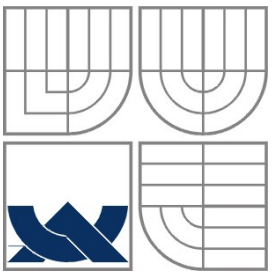
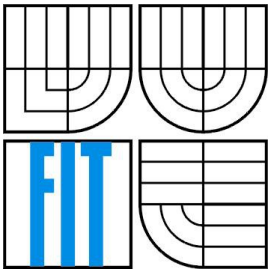


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO MĚŘENÍ ODSTUPU OD PŘEDCHOZÍHO VOZIDLA V PROVOZU

MOBILE APP FOR MEASURING THE RANGE FROM THE PRECEDING VEHICLE IN TRAFFIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Andrii Henry

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. Adam Herout, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá vývojem mobilní aplikace na platformě Android pro měření vzdálenosti k předchozímu vozidlu v provozu vizuální cestou. Byla řešena problematika implementace algoritmů počítačového vidění pro detekci a sledování objektů v obraze, pro detekci horizontu a měření vzdálenosti vizuální cestou bez použití jakýchkoliv jiných pomůcek na desktopových a mobilních zařízeních. Výstupem práce jsou detektory vozidel a horizontu implementované pomocí knihovny OpenCV na platformě Windows a návrh mobilní aplikace demonstrující přijatelné uživatelské rozhraní.

Klíčová slova

OpenCV, počítačové vidění, adaptivní řízení vozidel, měření vzdálenosti vizuální cestou, mobilní zařízení, Android, detekce objektů, sledování objektů, detekce horizontu.

Abstract

This master's thesis deals with development of mobile app for measuring the range from the preceding vehicle in traffic using visual-based methods. This paper describes implementation of computer vision algorithms of detection and tracing objects, detection of horizon on desktop and mobile devices. Also deals with visual-based range measuring without any other mechanisms. The output of the work is implemented detectors of vihicles and horizon using OpenCV library on the Windows platfom and draft of user inerface of a mobile phone aplication on the Android platform.

Keywords

OpenCV, computer vision, adaptive cruise control, visual-based range measuring, mobile devices, Android, object detection, object tracing, horizon detection.

Citace

Andrii Henry: Mobilní aplikace pro měření odstupu od předchozího vozidla v provozu, **diplomová práce, Brno, FIT VUT v Brně,** 2015

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Andrii Henry

Obsah

1 Úvod **5**

2 Úvod do problematiky **6**

2.1 Digitální kamery . . . 6

2.2 Rozpoznávání aut v obraze . . . 7

2.2.1 Haarovy příznaky . . . 8

2.2.2 Kaskádové klasifikátory . . . 9

2.2.3 Sledování objektů v obraze . . . 9

2.3 Výpočet vzdálenosti vizuální cestou . . . 16

2.3.1 Vzdálenost . . . 16

2.3.2 Odhad vzdálenosti . . . 17

2.3.3 Invariance výsledků výpočtu vůči poloze zařízení . . . 18

2.3.4 Očekávaná přesnost . . . 22

2.4 Mobilní aplikace . . . 23

2.4.1 Platforma Android . . . 23

2.4.2 Použité API . . . 26

3 Rozpoznávání aut **29**

3.1 Popis vstupu a sběr dat . . . 29

3.2 Implementovaná metoda rozpoznávání aut . . . 30

3.2.1 Přístup rozpoznávání bez klasifikátoru . . . 30

3.2.2 Rozpoznávání s klasifikátorem . . . 33

3.2.3 Spojení metody bez klasifikátoru a s ním . . . 34

3.2.4 Výstup detektoru aut . . . 35

3.3 Detekce horizontu . . . 35

3.4 Rychlost zpracování . . . 38

4 Realizace mobilní aplikace **40**

4.1 Kalibrační mód . . . 40

4.1.1 Princip kalibrace . . . 40

4.1.2 Návrh módu . . . 41

4.2 Demonstrační mód . . . 42

4.2.1 Práce se vstupními daty . . . 42

4.2.2 Návrh uživatelského rozhraní . . . 43

5 Testování **47**

5.1 Experimenty v kalibračním módu . . . 47

5.2 Testování přesnosti odhadu vzdálenosti . . . 48

6 Závěr **49**

7 Seznam příloh **54**

1 Úvod

Bezpečnost na cestách je pro každého řidiče čím dál více aktuální problém. V dnešní době se díky vývoji automobilového průmyslu zvyšují bezpečnostní standardy a řidiči se chtějí cítit bezpečně i při vyšších rychlostech. Ne každý si však může dovolit auto s integrovaným moderním bezpečnostním systémem, který je velice často jeho nejdražší součástí. Tato diplomová práce se bude zabývat možností určování vzdálenosti k předchozím vozidlům vizuální cestou bez použití jiných senzorů jako jsou laser, sonar atd.

Cílem práce je vypracovat algoritmus detekce vozidel v obrazu z kamery mobilního zařízení a navrhnout algoritmus výpočtu vzdálenosti k detekovaným autům. Následně navrhnout a implementovat mobilní aplikaci, která by uměla detekovat auta v provozu a kvalitně odhadnout vzdálenost k nim.

V následujících kapitolách je tento proces popsán podrobněji. V kapitole 2 si více specifikujeme řešený problém a probereme teoretické nástroje, které v řešení problému pomůžou. V kapitole 3 je popsaná realizace detektoru aut a horizontu pomocí knihovny OpenCV v operačním systému Windows. Dále v kapitole 4 naleznete popis návrhu mobilní aplikace a postup její implementace v operačním systému Android. V kapitole 5 jsou shrnuty informace o testování výsledných aplikací jak v reálných podmínkách, tak i reálnými uživateli.

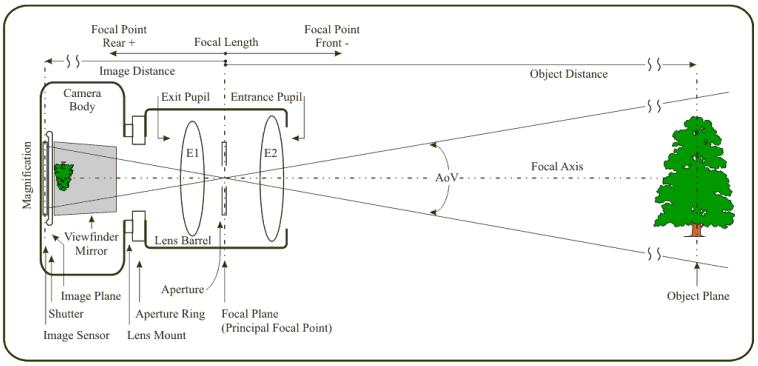
2 Úvod do problematiky

Lidský systém vnímání vizuálních informací o světě se skládá ze dvou receptorů — očí. Oči se nachází v určité vzdálenosti od sebe, průměrně 6-8 cm [2]. Jsou takto umístěny, aby dle principů stereovize člověk mohl odhadnou vzdálenosti od okolních objektů. Ve skutečnosti člověk přirozeně neumí kvalitně odhadnout vzdálenost k předmětům, které jsou vzdálené více než několik jednotek metrů. I přesto člověk dokáže být dobrým řidičem motorových vozidel nehledě na to, že pro bezpečnou jízdu potřebuje odhadnout vzdálenosti v intervalu od 1 do cca 200 metrů. Člověk dokonce může legálně získat řidičský průkaz na území České republiky i v případě, že má pouze jedno funkční oko [3], to znamená, že i s jedním okem člověk může řídit bezpečně. Je patrné, že z vlastních zkušeností dle zákonů perspektivy umí odhadnout vzdálenost dost kvalitně. Každý intuitivně chápe, že čím je objekt dál, tím menším se zdá být.

V této práci se objasňuje, zda je možné využít pro určení vzdálenosti objektů z obrazu kamery podobné principy, které využívá lidské oko, a to bez použití dalších zařízení pro měření vzdálenosti. V této kapitole je shromážděn dostatečný teoretický základ, který pomůže pochopit danou problematiku, a na kterém je stavěná realizace řešení specifikovaného problému. Podíváme se na možnosti digitálních kamer, které jsou nainstalované v mobilních zařízeních, dále probereme problematiku rozpoznávání aut v obraze. Dalším milníkem této kapitoly budou základy perspektivy, které pomohou pochopit princip odhadu vzdálenosti, a na závěr tu naleznete informace o zvolené mobilní platformě a další potřebné informace, které pomohou pochopit podrobnosti při implementaci mobilní aplikace.

2.1 Digitální kamery

Optické a digitální kamery jsou fakticky napodobeninou lidského oka. Světlo vstupuje do kamery skrz vstupní čočku, potom prochází clonou a přes výstupní čočku dopadá na film nebo senzor. Vzdálenost od senzoru ke cloně se nazývá fokální vzdálenost. Podrobněji je to vidět na Obrázku 1. [4]



Obrázek 1: Vnitřní ústrojí kamery [4]

Kamery v mobilních zařízeních fungují na stejném principu. Hlavním parametrem, který určuje kvalitu obrazu, je velikost a kvalita snímacího senzoru. Čím větší je senzor, tím víc světla má kamera možnost zachytit. Čím víc světla, tím je potenciálně i vyšší kvalita. Nezáleží tedy tolik na počtu pixelů senzoru, ale na jeho opravdové fyzické velikosti. Toto představuje problém pro kamery v mobilních zařízeních. Kvůli omezenému fyzickému prostoru v zařízení a reálným cenovým možnostem není možné do nich nainstalovat větší senzor [6]. V tabulce 1 jsou uvedené velikosti senzorů některých známých modelů mobilních telefonů, kompaktních fotoaparátů a profesionálních kamer.

Model	**Velikost senzoru [mm×mm]**	**Počet pixelů [Mp]**
Samsung Galaxy S III	4.54×3.42	8
Samsung Galaxy S4	4.69×3.53	13
HTC One X	4.54×3.42	8
Apple iPhone 5S	4.89×3.67	8
Samsung Galaxy Ace II	4.80×3.60	5
Fujifilm X-S1	8.80×6.60	12
Canon EOS 5D Mark II	36.00×24.00	22

Tabulka 1: Rozměry senzorů různých zařízení [7]

Jak je patrné z tabulky, přístroje, jejichž hlavní funkcí je focení, mají výrazně větší senzor. U všech mobilních zařízení je velikost senzoru přibližně stejná i přes to, že počet pixelů výsledného obrazu se odlišuje. Další faktor, který výrazně ovlivňuje výsledek, je kvalita senzoru.

Pochopitelně dražší zařízení mají kvalitnější a tím pádem i dražší senzor. Například kamera modelu Samsung Galaxy S III stojí přibližně 15 $, kvalita fotografií a cena je srovnatelná s mobilním telefonem iPhone 4, jeho kamera stojí přibližně 18 $. Na druhou stranu cena kamer novějších modelů stejných značek, Galaxy S4 a iPhone 5S, je přibližně 40 $. Víme, že kvalita obdržených snímků z těchto senzorů je poměrně vyšší než u jejich předchůdců. Informace o cenách je použita dle [8].

Kvalita vizuálního vstupu je jedním z nejdůležitějších předpokladů pro kvalitní odhad vzdálenosti založený na rozpoznání objektů v obraze. Z toho vyplývá, že existuje vysoká pravděpodobnost, že kvalita obrazu z levnějších zařízení nebude dostačující pro správnou detekci aut a tím pádem i odhad vzdálenosti nebude možné provést.

2.2 Rozpoznávání aut v obraze

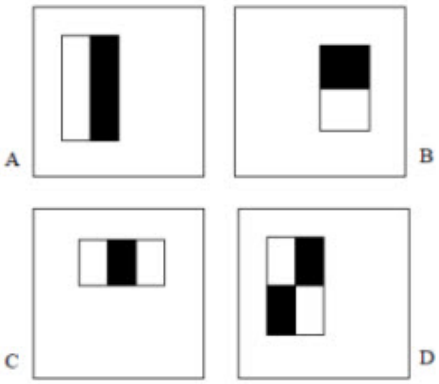
V této kapitole se budeme věnovat problematice detekce objektů v obraze, konkrétně aut. Základní používanou metodou v dnešní době jsou kaskádové klasifikátory založené na Haarových příznacích. Metoda byla poprvé navrhnuta Paulem Violou a Michaelem Jonesem

v roce 2001 [9]. Nejdříve se podíváme na Haarovy příznaky, potom podrobněji rozebereme, jak fungují kaskádové klasifikátory. Při implementaci jsem také použil metodu nalezení a sledování významných bodů v obraze Good Features to Track. Metoda byla navržená Jianbo Shi a Carlo Tomasi a poprvé prezentovaná na konferenci počítačového vidění (CVPR94) v roce 1994 [12]. Tato metoda je také popsána dále v této kapitole.

2.2.1 Haarovy příznaky

Používání příznaků namísto samotných pixelů přináší řadu výhod. V první řadě jde o to, že příznaky mohou využít konkrétní znalost typu detekovaného objektu k úspěšnějšímu rozpoznání. Například lze využít předpokladu, že světla vozidel jsou zpravidla světlejší, než okolní prostředí. Navíc použití příznaků je mnohem rychlejší, což je pro řadu problémů, včetně daného, velice žádané.

Na obrázku 2 je příklad příznaků, s jejichž pomocí se dají odhalit jejich hodnoty pro konkrétní obraz. Princip výpočtu je následující: konkrétní obdélníky se přiloží na místo v obraze, o kterém se ví, že by se v něm měl vyskytnout příznak. Spočítá se součet intenzit pixelů ležících pod bílým obdélníkem a odečte se od součtu intenzit pixelů pod černým obdélníkem. Počet obdélníků může být různý, také mohou být různě natočené. Na obrázku 2 jsou dvou-obdélníkové (A, B), tří-obdélníkové (C) a čtyř-obdélníkové (C) příznaky.[9]



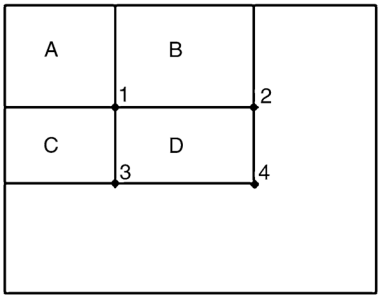
Obrázek 2: Příklad Haarových příznaků [9]

Pro dodatečné zrychlení výpočtu navrhli Viola a Jones v [9] novou metodu reprezentace obrazu, tzv. integrální obraz. Velikost integrálního obrazu je stejná jako původního. Hodnota integrálního obrazu na pozici *x,y* je součtem intenzit pixelů nahoru a vlevo od bodu *x,y* včetně daného bodu. Výpočet tedy probíhá pomocí vztahu:

$$II(x,y) = \sum_{a \leq x} \sum_{b \leq y} I(a,b) \tag{1}$$

Integrální obraz se spočítá jednou na začátku, poté se dá součet intenzit pixelů jakéhokoli obdélníku v obraze jednoduše spočítat jen pomocí čtyř zásahů do pole. Tento princip je

dobře vidět na obrázku 3, hodnota integrálního obrazu v bodě 1 je součet intenzit pixelů obdélníku A, hodnota v bodě 2 je $A + B$, hodnota v bodě 3 je $A + C$ a hodnota v bodě 4 je $A + B + C + D$. Tím pádem se součet intenzit pixelů v obdélníku D rovná $4 + 1 - 2 - 3$.



