit v průběhu běhu programu, jelikož gradient funkce (a tím i posuv ovlivněný koeficientem konvergence) se zmenšuje, čím blíže minimu se nacházíme.

3.3 Detekce čar

Výše popsané regresní postupy nám pomohou detekovat bílé body na vozovce. Běžná dopravní komunikace a křižovatka je ale plná bílých objektů, jako jsou automobily, značení autobusových zastávek, směrové šipky, lampy veřejného osvětlení aj. Ve výsledku zřejmě budou klasifikovány i takové pixely, které nenáleží žádné okrajové nebo dělící čáře na vozovce. Je tedy nutné zamyslet se nad vhodnou filtrační metodou. Jeden z možných přístupů je klasifikovat zkoumanou

křižovatku po pixelových okénkách, místo toho, aby se klasifikoval každý pixel zvlášť. Pixelová okénka obsahují vždy $n \times n$ pixelů a klasifikována je vždy průměrná hodnota barevných složek všech pixelů okénka. Pokud tedy velikost okénka nastavíme stejnou, jako je velikost okrajové čáry vozovky, můžeme následnou filtraci klasifikovaných bodů provést poměrně jednoduše, dle porovnávání počtu sousedních okének. Pro každé okénko klasifikovaných bodů zjistíme počet sousedních klasifikovaných okének. Pokud tento počet přeroste určenou mez s, není klasifikované okénko součástí čáry na silnici, ale s větší pravděpodobností je součástí automobilu, přechodu pro chodce či jiného velkého bílého objektu. Ukázku takovéto filtrace můžeme vidět na Obrázku 9. Zelené čtverce představují všechna klasifikovaná pixelová okénka a jsou ohodnoceny počtem sousedních okének. Sousedství pro úplnost znázorňují oranžové úsečky. V pravé části obrázku je filtrovaná množina pixelových okének s nastavením parametru $s = 3$. Z obrázku je patrné, že problémem tohoto přístupu je nastavení šířky okénka a parametru s.



Obrázek 9: Příklad filtrace bodů pro pixelová okénka s nastavením parametru $s = 3$.

3.3.1 Rekurzivní detekce

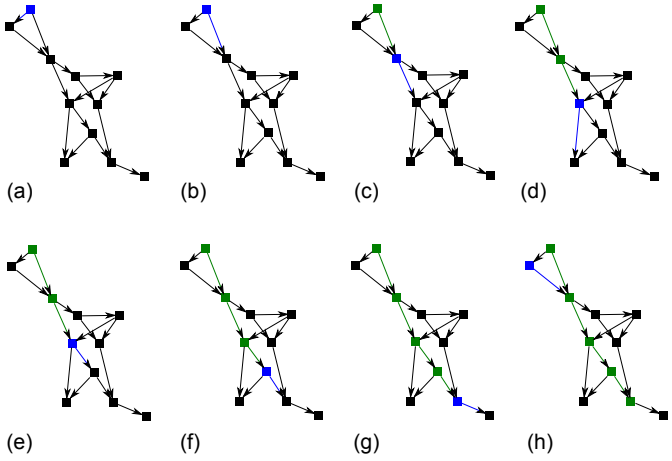
Po úspěšné detekci bílých bodů a jejich filtraci máme k dispozici množinu bodů, z nichž velká část je součástí okrajových nebo dělících čar vozovky a jen malá část je pozůstatkem ostatních bílých objektů. Naším dalším úkolem bude vytvořit z této množiny okrajové a dělící čáry na vozovce.

Jedním z přístupů detekce čar z bílých bodů je rekurzivní metoda založená na sledování vlastností bílého bodu. Zajímat nás bude vždy poloha bodu a množina normovaných směrových vektorů vedoucích ze zkoumaného bodu do bodů sousedních v blízkém okolí. Blízkým okolím rozumíme kruh o poloměru r pixelů se středem ve zkoumaném bodě. Výpočet normovaných směrových vektorů budeme provádět vždy z levé horní pozice, takže směrové vektory budou směřovat vždy k sousedům ležícím vpravo a dole). Po určení množiny směrových vektorů $S\epsilon(\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$ pro každý bod budeme porovnávat zkoumaný bod pro každý jeho směrový vektor $\boldsymbol{s}_i$ se všemi sousedními body v blízkém okolí. Nalezneme tedy sousední bod splňující následující podmínky:

1. Sousední bod obsahuje normovaný vektor $\boldsymbol{s}_j$ blízký zkoumanému normovanému směrovému vektoru $\boldsymbol{s}_i$.
2. Sousední bod v kontextu se směrovým vektorem $\boldsymbol{s}_j$ není součástí dříve detekované čáry.
3. Poloha sousedního bodu leží v blízkosti přímky tvořené polohou zkoumaného bodu a směrového vektoru $\boldsymbol{s}_i$.

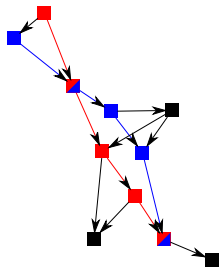
Pak do tohoto bodu přesouváme rekurzivní výpočet a opakujeme postup vyhledávání vhodného souseda. V momentě, kdy už žádný další vhodný bod nenajdeme, vracíme se zpět do prvotně zkoumaného bodu a celý proces opakujeme pro další směr. Průběh rekurzivního algoritmu je znázorněn na Obrázku 10. Algoritmus začíná v (a) v modrém bodě a hledá souseda dle podmínek zmíněných výše ve směru směrového vektoru vyznačeného modrou šipkou. Žádného vhodného souseda nenachází, proto hledá ve směru dalšího vektoru (b). Sousední bod vyhovuje podmínkám, zkoumaný bod v daném směru přiřadíme čáře (zelená barva) a vyhledávání přesouváme do nalezeného souseda (c). Opět byl nalezen vhodný sousední bod, přiřazujeme ho k rozpoznané čáře a přesouváme hledání (d). Danému směrovému vektoru nevyhovuje žádný soused, měníme směrový vektor (e)(f). Byl nalezen další sousední bod, ale nevlastní již žádného souseda, který by splňoval podmínky zmíněné výše. Přesouváme se proto do prvotního bodu. Ten už ale nemá žádný směrový vektor, takže algoritmus pokračuje do dalšího bodu, kde se bude snažit nalézt novou čáru (h).

Pozn.: Jeden bod může být součástí více čar najednou, jelikož jeho příslušnost čáře vychází z dvojice vlastností poloha a směrový vektor. Bod tedy může být součástí tolika čar, jaká je velikost jeho množiny směrových vektorů S.



Obrázek 10: Příklad průběhu rekurzivního algoritmu.

Výhodou algoritmu je schopnost přizpůsobit se zahnutým čarám libovolné délky. Velkou nevýhodou je ale samotná rekurze. Jelikož není předem možné odhadnout počet bílých klasifikovaných bodů, je velmi těžké omezit rekurzivní algoritmus zastavovací podmínkou a hrozí tak velká časová a výpočetní náročnost. Při nevhodném nastavení parametru s může navíc dojít k detekci velkého počtu sousedních bodů s podobnou pozicí a směrovým vektorem. Kromě vysoké výpočetní náročnosti hrozí i mnohonásobná detekce čáry (viz Obrázek 11) a je proto nutné výsledky algoritmu dále procházet a upravovat. Metoda navíc detekuje velké množství malých množin bodů, které ve skutečnosti tvoří velice krátké čáry (například přerušovaná dělicí čára). Proto je potřeba průběžně počítat vzdálenost každé čáry (suma vzdáleností mezi body) a brát v úvahu jen rozumně dlouhé čáry.



Obrázek 11: Problém mnohonásobné detekce rekurzivního algoritmu, stejná čára byla detekována dvakrát.

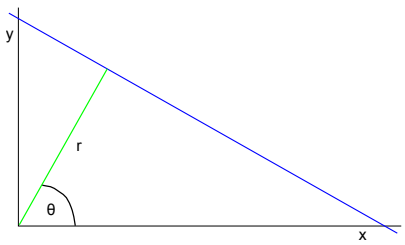
3.3.2 Metoda vzdálenosti od přímky

Je zřejmé že metoda rekurzivní detekce není pro aplikaci úplně vhodná, můžeme ji ale modifikovat tak, abychom získali lepší kontrolu nad algoritmem pomocí využití rovnice přímky. Základní postup je naprosto shodný s rekurzivní metodou. Pro každou čáru (tedy zkoumaný bod a směrový vektor $\boldsymbol{s}_i$) spustíme rekurzivní algoritmus vysvětlený výše s tím rozdílem, že stanovíme zastavovací podmínku rekurze, čímž zajistíme větší rychlost a kontrolu nad celým procesem. Pokud se nám podaří rekurzivní metodou přiřadit k čáře dostatečné množství bodů stanovené zastavovací podmínkou, zprůměrujeme směrové vektory nalezených bodů a jejich polohy. Z průměrného směrového vektoru a průměrné polohy vyjádříme rovnici přímky. Nakonec projdeme všechny zbývající detekované body, u kterých zjišťujeme jejich vzdálenost od vypočtené přímky. Pokud nalezneme dostatečné množství bodů, které této přímce odpovídají, pak jsme s vysokou pravděpodobností nalezli čáru na vozovce. Tento postup však již nepřipouští zahnuté čáry.

3.3.3 Houghova transformace

Zcela jinak k detekci čar přistupuje metoda využití Houghovy transformace k detekci čar [3]. Touto metodou se dají detekovat pouze přímé čáry a musí jí předcházet detekce hran. K tomu můžeme použít například Cannyho hranový detektor, jehož vysvětlení může čtenář nalézt v [4].

Houghova transformace je postup extrakce příznaků, který pracuje s polárními souřadnicemi pro vyjádření přímky. Parametry určující podobu přímky jsou r neboli výška trojúhelníku, který tvoří přímka s osami souřadnic, a θ neboli úhel, který svírá výška r a osa x. Význam těchto parametrů je graficky znázorněn na Obrázku 12.



Obrázek 12: Grafické znázornění polárních souřadnic pro vyjádření přímky

Abychom mohli vyjádřit obecnou rovnici přímky v polárních souřadnicích, vyjdeme z předpisu přímky v kartézských souřadnicích. Rovnici přímky v polárních souřadnicích pak snadno odvodíme následujícím postupem [3]:

$$\begin{gathered} y = a \cdot x + b \\ y = x \cdot tg\theta + b \\ y = \left(-\frac{cos\theta}{sin\theta}\right) x + \left(\frac{r}{sin\theta}\right) \\ y \cdot sin\theta = -x \cdot cos\theta + r \\ r = x \cdot cos\theta + y \cdot sin\theta \end{gathered} \tag{22}$$

Pro každý bod (x_0, y_0) pak můžeme vykreslit průběh vztahu $r_\theta = x_0 \cdot cos\theta + y_0 \cdot sin\theta$ do grafu na osách r a θ, čímž získáme sinusoidu. Jako definiční obor uvažujeme $D\epsilon\,(0, 2\pi)$ a jako obor hodnot $H\epsilon\,(0, \infty)$. Tímto způsobem vykreslíme všechny sinusoidy a budeme hledat průsečíky těchto křivek. Mějme sinusoidy i a j. Pokud nalezneme průsečík těchto sinusoid, znamená to, že body (x_{i0}, y_{i0}) a (x_{j0}, y_{j0}) leží na stejné přímce. Hodnoty průsečíku (θ, r) představují parametry rovnice přímky, na které body leží [3]. Čím více tedy získáme průsečíků v jednom bodě, tím více bodů leží v dané přímce. Dokážeme tak detekovat přímky a určit jejich význam na základě počtu křivek procházejících daným průsečíkem. Klíčovým krokem tohoto algoritmu je nastavení takového prahu počtu bodů náležících přímce, abychom mohli detekovanou přímku považovat za dostatečně významnou.

4 Detekce šipek na vozovce

Velice důležitými prvky křižovatky jsou směrové šipky kreslené na vozovku, jejichž úspěšnou detekcí bychom získali mocný nástroj pro rozpoznávání křižovatky. Jelikož je tvar směrových šipek jasně daný normou (viz Obrázek 2), můžeme se pokusit zaměřit na specifické vlastnosti každé ze šipek. Těmito vlastnostmi zřejmě budou ostré hrany šipek a jasně definované úhly mezi obrysovými čarami šipky. Podíváme-li se na tyto vlastnosti v širším kontextu, bude jejich výčet vždy definovat právě jednu ze známých šipek. Budeme tedy hledat klíčové body šipek, ve kterých jsou tyto vlastnosti nejvíce patrné a budeme je tak moci přímo porovnávat s klíčovými body nalezenými ve vzorových šipkách. Vzorovými šipkami rozumíme množinu šipek získanou z reálných fotografií.

4.1 Porovnávání klíčových bodů

Jak bylo popsáno výše, naším cílem je hledat takové body, které obsahují rohy, obliny, křížení hran atd. Důležitou vlastností rozpoznávacího algoritmu musí zřejmě být opakovatelnost, jelikož je potřeba najít stejný klíčový bod za různých podmínek (zdrojový a porovnávaný obraz mohou mít jinou kvalitu, jiné nasvícení vozovky, jinak pootočenou nebo jinak velikou směrovou šipku). Takový bod je následně popsán deskriptorem, tedy vícerozměrným vektorem popisujícím jednotlivé vlastnosti bodu [7]. Deskriptory bodů pak můžeme snadno porovnávat s dalšími deskriptory nalezenými v jiných obrázcích počítáním Euklidovské vzdálenosti vektorů. Níže budou popsány dvě různé metody (SIFT a SURF) pro detekci a porovnávání klíčových bodů.

4.1.1 SIFT

Algoritmus SIFT (Scale Invariant Feature Transform) je invariantní vůči škálování, osvětlení, rotaci a pokřivení hledaného obrazu a byl vymyšlen roku 2004 Davidem Lowem [5]. Algoritmus využívá teorie škálovatelného prostoru , která částečně vychází z biologického vnímání vizuální informace živočichů a která napomáhá úspěšně pracovat s rozpoznáváním obrazu. Její hlavní podstata vychází z faktu, že předměty reálného světa jsou tvořeny různými objekty různých velikostí, takže v různém rozlišení jsou vždy patrné jiné objekty. Rozpoznané klíčové body jsou tedy přímo závislé na blízkosti pozorování. Budeme-li mít fotografii zachycující křižovatku s rozlišením v metrech, budeme schopni rozpoznat jednotlivé automobily, postranní čáry a další. Bude-li však fotografie s rozlišením v řádu desítkách metrů, budou se automobily jevit pouze jako body a vozovka jako křivka. V úloze rozpoznávání obrazu je však téměř nemožné určit, které škálování je vhodné pro hledání klíčových bodů v daném obrazu, bez předchozí znalosti doplňujících informací (které ale většinou k dispozici nemáme). Abychom tedy byli schopni detekovat požadované klíčové body, musíme provést detekci v různých škálových úrovních. Jak bylo dokázáno pány Koendrikem a Lindebergem, jedinou možnou funkcí pro škálování prostoru je funkce Gaussova rozmazání [6].

Škálovaný prostor je pak definovaný jako funkce $L(x, y, \sigma)$, která je výsledkem konvoluce Gaussovy funkce G a původního zobrazení I [5]:

$$L(x, y, \sigma) = G(x, y, \sigma) \star I(x, y), \tag{23}$$

kde σ je rozptyl Gaussovy funkce G. Gaussovu funkci vypočteme jako:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{24}$$

Tento proces je následně aplikován na nové zobrazení a tím získáváme několik zobrazení v různém škálování. Abychom byli schopni úspěšně detekovat pozice klíčových bodů, využijeme funkci hledání maxim z rozdílu Gaussových funkcí D (DOG – Difference of Gaussian). Rozdíl spočteme ze dvou blízce škálovaných obrazů násobením jedné složky konstantou k:

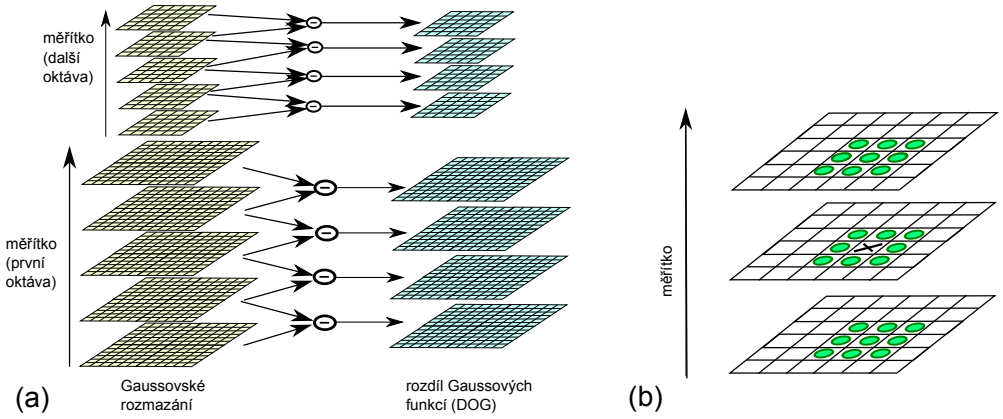
$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \star I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{25}$$

Výše popsanou úpravu zachycuje Obrázek 13.

Obrázek 13: Originální obrázek, obrázek upravený Gaussovo funkcí, rozdíl Gaussových funkcí (DOG)

Množinu obrazů stejného rozlišení s různými stupni rozmazání nazýváme oktávou. Dle doporučení autora algoritmu by se v oktávě mělo provádět 5 různých rozmazání obrazu [5]. Tím pro jednu oktávu získáme 4 různé DOG. Pro zisk další oktávy obrázek zmenšíme na polovinu a celý postup opakujeme.

Pro získání klíčových bodů je nutné lokalizovat maximum a minimum pro každý bod v DOG. Každý bod je porovnáván se všemi svými sousedy, ale i s blízkými body ve vyšší a nižší škálované vrstvě (celkově 26 porovnání). Pokud je bod extrémem, pak jsme nalezli klíčový bod, který je nejlépe rozpoznatelný v daném škálování. Struktura oktáv, DOG a lokalizace maxim je zobrazena na Obrázku 14.



Obrázek 14: (a) Znázornění oktáv a příslušných DOG, (b) Lokalizace extrémů pixelu. Zdroj [5]

Abychom byli schopni získat polohu klíčových bodů s větší přesností, než jsou pixely, aplikujeme na nalezené body Taylorův rozvoj druhého řádu [5]:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x, \tag{26}$$

a tento výraz následně minimalizujeme, abychom získali skutečnou polohu extrému [5]:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}, \tag{27}$$

kde $x = (x, y, \sigma)^T$ představuje odchylku od nalezeného kandidáta na klíčový bod a $\hat{x}$ je pozice extrému.

Množina nalezených bodů obsahuje velké množství nevhodných bodů (body nacházející se na hranách, body s nízkým kontrastem), které z ní potřebujeme vyřadit. Body s nízkým kontrastem odstraníme pomocí vztahu [5]:

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{28}$$

Pro odstranění bodů na hranách využijeme Hessovské matice $\boldsymbol{H}$, což je čtvercová matice druhých parciálních derivací [5]:

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{29}$$

K odstranění využijeme vlastnosti, že hlavní křivost musí být o mnoho větší při pohledu napříč hranou, než při pohledu podél hrany. Body tedy z množiny vyřadíme dle nerovnosti [5]:

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} = \frac{D_{xx+D_{yy}}}{D_{xx}D_{yy} - (D_{xy})^2} < \frac{(r+1)^2}{r}, \tag{30}$$

kde r představuje práh křivosti.

Aby byla zajištěna invariantnost rotace, musíme pro všechny body spočítat přiřazení orientace. K tomu využijeme zobrazení $L(x, y, \sigma)$ dané pro klíčový bod v příslušném rozmazání. Následně spočítáme velikost gradientu m a orientaci gradientu θ dle následujících vztahů:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \tag{31}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{(L(x, y+1) - L(x, y-1))}{(L(x+1, y) - L(x-1, y))}\right) \tag{32}$$

V oblasti okolo klíčového bodu pak vytvoříme histogram o 36 sloupcích (určující stupně od 10° do 360° s krokem 10°), které ohodnotíme zváženou velikostí gradientu m. Orientace bodu je pak přiřazena dle největšího sloupce. Obsahuje-li nějaký další sloupec alespoň 80% velikosti největšího sloupce, pak pro danou orientaci vytvoříme nový klíčový bod [5].