Obrázek 3: Integrální obraz

2.2.2 Kaskádové klasifikátory

Obyčejný klasifikátor je komponenta, která umí s určitou pravděpodobností určit, zda v nějaké oblasti na obraze je nebo není přítomný hledaný objekt. Nejdříve je klasifikátor trénován několika stovkami vzorových vzhledů příslušných objektů (v tomto případě auta), která jsou označována jako pozitivní vzory. Tyto vzory jsou transformovány na stejnou velikost. Klasifikátor je také trénován negativními vzory (obrázky stejné velikosti).

Po trénování se klasifikátor aplikuje na oblasti stejné velikosti, stejně jako vzory při trénování. Klasifikátor vrátí buď 1, pak se nejpravděpodobněji hledaný objekt v této oblasti vyskytuje, nebo 0, to pak klasifikátor tvrdí, že v dané oblasti hledaný objekt není nalezen. Standardně se okno klasifikátoru postupně posouvá přes celý obraz a v každé oblasti je zjišťována odpověď klasifikátoru. Klasifikátor je navržen tak, aby jeho velikost byla jednoduše měnitelná, aby mohl být schopný najít objekty různých velikostí v dané oblasti zájmu. Klasifikátor prochází několikrát obrazem v různých měřítkách tak, aby detekoval objekty různých velikostí.

Kaskádový klasifikátor funguje na principu sekvence jednoduchých klasifikátorů. Pokud první klasifikátor dává kladnou odpověď, pak rozhoduje druhý klasifikátor, jestli i jeho odpověď je kladná, předává ji třetímu, atd. V případě, že se sekvence přeruší zápornou odpovědí, kaskádový klasifikátor vrátí hodnotu 0. Když rozhodovací proces dojde až k poslednímu klasifikátoru a ten odpoví kladně, pak kaskádový klasifikátor vrací hodnotu 1. Jednotlivé prosté klasifikátory využívají Haarovy příznaky, které jsou popsány v předchozí kapitole. [10]

2.2.3 Sledování objektů v obraze

Základní ideou aplikování tohoto postupu pro řešení problému detekce aut vychází ze znalosti kontextových informací o vstupní sekvenci obrazů. O vstupu víme, že je to záběr

z kamery namontované za čelním sklem auta směrem ven, tím pádem ve scéně můžeme nalézt následující typy objektů:

- Objekty spojené s terénem;
- nebe (mraky, slunce atd.);
- silnice;
- vozidla.

Zajímají nás vozidla, proto bychom měli umět zbytek scény odfiltrovat. Do terénu zahrnujeme: stromy, keře, budovy, atd. Jelikož se kamera ve scéně pohybuje dopředu, objekty terénu postupně mizí a místo nich se objevují další. Pokud bychom bod nacházející se v oblasti terénu začali sledovat, zjistili bychom, že se relativně rychle pohybuje od středu obrazu k jeho okraji. Tak bychom mohli terén odfiltrovat.

Nebe je zpravidla obarvené menším množstvím barev než zbytek obrazu a navíc o něm víme, že se nachází v horní části obrazu, proto odstranění nebe ze scény není principiálně těžká záležitost.

Silnice je, podobně jako nebe, obarvená zpravidla užším spektrem barev, proto existuje jen velmi malá pravděpodobnost, že významný bod se vyskytne na povrchu silnice. Nicméně i v případě, že se to stane, tento bod se zachová podobně jako bod terénu - rychle zmizí za okrajem obrazu.

Zbývající část scény jsou vozidla. Významné body ležící na vozidle by zůstávaly delší dobu ve scéně a jen mírně by se pohybovaly do všech směrů. Navíc shluky takových bodů bychom mohli detekovat jako vozidla. V případě, že auto, ve kterém je namontovaná kamera, by jelo rychleji než vozidlo vepředu, shluk významných bodů by se postupně zvětšoval a posouval se níž v obraze. V opačném případě by se shluk zmenšoval a posouval se výš (blíž k horizontu).

Výše popsaný princip může při různých okolnostech selhávat, např. v zatáčkách (významné body části terénu v zatáčce se mohou začít chovat jako vozidlo), proto tato metoda samotná nemůže fungovat spolehlivě, ale mohla by se stát dobrým zpřesněním detekce aut s použitím klasifikátoru.

Metody hledání vektoru pohybu se v zásadě rozdělují na dvě kategorie: přímé (pixelově orientované) a metody vycházející z významných bodů v obraze [11]. S ohledem na složitost scény není vhodné používat přímé metody. Zaměříme se víc na metody založené na významných bodech. Intuitivně chápeme, že významné body jsou body, které se od svého okolí nejvíce odlišují nebo tvoří rohy. Vhodné je tedy použít detektor takových bodů a jejich pohyb sledovat porovnáním poloh těchto bodů ve dvou po sobě jdoucích snímcích vstupní sekvence.

Dále podrobněji rozebereme Harrisonův - Stephensův rohový detektor, který je základem algoritmu Good Features to Track [11] [12]. Tento algoritmus je potom probírán

v dalším paragrafu. Pro sledování nalezených významných bodů existuje řada algoritmů. V dalších paragrafech se podíváme podrobněji na algoritmus sledování významných bodů, který poprvé uvedli Bruce D. Lucas a Takeo Kanade v roce 1981 [13].

Harrisonův - Stephensův rohový detektor

Detektor byl navržen v roce 1988 a poprvé prezentován v [14]. Rohy reprezentují změny gradientu v obraze. Pokud hledáme rohy, potom hledáme tyto změny. Nechť vstupní obraz I je ve stupních šedi. Dále ať je okno o velikosti (u, v), které je posouvá podél (x, y). Vážený součet druhých mocnin rozdílu intenzit uvnitř tohoto okna $S(x, y)$ se dá vyjádřit vztahem 2 [14] [11].

$$S(x,y) = \sum_u \sum_v w(u,v) \cdot (I(u+x, v+y) - I(u,v))^2 \tag{2}$$

Ať I_x a I_y jsou parciální derivace I, potom $I(u+x, v+y)$ můžeme aproximovat pomocí Taylorova rozvoje následovně:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v) \cdot x + I_y(u,v) \cdot y \tag{3}$$

Z 3 plyne aproximace:

$$S(x,y) \approx \sum_u \sum_v w(u,v) \cdot (I_x(u,v) \cdot x + I_y(u,v) \cdot y)^2 \tag{4}$$

Vztah 4 se dá převést do maticového tvaru následovně:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix}, \tag{5}$$

kde M je autokorelační matice:

$$M = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{6}$$

Vlastní čísla [15] matice M odráží hladkost okolí zkoumaného bodu. Aby se daný bod mohl považovat za roh, M by měla obsahovat alespoň dvě relativně velká vlastní čísla (λ_1, λ_2). V závislosti na jejich velikosti můžeme vyvodit jeden z následujících závěrů:

1. Pokud $\lambda_1 \approx 0$ a $\lambda_2 \approx 0$, potom bod (x, y) není významným bodem.
2. Pokud $\lambda_1 \approx 0$ a λ_2 má velkou kladnou hodnotu, potom v bodě (x, y) je hrana.
3. Pokud λ_1 a λ_2 mají velkou kladnou hodnotu, potom v bodě (x, y) je nalezen roh. [11]

Výpočet vlastních čísel je operace výpočetně velice náročná, protože vyžaduje výpočet druhé odmocniny. Proto v [14] bylo navrženo použít následující funkci R:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 = det(M) - k \cdot trace^2(M), \tag{7}$$

kde hodnota konstanty k je zjištěná empiricky, v literatuře se doporučuje volit tuto hodnotu v rozsahu $0.04 - 0.15$. Hodnota 0.04 je používaná nejčastěji [11]. Algoritmus tedy ve skutečnosti nepočítá hodnotu vlastních čísel matice M, místo toho se ukázalo, že pro detekci rohů je dostatečné spočítat determinant a stopu (součet prvků na hlavní diagonále) matice M.

Good Features to Track

Algoritmus hledá nejvýraznější body v obraze vhodné ke sledování. Algoritmus má za cíl najít dostatečný počet dostatečně kvalitních bodů. Kontroluje počet nalezených významných bodů v obraze, aby nedocházelo k jeho rapidnímu nárůstu, a tím pádem i ke zvýšení výpočetní náročnosti.

Jak už bylo zmíněno, algoritmus vychází z Harrisonova - Stephensovova detektoru rohů popsaného v předchozím paragrafu. Jediná změna je ve výpočtu hodnoty R, kterou zmíněný rohový detektor vypočítává dle vztahu 7. Good Features to Track používá následující vztah:

$$R = min(\lambda_1, \lambda_2) \tag{8}$$

Vychází ze závěru, že pokud menší vlastní číslo je dostatečně velké, aby prošlo podmínkou ($min(\lambda_1, \lambda_2) > \lambda$, kde λ je nějaký práh), potom je zpravidla tento bod dostatečně stabilní (kvalitní roh). Hodnota R se vypočítává pro každý pixel obrazu, přitom se zaznamenává nejlepší roh v celém obraze, tj. bod s největší hodnotou R. Dále se dle nastaveného prahu kvality vybírá skupina bodů, které jsou dostatečně kvalitní v porovnání s nejlepším rohem. Následně se z těchto bodů vybírá jen určitý předem stanovený počet N_{MAX} nejlepších bodů. Na konci se odstraňují body, které jsou k sobě blíž než zadaná minimální vzdálenost, zůstávají pouze nejsilnější. [12] [11]

Popsaný algoritmus poskytuje kvalitní řešení s poměrně nízkou výpočetní náročností. Dokáže velice efektivně rozhodnout mezi špatným a dobrým významným bodem v obraze. Samozřejmě algoritmus neřeší všechny problémy spojené se sledováním významných bodů, například světlé fleky na lesklém povrchu by algoritmus nejpravděpodobněji zařadil do množiny kvalitních významných bodů přesto, že do ní nepatří.

Sledování významných bodů

Optický tok je zobrazení pohybu objektů, ploch a hran ve scéně, který byl způsoben relativním pohybem pozorovatele (v našem případě kamery) vůči scéně. Existují dva základní přístupy pro sledování tzv. optického toku [11]:

- dense optical flow;
- sparse optical flow.

Metody typu dense optical flow počítají optický tok pro každý bod vstupního obrazu. Výpočetní náročnost takových metod je velmi vysoká, zároveň ale poskytují lepší kvalitu výsledků. V obraze se občas vyskytují jednobarevné plošné objekty, pro které je dost obtížné vypočítat optický tok a výsledek může být nejednoznačný [11].

Algoritmy založené na přístupu sparse optical flow počítají optický tok pouze pro předem definovanou množinu bodů, o kterých se předpokládá, že jsou jednoduše sledovatelné. Tato skupina algoritmů je mnohem rychlejší za cenu nižší kvality [11], i přesto se dá dosáhnout uspokojivých výsledků. Pro řešenou problematiku je velice důležité, aby použitý algoritmus mohl běžet v reálném čase, proto samozřejmě jedinou možností je použit algoritmus ze skupiny sparse optical flow. Vstupem takového algoritmu je množina bodů, které má sledovat. Zpravidla to jsou významné body, které se zjišťují dle postupu popsaného v předchozích dvou paragrafech. Nejlepší algoritmus v této oblasti, který zaručuje nejlepší kvalitu výsledku s přijatelnou výpočetní složitostí, je algoritmus optického toku s rozptýlenými body využívající iterační pyramidový algoritmus Lucas-Kanade [13] [11]. Na tento algoritmus se teď podíváme podrobněji.

Mějme dva obrazy ve stupních šedi I a J. Tyto obrazy jsou reprezentovány maticemi diskrétních hodnot a přistupovat k intenzitám jednotlivých pixelů můžeme takto: $I(x, y)$ a $J(x, y)$ pro bod se souřadnicemi (x, y). Obrazy I a J mají stejnou velikost a úkolem sledování je bod z obrazu I nalézt v obrazu J. Pokud $u = (u_x, u_y)$ odpovídá bodu z obrazu I, potom cílem je najít bod $v = (u_x + d_x, u_y + d_y)$ v obrazu J. Vektor $d = [d_x \quad d_y]^T$ je obrazová rychlost nebo jinak tzv. optický tok [13] [11]. Kvůli tomu, že se může objevit tzv. aperaturní problém (angl. aperture problem) [20], je potřeba počítat také s blízkým okolím bodu: w_x pixelů doleva a doprava, w_y nahoru a dolů. Vektor d je tedy vektor, který minimalizuje níže uvedenou funkci ϵ: [13]

$$\epsilon(d) = \epsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x, y) - J(x + d_x, y + d_y))^2. \tag{9}$$

Výpočet tedy probíhá uvnitř tzv. integračního okna velikosti $(2w_x+1) \times (2w_y+1)$. Typicky hodnoty w_x a w_y se volí v rozmezí 2-7 pixelů. [13]

Původní algoritmus Lucas-Kanade vychází z následujících podmínek [11]:

- konstantní jas;
- relativně malé pohyby;
- prostorová koherence.

Podmínka konstantního jasu znamená, že jednotlivé intenzity stejných bodů (pixelů) se skokově nemění. Zachovává se relativně stejný jas. Jinak řečeno platí následující vztah [11]:

$$I(x(t), t) = I(x(t + \Delta t), t + \Delta t), \tag{10}$$

kde $I(x(t), t)$ je intenzita pixelu x v čase t v obraze I v čase t.

Podmínka malého pohybu znamená, že rychlost změny polohy bodu mezi obrazy je relativně malá. Dají se sledovat změny polohy pouze do velikosti integračního okna. Tato nevýhoda je překonaná modifikací algoritmu pomocí tzv. pyramidové implementace [11]. Více o tom uvedeno v této kapitole.

Podmínka prostorové koherence říká, že by většina okolních pixelů sledovaného bodu měla ležet ve stejné rovině a pohybovat se stejným směrem se stejnou rychlostí.

Pro implementaci rozšíření algoritmu Lucas-Kanade je nutné definovat pojem pyramidová reprezentace obrazu. Jak už bylo zmíněno, toto rozšíření bylo navrženo, aby se daly zachytit větší pohyby bodů než velikost integračního okna. Mějme obraz I velikosti $n_x \times n_y$. Ať obraz na nejnižší nulté úrovni se značí $I^0 = I$. Je to původní obraz s největším rozlišením v nejlepší možné kvalitě. Má tedy velikost $n_x^0 = n_x$ a $n_y^0 = n_y$. Pyramida se potom buduje rekurzivně, tedy z I^0 se vypočítá I^1, z I^1 se vypočítá I^2 atd. Nechť $L \in \mathbb{N}$ je obecná úroveň pyramidy, potom I^{L-1} je obraz na úrovni $L-1$ velikosti $n_x^{L-1} \times n_y^{L-1}$. Intenzita pixelu v obraze I^L se skládá ze tří složek: čtvrtina intenzity odpovídajícího pixelu v obrazu nižší úrovně, osmina součtu intenzit pixelů ve čtyřokolí daného pixelu a šestnáctina součtu intenzit pixelů v osmiokolí daného pixelu vyjma čtyřokolí. Matematický zápis je ve vztazích 11, 12 a 13 [13].

$$u_1(x, y) = \frac{1}{4} I^{L-1}(2x, 2y) \tag{11}$$

$$\begin{aligned} u_2(x, y) = &\frac{1}{8}(I^{L-1}(2x-1, 2y) + I^{L-1}(2x+1, 2y) + \\ &I^{L-1}(2x, 2y-1) + I^{L-1}(2x, 2y+1)) \end{aligned} \tag{12}$$

$$\begin{aligned} u_3(x, y) = &\frac{1}{16}(I^{L-1}(2x-1, 2y-1) + I^{L-1}(2x+1, 2y-1) + \\ &I^{L-1}(2x+1, 2y+1) + I^{L-1}(2x-1, 2y+1)) \end{aligned} \tag{13}$$

Obraz I^L se potom dá vypočítat následovně:

$$I^L(x, y) = u_1(x, y) + u_2(x, y) + u_3(x, y). \tag{14}$$

Výše uvedené rovnice poukazují na použití antialiasingového filtru s jádrem: $\begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix} \times \begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix}^T$. V praxi se používá větší jádro antialiasingového filtru: $\begin{bmatrix} 1/16 & 1/4 & 3/8 & 1/4 & 1/16 \end{bmatrix} \times \begin{bmatrix} 1/16 & 1/4 & 3/8 & 1/4 & 1/16 \end{bmatrix}^T$ [13].

V zavedené notaci je jedna nesrovnalost, funkce pro výpočet obrazu na nové úrovni zasahuje o jeden pixel vedle obrazu. Proto v [13] je uvedeno následující doplnění pro speciální případy, aby funkce prováděla výpočet pouze pro $0 \leq x \leq n_x^{L-1} - 1$ a $0 \leq y \leq n_y^{L-1} - 1$:

$$I^{L-1}(-1, y) = I^{L-1}(0, y)$$

$$I^{L-1}(x,-1) = I^{L-1}(x,0)$$
$$I^{L-1}(n_x^{L-1},y) = I^{L-1}(n_x^{L-1}-1,y)$$
$$I^{L-1}(x,n_y^{L-1}) = I^{L-1}(x,n_y^{L-1}-1)$$
$$I^{L-1}(n_x^{L-1},n_y^{L-1}) = I^{L-1}(n_x^{L-1}-1,n_y^{L-1}-1)$$

Vztah 14 je definován pouze pro takové x a y, že $0 \leq 2x \leq n_x^{L-1}-1$ a $0 \leq 2y \leq n_y^{L-1}-1$. Proto n_x^L a n_y^L jsou největší celá čísla splňující podmínky:

$$n_x^L \leq \frac{n_x^{L-1}+1}{2}, \tag{15}$$

$$n_y^L \leq \frac{n_y^{L-1}+1}{2}. \tag{16}$$

Vztahy 11, 12, 13, 14, 15, 16 se používají pro výpočet pyramidové reprezentace obrazu I: $\{I^L\}_{L=0,...,L_m}$, kde L_m je maximální výškou pyramidy. Tato hodnota se zpravidla volí v rozmezí 2-4. Větší hodnota není pro většinu vstupních obrazů praktická. Zavedení pyramidové reprezentace má za cíl umožnit sledování větších pohybů pixelů než dané integrační okno velikosti $w_x \times w_y$. Proto hodnota L_m se volí v závislosti na maximální očekávané rychlosti obrazu (optický tok). [13]

Dále popíši činnost algoritmu s použitím pyramidové reprezentace obrazu dle [13]. Úkolem je pro daný bod u obrazu I najít jeho odpovídající polohu $v = u + d$ v obraze J. Stačí tedy vypočítat vektor přemístění d. Ať $u^L = [u_x^L \quad u_y^L]$ pro $L = 0, ..., L_m$ jsou souřadnice bodu u v pyramidovém obraze I^L. Dle 11, 12, 13, 14, 15 a 16 se u^L dá vypočítat následovně:

$$u^L = \frac{u}{2^L} \tag{17}$$

Algoritmus probíhá následovně: nejdříve se vypočítá optický tok na nejhlubší úrovni pyramidy L_m. Potom výsledek postupuje do vyšší úrovně $(L_m - 1)$ ve formě počátečního odhadu. Pomocí tohoto odhadu se vypočítá kumulovaný optický tok na úrovni $L_m - 1$ a výsledek se předává vyšší úrovni $(L_m - 2)$. Tak to pokračuje až do úrovně 0. [13]

Popišme teď výše naznačenou rekurzivní operaci mezi dvěma obecnými úrovněmi $L-1$ a L. Nechť počáteční odhad optického toku na úrovni L je $g^L = [g_x^L \quad g_y^L]^T$, který je dostupný z výpočtů na úrovni $L+1$. Pro výpočet optického toku na úrovni L je nutné nalézt pohybový vektor $d^L = [d_x^L \quad d_y^L]^T$, takový, který minimalizuje modifikovanou chybovou funkci: [13]

$$\epsilon^L(d^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-w_x}^{u_x^L+w_x} \sum_{y=u_y^L-w_y}^{u_y^L+w_y} (I^L(x,y) - J(x+g_x^L+d_x^L, y+g_y^L+d_y^L))^2. \tag{18}$$

Integrační okno, jak je vidět, má stejnou velikost na všech úrovních L pyramidy $((2w_x+1) \times (2w_y+1))$. Vektor počátečního odhadu optického toku g^L se používá jen

k předtranslaci, proto konkrétně daný pohybový vektor d^L je malý a je možné ho vypočítat standardním krokem algoritmu Lucas-Kanade.

Po předání vypočtené hodnoty d^L na vyšší úroveň $L-1$, hodnota počátečního odhadu g^{L-1} se vypočítává následovně:

$$g^{L-1} = 2(g^L + d^L). \tag{19}$$

Hodnota d^{L-1} se potom vypočítává, jak bylo řečeno dříve, minimalizací funkce $\epsilon^{L-1}(d^{L-1})$ 18. Takto algoritmus pokračuje až k úrovni $L = 0$. Algoritmus se inicializuje nastavením počátečního odhadu úrovně L_m na nulu ($g^{L_m} = [0 \quad 0]^T$). [13]

Konečná hodnota pohybového vektoru d se vypočítává takto:

$$d = g^0 + d^0. \tag{20}$$

Výhodou tohoto postupu je to, že v každém kroku vypočtená hodnota vektoru d^L je relativně malá i přesto, že probíhá výpočet velké hodnoty přemístění pixelu (d). Pokud v každém kroku můžeme spočítat hodnotu zdvihu pixelu maximálně do d_{max}, potom pomocí pyramidové implementace je možné sledovat pohyb až $d_{MAX} = (2^{L_m+1} - 1)d_{max}$. Například pro hloubku pyramidy $L_m = 3$, maximální sledovatelný zdvih je 15 krát větší než při použití algoritmu Lucas-Kanade ve standardním tvaru. [13] [11]

2.3 Výpočet vzdálenosti vizuální cestou

Většina klasických způsobů měření vzdálenosti spadá do kategorie tzv. přímých metod, které využívají speciální senzory: vlnový radar, laserový radar nebo stereo kamery. Tyto senzory provádí přímé měření vzdálenosti pro každý bod zájmu ve scéně, tím pak velice zjednodušují segmentaci objektů od pozadí. Přímé metody měření vzdálenosti se aktivně používají při řešení problému adaptivního řízení.

V této kapitole se budu zabývat řešením problému měření vzdálenosti pomocí zákonů perspektivy, čili nepřímým způsobem. Víme, že přesnost přímých způsobů je více než dostačující pro adaptivní řízení vozidel, ale existuje dost vysoká pravděpodobnost, že přesnost nepřímých metod bude také postačující a tím se značně sníží cena výsledného produktu. [1]

2.3.1 Vzdálenost

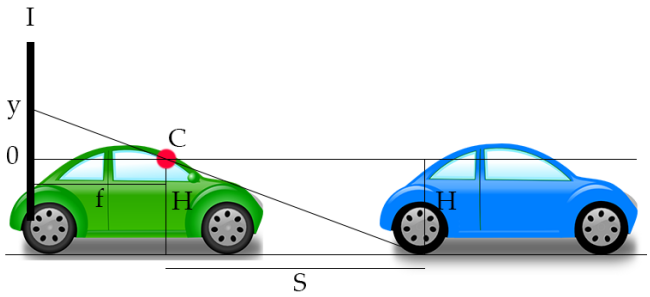
Jelikož je k dispozici jen jedna videokamera, odhad vzdálenosti proběhne pomocí zákonů perspektivy. Aby bylo možné odhadnout vzdálenost, je potřeba znát buď velikost vozidla v obraze nebo pozici spodní části vozidla v obraze. Velikost vozidla přesně neznáme a ani ji nemůžeme odhadnout, jelikož se může pohybovat v dost velkém rozmezí.

Lepším způsobem je použití bodu kontaktu vozidla s vozovkou a geometrii vozovky. Vycházíme z předpokladu, že povrch vozovky je plošný a kamera je namontována směrem

ven z auta. Pak se vzdálenost S k objektu vypočítá jako:

$$S = \frac{fH}{y}. \tag{21}$$

Obrázek 4 znázorňuje dvě auta. V prvním je nainstalována kamera v bodě C, f je fokální vzdálenost kamery k "promítacímu plátnu"I. H je výška, ve které je nainstalována kamera, y je vzdálenost mezi nulou na "plátně"I a bodem, kam se promítá spodní část (kola) pozorovaného auta. [1] Nula na plátně I je poloha horizontu v obraze, určení správné polohy tohoto bodu je pro přesné určení vzdálenosti pochopitelně nezbytné. Pokud bychom dokázali zajistit, aby zařízení vždy bylo namontováno tak, že horizont je např. přesně uprostřed obrazu, potom by problém nevznikl a vždy bychom dokázali odhadnou vzdálenost přesně. Mít tak velký nárok na potencionálního uživatele by bylo nerozumné, proto zatím předpokládejme, že polohu horizontu určit umíme, podrobněji se k tomuto problému vrátíme v dalších kapitolách.



Obrázek 4: Schématický diagram

Vzorec 21 se dá jednoduše odvodit z podobností trojúhelníků.

$$\frac{y}{f} = \frac{H}{S} \tag{22}$$

2.3.2 Odhad vzdálenosti

Pro výpočet vzdálenosti je nutné znát fokální vzdálenost kamery, pomocí které je pořízen záznam. Tuto hodnotu bychom měli umět převést na pixely. Aby to bylo možné, je potřeba vypočítat horizontální FOV (Field Of View). FOV se vypočítává dle vzorce:

$$FOV(horizontal) = 2 \cdot \arctan\left(\frac{frameWidth}{2 \cdot f}\right), \tag{23}$$

kde $frameWidth$ je šířka senzoru, f je fokální vzdálenost kamery [35]. Převod hodnoty fokální vzdálenosti z milimetrů na pixely se provádí dle následujícího vzorce:

$$f(pixels) = \frac{imageWidth \cdot 0.5}{\tan\left(FOV \cdot 0.5 \cdot \frac{\pi}{180}\right)}, \tag{24}$$

kde *imageWidth* je šířka obrazu v pixelech.

Po výpočtu *f (pixels)* už nic nebrání tomu, abychom mohli vypočítat vzdálenost kamery od objektu ve scéně dle vzorce 21.

2.3.3 Invariance výsledků výpočtu vůči poloze zařízení

Jak už bylo zmíněno v předchozích kapitolách, pro odhad vzdálenosti vizuální cestou je potřeba znát polohu horizontu v obraze, poněvadž odhad vzdálenosti spočívá v určení počtu pixelů mezi spodní částí objektu a horizontem. Čím větší je počet pixelů, tím blíž se objekt nachází ke kameře. Není rozumné předpokládat, že uživatel namontuje svůj mobilní přístroj přesně tak, aby horizont byl v konkrétním místně na obrazovce. Pro zvýšení uživatelské přívětivosti je nutné umět polohu horizontu v obraze odhadnout nezávisle na tom, jak uživatel namontuje zařízení. Cílem této kapitoly je prodiskutovat možnosti zajištění invariance výpočtů vzhledem k poloze zařízení.

Jedinou možnosti je polohu horizontu určovat přímo ze vstupního obrazu. Nejdříve se podíváme na dostupné teoretické metody, potom zmíníme tzv. Houghovou transformaci, která bude zapotřebí při implementaci modulu určení horizontu a vzdálenosti.