Nyní už konečně můžeme vytvořit deskriptory detekovaných klíčových bodů. Pro každý bod si nalezneme nejbližší rozmazané zobrazení a okolí daného bodu si rozdělíme na 8 čtverců o roz-

měrech 4×4 body. Pro každý čtverec vytvoříme histogram s osmi sloupci, který budeme hodnotit velikostí gradientu dané oblasti a relativní orientací vůči orientaci klíčového bodu. Získáváme tak popisný vektor o 128 prvcích. Tento vektor následně normalizujeme, čím získáváme schopnost in-variance vůči osvětlení objektu. Takto vytvořené deskriptory již můžeme libovolně porovnávat s dalšími deskriptory z jiných obrazů [5].

4.1.2 SURF

Z důvodu potřeby snižovat velikost deskriptorů a přitom zanechat jejich popisné vlastnosti jsou neustále vymýšleny další algoritmy pro vyhledávání klíčových bodů a jejich popis. Jedním z těchto algoritmů je i SURF (Speeded Up Robust Feature), který byl představen v roce 2006 [7] a oproti algoritmu SIFT dosahuje poloviční velikosti deskriptoru (tedy 64 prvků vektoru).

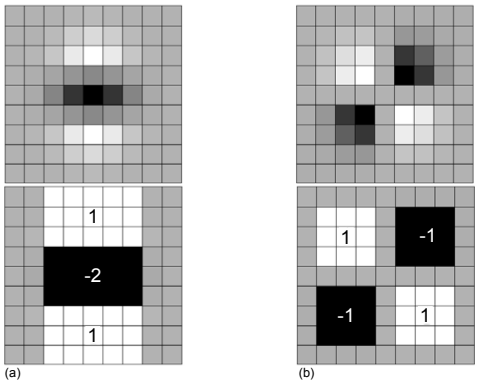
SURF pracuje s integrálním obrazem. Integrální obraz je takový obraz, který v hodnotě každého pixelu obsahuje sumu všech hodnot pixelů od počátku do daného pixelu. Sumu můžeme tedy zapsat jako:

$$I_{\Sigma}(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j) \tag{33}$$

Dalším krokem je spočtení Hessiánu, tedy determinantu Hessovy matice $\boldsymbol{H}$ [5], která již byla uvedena výše u algoritmu SIFT. Hessián spočteme jako:

$$\det(\boldsymbol{H}) = \det \begin{pmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{pmatrix} \tag{34}$$

Gaussovy funkce budeme aproximovat pomocí 2D filtrů s celočíselnými koeficienty, čímž se nám zjednoduší výpočetní postupy. Tuto aproximaci zachycuje Obrázek 15, kde (a) zobrazuje zobrazení L_{yy}(nahoře) a jeho aproximaci D_{yy}(dole), (b) představuje zobrazení L_{xy} a jeho aproximaci D_{xy}. Číselné hodnoty představují váhy čtverců. Šedé čtverce mají váhu 0.



Obrázek 15: Aproximace Gaussových funkcí. Zdroj [7]

Tato aproximace má pochopitelně vliv na výpočet Hessiánu. Pro zachování přesnosti výpočtu lze výpočet Hessiánu opravit následujícím vztahem [7]:

$$\det(\boldsymbol{H}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{35}$$

Invariantnosti vůči zvětšení hledaného obrazu dosahuje SURF pomocí výše zmiňovaného škálovaného prostoru, který byl popsán u algoritmu SIFT. Rozdíl je v tom, že se ke konvoluci s Gaussovými funkcemi používá integrovaný obraz místo obrazu zdrojového. Nejmenší možný 2D filtr obsahuje 9×9 pixelů a odpovídá hodnotě parametru Gaussovy funkce $\sigma = 1.2$. Toto nastavení parametru σ budeme nazývat jako základní přiblížení s. Mezi jednotlivými oktávami vždy zvětšujeme 2D filtr o 6 pixelů [7].

Po výpočtu Hessiánů pro různé body obrazu v různých zvětšeních se můžeme zaměřit na detekci klíčových bodů. Celý obraz nejprve ořízneme dle určeného prahu (nutno určit dle charakteru aplikace) a následně hledáme lokální maxima. Toto hledání je opět jako u algoritmu SIFT prováděno v okolí bodu a v sousedních zvětšeních (viz Obrázek 14). Hodnotu Hessiána tedy porovnáváme s 26 body. Nakonec interpolujeme polohu maxima s větší přesností než jsou pixely použitím Taylorova rozvoje druhého stupně [5]:

$$\mathcal{H}(x) = \mathcal{H} + \frac{\partial \mathcal{H}^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 \mathcal{H}}{\partial x^2}x, \tag{36}$$

tím, že tento vztah budeme minimalizovat:

$$\hat{x} = -\frac{\partial^2 \mathcal{H}^{-1}}{\partial x^2}\frac{\partial \mathcal{H}}{\partial x} \tag{37}$$

kde $\hat{x}$ představuje pozici nalezeného extrému.

Interpolaci budeme provádět, dokud každá složka řešení $\hat{x}$ v absolutní hodnotě nebude menší než 0.5 (případně dokud nedosáhneme maximálního povoleného počtu interpolačních kroků).

Invariantnost vůči rotaci zajistíme vypočítáním odezev Haarových vlnek [7] v kruhovém sousedství bodu v rozsahu $6s$ od zkoumaného bodu dle os x a y, kde s představuje měřítko ve kterém byl bod detekován. Jakmile máme spočtené jednotlivé odezvy v horizontálním i vertikálním směru, přidáme k nim váhu danou Gaussovo funkcí centrovanou na zkoumaný bod s parametrem $\sigma = 2.5s$. Odezvy chápeme jako vektory. Dominantní orientaci zjistíme spočtením sumy všech odezev v posuvném oknu orientace (jehož posun je roven s), které pokrývá úhel $\frac{\pi}{3}$. Takto sečteme odezvy horizontálního i vertikálního směru a ze sum tak získáváme nový dvouprvkový vektor pro každý posun okna orientace. Nejdelší z takto spočtených vektorů udává orientaci zkoumaného bodu [7].

Pro extrakci deskriptoru si nejprve vytvoříme čtverec se středem v klíčovém bodě o velikosti $20s$, orientovaném dle dominantní orientace bodu. Tuto oblast rozdělíme na 4×4 menších čtvercových oblastí. Každá podoblast tak obsahuje 5×5 rovnoměrně rozložených bodů. Pro každý bod jsou spočteny odezvy Haarových vlnek v rozsahu $2s$ dle os orientace d_x a d_y. K jednotlivým odezvám přidáme váhu Gaussovy funkce centrované do klíčového bodu s nastavením parametru $\sigma = 3.3$. Každá z čtvercových oblastí je tedy reprezentovány čtveřicí hodnot $\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|$. Výsledný deskriptor pak bude obsahovat celkem 64 hodnot popisujících klíčový bod. Tento popisný vektor nakonec ještě normalizujeme pro zajištění invariance vůči osvětlení [7].

4.2 Vyhledávání kontur

Zcela jiný přístupem rozpoznání směrových šipek na vozovce může být vyhledávání kontur (obrysů) v obrazu. Za předpokladu znalosti šířky jednoho pruhu vozovky (která je snadno a rychle zjistitelná ze zkoumaného obrazu za asistence uživatele) můžeme v obrazu vyhledat všechny kontury, jejichž poměr stran ohraničujícího obdélníka odpovídá poměru známého ze vzorových šipek a plocha odpovídá předpokládané hodnotě spočtené na základě znalosti šířky pruhu vozovky. Než se ale zaměříme na samotný algoritmus vyhledávání kontur, který v roce 1985 představil Satoshi Suzuki [8], je potřeba osvětlit vytvoření binárního obrazu.

Algoritmus vyhledávání kontur vyžaduje ke své práci binární obraz. Binární obraz vznikne převodem původního obrazu do stupňů šedi a následným prahováním. Tímto postupem získáváme takový obraz, jehož matice hodnot pixelů obsahuje pouze hodnoty 1 a 0 (představující bílou, resp. černou barvu). Bitová hloubka takového obrazu je proto vždy rovná jedné. Nyní budou popsány různé způsoby převodu obrazu na obraz černobílý a následně se zaměříme na možnosti prahování obrazu.

4.2.1 Převod obrazu na černobílý

Černobílý obraz v každém svém pixelu uchovává pouze informaci o intenzitě barvy, která je stejná pro každou ze tří barevných složek (RGB). Intensita pixelu se pohybuje v rozmezí minima a maxima, což pro 8-bitovou barevnou informaci znamená rozmezí $\langle 0, 255 \rangle$. Způsobů převodu barevného obrazu na černobílý je obecně známých několik, mezi nejpoužívanější patří:

- *Zprůměrování hodnot barevných složek* – nejjednodušší postup. Každé z RGB složek pixelu se nastaví hodnota spočtená jako [10]:

$$grayValue = (pixel.red + pixel.green + pixel.blue)/3 \tag{38}$$

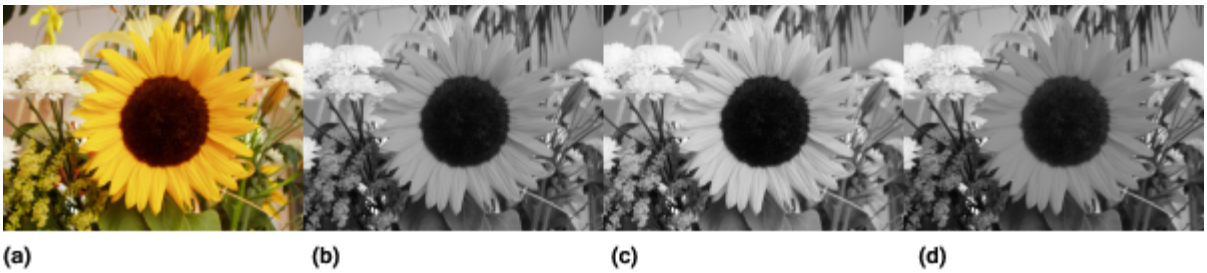
- *Luminance* – postup vycházející z biologických vlastností lidského oka, konkrétně z faktu, že schopnost lidského oka vnímat různé barvy se velmi liší. Zelenou barvu vnímáme mnohem silněji než červenou a červenou barvu zase vnímáme více než modrou barvu. Výpočet hodnoty černobílého obrázku pro každou barevnou složku vypadá následovně [10]:

$$grayValue = (pixel.red \cdot 0.299 + pixel.green \cdot 0.587 + pixel.blue \cdot 0.114) \tag{39}$$

- *Světlost* – převod založený na konvertování barevných složek RGB do barevného systému HSL (Hue Saturation Lightness) [9] s nastavenou hodnotou S na nulu. Pixel pak převedeme do stupňů šedi touto úpravou:

$$grayValue = (\max(pixel.red, pixel.green, pixel.blue) + \min(pixel.red, pixel.green, pixel.blue))/2 \tag{40}$$

Rozdíl těchto převodních postupů zachycuje Obrázek 16, kde (a) zobrazuje původní obraz, (b) výsledek zprůměrování, (c) luminaci, (d) světlost.



Obrázek 16: Výsledky různých způsobů převodu obrazu do stupňů šedi

4.2.2 Prahování

Prahování je nejjednodušší a nejpoužívanější segmentační metoda, tedy metoda, která se v černobílém vstupním obrazu snaží oddělit regiony obsahující objekty od pozadí [11]. Její výhoda je v nenáročnosti na hardware a výpočetní čas. Separace objektů je založená na velkém rozdílu mezi barevnou intenzitou objektu a pozadí. Hned z tohoto faktu je zřejmé, že se metoda prahování nedá aplikovat na jakoukoli třídu obrazů, nýbrž jen na ty, které mají objekty a pozadí snadno jasově rozlišitelné. Abychom oddělili objekty od pozadí, budeme porovnávat hodnotu každého pixelu s předem daným prahem. Volba prahu ale není triviální záležitost a mnohdy ji nelze provádět automaticky. O automatické určení prahu se můžeme pokusit například implementací algoritmu, který v roce 1979 představil Nobuyuki Otsu [12]. Výstupem některých typů prahování je binární obraz, který můžeme přímo využít jako vstup algoritmu vyhledávání kontur.

V následujícím popisu jednotlivých typů prahování budeme používat následující značení:

- $f(i, j)$ - hodnota pixelu na pozici i, j.
- P - konstantní hodnota prahu.

Nejzákladnější typy prahování jsou [11]:

- *Binární prahování* – V tomto typu prahování je každá hodnota pixelu $f(i, j)$ porovnávána s prahem P. Hodnota $f(i, j)$ je pak přenastavena dle následující podmínky:

$$f(i,j) = \begin{cases} 1 & pokud\ f(i,j) \geqq P \\ 0 & pokud\ f(i,j) \leqq P \end{cases} \tag{41}$$

- *Invertované binární prahování* – Typ odvozený od binárního prahování. Získáme ho invertováním podmínky stanovené pro binární prahování.
- *Osekávání* – Tento typ prahování už neposkytuje jako výstup binární obraz, jelikož spočívá v osekávání hodnot, které překročili práh P. Osekávání probíhá dle pravidla:

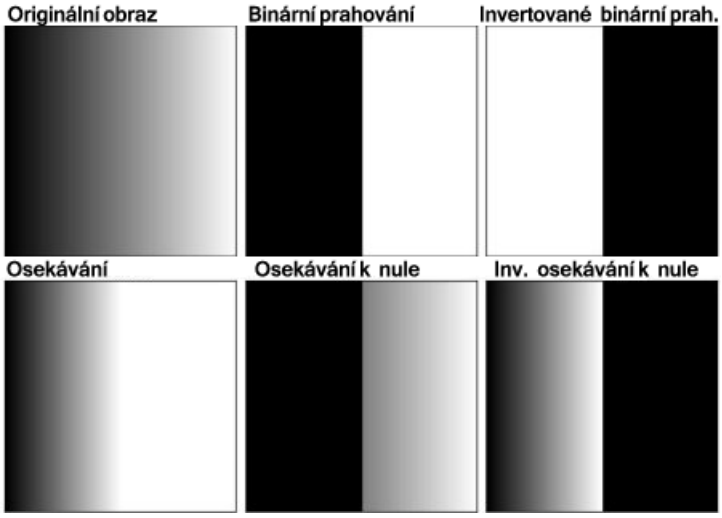
$$f(i,j) = \begin{cases} P & pokud\ f(i,j) > P \\ f(i,j) & jinak \end{cases} \tag{42}$$

- *Osekávání k nule* – Obdoba předchozího typu osekávání s mírně modifikovaným porovnávacím pravidlem:

$$f(i,j) = \begin{cases} f(i,j) & pokud\ f(i,j) > P \\ 0 & jinak \end{cases} \tag{43}$$

- *Invertované osekávání k nule* – Typ odvozený od Osekávání k nule, který získáme invertováním porovnávacího pravidla.

Všechny výše uvedené typů prahování jsou zobrazeny na Obrázku 17.



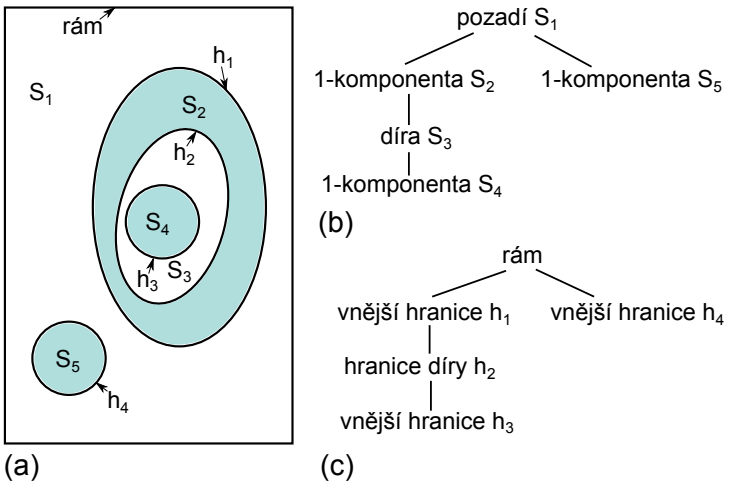
Obrázek 17: Různé typy prahování

4.2.3 Definice pojmů

Než se podíváme na samotný algoritmus vyhledávání kontur, je potřeba definovat základní pojmy, jejichž znalost bude potřeba k pochopení algoritmu [8].

- *Označení pixelu* – Každý pixel binárního obrazu nabývá hodnoty 0 nebo 1. Označení pixelu s danou hodnotou můžeme tedy zkráceně vyjádřit jako 0-pixel nebo 1-pixel.
- *Označení komponenty* – Všechny sousedící pixely stejné hodnoty označujeme jako komponenty. Komponenty značíme obdobně jako pixely 0-komponenta nebo 1-komponenta. Pokud 0-komponenta obsahuje rám obrazu, nazýváme ji pozadím obrazu, jinak dírou.
- *Hraniční bod* – Jedná se o takový 1-pixel, který má ve svém přímém okolí 0-pixel. Tento pixel tvoří hraniční bod mezi 1-komponentou S_1 a 0-komponentou S_2.
- *Obklopování komponent* – Komponenta S_1 obklopuje komponentu S_2, pokud existuje pixel v komponentě S_2 takový, že pro jakýkoliv ze čtyř směrů z daného pixelu nalezneme mezi komponentou S_2 a rámem obrazu pixel komponenty S_1. Pokud S_1 obklopuje S_2 a existuje mezi nimi hraniční bod, pak tvrdíme, že S_1 přímo obklopuje S_2.
- *Vnější hranice* – Množina hraničních bodů 1-komponenty a 0-komponenty, která 1-komponentu přímo obklopuje, se nazývá vnější hranice. Každá vnější hranice je unikátní.
- *Hranice díry* – Množina hraničních bodů 0-komponenty a 1-komponenty. Čtenář by měl být upozorněn, že hranice díry je také reprezentována 1-pixely (nikoliv 0-pixely) nacházejících se v 1-komponentě. Každá hranice díry je unikátní.
- *Rodičovská hranice* – Máme-li 1-komponentu S_1 pak její rodičovská hranice je hranice díry S_2, která S_1 přímo obklopuje. Pokud je S_2 pozadí obrazu, pak je rodičovská hranice rám obrazu.
- *Obklopování hranic* – Hranice h_n obklopuje hranici h_0, pokud existuje posloupnost hranic $h_0, h_1, \ldots, h_n$ taková, že $\forall k \epsilon \{1, 2, \ldots, n\} : h_k$ je rodičovskou hranicí pro hranici h_{k-1}.

Schématické znázornění komponent, vnějších hranic a rodičovských hranic lze vidět na Obrázku 18.



Obrázek 18: Schématické znázornění komponent komponent a hranic v obrazu (a), obklopování komponent (b) a obklopování hranic (c). Zdroj [8]

4.2.4 Algoritmus vyhledávání kontur

Algoritmus vyhledávání kontur prochází sekvenčně po řádcích (index i) a sloupcích (index j) zdrojový obraz. Každá pozice pixelu bude zapisována jako $p\,(i,j)$, hodnota pixelu na této pozici bude zapisována jako $f\,(i,j)$. Procházení začíná v levém horním rohu obrazu na pozici $p\,(0,0)$. Algoritmus sestává z následujících kroků [8]:

1. Procházej jednotlivé pixely obrazu, dokud nenajdeš takový pixel, který splňuje jednu z následujících podmínek startovacího bodu:

   (a) Pixel na pozici $p\,(i,j)$ je součástí vnější hranice, pokud platí, že hodnota pixelu $f\,(i,j-1)=0$ a zároveň $f\,(i,j)=1$.

   (b) Pixel na pozici $p\,(i,j)$ je součástí hranice díry, pokud platí, že hodnota pixelu $f\,(i,j) \geqq 1$ a zároveň $f\,(i,j+1)=0$.

   Nalezený bod nazýváme jako startovní bod a přiřadíme mu unikátní identifikační číslo, které bude představovat hodnotu nově nalezené hranice. Toto číslo nazýváme sekvenční číslo a pro další potřeby algoritmu si kromě aktuální hodnoty budeme pamatovat i předchozí hodnotu tohoto čísla.

2. Urči rodičovskou hranici pro nově nalezenou hranici. Sekvenční číslo posledně nalezené hranice si uchováváme v paměti a tato hranice musí být buď rodičovskou hranicí nově nalezené hranice, nebo s nově nalezenou hranicí bude sdílet stejnou rodičovskou hranici. Rozhodovací pravidlo přiřazení rodičovské hranice zachycuje Tabulka 3.