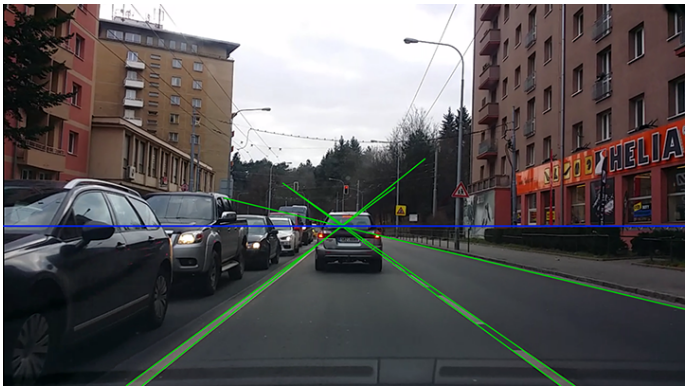
Horizont

Horizont je linie, která odděluje nebe od země [16]. Rozlišujeme lokální horizont a horizont zeměpisný [16]. První může zahrnovat stromy, hory, budovy, jeho určení je v podstatě jednoduché, protože nebe zpravidla neobsahuje širokou paletu barev. Pro řešení daného problému je lokální horizont irelevantní, protože často může být velice matoucí a reálný zeměpisný horizont se za ním může úplně skrývat. Horizont zeměpisný je naopak nehledě na stromy, hory a budovy vždy na stejném místě. Jeho detekce v obraze je bez známosti kontextové informace o scéně úloha velice těžká. Naštěstí pro řešení daného problému informace o tom, že zabíraná scéna je ze silničního provozu, může stačit pro určení polohy zeměpisného horizontu. Tento princip je vidět na obrázku 5. Stačí detekovat vodorovné dopravní značení na silnici nebo polohu krajnic. V místě, kde se tyto čáry kříží, se teoreticky nachází zeměpisný horizont.

Krajnice a relevantní vodorovné značení mají tvar přímek. Pro detekci přímek v obraze se nejčastěji používá Houghova transformace. Vstupem pro tuto speciální metodu je černobílý obraz neboli binární mapa, která obsahuje pouze hrany v obraze. V dalším paragrafu se budeme věnovat detekci hran v obraze, tento krok je předzpracováním vstupu pro Houghovou transformaci.

Detekce hran

V tomto paragrafu popíši princip činnosti optimálního detektoru hran [17]. Tento algoritmus má název Canny Edge Detector. Nejdřív se na vstupní obraz aplikuje Gaussův filtr



Obrázek 5: Určení horizontu z polohy silničních čar

obvykle s velikostí jádra 5. Potom se spočítá gradient intenzit aplikováním konvolučních matic 25 ve směru x a y.

$$MG_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad MG_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{25}$$

Pak se vypočítá síla (G) a směr (Θ) gradientu dle 26 a 27.

$$G = \sqrt{G_x^2 + G_y^2} \tag{26}$$

$$\Theta = atan2(G_y, G_x) \tag{27}$$

G_x a G_y jsou výsledky aplikace matic MG_x a MG_y na vstupní obraz. Směr hrany se zaokrouhluje na jeden ze čtyř úhlů reprezentujících vertikální, horizontální a dva diagonální směry. Jelikož hrany jsou po tomto kroku ještě celkem rozmazané, dalším krokem je ztenčování hran, aby každé nalezené hraně odpovídala právě jedna skutečná. Probíhá to tak, že všechny gradienty kromě největších v okolí se potlačují. Tím pádem zůstávají jen nejsilnější.

Po procesu ztenčování nalezené hrany už docela přesně napodobují reálné hrany, ale i přesto v obraze zůstávají hranové pixely vzniklé kvůli šumu nebo prudkým barevným změnám. Proto nasleduje tzv. dvouúrovňové nebo dvojité prahování. Nastaví se dva prahy: horní a spodní. Pixely s hodnotou gradientu, která je nižší než spodní práh, se rovnou vyřazují. Pixely s hodnotou gradientu větší než horní práh bezpodmínečně zůstávají. Ostatní pixely zůstávají pouze v případě, že mají ve svém nejbližším okolí pixely s gradientem větším než hodnota horního prahu. [17]

Výše popsaný algoritmus tedy zajišťuje následující hlavní kriteria:

- Detekce pouze skutečně existujících hran.
- Vzdálenosti mezi pixely jedné hrany jsou minimalizované.

- Právě jedna nalezená hrana pro jednu reálnou. [17]

Získaný obraz s hranami po převodu na binární mapu je vstupem Houghovy transformace, ta je popsána v dalším paragrafu.

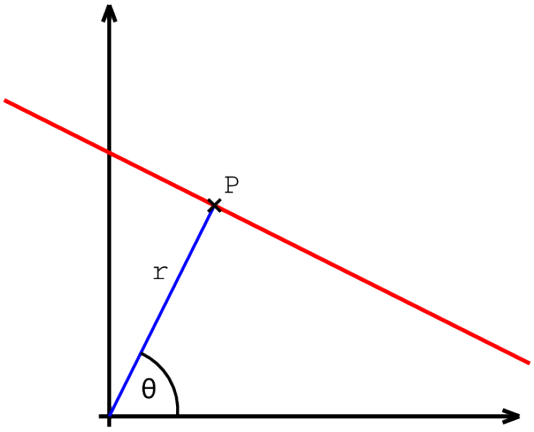
Houghova transformace

Jak už bylo řečeno, standardní Houghova transformace se používá pro detekci přímek, její modifikace se však používají pro detekci kružnic a elips. V tomto paragrafu rozebereme standardní Houghovu transformaci [19] [18], pomocí které ze vstupní binární mapy odvodíme polohy vodorovného dopravního značení.

Standardní reprezentace přímky je vyjádřena vztahem 28, kde m je směrnice a b je bod, ve kterém přímka protíná osu y.

$$y = m \cdot x + b \tag{28}$$

Tímto jednoduše použitelným vztahem však není možné vyjádřit vertikální přímky, proto pro Houghovou transformaci tato reprezentace není vhodná. Tento problém se dá vyřešit zavedením polárních souřadnic. Na obrázku 6 je znázorněná reprezentace přímky pomocí vzdálenosti přímky od počátku soustavy souřadnic r a úhlu θ mezi osou x a vektorem, který spojuje počátek souřadnic a nejbližší bod P na přímce. [18]



Obrázek 6: Polární souřadnice [18]

Promocí vztahů 29 lze přímku ze standardního tvaru převést do tvaru popsaného vztahem 30.

$$m = -\frac{cos(\theta)}{sin(\theta)} \quad b = \frac{r}{sin(\theta)} \tag{29}$$

$$y = -\frac{cos(\theta)}{sin(\theta)} \cdot x + \frac{r}{sin(\theta)} \tag{30}$$

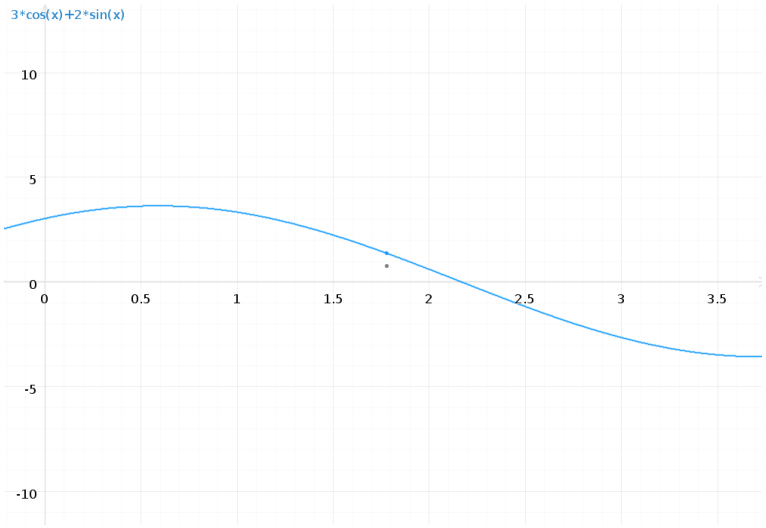
Pomocí jednoduchých úprav vztahu 30 dostaneme 31.

$$r = x \cdot cos(\theta) + y \cdot sin(\theta) \tag{31}$$

Obecně se dá říct, že pro každý bod (x_a, y_a) můžeme definovat soubor přímek takto:

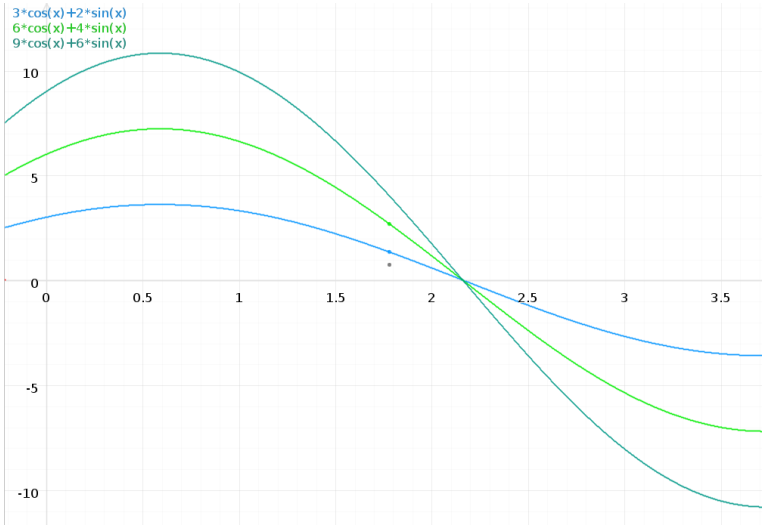
$$r_a = x_a \cdot cos(\theta_a) + y_a \cdot sin(\theta_a) \tag{32}$$

Každý pár (r_a, θ_a) reprezentuje tedy jednu přímku procházející bodem (x_a, y_a). Pro $x_a = 3$ a $y_a = 2$ by graf $r_a = 3 \cdot cos(\theta_a) + 2 \cdot sin(\theta_a)$ v osách r a θ vypadal následovně:



Obrázek 7: Graf pro $x_a = 3, y_a = 2$

Můžeme tuto operaci provést pro každý černý bod (s intenzitou 255) ve vstupní bitové mapě. Pokud se grafy dvou různých bodů překříží v osách r a θ, potom dané body leží na jedné přímce. Například do grafu na obrázku 7 vložíme grafy dvou dalších bodů: $x_b = 6, y_b = 4$ a $x_c = 9, y_c = 6$. Výsledek je na obrázku 8. Tyto tři grafy se kříží



Obrázek 8: Graf pro 3 body: $x_a = 3, y_a = 2$, $x_b = 6, y_b = 4$ a $x_c = 9, y_c = 6$

v bodě $(2.16, 0)$. Tyto souřadnice odpovídají parametrům (θ, r), které reprezentují přímku,

na které leží body (x_a, y_a), (x_b, y_b) a (x_c, y_c). [19]

Detekce přímky spočívá v nalezení určitého počtu překřížení grafů jednotlivých bodů na hranách v obraze. Čím více grafů se kříží v jednom bodě, tím víc bodů obsahuje přímka, takže je výraznější. Obecně řečeno, nastaví se určitý práh minimálního počtu překřížení pro detekci přímky. Houghova transformace počítá překřížení a pokud v nějakém bodě jich je více než nastavený práh, potom se ten bod zapamatuje a po skončení výpočtu máme k dispozici množinu přímek ve tvaru (θ, r). [18] [19]

Pro detekci vodorovného dopravního značení můžeme použít výše popsanou metodu. Abychom eliminovali přímky, které jsou pro nás irelevantní, využijeme informace, které o vodorovných čarách našeho zájmu víme. Jsou to dlouhé čáry, které v obraze mají spíše vertikální orientaci. Takže vhodné by bylo nastavit práh detektoru přímek na vyšší hodnotu a potom dle úhlu θ odfiltrovat přímky s horizontální orientací. Tím pádem nám teoreticky zůstanou pouze přímky, které svým překřížením určí polohu horizontu, jak bylo znázorněno na obrázku 5.

2.3.4 Očekávaná přesnost

Přesnost určené vzdálenosti záleží na řadě faktorů:

1. Kvalita senzoru kamery mobilního zařízení;
2. Známost přesné hodnoty fokální vzdálenosti kamery nebo hodnoty horizontální FOV;
3. Přesnost určení polohy auta v obraze;
4. Přesnost určení polohy horizontu.

Rozebereme výše uvedené body podrobněji. O kvalitě senzorů digitálních kamer je více psáno v kapitole 2.1. Chtěl bych jen připomenout, že čím kvalitnější senzor kamera obsahuje, tím je větší pravděpodobnost, že se podaří dostatečně kvalitně detekovat auta a vodorovné značení, což je nezbytné pro kvalitní odhad vzdálenosti.

Mnou zvolená mobilní platforma (Android) má dostatečně široké API a poskytuje oba parametry: fokální vzdálenost a výchozí Field Of View kamery. S počítáním aktuálního FOV z hodnoty výchozího může být komplikováno odlišným poměrem stran senzoru a komponenty pro zobrazení obrazu, ale hodnotu FOV se dá vždy vypočítat z fokální vzdálenosti kamery.

Odhadnutá vzdálenost k autu se přímo odvozuje ze vzdálenosti mezi horizontem a místem, ve kterém se auto dotýká vozovky. Pokud se tyto body nepodaří získat přesně, potom vznikne určitá chyba. Dále tuto chybu pro různé podmínky vypočteme.

Při výpočtech budeme vycházet z předpokladu, že fyzická fokální vzdálenost kamery je 4.5 mm, velikost senzoru je 4.69×3.53 mm, obraz má velikost 720×480 px a kamera v autě je namontována ve výšce 1.5 m.

Dle 23 spočítáme hodnotu horizontálního FOV:

$$FOV(horizontal) = 2 \cdot arctan(\frac{4.69}{2 \cdot 4.5}) = 55.05° \tag{33}$$

Potom dle 24 vypočítáme hodnotu fokální vzdálenosti v pixelech:

$$f(px) = \frac{720 \cdot 0.5}{\tan(55.05 \cdot 0.5 \cdot \frac{\pi}{180})} \approx 749\ px \tag{34}$$

V tabulce 2 jsou vypočtené hodnoty vzdálenosti pro případy správného určení poloh horizontu a auta a také pro případy chyb při detekci. V řádcích zvýrazněných šedě je výpočet pro bezchybnou detekci, v bílých pro chybně určené polohy. Sloupce tabulky mají následující význam: $y_v[px]$ je vypočtená hodnota y v pixelech, $y_s[px]$ je skutečná hodnota y, $E_p[px]$ je chyba y v pixelech, $E_p[\%]$ je chyba y v procentech, $S[m]$ je odhadnutá vzdálenost v metrech, $E_m[m]$ je chyba odhadu v metrech, $E_m[\%]$ je chyba odhadu v procentech.