Posledně nalezená hranice h_i (horizontálně) / Nově nalezená hranice h_{i+1} (vertikálně)	**Vnější hranice**	**Hranice díry**
Vnější hranice	Sdílení rodičovské hranice s h_{i+1}	Rodičovská hranice je h_i
Hranice díry	Rodičovská hranice je h_i	Sdílení rodičovské hranice s h_{i+1}

Tabulka 3: Rozhodovací pravidlo přiřazení rodičovské hranice

3. Následuj hranici ze startujícího bodu (dle algoritmu následování hranice, například algoritmem představeným v roce 1982 Rosenfeldem a Kakem v knize Digital Picture Processing [13]). Každý bod, který je součástí hranice přenastav na hodnotu sekvenčního čísla, dle následujících podmínek:

(a) Nalezená hranice je vnější hranicí. Pakliže se po hraně pohybujeme v jakémkoliv jiném směru než dolů, nastav hodnotu hraničních pixelů na hodnotu sekvenčního čísla, jinak hodnotu hraničních pixelů nastav na zápornou hodnotu sekvenčního čísla.

(b) Nalezená hranice je hranicí díry. Pakliže se pohybujeme zleva doprava, nastav hraničnímu pixelu zápornou hodnotu sekvenčního čísla, jinak hraničnímu pixelu nastav kladnou hodnotu sekvenčního čísla.

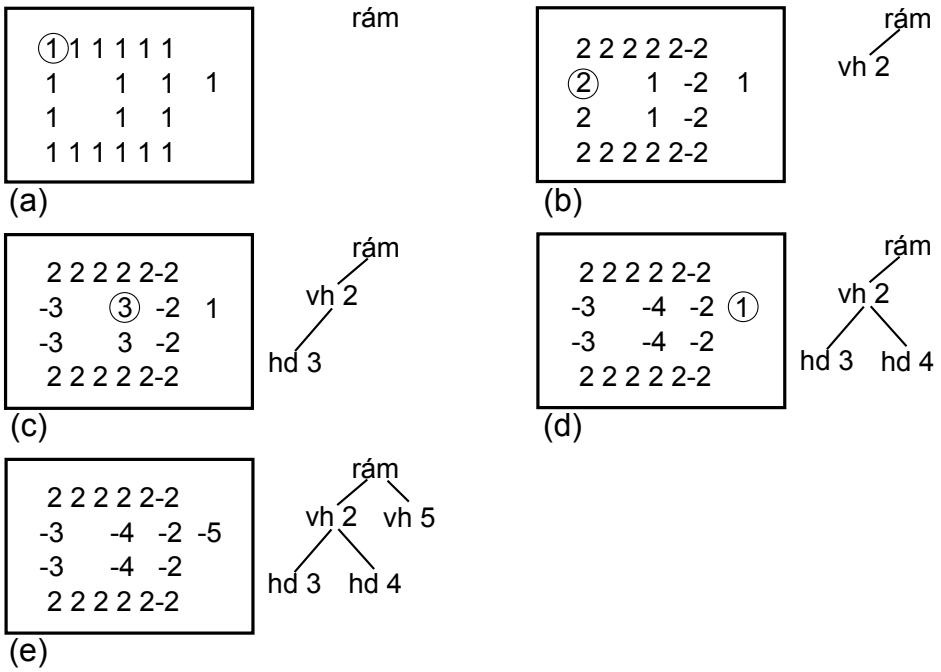
4. Opakuj od kroku 1., dokud neprojdeš celý obraz.

Příklad průběhu algoritmu je k vidění na Obrázku 19. Počáteční stav po detekci prvního hranového bodu vidíme v (a). K detekci 1-pixelu došlo po splnění podmínky zmíněné v prvním kroku algoritmu. Jedná se tedy o detekci vnější hrany. Hodnoty vnější hrany nastavíme na 2, jelikož hodnota 1 je vyhrazena pro rám obrazu. Hodnoty pixelů v pravé části obrazu budou záporné, dle podmínky (a) ze třetího kroku algoritmu. Zároveň je určena rodičovská hrana nově nalezené vnější hrany, tedy rám obrazu. Výsledný stav po aplikaci algoritmu procházení hrany je zobrazen v (b). V dalším stavu je jako startovací bod nalezen zakroužkovaný pixel s hodnotou 2, jelikož splnil podmínku pro hranici díry popsanou v prvním kroku algoritmu. Jako rodičovská hranice je určena vnější hranice nalezená v předešlém kroku. Následuje algoritmus sledování hrany díry, jehož výsledek zachycuje (c). Obsahuje dvě záporné hodnoty -3 a dvě kladné hodnoty 3, dle pravidla zmíněného ve třetím kroku algoritmu vyhledávání kontur. Zbytek algoritmu již proběhne ve stejném duchu, výsledný stav zachycuje (e).

Validita tohoto algoritmu byla ověřena důkazy [8] následujících tvrzení:

- Každý pixel $p\,(i,j)$, jehož soused na pozici $p\,(i,j+1)$ je nejvíce vlevo na nejvyšším možném řádku součástí jakékoliv díry S_1 splňuje podmínku, že hodnota pixelu $f\,(i,j) \geqq 1$ a zároveň $f\,(i,j+1) = 0$.
- Hranice h sledovaná z $p\,(i,j)$ je hranicí díry. Po sledování této hranice už ta samá hranice díry nebude nikdy sledována.
- Když má být určena rodičovská hranice, pak v aktuálně udržovaném sekvenčním čísle je vždy uložená buď rodičovská hranice nové hranice, nebo hranice, se kterou nová hranice rodičovskou hranici sdílí.

Po získání všech kontur, které se nachází v obrazu, provedeme jednoduché vyfiltrování nevhodných kontur. Tedy takových kontur, jejichž šířka a výška, poměr těchto veličin nebo plocha, kterou kontura zaujímá, neodpovídají předpokládaným hodnotám získaných ze vzorových šipek. Jak již bylo řečeno výše, k tomuto kroku potřebujeme znát šířku vozovky. Jelikož jsou rozměry směrových šipek dány normou (viz Kapitola 2.1.4), dokážeme přibližné odvodit hodnoty, kterých



Obrázek 19: Znázornění průběhu algoritmu vyhledávání kontur. Levá část obrázku znázorňuje hodnoty pixelů, pravá část extrahované struktury a typy hranic (vh – vnější hranice, hd – hranice díry). Zakroužkované pixely jsou vždy startovním bodem pro algoritmus následování hranice. Zdroj: [8]

musí kontury nabývat. Samotná filtrace je pak triviální úlohou, proto ji nebudeme dále rozvádět. Výstupem celého algoritmu po odstranění nevhodných kontur budou obrysy, které představují kandidáty na směrové šipky.

4.3 Podobnost pod-obrazů

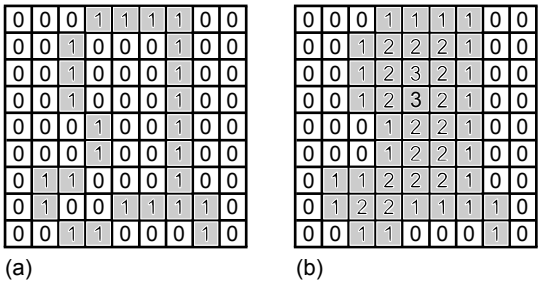
Po získání kandidátů na směrové šipky je potřeba zjistit, které kontury doopravdy představují směrové šipky a které jsou nějakým jiným objektem. Z původního zdrojového obrazu se budeme snažit vyříznout oblasti, které zachycují detekované kontury. Abychom byli schopni takovýto výřez provést a následně ho porovnávat se vzorovými obrazy šipek, musíme nejprve určit minimální ohraničující obdélník kolem kontury. Způsobů detekce minimálního ohraničujícího obdélníka je mnoho, povětšinou ale vycházejí z práce Toussainta [14] a tato práce se jimi nebude dále zabývat.

Pro zjištění, zda je kandidát na šipku skutečně směrovou šipkou na vozovce, bude potřeba provést porovnání obrázků s množinou vzorových směrových šipek. Tuto množinu snadno získáme z reálných fotografií. Pro každý typ směrové šipky si vytvoříme množinu obrazových předloh, kterou budeme následně porovnávat s kandidátem na šipku. Bude potřeba vytvořit další množinu porovnávacích obrazů, která bude obsahovat falešné detekce, tedy přechody pro chodce, automobily a tak dále. Kandidáta na šipku prohlásíme rozpoznanou směrovou šipkou, pokud jeho podoba s některou ze šipkových množin bude větší, než podoba s libovolným prvkem z množiny falešných detekcí, za využití příslušné metriky podobnosti (viz níže).

Porovnání obrazů můžeme provést různými způsoby. Jako první se jistě nabízí přímé porovnání kontury kandidáta s konturami známých směrových šipek. Jiným přístupem tedy zřejmě bude přímé porovnání výřezu (jehož polohu, úhel a rozměry určuje minimální ohraničující obdélník detekované kontury) s obrazy známých šipek. V tomto případě máme k dispozici různé metriky podobnosti. Nejjednodušší metrikou je Euklidovská vzdálenost obrazů, složitější přístup využívá podobnosti histogramů obrazů, poslední postup bude porovnávat tvary v binárním obrazu. Nyní bude následovat podrobnější popis zmíněných metod.

4.3.1 Přímé porovnání kontur

Kontury můžeme porovnávat na základě distanční transformace [15]. Využijeme zdrojového obrazu v binárním tvaru. Distanční transformaci pak provedeme tak, že pro každý pixel vyskytující se uvnitř kontury ohodnotíme minimální Euklidovskou vzdáleností od nejbližší hrany kontury (viz Obrázek 20) . Podobnost kontur pak spočteme sumou rozdílů hodnot pixelů obou obrazů. Určení vzdálenosti od hrany kontury můžeme volit na základě dalších distančních principů, které jsou k nahlédnutí v [15].



Obrázek 20: Příklad distanční transformace kontury v binárním obraze

4.3.2 Euklidovská vzdálenost obrazů

Tato metrika vyžaduje, aby oba obrazy měly stejné rozměry, přiblížení a nasvícení. Je to základní metrika, která od sebe odečítá hodnoty jednotlivých barevných kanálů obou obrazů, a počítá tak barevnou vzdálenost dvou pixelů. Vzdálenost D všech bodů obrazů O_1, O_2 spočteme sekvenčním průchodem všech pixelů porovnávaných obrazů, v nichž budeme vyjadřovat sumu Euklidovských vzdáleností těchto bodů:

$$D(O_1, O_2) = \sum_{i=0}^{width} \sum_{j=0}^{height} \sqrt{R_{ij}^2 + G_{ij}^2 + B_{ij}^2}, \tag{44}$$

kde R_{ij} představuje rozdíl červené barevné složky obrazů pixelu na pozici$i$, j a spočteme ho pomocí funkce $r(O, i, j)$, která vrací hodnotu červeného barevného kanálu pixelu na pozici (i, j) v obraze O. Vzorec pak zapíšeme jako:

$$R = r(O_1, i, j) - r(O_2, i, j) \tag{45}$$

Obdobně budeme postupovat pro výpočet rozdílu zelených barevných složek G_{ij} a modrých barevných složek B_{ij}. Nevýhodou tohoto výpočtu je velká časová náročnost pro velký počet kandidátů a známých vzorových směrových šipek, jelikož pro každý pixel musíme provést tři operace umocnění a jednu operaci odmocnění.

4.3.3 Porovnání histogramů obrazů

Tato metrika porovnává histogramy zkoumaných obrazů. Výhodou metody je, že nevyžaduje obrazy stejné velikosti, jelikož histogramy můžeme snadno normovat. Protože porovnáváme barevné obrazy, budeme vytvářet pro každý barevný kanál jeden histogram (v případě práce s černobílými obrazy vytvoříme pro každý obraz pouze jeden histogram). Histogram každého kanálu bude mít 0 až 256 sloupců. Hodnota každého sloupce představuje počet výskytů pixelu s danou hodnotou v příslušném barevném kanálu [16]. Poté budeme provádět trojí porovnání podobnosti histogramů příslušných barevných kanálů.

Tento přístup je ale velmi citlivý na změny osvětlení a barevných složek. Navíc při nalezení podobného obrazu zjišťujeme jen to, že obrazy mají velmi blízké rozložení barev. O podobnosti objektů na obrazech nedostaneme žádnou informaci. Z tohoto důvodu není metoda vhodná pro porovnávání výřezů kandidátů se známými šipkami, ale může být do jisté míry použita jako filtrační prvek množiny kandidátů.

4.3.4 Porovnání tvarů binárních obrazů

Vstupem této metody je binární obraz zobrazující kandidáta a množina binárních obrazů vzorových směrových šipek. Metoda spočívá v sekvenčním porovnávání pixelů obou obrazů, proto vyžaduje obrazy stejné velikosti. Jelikož každý pixel může nabývat pouze hodnoty 0 nebo 1, získáváme pro každý obrys obraz obsahující černé pozadí a bílý tvar objektu (případně další objekty, které se na fotografii vyskytují v blízkosti detekované kontury) [17]. Tvar objektu v našem případě představuje samotnou šipku, takže provádíme přímé porovnání tvaru kandidáta na tvar známých šipek. Vzdálenost porovnávaných obrazů je potom suma pixelů pozadí jednoho obrázku, které v druhém obrázku představují objekt a naopak. Tento vztah vyjádříme pomocí funkce $p(O, i, j)$ vracející hodnotu binárního pixelu obrazu O na pozici (i, j) jako:

$$D(O_1, O_2) = \sum_{i=0}^{width} \sum_{j=0}^{height} |p(O_1, i, j) - p(O_2, i, j)| \tag{46}$$

5 Detekce upřesňujících vlastností

V této kapitole budou popsány možnosti detekce složitějších prvků křižovatky, jako jsou jednotlivé řadicí pruhy a ramena křižovatky. Pro každý detekovaný prvek navíc budeme určovat další důležité vlastnosti. Nejprve se zaměříme na způsob orientace směrových šipek, dále se podíváme na možnost spojení více detekovaných šipek do stejného řadicího pruhu a budeme určovat jeho orientaci. Nakonec si vysvětlíme, jakým způsobem je možné detekovat sousední řadicí pruhy a vytvářet ramena křižovatky, na základě přidružování řadicích pruhů.

5.1 Určení orientace směrových šipek

Po krocích uvedených výše získáváme malou množinu rozpoznaných šipek, u kterých známe jejich typ. Zatím ale nevíme nic o jejich orientaci. Tuto vlastnost určíme pomocí nalezení středu křižovatky následujícím postupem:

1. Pro každou šipku vyjádříme úhel, který je shodný s úhlem delší hrany minimálního ohraničujícího obdélníka. Směr úhlu volíme zprava doleva a zdola nahoru.

2. Pro každou šipku spočteme normálovou rovnici přímky $y = kx + q$. Pro výpočet použijeme středový bod šipky a normálový úhel spočtený z vypočteného úhlu v předchozím kroku. Tuto přímku budeme nadále nazývat *normálovou přímkou šipky.*

3. Pro danou šipku spočteme normálovou rovnici přímky $y = ax + b$. Pro výpočet použijeme středový bod šipky a úhel spočtený v prvním kroku. Tuto přímku budeme nadále nazývat *přímkou šipky.*

4. Projdeme množinu všech rozpoznaných šipek a pokud nalezneme dvojici šipek, jejichž rozdíl úhlů vyhovuje definovanému rozmezí (v našem případě $\langle 60°, 120° \rangle$, pak hledáme průsečík přímek šipek.

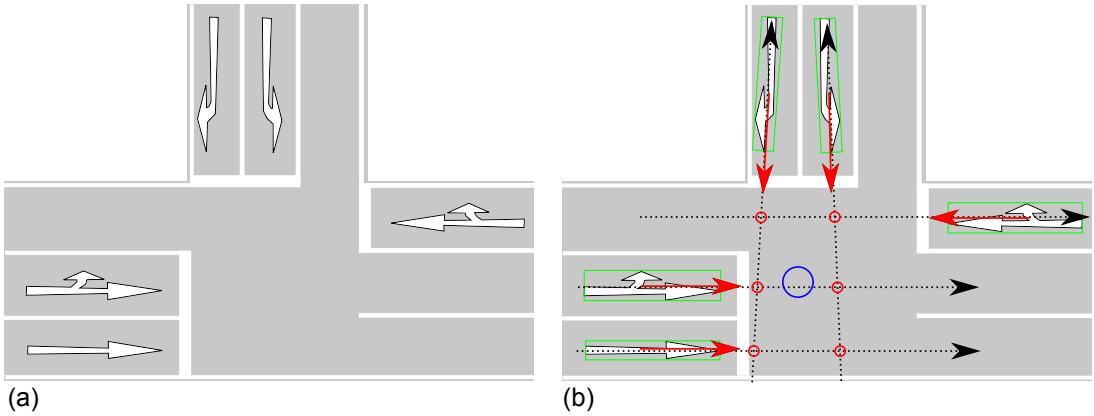
5. Pokud jsme nalezli alespoň dva průsečíky, provedeme zprůměrování pozic všech nalezených průsečíků, čímž získáváme pozici středu křižovatky. Pokud jsme nalezly jediný průsečík, prohlašujeme ho za střed křižovatky

6. Pro každou normálovou rovnici přímky budeme zjišťovat vzdálenost středového bodu křižovatky d od dané přímky. Orientaci šipky s úhlem a pak určíme dle následujících podmínky:

$$a = \begin{cases} a - 180° & pokud\ d < 0 \\ a & jinak \end{cases} \tag{47}$$

Grafické znázornění určení orientace šipek zobrazuje Obrázek 21. V (a) můžeme vidět náčrt křižovatky, kterou se snažíme rozpoznat. V (b) je výsledek výše popsaného postupu. Zelenou barvou jsou vyznačeny detekované minimální ohraničující obdélníky, černou barvou je zobrazen

průběh přímek šipek – orientace přímky udává polaritu spočteného úhlu dle prvního kroku algoritmu, červenými kroužky průsečíky těchto přímek a modrým kruhem střed křižovatky. Červené šipky představují určenou orientace šipky dle podmínek stanovených v pátém kroku algoritmu.

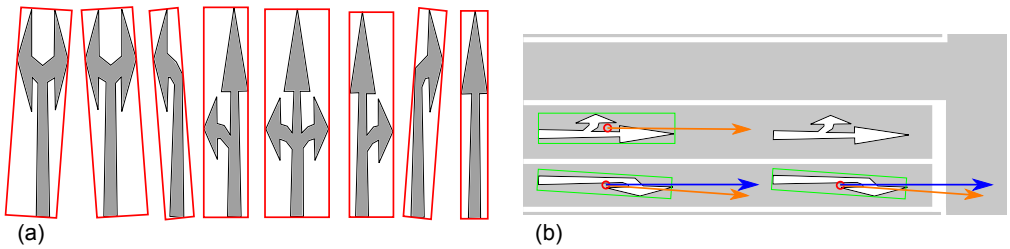


Obrázek 21: Grafické znázornění určení orientace šipek.

5.2 Rozpoznání řadicího pruhu

Jelikož už známe typ a orientaci detekovaných šipek, můžeme se pokusit o spojení více šipek do jednoho jízdního pruhu. Spojení můžeme provést, pakliže jsme detekovali více šipek, které leží za sebou v dané vzdálenosti. U těchto šipek se dá předpokládat, že náleží jednomu jízdnímu pruhu. Předpokládejme nyní, že úhel šipky vždy ukazuje ve směru pruhu a tudíž i na následující směrovou šipku (pokud taková existuje). Nalezení šipky stejného pruhu pak provedeme jednoduchým průchodem množiny detekovaných šipek, kdy budeme zjišťovat, zda zkoumaná dvojice šipek má blízký úhel a zda vzdálenost středů šipek vyhovuje normě (viz Kapitola 2.1.4). Abychom se vyhnuli chybnému spojování šipek, je vhodné tyto podmínky implementovat pomocí zkoumání náležitosti středového bodu jedné šipky dříve spočtené přímce druhé šipky (viz Kapitola 5.1). Nalezneme-li takovou dvojici, můžeme prohlásit, že dané šipky náleží jednomu jízdnímu pruhu. U takovéhoto pruhu si budeme vždy uchovávat jeho středový bod (zprůměrované hodnoty středových bodů šipek náležících pruhů) a úhel (zprůměrované hodnoty úhlů šipek).

Z Obrázku 21 je ale patrné, že směr minimálního ohraničujícího obdélníka vždy neodpovídá směru jízdního pruhu. Z tohoto důvodu uvádím v Obrázku 22a přehled tvarů a pootočení minimálních ohraničujících obdélníků pro nejvíce používané šipky. Na obrázku vidíme, že problém pootočeného ohraničujícího obdélníka bude nastávat pouze v případech šipek odbočování vpravo, vlevo a vpravo i vlevo zároveň. Jelikož jsou ale šipky na vozovku kresleny dle šablon a norem, bude otočení minimálního ohraničujícího obdélníka vždy stejné a můžeme proto provádět opravu úhlu šipky o předem daný úhel. Tuto úpravu zobrazuje Obrázek 22b, na němž můžeme vidět rozpoznané šipky (zelené obdélníky), spočtený úhel šipky (oranžová šipka), případně opravený úhel šipky (modrá šipka).



Obrázek 22: Pootočení minimálního ohraničujícího obdélníka u některých druhů šipek (a) a oprava tohoto jevu (b).

Pro další práci s řadicím pruhem, bude potřeba znát informace o jeho vlivu na ostatní ramena křižovatky. Pro každý řadicí pruh si budeme uchovávat třírozměrné pole binárních hodnot, které bude vyjadřovat směřování šipek. Tabulka 4 definuje pole pro nejvíce používané šipky.

Typ šipky / Pozice v poli	**0**	**1**	**2**
falešná detekce	0	0	0
pravá	0	0	1
přímá	0	1	0
přímá a pravá	0	1	1
levá	1	0	0
levá a pravá	1	0	1
levá a přímá	1	1	0
levá, přímá a pravá	1	1	1

Tabulka 4: Vliv pruhu na křižovatku reprezentovaný polem

5.3 Určení směru řadicího pruhu

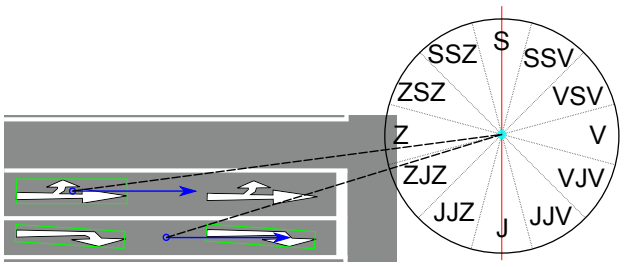
Pro každý rozpoznaný řadicí pruh můžeme nyní bez problémů určit jeho relativní směr vůči středu křižovatky. Abychom to mohli udělat, potřebujeme nejprve pro každý pruh vyjádřit *středový úhel* sa. Středový úhel sa je takový úhel, který svírá přímka procházející středem řadicího pruhu s a centrem křižovatky c s osou y a spočteme ho jako:

$$sa = \arctan(\frac{s_y - c_y}{s_x - c_x}) \tag{48}$$

Určení směru řadicího pruhu pak provedeme dle porovnání směrového úhlu se známými úhlovými hodnotami ze směrové růžice. Toto porovnání schématicky znázorňuje Obrázek 23: čerchované čáry představují přímky procházející středem řadicího pruhu a centrem křižovatky, které spolu s červenou osou y určují středový úhel pruhu.

5.4 Přidružování řadicích pruhů a tvorba ramene křižovatky

Jelikož se v rameni křižovatky většinou vyskytuje více řadicích pruhů, je potřeba jednotlivé jízdní pruhy přidružit k pruhům, se kterými sousedí. Abychom byli schopni prohlásit pruhy za sousední, budeme porovnávat jejich středové úhly, jelikož u sousedních pruhů si tyto úhlu budou velmi blízké (viz Obrázek 23).



Obrázek 23: Určení světové strany pro řadicí pruhu

Pokud jsme nalezli více sousedních řadicích pruhů, bude potřeba určit, který z pruhů zaujímá nejvíce levou pozici. Tuto informaci totiž využijeme v dalším kroku pro odhad počtu výstupních pruhů daného ramena. Nejvíce levý pruh se jistě dá zjistit na základě porovnávání středových úhlů pruhů, ale s tím jsou spojeny drobné komplikace s reprezentací středového úhlu (ten nabývá hodnot $\langle -180°, 180° \rangle$).

Lepším způsobem z důvodu jasné reprezentace výsledků je porovnávání náležitosti bodu polorovině. Abychom zjistili, zda je pruh nejvíce levý, budeme porovnávat náležitost středového bodu pruhu s se všemi polorovinami, určenými přímkami sousedních pruhů. Toho docílíme poměrně jednoduchým výpočtem vzdálenosti bodu s od přímky reprezentující sousední pruh. Je-li vzdálenost záporná, pak bod s leží v levé polorovině od přímky. Dostaneme-li tedy vzdálenosti bodu s od všech přímek sousedních pruhů zápornou, pak pruh, jemuž s náleží prohlašujeme za nejvíce levý pruh.

Abychom byli schopni určit vliv ramene na okolní ramena křižovatky, budeme si obdobně jako u řadicích pruhů uchovávat třírozměrné pole celých čísel. Při každém přidružení sousedního řadicího pruhu přičteme k jednotlivým buňkám pole ramena hodnoty buněk z pole přidávaného pruhu.

5.5 Určení počtu výstupních pruhů v ramene

Výstupní pruhy ramene křižovatky můžeme odhadnout na základě rozpoznávání obrazu a určit pomocí dříve zjištěných informací. Pro odhad výstupních pruhů využijeme nejvíce levý řadicí pruh ramene, ze kterého se podíváme na polopřímku levé kolmice procházející středovým bodem pruhu. Za využití klasifikátoru logistické regrese (viz Kapitola 3.3.2), který natrénujeme na šedé body vozovky, budeme počítat největší možnou posloupnost šedých bodů v daném směru. Jelikož jsme od uživatele již dříve vyžadovali informaci o šířce silnice v pixelech, dokážeme z velikosti posloupnosti určit, kolik obsahuje pruhů. Tato metoda je velice naivní, ale v případně malého vlivu okolních ramen (viz níže) nám může poskytnout alespoň základní odhad.

Více robustní způsob je určování výstupních pruhů ramena dle vlivu ostatních ramen křižovatky. Pro každý řadicí pruh každého ramena v křižovatce jsme si dle Tabulky 4 vyjádřili vliv pruhu na okolí, z něhož jsme následně určili vliv ramena na okolí. Stačí nám tedy jen zjistit, zda dané rameno má pravého souseda a pokud ano, jakou hodnotu obsahuje sousedovo pole na

pozici 0 (levé odbočky). Obdobně budeme postupovat pro protilehlá ramena (pozice 1 v poli) a levá ramena (pozice 2 v poli).
Definice vztahů ramen v křižovatce:

- Za pravého souseda ramene prohlásíme takové rameno, jehož vzdálenost proti směru hodinových ručiček je co nejmenší.
- Protilehlými rameny nazýváme taková ramena, jejichž úhly jsou co nejvíce protilehlé (tedy rozdíl úhlů se v absolutní hodnotě co nejvíce blíží 180°) a jejichž rozdíl úhlů v absolutní hodnotě je větší než předem daná mez (v našem případě $\frac{2}{3}\pi$).
- Za levého souseda ramene prohlásíme takové rameno, jehož vzdálenost ve směru hodinových ručiček je co nejmenší.
- Jakékoliv další rameno v křižovatce už nemůžeme využít pro určení počtu výstupních pruhů zkoumaného ramena.

5.6 Odhad dalších ramen

Z vlivů ostatních ramen můžeme odhadnout výskyt ramen, které se nám nepovedlo detekovat (například se nezdařilo rozpoznání směrových šipek), nebo nejsou viditelné. Pro každé rozpoznané rameno známe počet pruhů směřujících doprava, rovně a doleva a jsme schopni k němu vyhledat levé sousední, pravé sousední a protilehlé rameno, pokud existuje. Pokud takové rameno neexistuje, ale vliv pruhu do daného směru je větší nule, pak se v tomto směru musí nacházet rameno. U tohoto ramena můžeme odhadnout alespoň počet výstupních pruhů. Odhad určíme jako maximum vlivu ostatních ramen do daného směru.

6 Dostupné nástroje

Po prozkoumání možností dostupných zdrojů, které nám mohou usnadnit řešení daného problému jsem vybral množinu vhodných programů a knihoven. Nejprve představím jazyky Matlab a Octave, dále se zaměřím na C/C++ knihovnu OpenCV (a její Java rozhraní JavaCV). Nakonec se podíváme na open source knihovny pro řešení dílčích úloh.

6.1 Matlab

Matlab (Matrix laboratory) je vysokoúrovňový programovací jazyk zaměřený na matematické algoritmy a jejich vizualizaci [18]. Tento jazyk je optimalizovaný pro práci s maticemi a při vhodném návrhu a zapsání algoritmu (nutno psát algoritmy vektorizovaně) poskytuje zpravidla rychlejší výsledky, než tradiční programovací jazyky (C/C++, Fortran aj.). Matlab je často používaný pro zpracování obrazu, ale i modelování a analyzování finančních dat, biologických a dalších procesů.

6.1.1 Výhody

Funkce jazyka Matlab se dají rozšířit o specializované toolboxy, tedy sady funkcí zabývající se konkrétním problémem. Kód Matlabu může být integrován s jinými programovacími jazyky a aplikacemi.
Matlab poskytuje [18]:

- Vysokoúrovňový programovací jazyk pro technické výpočty.
- Vývojové prostředí pro vývoj programového kódu, správu souborů a dat.
- Matematické funkce z oblasti lineární algebry, statistiky, Fourierovy analýzy, filtrování, optimalizace a numerického integrování.
- Vizualizaci grafů ve 2D a 3D.
- Nástroj pro tvorbu uživatelského rozhraní aplikace.
- Funkce pro integraci programů vytvořených v Matlabu s jinými aplikacemi a programovacími jazyky (C, C++, Fortran, Java, Microsoft Excel).

Jedním ze zmíněných toolboxů je i sada funkcí týkajících se zpracování obrazu (Matlab Image Processing Toolbox), která poskytuje základní množinu standardních algoritmů, funkcí a aplikací z této obalsti. Mezi hlavní vymoženosti patří [20]:

- Analýza obrazu – segmentace (vyhledávání kontur, prahování), morfologické funkce (změna kontrastu, ostranění šumu, aj.), analýza objektu (vyhledávání hran a rohů, Houghova transformace, detekce kruhových tvarů, aj.) statistiky a měření.
- Vylepšení obrazu – filtrování obrazu, zaostření obrazu.

- Geometrické transformace obrazu.
- Funkce využívající více jader procesoru a GPU.

Pokud bychom chtěli k detekci směrových šipek použít metody detekce klíčových bodů, můžeme využít souborů funkcí počítačového vidění (Matlab Computer Vision System Toolbox) [21]. Tento nástroj poskytuje celou řadu metod detekce klíčových bodů (FAST, SURF, BRISK, Harris a další), ke kterým nabízí další funkce jako například sledování pohybu objektu ve videu. Dále obsahuje některé základní morfologické funkce a statistické funkce, ale již neobsahuje segmentační funkce.

6.1.2 Nevýhody

Největší nevýhodou tohoto jazyka je nutnost zakoupení licence. V době psaní této práce vychází studentská licence na základní vývojové prostředí jazyka na $45. Pokud bychom nechtěli dále programovat další potřebné algoritmy a zakoupili některý (nebo oba) ze zmíněných toolboxů, museli bychom zaplatit $26 za každý.

6.1.3 Shrnutí

Základní vývojové prostředí programovacího jazyka Matlab s případným rozšířením toolboxů by pro řešení našeho problému poskytovaly dostatek použitelných funkcí. Vývojové prostředí je dobře zdokumentované, a podpora produktu je na vysoké úrovni. Nevýhodou je nutnost zakoupení licence, která je v našem případě rozhodující.

6.2 GNU Octave

GNU Octave je vysokoúrovňový programovací jazyk pro obecné numerické počty. Je to tedy obdoba výše zmíněného Matlabu, se kterým je i částečně kompatibilní. Na rozdíl od Matlabu je GNU Octave zcela zdarma a vývojáři mají povoleno tento software volně rozšiřovat. Octave je možné nainstalovat na systémy Linux, Windows a Mac OS X. Octave poskytuje funkce pro řešení problémů z lineární algebry, řešení nelineárních rovnic, integrální a diferenciální počty . Octave je možné rozšiřovat funkcemi psanými přímo v Octave, nebo nahráním modulů napsaných v C, C++, Fortranu aj. Teprve v nedávné době bylo do softwaru přidáno grafické uživatelské rozhraní (verze 3.8.0 a vyšší).

6.2.1 Výhody

Skripty napsané v Octave umožňují tvoření datových struktur a tím i objektově orientované programování. Pokud budeme dbát na syntaxi, můžeme snadno napsat skript, který bude spustitelný v GNU Octave, ale i v Matlabu. Uživatelská komunita může volně rozšiřovat funkce produktu.

6.2.2 Nevýhody

Jazyk Octave je interpretovaný programovací jazyk, jehož struktura je podobná jazyku C, ke svému běhu vyžaduje interpret. Některé funkce implementující algoritmy popsané v Kapi-

tole 3 v Octave bohužel nenalezneme. Tyto funkce by bylo potřeba implementovat nebo převzít od komunity. S tím je spojené riziko neefektivní implementace algoritmu.

6.2.3 Shrnutí

GNU Octave je zcela zdarma s širokou uživatelskou základnou, která jej může volně rozšiřovat. Octave ve svém základu neposkytuje tak širokou paletu funkcí jako například Matlab a bylo by nutné je implementovat.

6.3 OpenCV

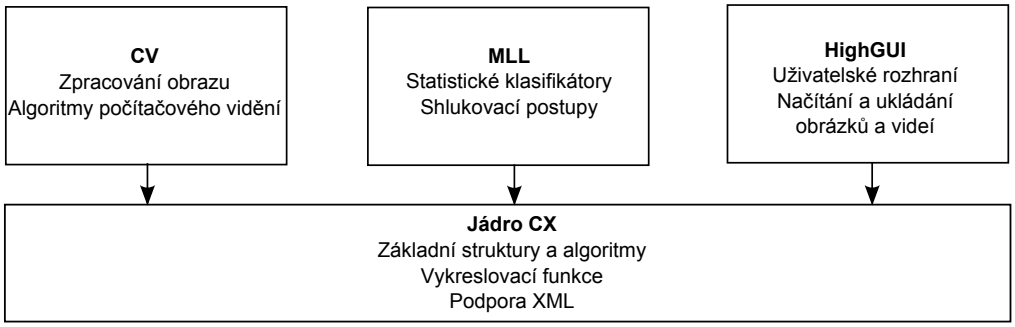
OpenCV (Open source Computer Vision) je knihovna zabývající se programovým viděním a analýzou obrazu. Vývoj knihovny začal v roce 1999, ale první oficiální vydání proběhlo až v roce 2006. Knihovna je napsána v jazyce C a C++ a je ji možno nainstalovat na platformy Linux, Windows a Mac OS X. Aktivně se pracuje na vývoji rozhraní knihovny pro jiné programovací jazyky jako jsou Java, Python, Ruby nebo Matlab. OpenCV se zaměřuje na aplikace běžící v reálném čase a klade proto důraz na vysokou výpočetní účinnost knihovních funkcí s čímž je spojena i snaha optimalizace výpočetních procesů pro vícejádrové procesory. Pokud si uživatel přeje ještě více optimalizovat výpočetní postupy a vlastní procesor firmy Intel, může si zakoupit knihovny IPP (Intel's Integrated Performance Primitives) [22], které nahrazují některé funkce nízké úrovně funkcemi optimalizovanými pro daný procesor a zrychlují tak běh většiny algoritmů. Pokud je tato knihovna nainstalována, OpenCV bude využívat přednostně její funkce [23].

6.3.1 Výhody

OpenCV je registrováno pod BSD licencí, což znamená, že je povoleno tuto knihovnu libovolně užívat pro osobní i komerční účely a vývojář není povinen sdílet vyvíjený kód komunitě. Veškerá funkčnost je dobře zdokumentována a díky open source modelu se kolem knihovny vytvořila nápomocná komunita, která vývojáři pomáhá řešit případné problémy.

Knihovna obsahuje přes 500 funkcí, které kromě široké škály funkcí zabývajících se rozpoznávání obrazu (detekce hran, zaostřování, rozostřování obrazu, prahování, hledání kontur, hledání klíčových bodů, aj.) poskytuje mimo jiné i funkce kalibrace kamery či vytvoření uživatelského rozhraní. Jelikož rozpoznávání obrazu je velice blízké problematice umělé inteligence, obsahuje OpenCV mimo jiné i knihovnu MLL (Machine Learning Library), která poskytuje funkčnost pro učící se algoritmy umělé inteligence, konkrétně statistické rozpoznání obrazu a metody shlukování [23]. Základní strukturu OpenCV zachycuje Obrázek 24.

Velice zajímavou výhodou knihovny jsou již zmíněná rozhraní pro jiné programovací jazyky. Pokud se vývojáři nezamlouvá programování v jazyce C/C++ může bez problémů využít veškerou funkčnost knihoven OpenCV využitím rozhraní JavaCV. OpenCV navíc podporuje vývoj mobilních aplikací v systému Android, na který většina vývoje probíhá právě v jazyce Java.



Obrázek 24: Základní struktura OpenCV

6.3.2 Nevýhody

Jedinou nevýhodou snad může být nedostatečná dokumentace pro rozhraní JavaCV, pro které nejsou k dostání žádné knihy a jen velmi málo návodů. Některé funkce jsou navíc zcela jinak volány nebo jsou součástí jiných balíků, než v C/C++ verzi.

6.3.3 Shrnutí

OpenCV je soubor knihoven, které nám poskytují velice mocný nástroj v oblasti rozpoznání obrazu. Využití knihoven je zadarmo pro osobní i komerční účely bez dalších závazků vývojáře. Jednotlivé funkce jsou implementovány s důrazem na účinnost kódu a výsledné operace proto trvají velmi krátkou dobu. Ačkoliv jsou nativní knihovny OpenCV dobře okomentovány, v případě využití rozhraní JavaCV budeme čelit drobným komplikacím z důvodu chybějící řádné dokumentace. S implementačními problémy se ale můžeme obracet na komunitu okolo projektu OpenCV, která je velmi velká (v současné době knihovnu aktivně využívá přes 50 000 vývojářů) a ochotná pomoci.

6.4 Přehled dalších knihoven pro řešení dílčích problémů

Nástrojů pro zpracování obrazu se v dnešní době dá nalézt velmi mnoho. Obsahem této kapitoly bude příklad menších projektů, než jsou Matlab, Octave a OpenCV. Konkrétně budou představeny knihovny Liblinear a Point Cloud Library.

6.4.1 Liblinear

Nástroj Liblinear A Library for Large Linear Classification) je knihovna pro klasifikaci založenou na lineární regresi rozsáhlých dat [24]. Knihovna je volně dostupná a podporuje logistickou regresi a metodu SVM (Support Vector Machines) [2]. Knihovna poskytuje rozhraní pro jazyky Matlab, Octave, Java, Python, Ruby a další.

6.4.2 Point Cloud Library

PCL (Point Cloud Library) je volně dostupná knihovna pro zpracovávání 2D a 3D obrazů. Knihovna poskytuje mnoho algoritmů včetně filtrování, odhadů funkcí, segmentace, porovnávání

histogramů a nalezení klíčových bodů (SURF, SIFT) a jejich popis deskriptory [25]. PCL je vystavěna modulárně a je proto možné využívat pouze ty moduly, které jsou skutečně potřeba v aplikaci. PCL je multiplatformní knihovnou a je možné ji spustit na systémech Linux, MacOS, Windows a Android. Knihovna je psaná pro jazyk C++.

7 Analýza aplikace

Jedním z bodů zadání této práce je vytvoření aplikace, která musí splňovat několik dílčích vlastností. Musí umožnit načíst satelitní snímek z disku počítače, tento snímek zanalyzovat a pokusit se určit strukturu křižovatky, která se na snímku vyskytuje. Aplikace musí umožnit uložit strukturovanou křižovatku do souboru.

Načtení satelitního snímku, grafické uživatelské rozhraní a uložení detekované křižovatky do souboru jsou operace, pro jejichž vykonání můžeme použít standardní nástroje, které jsou součástí vyšších programovacích jazyků. Proto tyto vlastnosti blíže popíšeme až v následující kapitole, která se bude zabývat implementací aplikace.

V této kapitole bude zkoumána možnost reálného využití rozpoznávacích algoritmů představených v Kapitolách 2 a 3 v aplikaci. Nejprve se zaměříme na detekci okrajových čar vozovky a pak se podíváme na detekci směrových šipek.

7.1 Detekce okrajových čar vozovky

V Kapitole 3 byly představeny rozpoznávací algoritmy, které nám mohou pomoci detekovat okrajové čáry vozovky. Z analýzy těchto algoritmů je patrné, že se buď můžeme vydat cestou implementace logistické regrese pro získání bílých pixelů v obraze a jejich následným zpracováním, nebo cestou hranové detekce, na jejíž výsledek následně použijeme Houghovu transformaci. Výhody a nevýhody těchto přístupů budou popsány níže.