Jak je vidět v tabulce, pro vzdálenosti 5-12 metrů chyba v detekci nemá tak kritické následky, 10pixelová nepřesnost má za následek chybu kolem 10 %.

U větších vzdáleností je to horší, např. u vzdálenosti kolem 38 metrů, 10pixelová nepřesnost by způsobila až 50% chybu, což je nepřípustné. U vzdálenosti kolem 100 metrů chyba o 3 pixely by byla kritická, protože by měla za následek 40-50% výslednou chybu.

Aby výsledná aplikace byla použitelná, průměrná chyba detektoru aut a horizontu by neměla přesahovat 15 %.

2.4 Mobilní aplikace

Součástí této práce je také realizace mobilní aplikace na platformě Android. V této kapitole probereme základní nástroje zvolené platformy, které budou využity při realizaci. Nejdříve se podíváme na základní aspekty vývoje aplikací na platformě Android, potom nasleduje popis zvláštního API, které bylo použito: práce s videokamerou a GPS.

2.4.1 Platforma Android

V této kapitole naznačím architekturu platformy, a potom se podíváme na strukturu aplikací vyvíjených na této platformě.

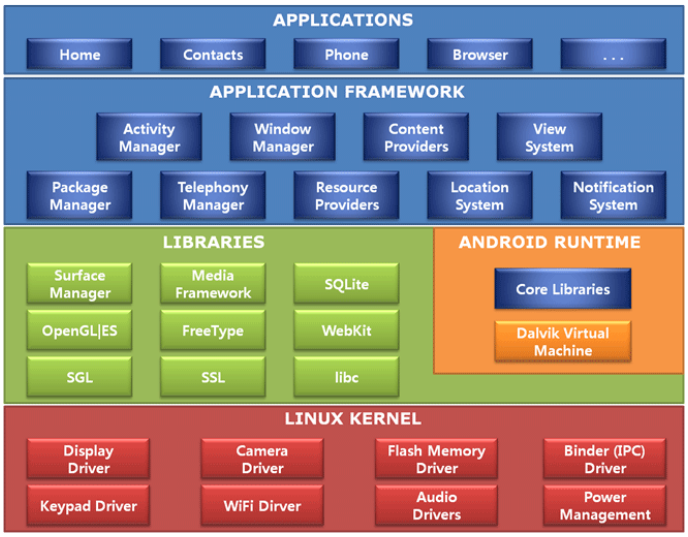
Architektura

Architektura [22] [23] platformy Android se skládá ze základních pěti vrstev (od nejnižší k nejvyšší), přehledněji také na obrázku 9:

- Linuxové jádro;
- knihovny;

$y_v[px]$	$y_s[px]$	$E_p[px]$	$E_p[\%]$	$S[m]$	$E_m[m]$	$E_m[\%]$
200	200	0	0	5.62	0	0
195	200	-5	-2.5	5.76	+0.14	+2.5
190	200	-10	-5	5.91	+0.29	+5.2
180	200	-20	-10	6.24	+0.62	+11.03
220	200	+20	+10	5.11	-0.51	-9.07
70	70	0	0	16.05	0	0
60	70	-10	-14.29	18.73	+2.68	+16.7
50	70	**-20**	-28.57	22.47	**+6.42**	**+40**
80	70	+10	+14.29	14.04	-2.01	-12.52
90	70	+20	+28.57	12.48	-3.57	-22.24
30	30	0	0	37.45	0	0
27	30	-3	-10	41.61	+4.16	+11.11
25	30	-5	-16.67	44.94	+7.49	+20
20	30	**-10**	-33.33	56.18	**+18.72**	**+50**
35	30	+5	+16.67	32.1	-5.35	-14.29
40	30	+10	+33.33	28.09	-9.36	-24.99
10	10	0	0	112.35	0	0
9	10	-1	-10	124.83	+12.48	+11.11
7	10	**-3**	-30	160.5	**+48.15**	**+42.86**
13	10	+3	+30	86.42	-25.93	-23.08

Tabulka 2: Odhad přesnosti



Obrázek 9: Architektura platformy Android [22]

- Android runtime;
- aplikační framework
- aplikace.

Linuxové jádro tvoří vrstvu mezi hardware a vyššími vrstvami. Využívá systémové prostředky pro ovládání displeje, kamery, Wi-Fi, správy paměti, napájení atd.

Další vrstva poskytuje sadu knihoven v jazyce C/C++. Poskytují funkce pro zpracování zvuku, obrazu, 2D a 3D grafiky, SQLite databázi atd. Funkce těchto knihoven poskytují operace na nejnižší úrovni linuxového jádra.

Na stejné úrovni s knihovnami se nachází Android runtime. Hlavní součástí této vrstvy je Dalvik Virtual Machine, je to upravené Java prostředí pro běh aplikací. Tuto vrstvu tvoří knihovny jazyka Java, které jsou bezprostředně dostupné při vývoji aplikací přes aplikační framework.

Aplikační framework poskytuje vývojáři konkrétní stavební bloky aplikace. Pro programátory se jedná o nejdůležitější vrstvu, protože právě s ní jednají v průběhu celého vývoje.

Struktura aplikací

V tomto paragrafu se zaměříme na základní stavební bloky aplikací vyvíjených pro platformu Android. Povinnou komponentou každé aplikace je soubor *AndroidManifest.xml*. Každá aplikace jej musí mít v kořenové složce. Systém nemůže spustit kód aplikace dřív, než zjistí informace z manifestu aplikace. Manifest dává název paketu Java, který je identifikátorem aplikace v systému, deklaruje tzv. povolení (angl. permissions), která udávají, co všechno aplikace může použít a další (viz [24]).

Aktivita (angl. Activity) je součást aplikace, pod kterou často rozumíme konkrétní jednu obrazovku aplikace. Aktivita obsahuje uživatelské rozhraní, pomocí kterého probíhá komunikace uživatele s aplikací. Principiálně může být v aplikaci více aktivit, jedna je vždy označená za výchozí a při startu aplikace se vždy spouští automaticky. Spuštění další aktivity znamená zastavení předchozí, znamená to, že systém ji uloží do zásobníku aktivit, což umožňuje rychlý návrat zpátky, pokud je to potřeba. Aktivita se může nacházet v jednom ze tři stavů:

- Resumed - běží na popředí a interaguje s uživatelem.
- Paused - jiná aktivita je na popředí, tato je však ještě z části viditelná.
- Stoped - aktivita je zcela neviditelná.

Aktivity si můžou vyměňovat data pouze při spuštění a ukončení. Když jedna aktivita spouští druhou, tak jí předává zprávu Intent, do které zabalí data, která jí chce předat. Při skončení aktivita zabalí návratová data do zprávy Intent a odešle ji aktivitě, která ji spustila.

Intent je další součástí aplikace, která propojuje jednotlivé komponenty aplikace s jinými komponentami stejné či jiné aplikace. Pro aktivity intenty definují akci, která se má provést, a také mohou odkázat na data, která by se měla zpracovat. [21]

Kromě zdrojového kódu aplikace může obsahovat další data, např. grafická data (obrázky, ikony, pozadí atd), soubory, atd. Těmto datům se říká zdroje (angl. resources). Jako zdroj v Androidu jsou chápané také XML soubory popisující rozložení prvků GUI na obrazovce a definující obecně styl aplikace. Tyto soubory se nachází v adresáři *layout*. Ve zdrojích se také dají uchovávat různé konstantní hodnoty, které jsou potom používané v aplikaci. XML soubory s těmito hodnotami jsou uložené ve složce *values*. V adresáři *drawable* se nachází grafické prvky. [23]

Více o struktuře platformy a aplikací, životním cyklu aktivit a dalších komponentách se dá zjistit z oficiálních zdrojů, např. [25].

2.4.2 Použité API

Android API je známé svou rozsáhlostí a jednoduchostí v používaní. Poskytuje přístup k většině hardwarových prvků přístroje zejména kamera, lokalizační služby (GPS), baterie, komunikační prvky (Wi-Fi, Bluetooth atd.) ... V této kapitole popíši API, které bylo použito při vývoji aplikace, konkrétně kamera a lokalizační služby.

Kamera

Při práci s kamerou jsem použil třídu, která je od API úrovně 21 (verze Android 5.0) zastaralá, aby aplikace mohla fungovat i na starších přístrojích, které tuto úroveň API nepodporují. Jedná se o třídu *android.hardware.Camera*. Pro přístup ke kameře je potřeba

tento záměr deklarovat v Android manifestu například takto:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
```

Pro start kamery je potřeba zavolat metodu *open(c)* nad třídou *Camera*, kde *c* je identifikátor kamery, pro hlavní kameru zařízení bývá zpravidla *c=0*. Metoda vytvoří a vrátí instanci třídy *Camera*.

Potom je potřeba přiřadit kameře vzhled (angl. view), na který se bude zobrazovat náhled (angl. preview) z kamery. Většinou to bývá *SurfaceView*. Pomocí metody *camera.setPreviewDisplay(holder)* kameře se přiřadí *SurfaceHolder holder* toho *SurfaceView*, na který se bude zobrazovat náhled. Potom voláním *startPreview()* a *stopPreview()* se začne zachytávat obraz z kamery, respektive zastaví. Voláním *release()* se kamera uvolní, aby ji mohla použít jiná aplikace.

Po vytvoření instance kamery se dají vyčíst anebo upravit její parametry. Volání metody *getParameters()* vrátí instanci třídy *Camera.Parameters*. Z tohoto objektu potom lze vyčíst užitečné informace o kameře, konkrétně: fokální vzdálenost a Field Of View (FOV).

Fokální vzdálenost se dá vyčíst voláním metody *getFocalLength()*, vrátí se hodnota v milimetrech. Metody *getHorizontalViewAngle()* a *getVerticalViewAngle()* vrátí hodnoty horizontálního a vertikálního FOV ve stupních. Důležité je chápat, že nejsou dostupné informace o přesnosti vyčítaných hodnot a navíc bez ohledu na aktuální přiblížení. Bohužel, nelze vyčíst fyzickou velikost nainstalovaného senzoru, což nám brání k výpočtu hodnoty FOV dle 23. Zbývá jediná možnost, spoléhat na to, že poskytnuté údaje jsou pravdivé a přepočítat zjištěnou hodnotu FOV dle aktuálního přiblížení pomocí vztahu:

$$\alpha_z = 2 \arctan\left(\left(\frac{z_n}{z_a}\right) \cdot \tan\left(\frac{\alpha}{2}\right)\right), \tag{35}$$

kde α_z je aktuální hodnota FOV včetně přiblížení, z_n je výchozí hodnota přiblížení (pro Android se rovná 100, což znamená žádné přiblížení), z_a je aktuální přiblížení, α je FOV bez přiblížení.

Lokalizační služby

Android API nabízí velice jednoduchý přístup k informaci o poloze zařízení. Informaci o poloze zařízení dává tzv. poskytovatel (angl. provider). Poskytovatelem může být modul GPS anebo síť (angl. network provider).

Z aktuálního kontextu se může vyčíst objekt třídy *LocationManager* přes metodu *context.getSystemService(LOCATION_SERVICE)*. Přes tento objekt se dá požádat o pravidelné obnovení polohy (metoda *requestLocationUpdates(...)*) anebo se dá zjistit poslední známá poloha zařízení voláním metody *getLastKnownLocation(provider)*, kde *provider* je

buď *NETWORK_PROVIDER* anebo *LGPS_PROVIDER*, metoda vrátí aktuální zeměpisnou šířku a délku v podobě objektu třídy *Location*.

Třída *Location* poskytuje metodu *distanceTo(Location dest)*, která vrátí přibližnou vzdálenost dvou poloh. Výpočet probíhá s použitím elipsoidy WGS84 [26].

3 Rozpoznávání aut

V této kapitole jsou popsány různé přístupy k detekci aut v obrazu a hodnocení použitelností těchto metod. Všechny prezentované výstupy v této kapitole jsou realizované v jazyce C++ pomocí knihovny OpenCV verze 2.4.10.

Metody pro řešení dané problematiky jsem rozdělil na metody s klasifikátorem a bez něj. V kapitole 2.1 je vysvětlen očekávaný vstup. V kapitole 2.2 jsou popsány metody bez klasifikátoru, v kapitole 2.3 metody s klasifikátorem a v kapitole 2.4 kombinace obou metod.

3.1 Popis vstupu a sběr dat

Vstupem celého procesu rozpoznávání aut je obraz z kamery mobilního telefonu uchyceného v držáku na čelním skle auta směrem k cestě a na obrázku 10 je tato koncepce graficky znázorněna. Kamera je umístěná uprostřed vozidla nasměrována tak, aby jízdní pruh byl co nejblíže středu obrazu.

Obrázek 10: Mobil na držáku v autě

Data byla nasbírána pomocí kamery mobilního přístroje Samsung Galaxy Note 3 v Brně a na dálnici D1. Celkem bylo nasbíráno více než hodina a půl video záznamu s rozlišením 1280×720 v různých světelných podmínkách. Velkým problémem už ve fázi sběru dat se ukázalo ostré slunce (kolem čtvrté hodiny odpolední), které vyběluje obraz ve chvíli, kdy svítí do objektivu. Ve chvíli, kdy slunce svítí zboku, je kamera přístroje nucená zmenšit světelnou citlivost a obraz se výrazně ztmavuje (především v oblasti, kde je vozovka, jelikož je dost tmavá). Při testování potom kvůli tomu může vzniknout problém s rozpoznáním aut.

3.2 Implementovaná metoda rozpoznávání aut

3.2.1 Přístup rozpoznávání bez klasifikátoru

Metoda vychází z předpokladu, že na vstupní sekvenci snímků bude vidět jak krajina, tak vozovka s auty. Vozovka se hýbe celkem vysokou rychlostí a postupně mizí, kdežto relativně statické nebo pomalu pohybující se objekty vůči kameře se nebudou proměňovat a mizet tak rychle. Budou se postupně přemísťovat, zvětšovat nebo zmenšovat, ale zůstanou delší dobu pozorovatelné.

Obrázek 11: Pozorovatelné body v obraze

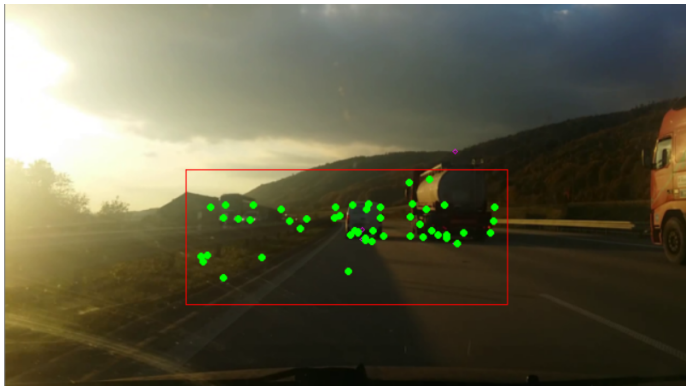
Nejjednodušší postup je pomocí funkce knihovny OpenCV *goodFeaturesToTrack()*, která využívá principů popsaných v kapitole 2.2.3, najít v obraze dobře pozorovatelné body a pomocí funkce *calcOpticalFlowPyrLK()* využívající metody popsané v kapitole 2.2.3 zjišťovat jejich přemístění v obraze. Dále následuje analýza shluků, které se příliš nepohybují, dle různých kritérií (množství bodů, tvar objektu, rychlost pohybu atd.) a filtrace objektů, o kterých je rozhodnuto, že jde o automobily.

Tento postup selhává v okamžiku, kdy se v obraze nachází složitá krajina (stromy, keře, zábradlí, lidé atd.) a auta naopak nejsou ničím výrazná (jednobarevná auta často podobná vozovce, zaoblené tvary bez hran). V důsledku se vytváří shluky bodů mimo vozovku, které vytváří dojem aut mimo cestu, naopak skutečná auta na vozovce jsou přehlédnuta. Tato situace je zobrazena na obrázku 12.

Pro řešení tohoto problému jsem se rozhodl odříznout horní část obrazu a dále krajinu zleva a zprava. Poté se mohu zaměřit na hledání pozorovatelných bodů jen v přímé blízkosti vozovky, čili uprostřed obrazu, příp. trochu níž. Při zvýšení počtu bodů a tím pádem pravděpodobnosti, že se auto bude nacházet v dané oblasti, jsem předpokládal, že čím blíže k centru obrazu se sledované body nacházejí, tím vyšší je pravděpodobnost, že se v dané oblasti nachází automobil. Proto jsem u centra nechal vysoký počet bodů a snižoval jejich počet směrem k okrajům. Výsledek můžeme vidět na obrázku 13.

Problém selhání metody při použití se složitou krajinou je částečně vyřešen, když se auta v obraze nacházejí blízko ke kameře. Následně vyvstává další problém, že body nejsou

Obrázek 12: Výsledek volání goodFeaturesToTrack()



Obrázek 13: Zmenšená oblast pro pozorování významných bodů

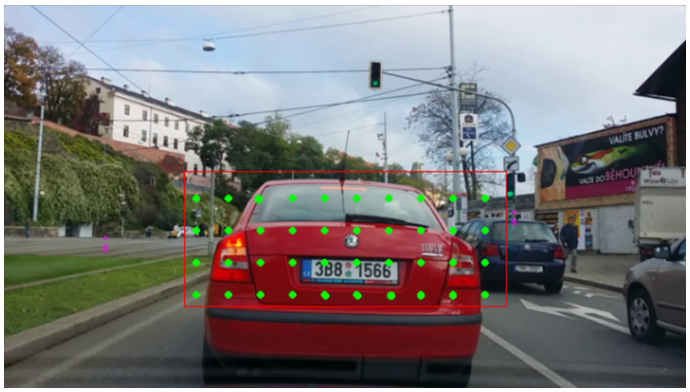
rovnoměrně rozložené po nalezených objektech, a proto je přesnější určení polohy a velikosti auta nemožné.

Ve chvíli, kdy je objekt dále od kamery, je jeho identifikace téměř nemožná, protože i přes to, že poměrně dost pozorovatelných bodů se shromažďuje na autech, nelze je určit jako shluky bodů. Tento problém vidíme na obrázku 14.

Shluková analýza v tomto případě nepřinese uspokojivý výsledek, neboť přestože dokáže identifikovat shluk bodů jako hledaný objekt, nedokáže odhadnout rozměry objektu správně nebo dva shluky patřící jednomu autu rozdělí na dva menší objekty.

Za účelem rovnoměrného rozložení sledovaných bodů na hledaném objektu se použití funkce goodFeaturesToTrack() ukázalo jako nevhodné. Proto jsem se rozhodl použít metodu sledování skupiny bodů rovnoměrně rozložených v mřížce, kde je každý třicátý pixel v každém třicátém řádku zvolen k sledování. Zachoval jsem koncepci sledování bodů jen ve vybrané oblasti, protože se to osvědčilo jako dobrý krok, ale přidal jsem k ní navíc mřížku. Příklad takové mřížky je na obrázku 15. Tento postup se dobře osvědčil v případě, že se vozidlo nacházelo blízko ke kameře (obrázek 16). Naopak v případě, kdy je objekt vzdálený, metoda selhává kvůli velkému počtu bodů na vozovce, které nejsou sledovatelné (vozovka má často všude stejnou barvu, proto body na vozovce zůstávají na místě a vytváří falešné

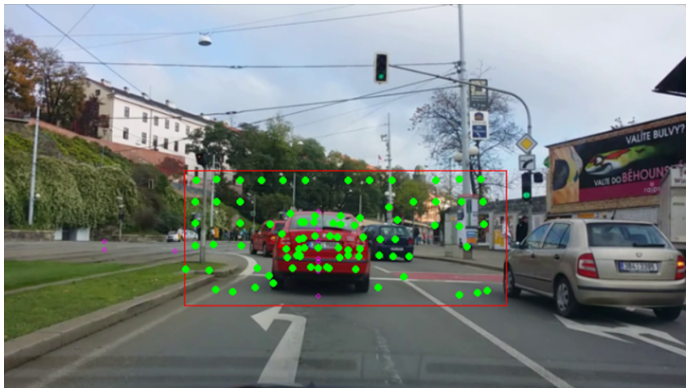
Obrázek 14: Problém velké vzdálenosti objektu od kamery



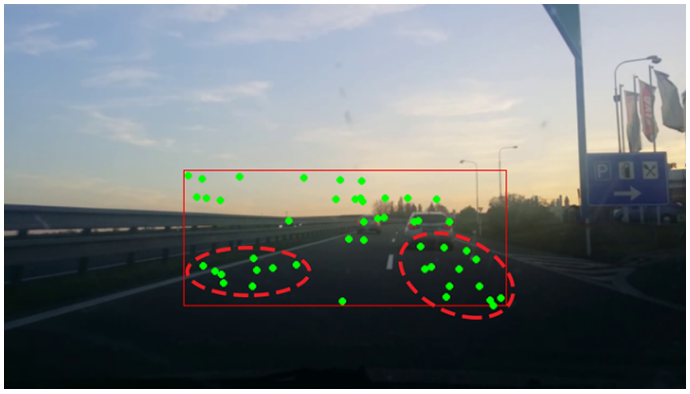
Obrázek 15: Mříž sledovaných bodů

shluky, obrázek 17).

Závěrem bych shrnul, že žádný z vyzkoušených přístupů nedává jednoznačný výsledek, proto pro řešení problému rozpoznání aut v obraze je nutné zvolit jiný přístup.



Obrázek 16: Vozidlo blízko kamery, úspěch metody



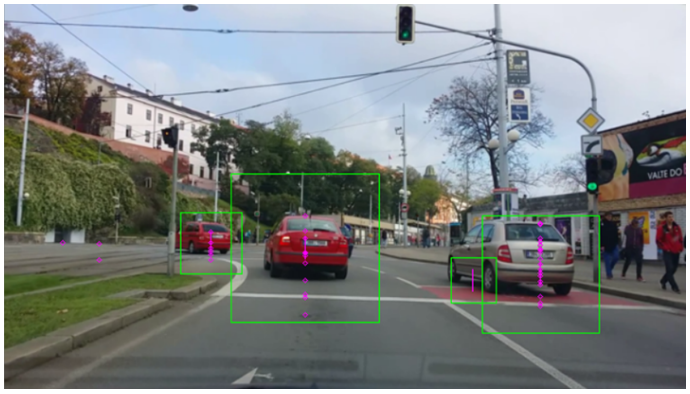
Obrázek 17: Vozidlo daleko od kamery, falešné shluky

3.2.2 Rozpoznávání s klasifikátorem

Za účelem testování jsem použil již hotový natrénovaný klasifikátor z opensource projektu *opencv-lane-vehicle-track*. Klasifikátor je trénován na 516 obrázcích aut velikosti 128×128. XML soubor s rozhodovacím stromem klasifikátoru je dostupný ke stažení na stránkách projektu [33]. Klasifikátor funguje na principech popsaných v kapitole 2.2.2

V projektu je použita zajímavá metoda verifikace nalezených aut pomocí horizontálních hran a vertikální symetrické osy auta. Při použití daného klasifikátoru a verifikace metoda dokáže rozpoznat pouze některá auta (obrázek 18). Stěžejní pro tuto metodu je kvalita osvětlení a barev. Pokud zboku nepatrně svítí ostré slunce, je obraz hodně světlý a klasifikátor může selhávat. Při soumraku je obraz příliš tmavý a klasifikátor opět nemůže najít všechna auta (obrázek 19). Funkční je pouze za denního světla s nepřímým sluncem, bohužel se tak nedá říci, že by metoda fungovala spolehlivě.

Navíc i při dobrých světelných podmínkách je běžné velké množství falešných nálezů. Řešením může být spojení metod rozpoznávání bez klasifikátoru popsaných v kapitole 3.2.1 a metody s klasifikátorem.



Obrázek 18: Detekovaná vozidla pomocí klasifikátoru opencv-lane-vehicle-track

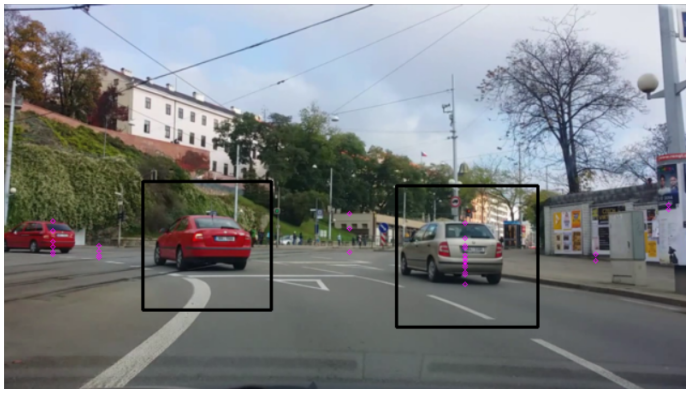
Obrázek 19: Špatné světlné podmínky - soumrak. Klasifikátor opencv-lane-vehicle-track nedetekuje žádné z aut

3.2.3 Spojení metody bez klasifikátoru a s ním

Testováním se ukázalo, že samotná metoda detekce aut založena na sledování významných bodů v obraze není efektivní ani spolehlivá i přesto, že významné body se shlukují na autech. Falešné shluky kolem však neustále brání v identifikaci auta. Samotné rozpoznávání pomocí klasifikátoru se také neosvědčilo, jelikož funguje pouze při perfektních světelných podmínkách, při kterých ale stejně generuje velké množství falešných nálezů.

Pokusil jsem se o spojení obou přístupů s následujícím konceptem. Řídicí jednotkou je klasifikátor, který vybere oblasti, ve kterých se podle něj nachází auto. Pak jsou tyto oblasti verifikovány pomocí metody bez klasifikátoru. Pokud se v dané oblasti vytvoří shluk sledovaných bodů, pak se tato oblast potvrzuje jako auto, v opačném případě se výskyt auta vylučuje.

Tento postup už z principu nemůže vyhledat auta, která klasifikátor vynechal, ale upřesní proces rozpoznání tím, že odstraní většinu falešných nálezů. Pro úspěch tohoto postupu je nutný lépe natrénovaný klasifikátor, který prokazuje lepší invariantní vlastnosti vůči světelným podmínkám. Hledání nebo trénování takového klasifikátoru by mohlo být



Obrázek 20: Spojení metod. Přesnější detekce, méně falešných nálezů