7.1.1 Detekce pomocí logistické regrese

Logistická regrese je nástroj, který nám ze zdrojového obrazu dokáže po vhodném natrénování poskytnout množinu bílých bodů, které obraz obsahuje. Tento algoritmus byl podrobně popsán v Kapitole 3.2. Implementace tohoto algoritmu není složitá a lze provést přímým přepisem pseudokódu Algoritmu 1 do kódu programovacího jazyka. Pokud bychom se nechtěli zabývat implementací logistické regrese, můžeme využít například knihovnu Liblinear popsanou v Kapitole 6.4. Jelikož bude tento algoritmus důležitou součástí detekce směrových šipek (viz Kapitola 7.2) a rozpoznání pixelu je velmi rychlá operace, doporučuji tento algoritmus implementovat vlastními silami. Časově náročnější na výpočet je pouze trénování algoritmu, ale to je prováděno jen při počátečním nastavení klasifikátorů nebo při případné změně trénovací množiny a jedná se tak o ojedinělé události, které nebudou ovlivňovat rychlost běhu aplikace.

Po úspěšné implementaci logistické regrese budeme pracovat s množinou rozpoznaných bílých bodů. Tyto body je ještě potřeba filtrovat z důvodů uvedených v Kapitole 3.3. Po filtraci bude nutné implementovat rekurzivní algoritmus detekce okrajových čar (viz Kapitola 3.3.1), který můžeme použít samostatně, nebo ho využijeme pro metodu vzdálenosti od přímky zmíněné v Kapitole 3.3.2. Implementace tohoto algoritmu není složitá, ale výstupy algoritmu bohužel neposkytují dostatečně spolehlivá data (z důvodu nízkého rozlišení satelitních snímků), která by se dala dále použít pro rozpoznávání křižovatky. Tyto jevy budou podrobněji popsány v Kapitole 9.

7.1.2 Houghova transformace

Houghova transformace je metoda, která slouží k detekci přímých čar v obraze a byla popsána v Kapitole 3.4. Abychom ji mohli použít, musíme nejdříve souřadnice zkoumaných bodů převést do polárních souřadnic a provést detekci hran obrazu. Převod souřadnic není implementačně složitý. Jiná situace platí pro detekci hran, u které doporučuji pro zajištění rychlého běhu algoritmu použít některou z knihoven popsaných v Kapitole 6. Samotná Houghova transformace je pak hledání průniků sinusoid, které jsou tvořeny na základě dříve spočtených polárních souřadnic. Při nalezení dostatečného počtu průniků jsme nalezli přímku, pro níž známe přesné parametrické vyjádření pomocí polárních souřadnic daného průsečíku (viz Kapitola 3.4). Implementace tohoto postupu není složitá, ale výsledky algoritmu při aplikaci na oblast rozpoznávání křižovatky neposkytují dostatečně dobrá data. Výsledky jsou ovlivněny nepříjemnými jevy, které budou popsány v Kapitole 9.

7.2 Detekce směrových šipek

Jelikož se detekce okrajových čar ukázala být nespolehlivou, zaměříme se na detekci směrových šipek. Pokud by se nám za pomocí algoritmů představených v Kapitole 4 podařilo detekovat směrové šipky, byli bychom schopni detekovat další vlastnosti křižovatky (viz Kapitola 5). Níže budou popsána analýza implementace metody porovnávání klíčových bodů a metody vyhledávání kontur.

7.2.1 Porovnávání klíčových bodů

Porovnávání klíčových kontur bylo probráno v Kapitole 4.1. Ve zkratce se jedná o postup, který pro každý bod určí jeho významnost pro blízké okolí. Body představující hrany a rohy budou významnější než jiné body. Tyto klíčové body pak popisujeme deskriptory, které můžeme dále porovnávat. Implementace algoritmu není triviální a je proto doporučeno využít některou z knihoven popsaných v Kapitole 6, kde jsou tyto algoritmy většinou vektorově optimalizovány.

Při aplikaci algoritmů na oblast rozpoznávání křižovatky se setkáme s problémem, který vychází z nedostatečného rozlišení satelitních snímků. Pokud tento fakt spojíme s nekvalitními nákresy šipek na vozovce (tzn. staré, zašlé a ošoupané šipky, u kterých nenalezneme ostrou hranu ani rohy), záhy zjistíme, že algoritmus neposkytuje dostatečné množství klíčových bodů šipky, které bychom byli schopni použít k porovnání se známými vzorovými šipkami. Tento jev bude názorně předveden v Kapitole 9.

7.2.2 Vyhledávání kontur

Nadějným postupem pro detekci šipek na vozovce je vyhledávání kontur. Tento postup byl představen v Kapitole 4.2. K jeho aplikaci potřebujeme nejprve převést obraz do stupňů šedi a provést prahování. Tyto úpravy nejsou implementačně náročné a často bývají součástí grafických knihoven pro úpravu obrazu (viz Kapitola 6). Samotný algoritmus vyhledávání kontur je pak možno implementovat dle popisu v Kapitole 4.2, nebo použít jeho implementaci v některé ze zmíněných knihoven.

Algoritmus vyhledávání kontur bude pro detekci směrových šipek nejlepším nástrojem, z důvodu jeho dobrých výsledků na dostupných satelitních snímcích, jasného určení polohy a velikosti kandidáta na šipku a možnosti porovnání kandidáta se známými šipkami. Výsledky použití této metody jsou k nahlédnutí v Kapitole 9. Velkou výhodou tohoto přístupu je navíc možnost prahovat zdrojový obrázek dle více různých prahů a tím zajistit úspěšnou detekci směrových šipek na různě nasvícených satelitních snímcích a v jízdních pruzích, které obsahují stíny okolních budov, stromů aj. Použitím několikanásobného prahování s následným vyhledáváním kontur nám poskytne velkou množinu obrysů, která bude tvořena duplicitními nálezy a konturami jiných objektů, než jsou směrové šipky. Takto nalezenou množinu tedy bude vhodné filtrovat.

7.3 Porovnání kandidáta na šipku

K porovnání kandidáta na šipku se známými vzorovými šipkami je nejprve potřeba nalézt minimální ohraničující obdélník kolem kontury kandidáta (viz Kapitola 4.3). Tento úkon doporučuji zajistit využitím některé ze zmíněných knihoven v Kapitole 6. Po nalezení minimálního ohraničujícího obdélníka můžeme už přistoupit k porovnání kandidáta se známými šipkami. Různé možnosti porovnávání byly probrány v Kapitole 4.3 a zde bude probrána jejich vhodnost pro implementaci v oblasti rozpoznávání křižovatky.

7.3.1 Přímé porovnání kontur

Přímé porovnání kontur se jeví jako nejsnadnější a nejefektivnější metrika podobnosti, ale nese sebou několik nepříjemností. Kontury kandidáta a známých šipek budou pravděpodobně jinak veliké a museli bychom proto řešit přeškálování kontury kandidáta do velikosti kontur známých šipek. Šipky na silnici bývají často nekvalitní (staré, zašlé a ošoupané šipky bez ostrých hran a rohů) a ohodnocení jednotlivých bodů na základě vzdálenosti od hrany kontury je tedy potenciálně nespolehlivé.

7.3.2 Euklidovská vzdálenost

Tato metrika by pro porovnání kandidáta se známými šipkami byla dostatečná. Její velkou nevýhodou je ale výpočetní náročnost, jelikož obsahuje tři operace umocnění a jednu operaci odmocnění. Tyto operace pak musíme provádět pro každé porovnání kandidáta se známou šipkou pro každý pixel dvojice. Výpočetní náročnost tohoto postupu je proto nepřijatelná.

7.3.3 Porovnání histogramů obrazu

Výhoda tohoto přístupu oproti ostatním metrikám v této kapitole je, že nemusíme upravovat velikost porovnávaných obrazů na stejnou, jelikož můžeme histogramy bez problémů normovat. Nevýhodou ale je, že nám porovnání histogramů poskytne pouze informaci o barevné podobnosti obrazů, ale ne o podobnosti objektů v obrazech. Tuto metriku tedy můžeme rovnou zavrhnout.

7.3.4 Porovnání tvarů binárních obrazů

Tato metrika vychází z předpokladu, že kandidát na šipku, pokud je skutečně šipkou, obsahuje bílou plochu představující šipku a šedou plochu obsahující okolní vozovku. Kandidáta pak můžeme snadno převést na binární obraz a můžeme provést porovnání tvarů v binárních obrazech kandidáta a známých směrových šipek. Výpočet této metriky bude rychlý, jelikož porovnávány budou vždy binární pixely v jediné barevné složce. Tato metrika byla zhodnocena jako nejvhodnější pro aplikaci.

7.4 Detekce upřesňujících vlastností

Po získání šipek je potřeba detekovat jednotlivé řadicí pruhy, přidružit sousední řadicí pruhy do jednotlivých ramen křižovatky a pokusit se odhadnout počet výstupních pruhů ramene. Aby bylo možno detekovat řadicí pruhy, je nejprve nutné určit orientaci rozpoznaných směrových šipek. Výstupní pruhy ramene budou odhadnuty na základě výstupu klasifikátoru šedých bodů a tento odhad bude následně porovnáván s vlivem ostatních rozpoznaných ramen na křižovatku. Veškeré postupy jsou přímo implementovatelné dle popisu sepsaného v Kapitole 5.

7.5 Shrnutí

Po zvážení všech popsaných způsobů rozpoznávání jednotlivých prvků křižovatky a zhodnocení jejich výhod a nevýhod, doporučuji provést implementaci aplikace použitím následující posloupnosti technik:

- Implementace logistické regrese (viz Kapitola 3.2) pro umožnění efektivního filtrování nalezených kontur (viz Kapitola 4.2) a pro odhad výstupních pruhů ramene křižovatky (viz Kapitola 5.5).
- Převod zdrojového obrazu do stupňů šedi a následné několikanásobné prahování pro zajištění detekce různě nasvícených křižovatek a redukci stínů a (viz Kapitola 4.2).
- Vyhledávání kontur v každém prahovaném obraze a následná filtrace duplicitních a jinak nevhodných kontur (viz Kapitola 4.2).
- Porovnání takto získaných kandidátů na šipku s množinou známých vzorových šipek dle metriky porovnání tvaru binárních obrazů (viz Kapitola 4.3.4).
- Zjištění orientace rozpoznaných směrových šipek (viz Kapitola 5.1) a rozpoznání jednotlivých řadicích pruhy, které šipky utvářejí (viz Kapitola 5.2).
- Přidružení rozpoznaných sousedních řadicích pruhů do ramene křižovatky (viz Kapitola 5.4), určení jeho orientace a počtu výstupních pruhů (viz Kapitola 5.5).

8 Implementace aplikace

V této kapitole se zaměříme na implementační specifika aplikace. Návrh implementace zachovává obecné charakteristiky objektově orientovaného programování, bez zaměření na konkrétní programovací jazyk, takže výslednou aplikaci je možné vytvořit v jakémkoliv objektově orientovaném jazyce s příslušnými knihovnami. Nejprve bude zmíněna technologie, která byla využita pro vytvoření reálné aplikace. Dále se zaměříme na strukturu aplikace a osvětlíme nejdůležitější procesy jednotlivých funkčních celků a jejich návaznosti.

8.1 Použité technologie

Dříve, než si vysvětlíme návrh implementace aplikace, bude vhodné zmínit technologie, kterých bylo pro implementaci skutečně využito. Aplikace byla vyvíjena pod operačním systémem Windows 7 ve vývojovém prostředí Eclipse Juno. Jako programovací jazyk byl zvolen jazyk Java spravovaný firmou Oracle. Výhodou tohoto jazyka je snadný a rychlý vývoj a přenositelnost naprogramované aplikace. Pro snadnější řešení dílčích problémů z oblasti rozpoznávání byly využity knihovny OpenCV, které byly podrobněji popsány v Kapitole 6. Rozpoznaná křižovatka je strukturovaně ukládána do XML dokumentu, k čemuž se využívá balík *javax.xml*, který je součástí Java Core API.

8.2 Struktura aplikace

Aplikaci dělíme do několika logických skupin, dle činností vykonávaných danou skupinou. Na aplikaci požadujeme, aby povolovala načtení satelitního snímku křižovatky z disku, aby tento snímek analyzovala, rozpoznaná data strukturovaně uložila a výsledek zobrazila uživateli na výstupu. Potřebujeme tedy sadu funkcí, která bude poskytovat diskové operace. Další sada funkcí se zabývá rozpoznáváním jednotlivých prvků vyskytujících se na křižovatce a rozpoznané prvky ukládá do množiny datových objektů, které prvky reprezentují. Výsledek rozpoznání je následně uložen za využití již zmíněné sady funkcí diskových operací a je vizualizován v grafickém uživatelském rozhraní, které zároveň musí poskytovat všechny prvky ovládání aplikace.

Jednotlivé třídy aplikace tedy dělíme dle jejich funkčnosti do následujících čtyř různých balíků:

- *balík diskových operací*
- *balík analýzy obrazu*
- *balík datových objektů*
- *balík vizualizace*

V následujících podkapitolách bude následovat podrobný výpis funkcí a požadavků, které musí balík splňovat a následný návrh tříd k implementování těchto funkčností.

8.3 Balík datových objektů

Jako první bude probrán *balík datových objektů*, jelikož jediným požadavkem na tento balík je zajištění všech potřebných datových objektů, které budou ostatní balíky aktivně využívat. Každý objekt má svoji sadu vlastností, které jej činí unikátním. Nejdůležitější vlastnosti každého objektu budou zmíněny a bude vysvětlena jejich funkce.

- *Kandidát na šipku* – Základní datový objekt, který je využíván ve většinové části aplikace. Objekt představuje rozpoznaný obrys na silnici, který by eventuálně mohl být směrovou šipkou v řadicím pruhu. U tohoto prvku si potřebujeme uchovávat minimální ohraničující obdélník (jeho výšku, šířku, úhel a středový bod), typ šipky (tedy je-li přímá, odbočovací, nebo kombinací zmíněných), směr šipky vůči středu křižovatky a rovnici přímky šipky.
- *Pruh* – Tento objekt přestavuje jeden řadicí pruh na vozovce a je tvořen kandidáty na šipku, o kterých bylo rozhodnuto, že šipkou skutečně jsou. U pruhu si potřebujeme uchovávat informace o jeho středu, úhlu, směru vůči středu křižovatky a jeho příslušnosti k nějakému ramenu křižovatky.
- *Rameno* – Spojením více pruhů vzniká rameno křižovatky. K jeho reprezentaci potřebuje znát jeho směr vůči středu křižovatky, počet výstupních pruhů a seznam vstupních pruhů, které ho tvoří.
- *Set vzorových šipek* – Tento objekt slouží jen pro uchování seznamu obrazů známých vzorových šipek a pro uchování některých pomocných proměnných.

Pro každý zmíněný datový objekt je vhodné implementovat samostatnou třídu.

8.4 Balík diskových operací

Na *balík diskových operací* máme následující funkční požadavky:

- *Načtení konfiguračního souboru klasifikátoru* – Každý klasifikátor logistické regrese vyžaduje svoje individuální nastavení, dle jeho stupně natrénování a charakteru trénovací množiny. Každému vytvořeném klasifikátoru je tedy potřeba přiřadit touto funkcí jeho konfigurační soubor s příslušnými hodnotami.
- *Načtení trénovací množiny příznaků* – Uživatel může požadovat natrénování klasifikátorů na nové, jím definované trénovací množině. Potřebujeme tedy i funkci, která z příslušného dokumentu načte trénovací množinu příznaků a poskytne je k dalšímu zpracování.
- *Uložení nového natrénování klasifikátoru do konfiguračního souboru* – Po natrénování klasifikátoru logistické regrese na nových trénovacích datech je nutné přepsat konfigurační parametry v konfiguračním souboru klasifikátoru na nové.
- *Načtení konfigurace aplikace* – Nastavení jednotlivých aplikačních parametrů je pro rozpoznávání křižovatky velice důležité a je proto potřeba zajistit, aby je mohl uživatel dle potřeby měnit a nemusel kvůli tomu znovu překládat celý programový kód. Proto jsou veškeré důležité konfigurační parametry načítány ze souboru.

- *Načtení známých vzorových šipek* – Pro porovnávání kandidátů na směrové šipky s množinou známých vzorových šipek je nejprve potřeba pro každý typ šipky načíst množinu obrazů dané šipky z disku a uložit je do příslušného datového objektu.

- *Export křižovatky* – Po úspěšném rozpoznání křižovatky je potřeba nalezené prvky uložit do strukturovaného souboru.

Vzhledem k možnosti jasného oddělení funkčností na dvě skupiny (načítání z disku a ukládání na disk) dělíme tyto funkce dle stejného pravidla do dvou tříd. Jedna třída obsahuje pouze operace vyžadující četní z disku, druhá naopak operace zapisování na disk.

8.5 Balík analýzy obrazu

Tato skupina tříd zřejmě obsahuje algoritmy rozpoznávání obrazu a je tak hlavním balíkem celé aplikace, který využívá funkcí všech okolních balíků. Požadavky na tento balík jsou tedy i cílem aplikační funkčnosti. Jsou to:

- *Klasifikace bílých bodů* – Je potřeba mít natrénovaný klasifikátor, který dokáže určit, zda je zkoumaný pixel součástí okrajové čáry vozovky nebo směrové šipky v řadicím pruhu.

- *Klasifikace šedých bodů* – Je zapotřebí mít ale i druhý natrénovaný klasifikátor, který se pokusí určit, zda pixel zobrazuje asfalt na vozovce, či nikoliv.

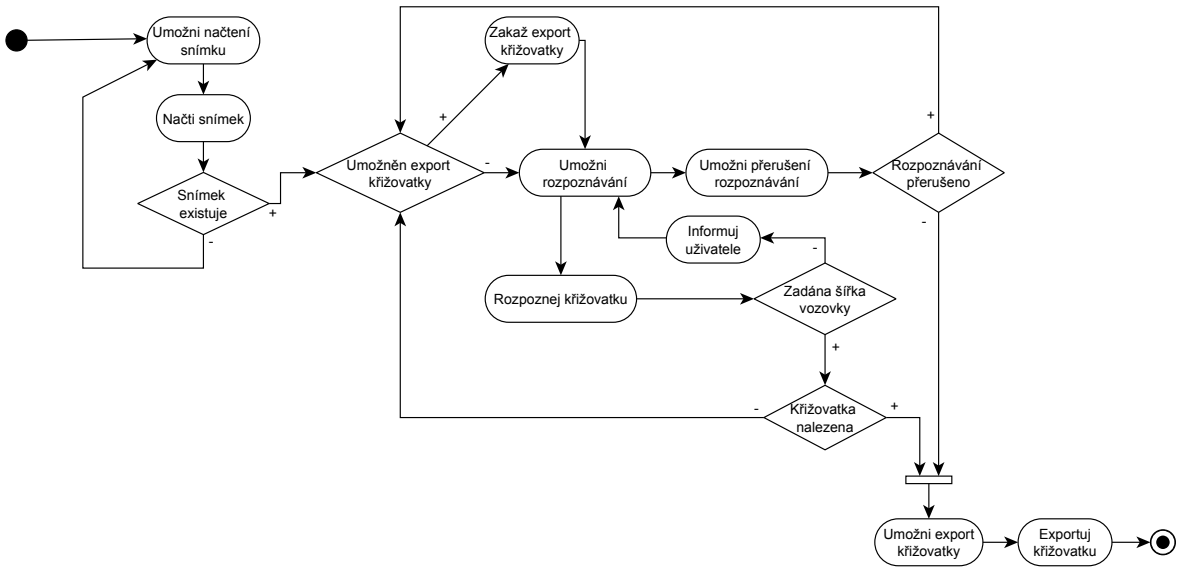
- *Rozpoznání obrazu* – Hlavní metoda, která postupně aplikuje všechny rozpoznávací algoritmy a z jejich výsledků určovat strukturu křižovatky. Tato metoda využívá naprostou většinu funkcí a datových objektů celé aplikace.

- *Změna zdrojového obrazu* – Pokud uživatel dokončí rozpoznávání křižovatky a chce započít rozpoznávání nové křižovatky je potřeba změnit zdrojový obraz, zresetovat všechny použité proměnné a přenastavit šířku pruhu vozovky na novou hodnotu z nového obrazu.

Zmíněné funkce jsou velmi obecné, ale postačí nám k základnímu popisu balíku. Je zřejmé, že jednou z tříd je klasifikátor bodů, jelikož potřebujeme více instancí natrénovaných na různých trénovacích množinách. Další třídou je hlavní klasifikační třída, která volá všechny potřebné klasifikační a pomocné metody. Klasifikační metody ponecháme této třídě a pomocné metody situujeme pro přehlednost do třídy poskytující dodatečnou funkčnost.