námětem pro další práci. Zatím počítejme, že takový klasifikátor máme.

3.2.4 Výstup detektoru aut

Realizovaný detektor aut popsaný v kapitolách výše, na vstup přijímá video záznam natočený z kamery namontované za čelním sklem auta a jako výstup produkuje textový soubor s informacemi o detekovaných autech. Jednotlivým záznamem v souboru je řádek. Každý řádek je v následujícím formátu:

frameID x y width height distance

Kde *frameID* je pořadové číslo snímku ve vstupní video sekvenci, x a y jsou souřadnice levého horního rohu detekovaného auta, *width* a *height* je šířka a výška obdélníku, který orámuje detekované auto a *distance* je přibližná vzdálenost k detekovanému autu za předpokladu, že horizont se nachází uprostřed obrazu.

Samozřejmě předpokládat, že horizont je uprostřed obrazu není rozumné, proto vypočítané vzdálenosti jsou velice nepřesné. V další kapitole je popsán proces detekce horizontu, známost polohy kterého je nezbytná pro přesný odhad vzdálenosti.

3.3 Detekce horizontu

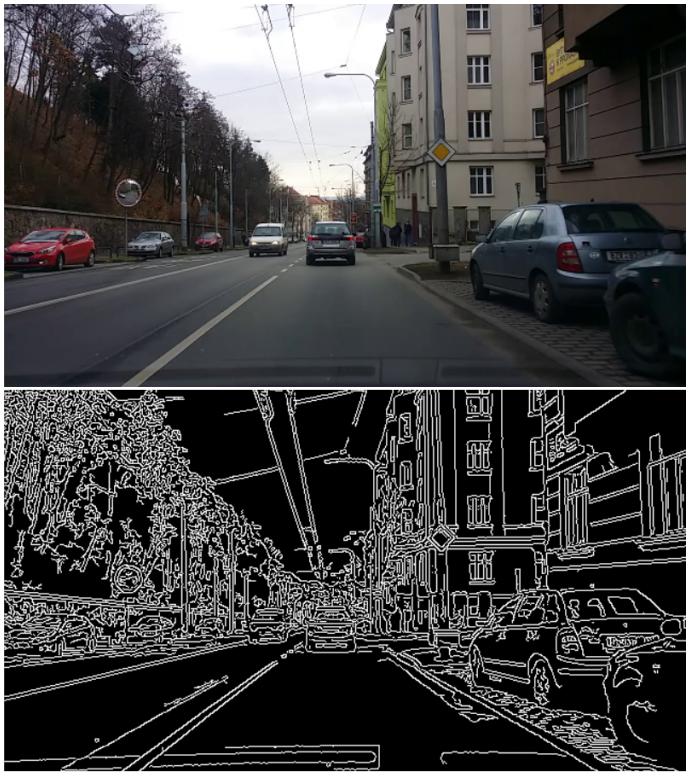
Jak bylo psáno v kapitole 2, pro přesné určení vzdálenosti je nezbytné určit polohu horizontu, protože hodnota vzdálenosti se přímo vypočítává z počtu pixelů mezi horizontem a místem, ve kterém se auto dotýká vozovky. V této kapitole bude popsán postup realizace detektoru horizontu v jazyce C++ pomocí knihovny OpenCV na platformě Windows. Všechny použité techniky vychází z teoretického základu popsaného v kapitole 2.3.3.

Vstupem realizovaného detektoru je videozáznam, který prošel detektorem aut, a textový soubor, který byl výstupem zmíněného detektoru. Textový soubor je pro detektor horizontu irelevantní, slouží pouze pro zobrazení detekovaných aut.

V kapitole 2.3.3 jsme došli k závěru, že horizont je linie oddělující nebe a zemi. V našem konkrétním příkladě v obraze jsou objekty, které nejsou důležité: budovy, stromy, keře, hory, lidé, atd. Proto z určení polohy horizontu se stává netriviální úloha. Jediný použitelný způsob, jak určit jeho polohu, je ukázán na obrázku 5 v kapitole 2.3.3. Metoda spočívá v detekci vodorovného dopravního značení a krajnic. V místě překřížení těchto linií potom prochází horizont.

Vodorovné značení většinou je bílé, a proto se odlišuje od barvy vozovky, která je zpravidla hodně tmavá. Hranový detektor by tedy měl vodorovné značení detekovat jako kvalitní hrany.

Pro detekci hran jsem použil Canny Edge Detector, princip jeho funkce je podrobně popsán v kapitole 2.3.3. Knihovna OpenCV obsahuje funkci *Canny(in, out, th1, th2)*, jejímž vstupem je obraz *in*, výstup zapíše do obrazu *out*, *th1* a *th2* jsou hodnoty prahů pro dvojité práhování. Na obrázku 21 můžeme vidět příklad vstupu a výstupu tohoto detektoru s parametry *th1=10* a *th2=50*.

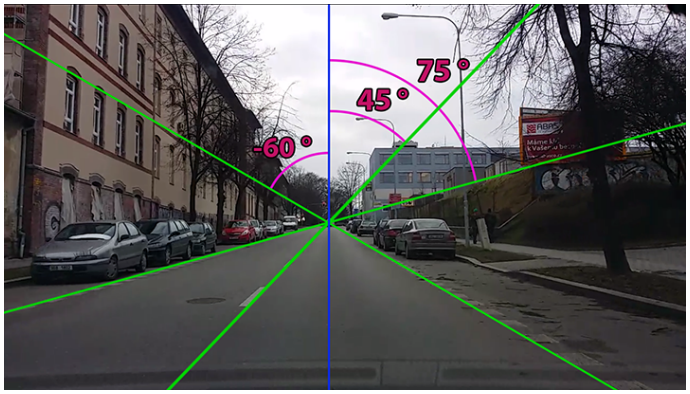


Obrázek 21: Canny Edge Detector - příklad vstupu a výstupu

Po detekci hran potřebujeme najít dostatečně dlouhé přímky, abychom o nich mohli usoudit, že jsou součástí vodorovného značení. Houghova transformace popsaná v kapitole 2.3.3 slouží právě k tomuto účelu. V knihovně OpenCV klasická Houghova transformace detekující přímky je implementována ve funkci *HoughLines(image, lines, rho, theta, th)*, kde *image* je vstupní obraz s detekovanými hranami, *lines* je pole, do kterého se zapíší

detekované přímky, *rho* je rozlišení vzdálenosti v pixelech, *theta* je rozlišení úhlů v radiánech a *th* je práh, který udává minimální délku úsečky, aby byla považována za přímku. Algoritmus dále postupuje následovně: nastaví vysoký práh (nejvíce se osvědčila hodnota 200), tím pádem za detekovanou přímku se budou považovat pouze dlouhé úsečky (delší než 200 pixelů).

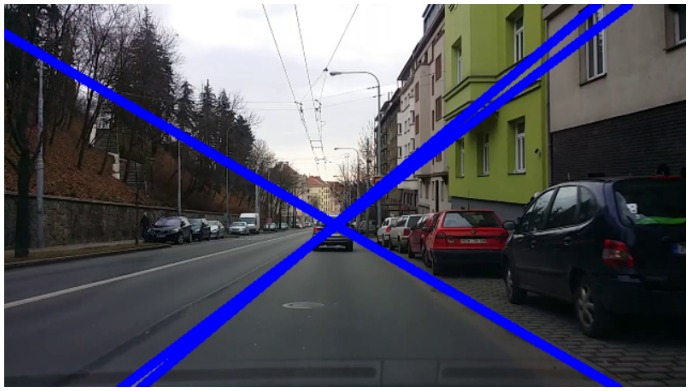
V každém kroku algoritmu (jeden snímek) potom probíhá filtrace přímek dle směru jejich natočení. Zajímají nás pouze ty, které nejsou horizontální, protože s větší pravděpodobností odpovídají vodorovnému značení, které potřebujeme k detekci horizontu. Každá přímka nalezena pomocí Houghovy transformace má dva parametry (polární souřadnice): r a θ, viz kapitolu 2.3.3. Z úhlu θ zjistíme, jakým směrem je natočena přímka. Na obrázku 22 vidíme úhly natočení správně detekovaných přímek. Zajímají nás tedy přímky, které jsou odkloněné od vertikály na $\pm 75°$, vůči horizontále je to rozmezí: $[15°, 165°]$, nechť tento úhel je α.



Obrázek 22: Detekované přímky a úhel vůči vertikále

Dle koncepce polárních souřadnic zavedených v kapitole 2.3.3, hodnota θ nalezené přímky by se měla přepočítat následovně: $\theta = (\alpha + 90) \mod 180$. Hledáme tedy přímky s hodnotou $5\pi/12 < \theta < 7\pi/12$ o nich můžeme předpokládat, že s největší pravděpodobností se jedná o součást vodorovného značení. Ostatní přímky zamítáme jako falešné nálezy. Pokud počet detekovaných přímek je menší než určitý počet n (ve většině případů jsem nastavil $n = 2$, protože stačí dvě přímky k určení polohy horizontu), potom se snižuje nastavený práh o 5 a detekce probíhá ještě jednou a opakuje se tak dlouho, dokud se nenajde alespoň n přímek. Jinými slovy, pro každý snímek zaručeně nalezneme alespoň n nejdelších úseček. Na obrázku 23 můžeme vidět odfiltrované přímky, které se kříží v jednom bodě. Dle dříve definovaných předpokladů, tímto bodem prochází horizont.

Tedy v každém kroku algoritmu, pokud jsou detekované alespoň dvě přímky, je modifikována poloha horizontu. Aby byl minimalizován vliv falešných detekcí, výpočet nové polohy horizontu probíhá pomocí váženého průměru. Stávající hodnota má váhu 50, nová zjištěná hodnota má váhu 1. Tím pádem nový horizont jen koriguje ten stávající a nenastávají velké skoky. Výsledek můžeme vidět na obrázku 24, horizont je vyznačen červenou



Obrázek 23: Relevantní detekované přímky

horizontální linií.

Obrázek 24: Nalezený horizont (červená linie)

Výstupem detektoru horizontu je také textový soubor definující polohu horizontu pro každý snímek videozáznamu.

3.4 Rychlost zpracování

Rychlost zpracování vstupních videosekvencí detektorem aut a horizontu je velice důležitá. Právě tato hodnota ukáže, jestli je šance, aby výše popsaný algoritmus fungoval v reálném čase na mobilním přístroji. Provedl jsem měření rychlosti výpočtů realizovaných v C++ pomocí OpenCV detektorů aut a horizontu na počítači s operačním systémem Windows. Parametry tohoto počítače jsou následující:

- Procesor: Intel® Core™ i7-4710HQ [27]
- Operační paměť: 16 GB
- Grafická karta: Intel® HD Graphics 4600 a NVIDIA GeForce GTX 860M
- Operační systém: Windows 8.1 64 bit

Při testování byla použita videa s rozlišením 1280×720, přitom byl každý snímek před začátkem zpracování zmenšen dvakrát, jak na šířku, tak i na délku. Fakticky se tedy zpracovávaly snímky o velikosti 640×360. Videosekvence použité při testování jsou v příloze A (DVD disk) v adresáři *Video*. Dle názvu souboru v tabulce 3 se dají identifikovat testované sekvence. Výsledky testování rychlosti jsou v tabulce níže:

Název	Detektor aut [fps]	Detektor horizontu [fps]	Celkem [fps]
video1.mp4	14.5	32.3	10.0
video2.mp4	17.7	34.4	11.7
video3.mp4	15.2	26.9	9.7
video4.mp4	18.5	36.3	12.3
video5.mp4	14.0	31.9	9.7

Tabulka 3: Testování rychlosti detekce - výsledky

Jak je vidět z tabulky 3 na plnohodnotném počítači zpracování videí detektorem probíhá poměrně pomalu, průměrně 10.7 snímku za sekundu. Je možné na algoritmus aplikovat různé optimalizační techniky, např. vynechávat každý druhý nebo každý třetí snímek, a je hypoteticky možné, že na výkonném stolním počítači detektor dokáže pracovat v reálném čase.

V dnešní době výkon mobilních zařízení je mnohém nižší než plnohodnotných počítačů (stolní počítač, notebook atd.). I přesto, že maximální frekvence některých mobilních procesorů se přibližuje k průměrné frekvenci normálních procesorů, mobilní zařízení nejsou schopna dlouhodobě zajistit dostatečně vysoký výkon pro zpracování podobných problémů v reálném čase, a to kvůli rychlému zvýšení teploty procesoru [28]. Proto realizace tohoto algoritmu na mobilním zařízení zatím nemá význam.

V další kapitole se budeme věnovat návrhu a vývoji mobilní aplikace jako jednoho ze stavebních bloků výsledného řešení daného problému. Kompozice všech bloků (detektor aut, detektor horizontu a mobilní aplikace) by mohla být předmětem další práce, až budou schopna mobilní zařízení daný problém fyzicky zpracovat.

4 Realizace mobilní aplikace

V této kapitole bude podrobněji popsán proces návrhu a realizace mobilní aplikace na platformě Android. Jak již bylo zmíněno dříve, realizovat algoritmus detekce na mobilní platformě nemá význam, aplikace by nemohla pracovat v reálném čase. Proto se omezíme na návrh aplikace a realizaci její kostry a dílčích součásti.

Pro výpočet vzdálenosti dle vztahu 21 je nutné znát přesnou výšku H, ve které je namontováno mobilní zařízení. Chyba v hodnotě H by mohla znamenat velkou chybu v odhadu vzdálenosti. Dá se spočítat dle vztahu 36 (dá se odvodit ze vztahu 21), kde H_1 je správná a H_2 je chybná hodnota výšky. Pro hodnoty $H_1 = 1.5$ a $H_2 = 1.6$ (průměrná výška pro osobní auto) by se chyba E rovnala 6.67 %.

$$E = \left(\frac{H_2}{H_1} - 1\right) \cdot 100 \tag{36}$$

Je relativně komplikované zjistit, v jaké výšce si uživatel namontoval mobilní přístroj. Tento údaj je docela těžko měřitelný, dá se pouze odhadnout, ale není zaručeno, že uživatel tuto výšku dokáže odhadnout správně. Proto je potřeba udělat kalibraci aplikace pro daný přístroj pro dané vozidlo, resp. polohu zařízení ve vozidle.

V další podkapitole popíši návrh a realizaci kalibračního módu aplikace. Potom bude následovat popis demonstračního módu, který byl vyvinut jako ukázka toho, jak by uživatelské rozhraní vypadalo, kdyby aplikace realizovala celý detekční algoritmus.

4.1 Kalibrační mód

4.1.1 Princip kalibrace

Principem kalibrace je nalezení hodnoty koeficientu k, který by zpřesňoval odhad vzdálenosti. Koeficient k se dá empiricky odhadnout pro každé zařízení a každou jeho polohu v autě. Pozměníme tedy vztah 21 přidáním koeficientu k:

$$S = k \cdot \frac{fH}{y} \tag{37}$$

Výpočet k je možný, pokud známe přesnou vzdálenost S k nějakému referenčnímu objektu, a také známe přibližné hodnoty f, H a y. Koeficient k se pak vypočítá dle vztahu:

$$k = \frac{Sy}{fH} \tag{38}$$

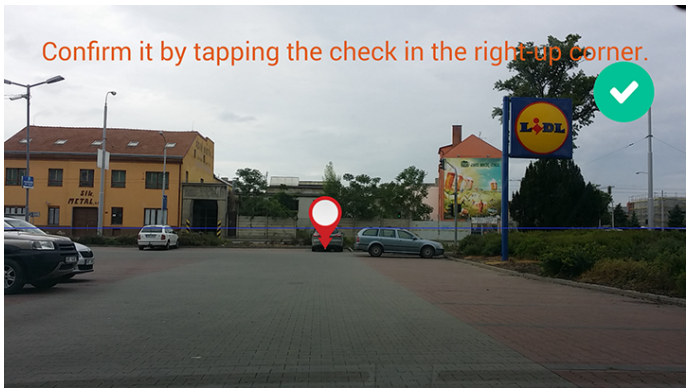
Přibližné nebo přesné hodnoty f, H a y umíme vypočítat, příp. vyčíst pomocí API. Přesnou vzdálenost k referenčnímu objektu S je potřeba změřit. Nemůžeme po uživateli žádat, aby poskytl přesné informace o vzdálenosti k nějakému objektu, protože i v případě, že by byl ochotný se tomu věnovat, nejpravděpodobněji nebude mít nástroje, aby přesně změřil vzdálenost délky v řádu desítek metrů.

Jinou možností je implicitní měření vzdálenosti pomocí lokalizačních služeb, které jsou dostupné ve většině mobilních zařízení s operačním systémem Android. Dle [29] je přesnost GPS v mobilních zařízeních v 95% případů lepší než 10 metrů. Na trhu jsou k dispozici aplikace, které dokážou změřit vzdálenost v řádu stovek metrů s dostatečnou přesnosti [29]. Pokusíme se tedy pomocí GPS změřit vzdálenost S a následně vypočítat koeficient k, který se potom bude používat při zpřesnění odhadnutých hodnot vzdálenosti. V kapitole 5 naleznete testování přesnosti tohoto postupu.

4.1.2 Návrh módu