8.6 Balík vizualizace

Balík vizualizace musí zajistit spuštění hlavní metody aplikace, která vytvoří instanci grafického uživatelského rozhraní a načte počáteční konfigurační parametry za využití funkce z *balíku datových operací*. Tyto parametry dále distribuuje dalším třídám, které je potřebují pro své funkce. Balík musí zároveň poskytnout veškeré vizualizační funkce. Abychom byli schopni určit jednotlivé ovládací požadavky na grafické uživatelské rozhraní, je nejprve potřeba uvědomit si všechny situace, které mohou při ovládání aplikace nastat. Jednotlivé akce a jejich vzájemné ovlivnění nejlépe znázorňuje diagram aktivit, který můžeme vidět na Obrázku 25.



Obrázek 25: Diagram aktivit grafického uživatelského rozhraní

Z analýzy tohoto diagramu diagramu vyplývají čtyři základní funkce ovládání. Tyto funkce se navzájem ovlivňují a jejich přístupnost uživateli je dána posledně provedenou operací. Pro efektivní ovládání aplikace tedy musíme zajisti následující funkce:

- Otevření zdrojového obrazu k rozpoznávání.
- Zadání (či určení) šířky jízdního pruhu.
- Zahájení a přerušení rozpoznávání.
- Uložení rozpoznané křižovatky do souboru.

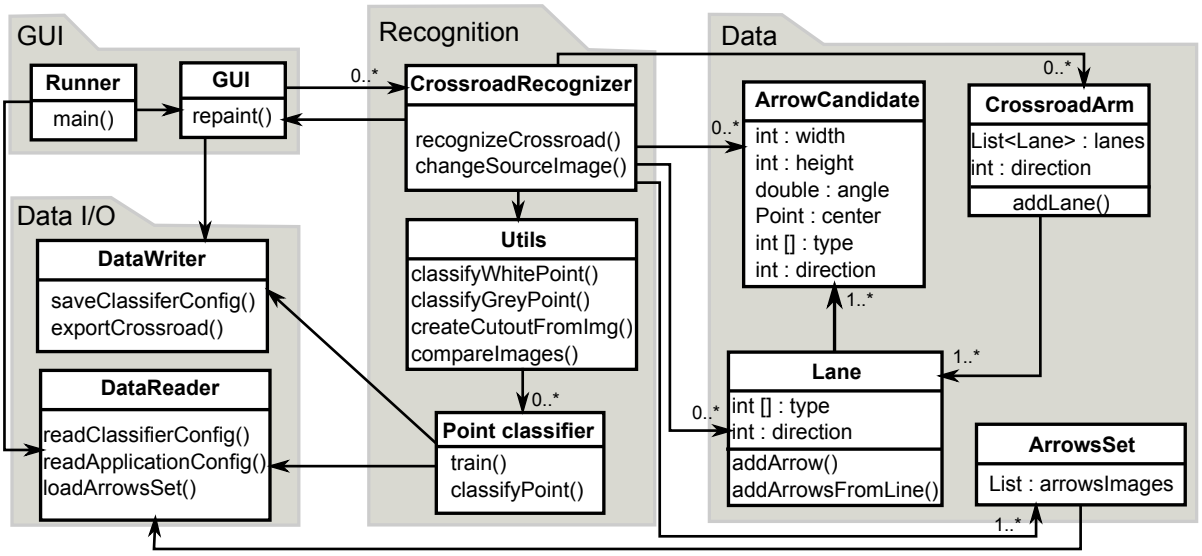
Pro zachování principů objektově orientovaného prostředí je nejvhodnější implementovat jednu třídu, která obsahuje hlavní metodu aplikace, načítá základní nastavení a vytváří instanci druhé třídy, která zajišťuje zobrazení grafického uživatelského rozhraní a vykreslení výsledků algoritmů na obrazovku.

8.7 Diagram tříd

V předchozí kapitole byly podrobněji popsány jednotlivé balíky aplikace a požadavky na jejich funkčnost. Základní zjednodušený digram tříd, který byl vytvořen dle popisů balíků v předchozí kapitole, je zobrazen na Obrázku 26. Balíky jsou znázorněny šedou barvou, třídy bílou.

Balík vizualizace obsahuje třídu *Runner*, ve které se nachází hlavní metoda aplikace. Tato třída potřebuje instanci *DataReader*, aby mohla načíst konfigurační nastavení aplikace. Následně třída vytváří instanci třídy *GUI*, která vytvoří uživatelské rozhraní a nabídne možnosti rozpoznání křižovatky za využití třídy *CrossroadRecognizer*. Třída *GUI* si uchovává instanci *DataWriter*, aby po úspěšném rozpoznávání zavolala funkci pro uložení vygenerované křižovatky do souboru.

Balík analýzy obrazu obsahuje hlavní třídu *CrossroadRecognizer*, která řídí průběh rozpoznávání. Tato třída vyžaduje instanci třídy zajišťující grafické uživatelské rozhraní z důvodu



Obrázek 26: Zjednodušený diagram tříd

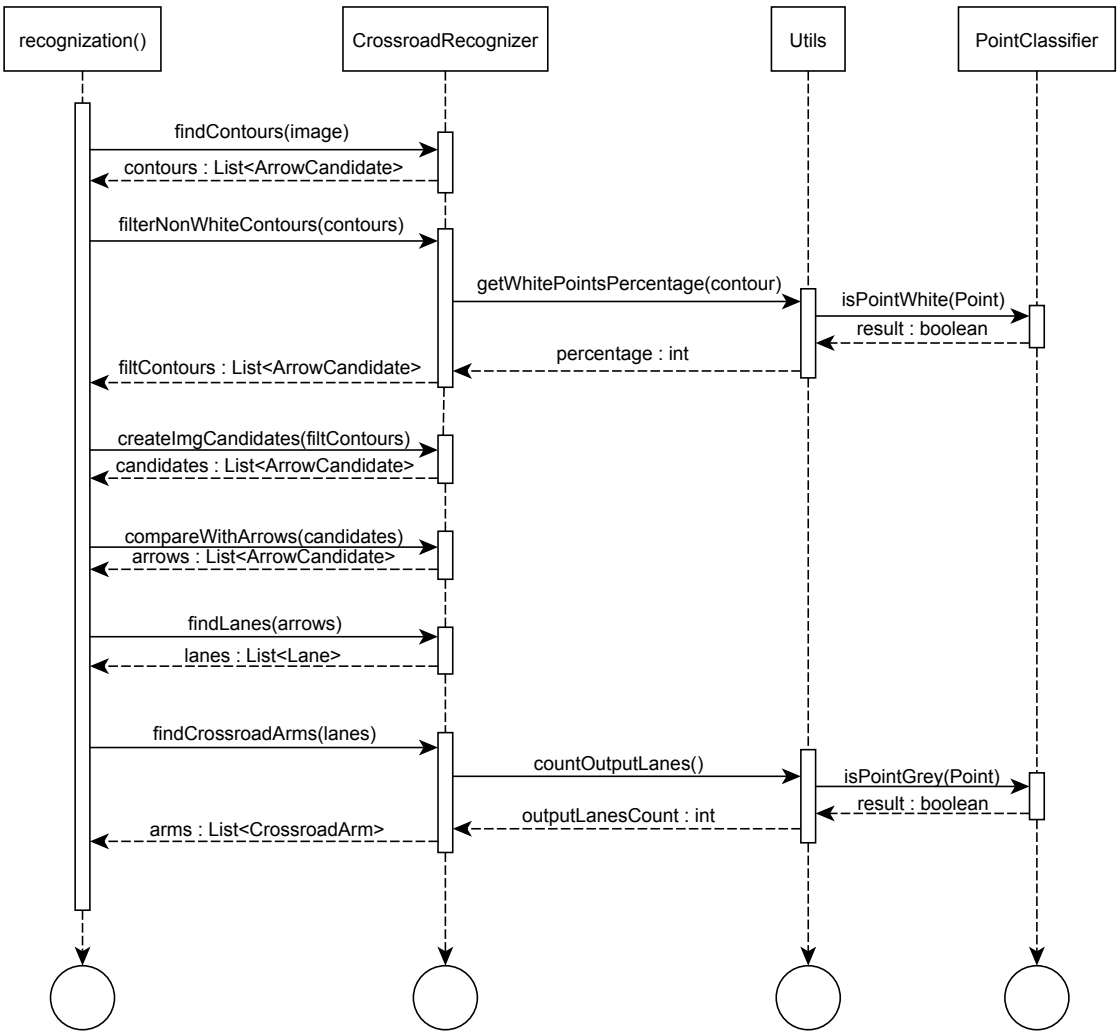
možnosti informování rozhraní o pokroku výpočtu rozpoznávání, takže uživatel vždy bude vědět, kolik z výpočtu přibližně zbývá a nezíská pocit, že aplikace je neaktivní. *CrossroadRecognizer* dále obsahuje seznamy všech objektů z *balíku datových objektů* a ukládá do nich mezivýsledky rozpoznávání. V neposlední řadě využívá funkce třídy *Utils*, která poskytuje jednoduché pomocné metody a spravuje klasifikátory bodů. *PointClassifier* je třída reprezentující klasifikátor bodů a uchovává si instance *DataReader* pro načtení konfiguračních souborů, případně i trénovacích dat a *DataWriter* pro uložení nových parametrů klasifikátoru po natrénování.

U balíku datových objektů můžeme vidět kaskádové spojení objektů *CrossroadArm*, *Lane* a *ArrowCandidate*, jelikož množina objektů *ArrowCandidate* je vždy součástí objektu *Lane* a obdobně množina objektů *Lane* je součástí objektu *CrossroadArm*. Třída *ArrowsSet* pak uchovává obrazy známých vzorových šipek, k jejichž načtení potřebuje vlastnit instanci třídy *DataReader*.

8.7.1 Popis třídy CrossroadRecognizer

Jak bylo popsáno výše, třída *CrossroadRecognizer* je nejdůležitější třídou celé aplikace, která řídí chod rozpoznávacích algoritmů a pracuje s jejich výsledky. Třída dále využívá většinu funkcí a datových objektů ostatních balíků. Je proto důležité uvědomit si jednotlivé kroky, které tato třída musí vykonat, aby bylo rozpoznávání křižovatky úspěšné. Popis třídy je nejlépe reprezentovatelný sekvenčním diagramem, který můžeme vidět na Obrázku 27.

Kořenovým objektem diagramu je metoda *recognization()*, kterou můžeme chápat jako hlavní metodu řídící rozpoznávání křižovatky. Tato metoda bude postupně volat funkce objektu *CrossroadRecognizer*, který dále využívá objektu *Utils*, který poskytuje pokročilejší operace nad objekty klasifikátorů *PointClassifier*. Zobrazená komunikace tříd *Utils* a *CrossroadRecognizer* je pro přehlednost velmi zjednodušená. Sekvenční diagram je vypracován pro řešení navržené v Kapitole 7.



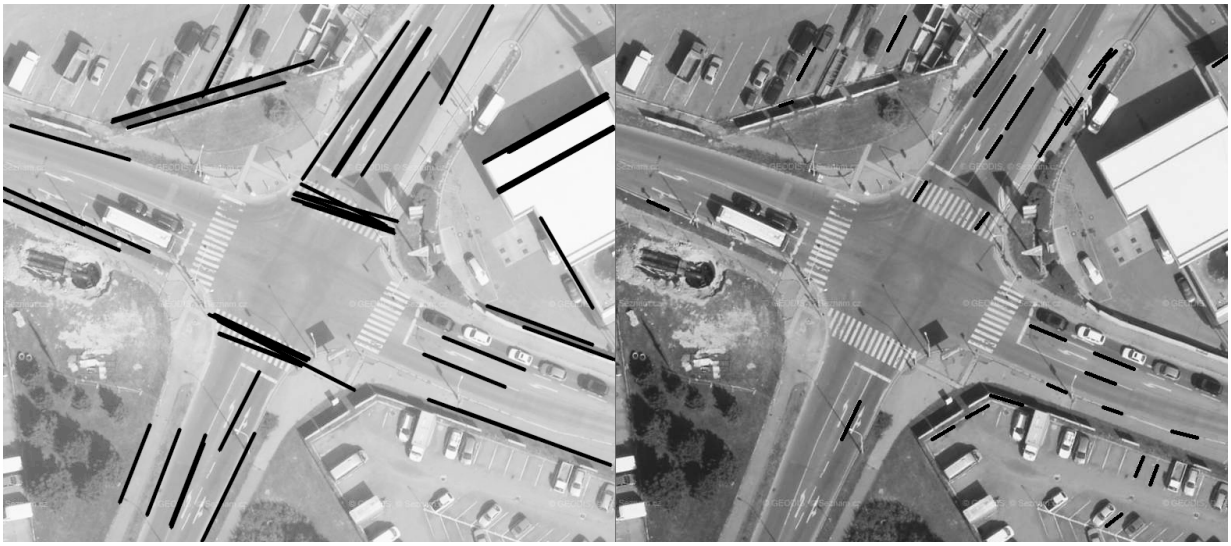
Obrázek 27: Sekvenční diagram postupu rozpoznávání

9 Testování a výsledky

V této kapitole budou probrány výsledky použití jednotlivých algoritmů a budou prezentovány nedostatečné výstupy některých postupů pro oblast rozpoznávání křižovatky (viz Kapitola 7). Nejprve si ukážeme výsledky algoritmů detekujících čáry na vozovce a porovnáme rekurzivní přístup s Houghovo transformací. Dále se zaměříme na detekování směrových šipek pomocí porovnávání klíčových bodů a pomocí vyhledávání kontur. Nakonec se zaměříme na výsledky celé aplikace a tyto výsledky zhodnotíme.

9.1 Detekce čar na vozovce

Pro detekci čar na vozovce byly vyzkoušeny metody Houghovy transformace a rekurzivní metoda detekce (viz Kapitola 3.3). Jak bylo zmíněno v Kapitole 7.1, ani jeden z těchto přístupů neposkytoval dostatečně spolehlivé výsledky pro oblast rozpoznávání křižovatky. Porovnání postupů detekce okrajových čar nejlépe zachycuje grafické znázornění (viz Obrázek 28). Metoda Houghovy transformace se projevuje detekcí přímých čar, ale některé čáry detekuje mnohonásobně a připouští i hrany chodníků a budov. Rekurzivní metoda umožňuje detekci zahnutých okrajových čar, ale je více citlivá na rozpoznané bílé body, které prošly filtrováním. Umožňuje proto více falešných detekcí v okolí křižovatky. U obou metod si můžeme všimnou veliké náchylnosti na hrany chodníků, přechodů a případných parkovišť. Tato náchylnost u většiny snímků velice ztěžuje využití výsledků výše zmíněných postupů.



Obrázek 28: Porovnání metod detekce čar. Vlevo – Houghova transformace, vpravo – rekurzivní metoda

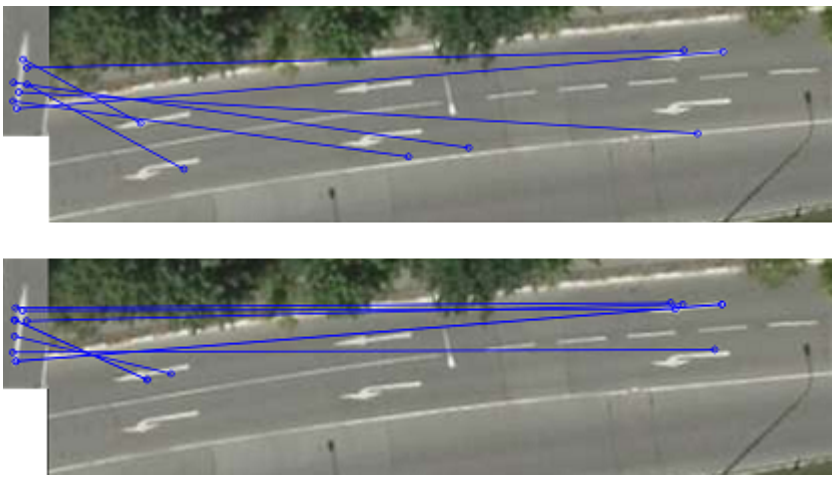
9.2 Detekce směrových šipek

V Kapitole 4 byla podrobně probrána možnost detekce směrových šipek na vozovce. Jedním navrhovaným přístupem je porovnávání klíčových bodů. Jak ale bude ukázáno níže, tento postup

se v oblasti rozpoznávání křižovatky neosvědčil. Mnohem lepší výsledky poskytuje algoritmus vyhledávání kontur. Výsledky zmíněných postupů jsou podrobněji ukázány níže.

9.2.1 Detekce klíčových bodů

Algoritmus detekce klíčových bodů je velmi dobře použitelný pro vyhledávání objektů v obraze, které obsahují zajímavé prvky. Nejlépe jsou využitelné pro detekci identického objektu v obraze. V ideálním případě by se za takový objekt dala jistě považovat i směrová šipka. Na snímcích ale bohužel nalezneme šipky opotřebené, se zašlými hranami a rohy. Šipky jsou navíc (vzhledem k rozlišení zdrojových obrazů) velmi malé a algoritmy z nich nedokáží extrahovat dostatečný počet klíčových bodů. Výsledky těchto algoritmů zachycuje Obrázek 29, u jehož levé strany nalezneme vzorovou šipku, kterou se snažíme detekovat v obrazech vpravo. Modré kruhy pak představují nalezené klíčové body ve vzorové šipce a podobné klíčové body detekované v obraze. Dvojice podobných bodů je vždy spojena přímkou.



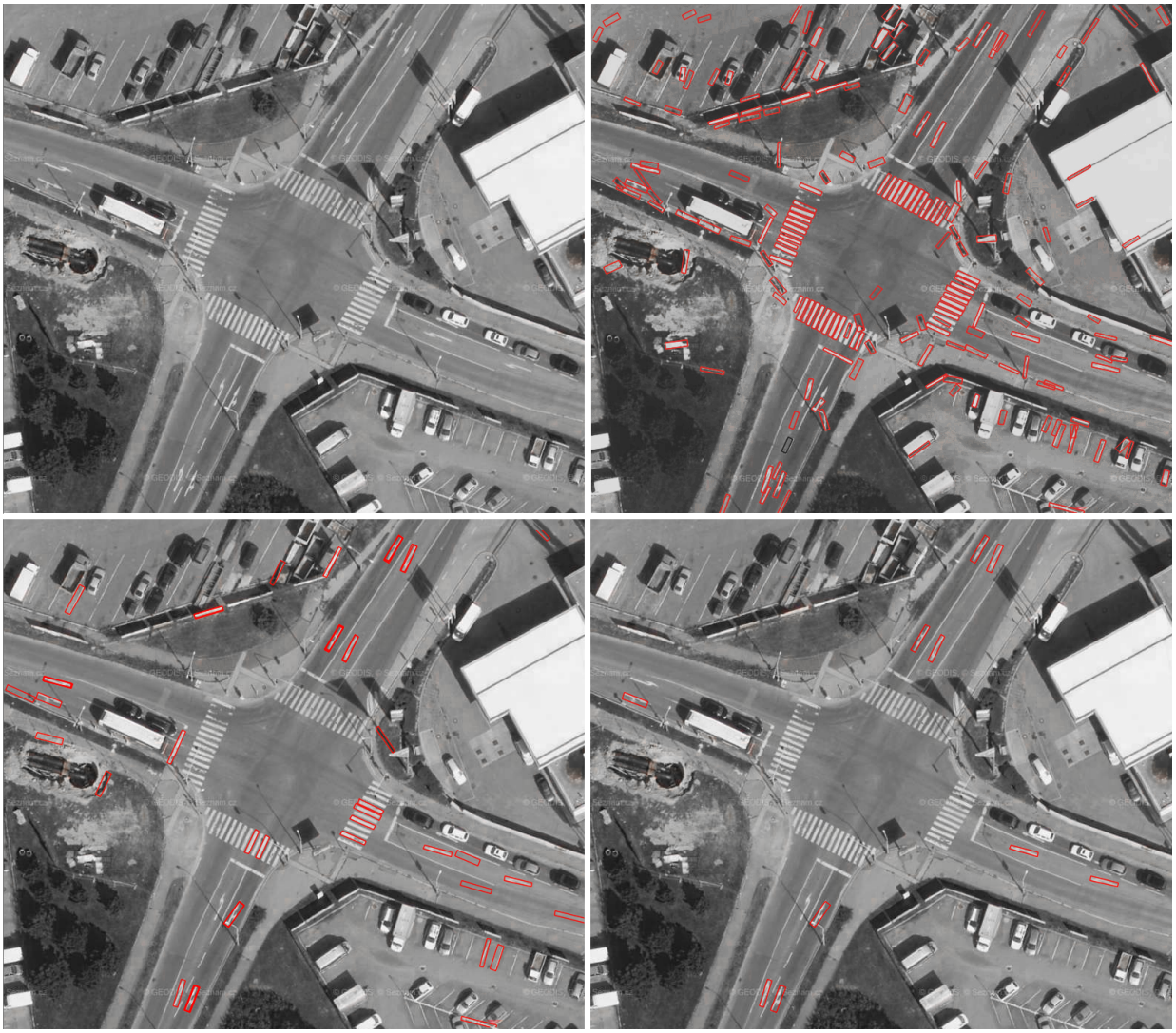
Obrázek 29: Výsledky algoritmů SURF (nahoře) a SIFT (dole).

9.2.2 Vyhledávání kontur

Nejvíce spolehlivým způsobem detekce směrových šipek na vozovce se ukázalo být vyhledávání kontur. Všechny detekované kontury je třeba filtrovat dle jejich rozměrů a obsahu bílých pixelů (viz Kapitola 4.2). Výsledky těchto postupů zachycuje Obrázek 30.

9.3 Výsledky

Na následujících stranách této práce budou zobrazeny výsledky implementované aplikace. Výsledky budou následně vyhodnoceny pomocí grafů úspěšnosti rozpoznávání. Abychom byli schopni grafy úspěšnosti vytvořit, musíme nejprve stanovit bodovací postup pro ohodnocení kvality rozpoznání křižovatky.



Obrázek 30: Výsledky algoritmu vyhledávání kontur. Vlevo nahoře je původní zdrojový obraz, vpravo nahoře jsou všechny detekované kontury, vlevo dole jsou filtrované kontury a vpravo dole jsou kontury rozpoznané jako směrové šipky.

9.3.1 Bodování rozpoznaného obrazu

Každou křižovatku budeme bodovat celočíselným bodováním založeným na počtu správně detekovaných objektů v křižovatce. Nejprve si pro každou křižovatku vyčíslíme maximální možný počet bodů, který bude odpovídat maximálnímu možnému počtu detekovatelných prvků. Body budou přidělovány dle Tabulky 5. Všimněme si, že se v tabulce nenachází rozpoznání ramene křižovatky nebo určení orientace ramene. Tyto činnosti totiž přímo vychází z detekovaných řadicích pruhů a můžeme předpokládat, že při správné detekci řadicího pruhu dojde i ke správné detekci těchto vlastností. Za správně rozpoznanou směrovou šipku budeme považovat takovou směrovou šipku, která je na vozovce jasně viditelná (není zčásti zakrytá automobilem, stínem, či lampou veřejného osvětlení). U takové šipky pak budeme určovat její typ. Správně rozpoznaný řadicí pruh představuje rozpoznaný pruh s minimálně jednou směrovou šipkou, který odpovídá pruhu na vozovce. V tomto testování nebereme v úvahu odhady ramen křižovatky, pokud neobsahují

žádné směrové šipky, nebo nejsou viditelné.

Úspěšnost rozpoznání pak budeme chápat jako poměr získaných bodů a počtu všech získatelných bodů. Úspěšnost tedy můžeme snadno vyjádřit procentuálně tak, že tento podíl vynásobíme 100.

Objekt	**Body**
Rozpoznaná směrová šipka	+1
Správně určený typ rozpoznané šipky	+1
Správně rozpoznaný řadicí pruh	+1
Správné určení počtu výstupních pruhů ramene	+1
Chybná detekce typu šipky nebo počtu výstupních pruhů	0
Falešná detekce šipky nebo řadicího pruhu	-1

Tabulka 5: Tabulka bodování rozpoznané křižovatky

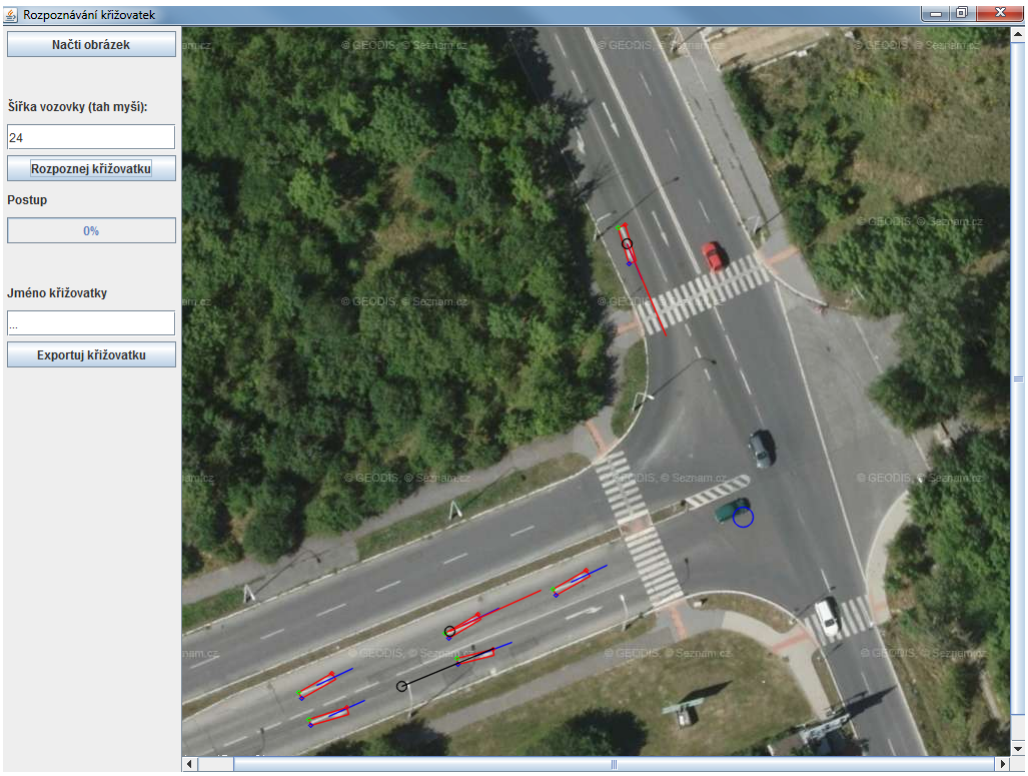
Obrázek 31 zobrazuje strukturovaný zápis rozpoznané křižovatky do xml souboru. Grafický výstup aplikace můžeme vidět na Obrázku 32. Na tomto výstupu si nyní ukážeme ukázkové bodování výstupu algoritmu pro reálný snímek. Maximální možný počet bodů na křižovatce ze snímku je 24 bodů, jelikož obsahuje 9 jasně viditelných směrových šipek, u kterých se snažíme správně určit 9 typů, dále 4 řadicí pruhy a 2 ramena, pro něž budeme určovat počet výstupních pruhů. Na obrázku můžeme vidět, že aplikace rozpoznala 6 směrových šipek, u nichž byly všechny typy určeny správně (celkem 12 b). Dále byly rozpoznány 3 řadicí pruhy (3 body). Z exportu křižovatky do strukturovaného souboru (viz níže) můžeme vyčíst, že počet výstupních pruhů byl odhadnut jako 1 a 2, tedy jen jeden odhad byl správně (1 bod). Bodové ohodnocení tohoto rozpoznání je tedy 16 bodů z možných 24, úspěšnost rozpoznání je tedy $\frac{16}{24} \cdot 100 \approx 67\%$.

```
<crossroad armscount="2" name="kopernikova">
  <arm clock="11" direction="NORTH_NORTH_WEST" id="0">
    <inputLanes count="1">
      <inputLane direction="R" id="0"/>
    </inputLanes>
    <outputLanes count="2"/>
  </arm>
  <arm clock="8" direction="WEST_SOUTH_WEST" id="1">
    <inputLanes count="2">
      <inputLane direction="R" id="0"/>
      <inputLane direction="L" id="1"/>
    </inputLanes>
    <outputLanes count="1"/>
  </arm>
  <arm clock="5" direction="SOUTH_SOUTH_EAST" id="2">
    <inputLanes count="0"/>
    <outputLanes count="1"/>
  </arm>
</crossroad>
```

Obrázek 31: Ukázkové strukturované uložení křižovatky do souboru

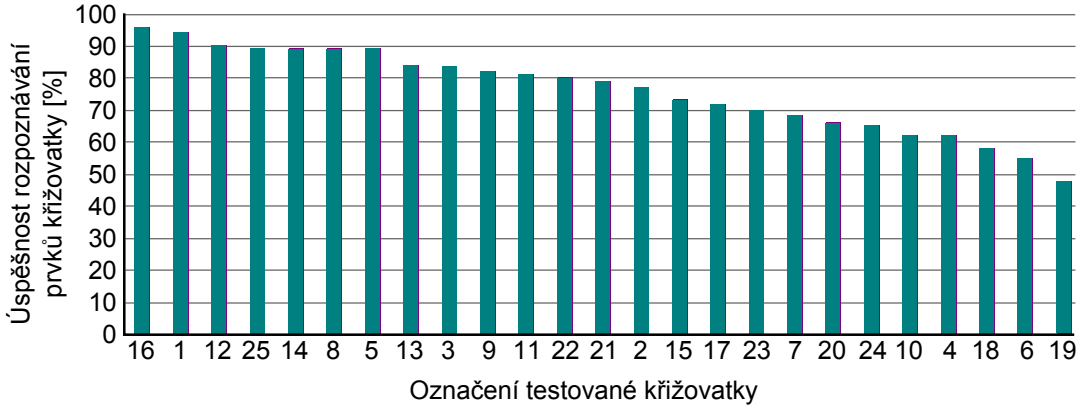
9.3.2 Graf úspěšnosti

Dle bodování vysvětleného výše bylo provedeno testování aplikace na testovací sadě 25 křižovatek z reálných satelitních snímků. Výsledky testování zachycuje graf na Obrázku 33. V grafu můžeme



Obrázek 32: Ukázkový grafický výstup aplikace

vidět, že kolem hranice 90% úspěšného rozpoznávání se umístilo 7 křižovatek. Naopak pod hranicí úspěšnosti 70% se vyskytuje 9 křižovatek. Zjišťujeme tedy, že aplikace poskytla velmi dobré rozpoznání viditelných prvků pro 28% testovaných křižovatek. Dostačující výsledky zajišťující částečné rozpoznání křižovatky získáváme pro 36% testovacích křižovatek. Nevhodné výsledky obsahující falešné detekce, nebo malé procento rozpoznaných prvků křižovatky jsme dostali pro 36% testovacích křižovatek. Přibližně pro dvě třetiny testovacích dat tedy aplikace zajišťuje alespoň 70% úspěšně rozpoznaných prvků křižovatky.



Obrázek 33: Graf úspěšnosti rozpoznávání

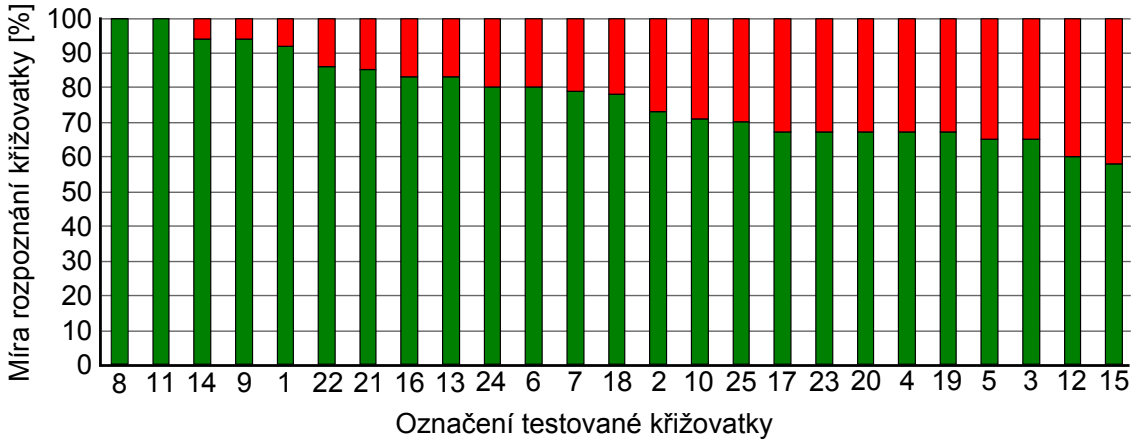
Výsledky tohoto testování poskytují pouze informaci o schopnosti algoritmu rozpoznat viditelné prvky na křižovatce, nikoliv o schopnosti programu usnadnit práci uživatele. Proto bylo

provedeno další testování, které bylo zaměřené přímo na výstupní xml soubor se strukturovanou křižovatkou. Hlavní důraz byl kladen na počet manuálních úprav, které musí uživatel provést, aby výstupní soubor obsahoval plně rozpoznanou křižovatku. Bodování probíhalo obdobně jako v předchozím testování, bylo ale potřeba pozměnit bodovací tabulku z Kapitoly 9.3.1 (viz Tabulka 6).

Objekt	**Body**
Správně rozpoznané rameno křižovatky	+1
Správně rozpoznaný řadicí pruh	+1
Správně rozpoznaný směr řadicího pruhu	+1
Správné určení počtu výstupních pruhů ramene	+1
Chybná detekce směru pruhu nebo počtu výstupních pruhů	0
Falešná detekce ramene křižovatky	-1

Tabulka 6: Upravené bodování rozpoznávání křižovatky

Toto testování hodnotí detekci všech ramen křižovatky (i těch, která nejsou viditelná), všech pruhů na křižovatce (i těch, které nemají směrové šipky, nebo jsou zcela zakryty auty či jinými objekty) a jejich směrů. Stává se tak odrazem funkčnosti aplikace z pohledu uživatele. Výsledky testování zachycuje graf na Obrázku 34. Zeleně je vyznačena úspěšnost rozpoznání křižovatky, tedy procento správných informací vyplněných ve výstupním souboru. Červeně je pak zobrazen doplněk, tedy procento informací, které musí uživatel sám doplnit nebo opravit.



Obrázek 34: Úspěšnost aplikace z pohledu uživatele

Z grafu je patrné, že aplikace poskytla pro polovinu testovacích křižovatek správné rozpoznání křižovatky z alespoň 75%. Rozpoznání ostatních testovacích křižovatek poskytlo většinou alespoň 60% správných informací o křižovatce.

Analýzou obou grafů zjišťujeme, že rozpoznání vysokého počtu prvků v křižovatce není zárukou pro dobré rozpoznání křižovatky. Například křižovatka s označením 12 byla v úspěšnosti rozpoznaných prvků křižovatky na 3. místě, ale z pohledu úspěšnosti rozpoznávání uživatele se umístila na předposledním místě s pouze 60% správně rozpoznaných informací o křižovatce.

Analýzou grafu úspěšnosti z pohledu uživatele také získáváme představu o vhodnosti rozpoznávané křižovatky pro aplikaci. Pokud porovnáme pět křižovatek s nejvyšší úspěšností s pěti

křižovatkami s nejnižší úspěšností, zjišťujeme, že největší překážkou úspěšného rozpoznání křižovatky je detekce přímých směrových šipek. Přímých šipek je rozpoznáno minimum, jelikož jejich členitost je minimální. Pokud navíc zkoumáme starší šipku, která má zašlé hrany a rohy, jeví se nám spíše jako část přerušované dělicí čáry. Aplikace navíc klade pro rozpoznání přímé směrové šipky přísnější podmínky z důvodu zamezení falešných detekcí okolních objektů, které ze všech možných šipek nejčastěji připomínají právě přímou směrovou šipku. Dále je pro aplikaci samozřejmě překážkou detekce pruhů, které jsou z většiny zakryty automobily a jinými objekty. Obrázek 35 zobrazuje křižovatku, která v testování dopadla nejlépe (označení 8) a nabízí porovnání s křižovatkou, která dopadla nejhůře (označení 15).

Obrázek 35: Nejvíce (nahoře) a nejméně (dole) rozpoznaná křižovatka

Z výše popsaného vyplývá, že nejvíce vyhovujícími křižovatkami pro rozpoznávání jsou křižovatky v nejvyšším možném rozlišení, které v ramenech obsahují jasně viditelné směrové šipky (tzn. šipky nejsou z části překryty jiným objektem, ani stínem) a v nichž je počet přímých směrových šipek minimální. Pro správnou detekci směrových pruhů je třeba volit takové křižovatky, jejichž ramena nejsou zahnutá (tedy šipky stejných řadicích pruhů leží na přímce). Jak bylo popsáno v Kapitole 2, nevhodnými jsou mimoúrovňové křižovatky a křižovatky s připojovacími pruhy (např. dálniční křižovatky).

10 Závěr

Diplomová práce měla za úkol vytvořit aplikaci umožňující rozpoznání křižovatky na základě satelitního snímku. Bylo tedy třeba prozkoumat známé metody rozpoznávání obrazu, prozkoumat dostupné nástroje, které můžeme použít k implementaci metod, vytvořit aplikaci umožňující rozpoznání křižovatky ze satelitního snímku a výstupy aplikace otestovat.

V teoretické části práce jsme se zaměřili na analýzu jednotlivých prvků křižovatky a definovali jsme jejich vlastnosti, které jsou vhodné pro detekci. Dále jsme si vysvětlili nejrůznější metody z oblasti zpracování obrazu a rozpoznávání, jejichž reálné použití v práci jsme posléze zvážili. Probrali jsme komerční i volně dostupné nástroje a vyzdvihli jsme jejich výhody a nevýhody.

V praktické části práce jsme se vrátili k analýze probraných metod zpracování obrazu a rozpoznávání a hodnotili jsme jejich přínos při reálném užití v aplikaci. Na základě této analýzy jsme navrhli doporučený postup řešení, který vede k rozpoznání křižovatky. Dále jsme se zaměřili na analýzu aplikace z implementačního hlediska. Aplikaci jsme budovali pro objektově orientované jazyky a navrhli jsme možnou implementaci rozdělenou do čtyř balíků odlišených specifickou funkčností. Po implementaci aplikace jsme provedli testování výstupů programu a tyto výstupy jsme zhodnotili.

Závěrečné testy ukázaly, že aplikace poskytuje pro polovinu testovacích křižovatek alespoň 75% úspěšnost rozpoznání křižovatky. Pro druhou polovinu testovacích křižovatek byla úspěšnost rozpoznávání alespoň 60%. Na základě analýzy testovacích křižovatek a jejich úspěšnosti rozpoznání byl stanoven popis vhodných křižovatek pro rozpoznání aplikací.

Prostor pro zlepšení této práce je eliminování závislosti rozpoznávání na přítomnosti směrových šipek v rameni křižovatky. Toho by se mohlo docílit vhodným spojení postupů logistické regrese pro detekování pixelů vozovky a Houghovy transformace pro detekci ramen křižovatky a dalších prvků na detekované vozovce.

Práce byla vypracovávána průběžně po celý rok a všechny body zadání byly splněny.

Seznam obrázků

Seznam tabulek

Reference

[1] MINISTERSTVO DOPRAVY. Zásady pro vodorovné dopravní značení na pozemních komunikacích [online]. 2. vyd. Říjen 2005 [cit. 2014-04-05]. ISBN 80-86502-25-2. Dostupné z: http://www.ibesip.cz/data/web/kampane/legislativa/besip-04-TP_133_2vydani.pdf

[2] EKŠTEIN, Kamil. Přednášky předmětu Teorie kognitivních systémů. Západočeská univerzita, Fakulta aplikovaných věd. 2012. Dostupné z: http://amos.fav.zcu.cz/tks-prelim

[3] DUDA, R. O. a P. E. HART. Use of the Hough Transformation to Detect Lines and Curves in Pictures, Comm. ACM. Leden 1972, Vol. 15, s. 11-15.

[4] CANNY, John. A Computational Approach to Edge Detection. Listopad 1986, Vol. PAMI-8, č. 6.

[5] LOWE, David G. Distinctive Image Features from Scale-Invariant Keypoints. Vancouver, B.C., Canada, Leden, 2004.

[6] LINDEBERG, T. Scale-space theory: A basic tool for analysing structures at different scales. In: Journal of Applied Statistics. 1994, s. 224-270.

[7] BAY, H., T. TUYTELAARS a L. V. GOOL. SURF: Speeded up robust features. Lecture Notes in Computer Science. Květen 2006, č. 3951, s. 404-417.

[8] SUZUKI, Satoshi a Keiichi ABE. Topological Structural Analysis of Digitized Binary Images by Border Following. In: Computer Vision, Graphics, and Image Processing. 30. vyd., 1985, s. 32-46.

[9] JOBLOVE, G.H. a D. GREENBERG. Color spaces for computer graphics. In: Computers Graphics. 12. vyd., 1987, s. 20-27.

[10] POYNTON, Charles. Digital Video and HDTV: Algorithms and Interfaces. 2. vyd. USA: Morgan Kaufmann Publishers, 2012.

[11] ŽELEZNÝ, Miloš. Přednášky předmětu Zpracování digitalizovaného obrazu. Západočeská univerzita, Katedra kybernetiky. 2013. Dostupné z: http://www.kky.zcu.cz/uploads/courses/zdo/ZDO_aktual_130215.pdf

[12] OTSU, Nobuyuki. A Threshold Selection Method from Gray-Level Histograms. In: IEEE Transactions on Systems, Man, and Cybernetics. Roč. 9. č.1, Leden 1979, s. 62-66.

[13] ROSENFELD, Azriel a Avinash C KAK. Digital Picture Processing. 1. vyd. London: ACADEMIC PRESS INC. (LONDON) LTD., 1982. 2. ISBN 0-12-597301-2.

[14] TOUSSAINT, Godfried. Solving Geometric Problems with the Rotating Calipers. In: Proceedings of IEEE MELECON'83. Athény, Řecko, Květen 1983.

[15] ROSENFELD, A. a J. L. PFALTZ. Distance Functions in Digital Pictures. In: Pattern Recognition. 1. vyd. Velká Británie: Pergamon Press, 1968, s. 33-61.

[16] KUNTTU, Iivari, Leena LEPISTÖ, Juhani RAUHAMAA a Ari VISA. Binary Histogram in Image Classification for Retrieval Purposes. In: Journal of WSCG. 11, č 1. Plzeň, ČR: Science Press, Únor 2003. ISSN 1213-6972.

[17] ESHOME, Mikiyas, Legesse ZERUBABEL a Kim Dong YOON. A simple binary image similarity matching method based on exact pixel matching. International Conference on Computer Engineering and ApplicationsIPCSIT. 2011.

[18] THE MATHWORKS, Inc. MATLAB®: Getting Started Guide [online]. 17. vyd. Září 2011 [cit. 2014-04-13]. Dostupné z: www.mathworks.com

[19] GNU Octave. GNU. Operační systém GNU [online]. [cit. 2014-04-13]. Dostupné z: http://www.gnu.org/software/octave/index.html

[20] Image Processing Toolbox. MATLAB. Domovská stránka Matlab [online]. [cit. 2014-04-13]. Dostupné z: http://www.mathworks.com/products/image/

[21] Computer Vision System Toolbox. MATLAB. Domovská stránka Matlab [online]. [cit. 2014-04-13]. Dostupné z: http://www.mathworks.com/products/computer-vision/

[22] Intel® Integrated Performance Primitives (Intel® IPP). Intel® Domovská stránka [online]. [cit. 2014-04-13]. Dostupné z: https://software.intel.com/en-us/intel-ipp

[23] BRADSKI, Gary a Adrian KAEHLER. Learning OpenCV. 1. vyd. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2008. ISBN 978-0-596-51613-0.

[24] FAN, Rong-En, Kai-Wei CHANG, Cho-Jui HSIEH, Xiang-Rui WANG a Chih-Jen LIN. LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning. 2008, č. 9.

[25] PCL: Point Cloud Library [online]. [cit. 2014-04-25]. Dostupné z: http://www.pointclouds.org

A Uživatelská dokumentace

V této příloze bude popsána instalace aplikace na uživatelský počítač, postup spuštění aplikace, ovládání aplikace a budou vysvětleny důležité konfigurační parametry pro přizpůsobení aplikace.

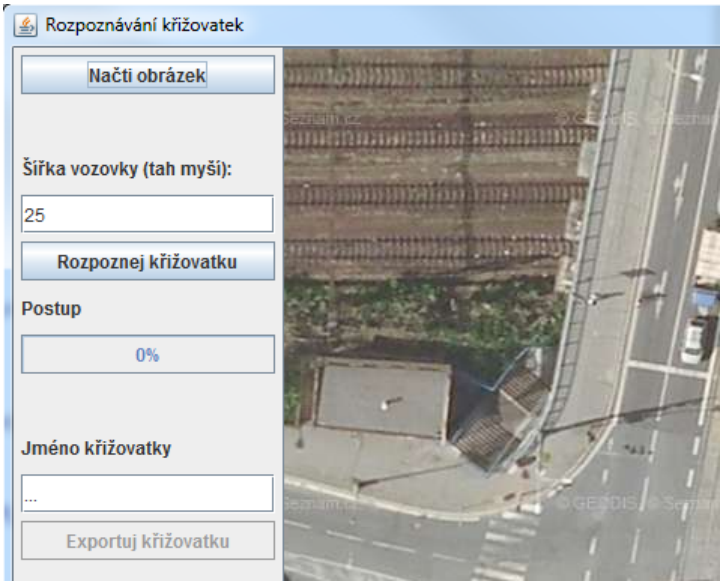
A.1 Instalace a spuštění aplikace

K běhu aplikace je vyžadována instalace knihoven OpenCV. Instalační soubor je možné nalézt na oficiálních stránkách Opencv (`http://opencv.org/downloads.html`) pro operační systémy Windows, Linux, MacOS a Android. Pro správnou funkci je zapotřebí stáhnout verzi 2.4.5 z důvodu správné kooperace s rozhraním JavaCV. V systému Windows může být zapotřebí nastavit cestu k instalaci OpenCV do systémových proměnných. Dále je nutné, aby byl v počítači nainstalován jazyk Java (stažení z: `https://www.java.com/en/download/`) alespoň ve verzi 1.6.

Následně je možné aplikaci spustit. To provedeme jednoduše otevřením souboru `CrossroadRecognition.bat`. Pokud byla instalace výše zmíněných nástrojů úspěšná, mělo by se nám po otevření souboru zobrazit okno se základními ovládacími prvky aplikace.

A.2 Ovládání aplikace

Ovládání aplikace je velice jednoduché a intuitivní. Celý program je ovládán tlačítky umístěnými v panelu v levé části obrazovky. Rozpoznávaná křižovatka je pak v pravé části aplikace. Rozložení ovládacích prvků je zobrazeno na Obrázku A.1. Uživateli jsou vždy zpřístupněny pouze aktuálně dostupné akce, vzhledem k právě prováděné akci, čímž je zajištěn správný běh aplikace.



Obrázek A.1: Ovládací prvky aplikace

Tlačítko *Načti obrázek* otevře manažer pro načítání obrázků. Uživatel má možnost filtrovat zobrazení souborů na PNG nebo JPG obrázky. Pokud je výběr obrázku v pořádku, zobrazí se v pravé části aplikace.

Před začátkem rozpoznávání křižovatky je doporučeno nastavit hodnotu *Šířka vozovky*. Tato hodnota je přednastavena na 25 z důvodu největšího zastoupení této šířky v dostupných satelitních snímcích. Nová šířka vozovky lze zadat přímo do pole, nebo lze změřit tahem myši na obrázku s křižovatkou v pravé části aplikace. Šířka vozovky musí být celočíselná.

Tlačítko *Rozpoznej křižovatku* provede samotné rozpoznávání křižovatky v načteném obrázku. Postup rozpoznávání zobrazuje v procentech grafický prvek pod tlačítkem. Rozpoznávání je kdykoliv možné přerušit tlačítkem *Zruš rozpoznávání*, které nahrazuje tlačítko *Rozpoznej křižovatku* po dobu rozpoznávání.

Uživatel může vyplnit pole *Jméno křižovatky*, do kterého může zapsat řetězec, který bude vyplněn jako atribut název křižovatky v exportovaném XML souboru.

Tlačítko *Exportuj křižovatku* otevře manažer, který umožňuje uložení křižovatky do strukturovaného XML souboru.

A.3 Konfigurační soubor

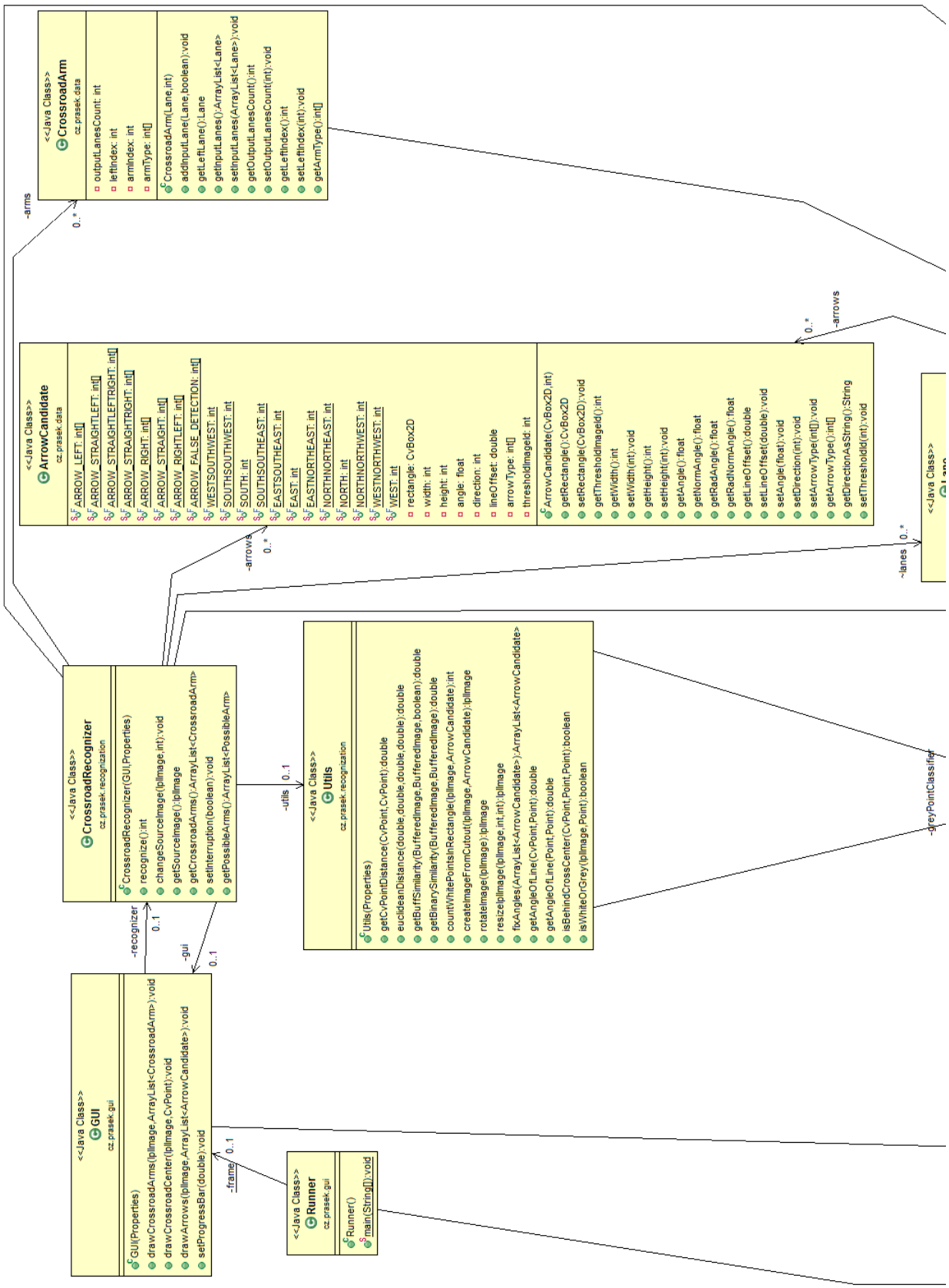
Nastavení jednotlivých parametrů určujících chování programu je uloženo v souboru `config.prop`. Uživatel má možnost přizpůsobit chování programu svým představám změnou těchto parametrů. V první části konfiguračního souboru jsou uloženy veškeré řetězce zobrazované v grafickém uživatelském rozhraní a je možno je přepsat na jiné. V druhé části se nachází konfigurační parametry, jejichž vysvětlení zobrazuje Tabulka A.1.

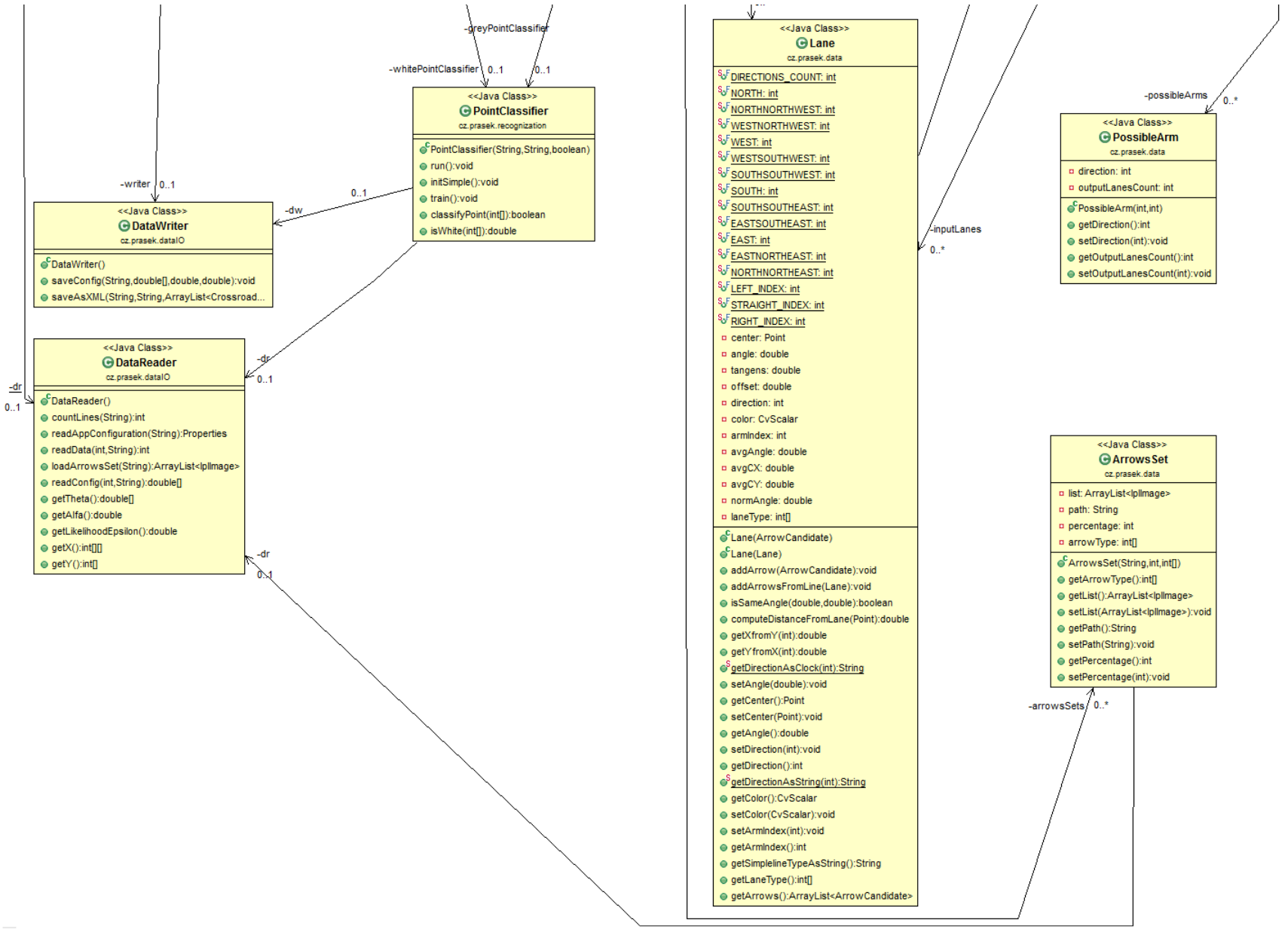
Název parametru	**Význam parametru**	**Hodnoty**
whiteClassifierTraining	Povolení trénování klasifikátoru bílých bodů	true / false
greyClassifierTraining	Povolení trénování klasifikátoru šedých bodů	true / false
laneMaxAngleDiff	Maximální rozdíl úhlů šipek v jednom pruhu	celé číslo (0 - 180)
minWhitepointsPercentage	Minimální povolený počet bílých bodů v kandidátovi na šipku	celé číslo (0 - 100)
maxWhitepointPercentage	Maximální povolený počet bílých bodů v kandidátovi na šipku	celé číslo (0 - 100)
minThreshold	Počáteční prahová hranice pro rozpoznávání	celé číslo (0 - 255)
maxThreshold	Koncová prahová hranice pro rozpoznávání	celé číslo (0 - 255)
minCrossAngle	Minimální rozdíl úhlů šipek pro detekci křížení	celé číslo (0 - 180)
maxCrossAngle	Maximální rozdíl úhlů šipek pro detekci křížení	celé číslo (0 - 180)
maxOutputLanesCount	Maximální počet detekovatelných výstupních pruhů pomocí klasifikátoru šedých bodů	celé kladné číslo

Tabulka A.1: Význam konfiguračních parametrů

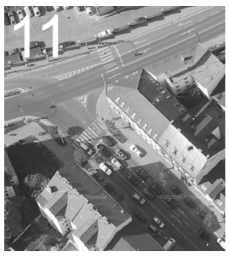
Ostatní konfigurační parametry v souboru představují cesty nebo pomocné parametry se kterými není doporučeno manipulovat bez větší znalosti implementace aplikace.

B Diagram tříd




-greyPointClassifier
-whitePointClassifier
0..1
0..1
<<Java Class>>
PointClassifier
cz.prasek.recognization
PointClassifier(String,String,boolean)
run():void
initSimple():void
train():void
classifyPoint(int[]):boolean
isWhite(int[]):double
-writer
0..1
0..1
-dw
<<Java Class>>
DataWriter
cz.prasek.dataIO
DataWriter()
saveConfig(String,double[],double,double):void
saveAsXML(String,String,ArrayList<Crossroad...
-dr
0..1
-dr
0..1
<<Java Class>>
DataReader
cz.prasek.dataIO
DataReader()
countLines(String):int
readAppConfiguration(String):Properties
readData(int,String):int
loadArrowsSet(String):ArrayList<IplImage>
readConfig(int,String):double[]
getTheta():double[]
getAlfa():double
getLikelihoodEpsilon():double
getX():int[][]
getY():int[]
-dr
0..1
<<Java Class>>
Lane
cz.prasek.data
DIRECTIONS_COUNT: int
NORTH: int
NORTHNORTHWEST: int
WESTNORTHWEST: int
WEST: int
WESTSOUTHWEST: int
SOUTHSOUTHWEST: int
SOUTH: int
SOUTHSOUTHEAST: int
EASTSOUTHEAST: int
EAST: int
EASTNORTHEAST: int
NORTHNORTHEAST: int
LEFT_INDEX: int
STRAIGHT_INDEX: int
RIGHT_INDEX: int
center: Point
angle: double
tangens: double
offset: double
direction: int
color: CvScalar
armIndex: int
avgAngle: double
avgCX: double
avgCY: double
normAngle: double
laneType: int[]
Lane(ArrowCandidate)
Lane(Lane)
addArrow(ArrowCandidate):void
addArrowsFromLine(Lane):void
isSameAngle(double,double):boolean
computeDistanceFromLane(Point):double
getXfromY(int):double
getYfromX(int):double
getDirectionAsClock(int):String
setAngle(double):void
getCenter():Point
setCenter(Point):void
getAngle():double
setDirection(int):void
getDirection():int
getDirectionAsString(int):String
getColor():CvScalar
setColor(CvScalar):void
setArmIndex(int):void
getArmIndex():int
getSimplelineTypeAsString():String
getLaneType():int[]
getArrows():ArrayList<ArrowCandidate>
-inputLanes
0..*
-possibleArms
0..*
<<Java Class>>
PossibleArm
cz.prasek.data
direction: int
outputLanesCount: int
PossibleArm(int,int)
getDirection():int
setDirection(int):void
getOutputLanesCount():int
setOutputLanesCount(int):void
<<Java Class>>
ArrowsSet
cz.prasek.data
list: ArrayList<IplImage>
path: String
percentage: int
arrowType: int[]
ArrowsSet(String,int,int[])
getArrowType():int[]
getList():ArrayList<IplImage>
setList(ArrayList<IplImage>):void
getPath():String
setPath(String):void
getPercentage():int
setPercentage(int):void
-arrowsSets
0..*

C Přehled testovacích křižovatek




16


21


17


22


18


23


19


24


20


25

D Obsah přiloženého CD

Text diplomové práce v elektronické podobě

- Umístěno ve složce: `Text`

Instalační soubory

- OpenCV 2.4.5
- Java 1.6
- Umístěno ve složce: `Instalace`

Aplikace rozpoznávání křižovatky

- Spustitelný přeložený kód aplikace
- Zdrojové soubory aplikace
- Konfigurační soubory
- Porovnávací množiny dat
- Umístěno ve složce: `Aplikace`