Do kalibračního módu se aplikace zapíná pouze při prvním zapnutí aplikace na tomto přístroji, případně lze na žádost uživatele provést kalibraci opakovaně. Po zapnutí aplikace se na obrazovce objeví obraz z kamery mobilního přístroje (práce s kamerou probíhá dle popisu v kapitole 2.4.2) s žádostí, aby uživatel zvolil referenční objekt na obrazovce dotykem prstu a následným přesouváním tohoto bodu.

Potom by měl uživatel potvrdit polohu objektu stisknutím potvrzovacího tlačítka v pravém horním rohu. Z důvodu, že algoritmus detekce horizontu nebyl implementován na mobilní platformě, aplikace dále žádá uživatele, aby nastavil polohu horizontu stejným způsobem, jakým nastavoval polohu referenčního objektu. Potom aplikace žádá o potvrzení polohy horizontu stejným tlačítkem jako v předchozím případě. Tato obrazovka je na obrázku 25.



Obrázek 25: Kalibrační mód — čekání na potvrzení polohy horizontu

Po potvrzení polohy aplikace ukládá aktuální polohu poskytnutou lokalizačními službami zařízení a provádí odhad vzdálenosti na základě informace o poloze objektu a horizontu. Dále je uživatel žádán, aby dojel k referenčnímu objektu, který byl zvolen v první fázi kalibrace. Po dojetí do konečné polohy by měl uživatel stisknout potvrzovací tlačítko, a tím ukončuje běh kalibračního režimu. Aplikace vyčítá novou polohu zařízení pomocí lokalizačních služeb a vypočítává přesnou vzdálenost k referenčnímu objektu (nástroj pro dané výpočty je uveden v kapitole 2.4.2), návrh této obrazovky je na obrázku 26.

Obrázek 26: Konec běhu kalibračního módu

Teď je známo všechno, co je nutné pro výpočet koeficientu k dle vztahu 38. Pokud kalibrace proběhla úspěšně, aplikace pokračuje demonstračním módem, v opačném případě (např. hodnota koeficientu se hodně odlišuje od 1) kalibraci je nutné zopakovat.

Kalibrační proces může být velice důležitý pro přesný odhad vzdálenosti a v daném návrhu je kladena velká zodpovědnost na uživatele. Metoda s menším vlivem uživatele na přesnost kalibrace by byla vhodnější. S prostředky, které jsou v dané úloze k dispozici, se jeví toto řešení jako jedno z mála dosažitelné a vyhovující.

Popis mobilní aplikace pokračuje podkapitolou, která se věnuje návrhu a realizaci demonstračního módu aplikace. Dále v kapitole 5 je diskutováno o tom, zda kalibrace skutečně zpřesňuje výsledky či nikoliv a zda je její přítomnost v aplikaci nezbytná.

4.2 Demonstrační mód

V této podkapitole popíši návrh funkčního režimu mobilní aplikace. Jak už bylo řečeno, z výkonnostních důvodů bylo rozhodnuto neimplementovat algoritmus detekce na mobilní platformě, proto představuji jen návrh tohoto režimu, který je ve výsledné aplikaci demonstrován na videozáznamu a výstupech z detektorů aut a horizontu implementovaných na platformě Windows pomocí knihovny OpenCV.

4.2.1 Práce se vstupními daty

Vstupem tohoto režimu je videozáznam, který byl natočen na kameru mobilního přístroje. Tento záznam předem prochází detektorem aut a následně i detektorem horizontu, což vygeneruje dva textové soubory. V prvním jsou data z rozpoznávače aut: pro některé snímky jsou definovány okénka zabalující detekovaná auta do sebe, formát je popsán v kapitole 3.2.4. V druhém souboru jsou polohy horizontu pro každý snímek videozáznamu.

Pro zobrazení videozáznamů v Androidu je speciální vzhled (angl. view) - *VideoView*. *VideoView* umožňuje provádět základní operace jako např. přehrávání, zastavení videa, atd. Mimo jiné také umožňuje vyčíst aktuální časovou pozici přehrávaného videa, toto za-

jišťuje metoda *VideoView.getCurrentPosition()*. Tato metoda vrátí aktuální časovou pozici přehrávání v milisekundách. Třída *VideoView* tedy nemá žádnou metodu, která by vrátila aktuální číslo snímku, je možné zjistit pouze časový údaj. Jelikož data ve výstupech detektorů jsou vázaná na číslo snímku, je nutné číslo snímku přepočítat na aktuální časový okamžik ve videu. To se dá udělat, pokud víme, kolik snímků za sekundu má video (angl. frames per second - fps). Tedy pokud video má $Ffps$, potom pro snímek číslo n se jeho časový okamžik ve videu bude rovnat: $t = \frac{1000}{F} \cdot n$. Po přiřazení čísel snímků lze zesynchronizovat video a výstupy z detektorů.

4.2.2 Návrh uživatelského rozhraní

Jedním z cílů této práce je návrh a realizace přívětivého uživatelského rozhraní pro danou aplikaci. Uživatelské rozhraní má být maximálně intuitivní a hlavně nerušit řidiče při jízdě. Aplikace by taky neměla vyžadovat od řidiče zvláštní pozornost, jinak by mohlo potenciálně hrozit nebezpečí nehody z důvodu nepozornosti řidiče.

Smyslem aplikace je pomoct řidiči naučit se intuitivně cítit bezpečnou vzdálenost k předchozím vozidlům a varovat ho v případě, že se přiblížil na příliš krátkou vzdálenost. Proto návrh rozhraní vypadá, jak ukazuje obrázek 27.

Obrázek 27: Návrh uživatelského rozhraní

Uživatelské rozhraní tedy obsahuje:

- okno s videem;
- zobrazení odhadnuté vzdálenosti;
- zobrazení aktuální rychlosti;
- indikátor přiblížení (barevný rámeček);
- oznámení pomocí vibrací.

Zobrazení vzdálenosti

Aktuální vzdálenost se zobrazuje ve spodní části obrazovky. Zobrazuje se tedy pouze jedna hodnota a platí pro vozidlo, které jede bezprostředně nejblíže před námi. Hodnota se dá zobrazovat v metrech a v sekundách, dotykem se mezi režimy přepíná. Zobrazení překonání vzdálenosti v sekundách může více vypovídat o aktuální situaci v provozu, např. jízda na vzdálenosti 20 metrů při rychlosti 30 km/h je celkem bezpečná, ale při rychlosti 130 km/h velice nebezpečná.

Pro přepočet jednotek z metrů na sekundy se musí použít vztah:

$$t = \frac{s}{v}, \tag{39}$$

kde t je hledaná hodnota dojezdu v sekundách, s je vzdálenost k předchozímu autu v metrech a v je aktuální rychlost jízdy v m/s. Pro výpočet nám tedy chybí hodnota rychlosti v.

Zobrazení rychlosti

V případě, že by v aplikaci byl realizován detekční algoritmus a aplikace by se používala za provozu, potom výpočet rychlosti by se dal udělat pomocí lokalizačních služeb. Vycházím z toho, že na trhu (Google Play) existuje řada jednoduchých aplikací pro měření rychlosti za provozu a dle zkušeností uživatelů fungují dostatečně přesně. V našem případě by se rychlost jízdy z videa odvodit nedala, resp. velice těžko. Proto se pro demonstraci v poli zobrazení rychlosti objevuje náhodná hodnota. Při startu se tato hodnota inicializuje na 50 km/h a potom se modifikuje každou sekundu následujícím způsobem, se stejnou pravděpodobností mohou nastat tři případy: hodnota rychlosti se sníží o 1, zůstane stejná nebo se zvýší o 1.

Indikátor přiblížení

Tento indikátor byl navržen tak, aby se řidič zbytečně nerozptyloval čtením čísel a písmen na obrazovce. Indikátor je tedy barevný rámeček na krajích obrazovky. Indikátor se různě zvětšuje a zmenšuje a mění barvu v závislosti na vzdálenosti k předchozímu vozidlu. Čím blíž je předchozí auto, tím je indikátor červenější a větší. A naopak čím bezpečnější je vzdálenost, tím je indikátor zelenější a menší. Na sadě obrázků 28 vidíme příklad vzhledu indikátoru při vzdálenostech: 45 metrů, 14 metrů a 2 sekundy.

Při obarvení indikátoru vycházím ze známého praktického pravidla dvou sekund [30] [31]. Odstup od předchozího vozidla by neměl být menší než dvě sekundy, jinak je vzdálenost nebezpečná. Proto je barva indikátoru na dvou vteřinách jemně oranžová, znamená to, že se blíží hranice bezpečnosti a řidič by měl raději zpomalit.

Obrázek 28: Příklad vzhledu indikátoru při vzdálenostech 45 metrů, 14 metrů a 2 sekundy

Oznámení vibrací

V případě, že řidič se dostává do nebezpečného stavu, aplikace by ho o tom měla varovat vhodným způsobem. Zvukové signály mohou být často nepříjemné a v případě, že aplikace špatně vyhodnotila situaci a řidič se pouze snaží předjet vozidlo, zvukový signál by mohl odradit řidiče od používání aplikace. Proto byl zvolen jiný přístup — vibrace.

V případě, že řidič překročí minimální bezpečnou vzdálenost (2 sekundy), mobilní přístroj začne vibrovat, aby upozornil řidiče, že by měl zpomalit. Pokud i přesto řidič zrychluje a vzdálenost je menší než sekunda, mobilní přistroj začne vibrovat intenzivněji.

5 Testování

V této kapitole popíši experimenty a průběh testování výsledné mobilní aplikace. Nejdříve se podíváme na experimenty s kalibračním módem aplikace, zjistíme jaké přesnosti se dá dosáhnout a zda ji kalibrační mód skutečně zlepší. Potom se podíváme na testování přesnosti výsledné metody odhadu vzdálenosti.

5.1 Experimenty v kalibračním módu

Smyslem existence kalibračního módu je zpřesnění výsledků odhadu vzdálenosti. Vznikl kvůli neznalosti přesné výšky, ve které je namontováno mobilní zařízení v autě. Tento mód byl realizován pomocí lokalizačních služeb (mobilní síť nebo GPS modul), podrobnější popis realizace je popsán v kapitole 4.1. V tomto režimu spoléháme na to, že lokalizační služby dokážou poskytnout dostatečnou přesnost, abychom mohli skutečně vylepšit přesnost odhadu.

Testování probíhalo pomocí zařízení Samsung Galaxy Note 3, více informací o tomto zařízení je zde [36]. Pro testování jsem namontoval mobilní zařízení na držák za čelním sklem v autě a na různých vzdálenostech od referenčního objektu (jiné auto) jsem prováděl odhad vzdálenosti pomocí zákonů perspektivy (kapitola 2.3), tento krok testování je vidět na obrázku 29. Reálná vzdálenost k objektu byla změřená přesně jiným měřicím nástrojem (pásmo pro měření délky). Následně jsem k referenčnímu objektu najel a pomocí lokalizačních služeb zjistil ujetou vzdálenost. Před každým testem jsem se ujistil, že GPS signál je dost silný (pomocí aplikace GPS Status).

Obrázek 29: Odhad vzdálenosti v kalibračním módu

V průběhu testování bylo zjištěno, že přesnost lokalizačních služeb pro měření tak malých vzdáleností (do 100 metrů) není dostatečná. Například pro reálnou vzdálenost 35 metrů aplikace provedla odhad 34 metrů. Při opakovaném měření vzdálenosti pomocí lokalizačních služeb byly naměřeny hodnoty: 16.3 m, 28 m, 0.2 m, 10.4 m, 329.5 m. Samozřejmě, hodnoty 0.2 m a 329.5 m se dají jednoduše odfiltrovat jako nereálné, ale i přesto měřit tak

krátkou vzdálenost pomocí lokalizačních služeb nemá smysl (hodnoty jsou nepřesné a je mezi nimi velký rozptyl). Experiment jsem opakoval i pro reálnou vzdálenost 55 metrů, odhad aplikace vizuální cestou byl 48.2 metrů. Provedl jsem také opakované měření pomocí lokalizačních služeb, výsledky byly rovněž nejednoznačné.

Z toho vyplývá, že rozumnější je dát uživateli možnost zadat přibližnou výšku, ve které má namontované zařízení, než provádět kalibraci, jak byla popsána výše. Z tohoto důvodu jsem po tomto testování do aplikace přidal možnost zadávání výšky. Ponechal jsem také kalibrační mód jen na ukázku, demonstrační mód na kalibrační data nebere ohled.

5.2 Testování přesnosti odhadu vzdálenosti

V této kapitole se podíváme na přesnost samotné metody odhadu vzdálenosti. Toto testování probíhalo na fotografiích pořízených z kamery mobilního přístroje namontovaného za čelní sklo auta, přitom je známá přesná vzdálenost k referenčnímu objektu na snímku (změřená předem ruletou). Vycházíme z předpokladu, že poloha horizontu a auta v obraze je určená správně. V následující tabulce je výběr výsledků experimentů. Byla provedena řada měření na vzdálenostech 3 až 55 metrů.

Skutečná vzdálenost [m]	Naměřená vzdálenost [m]	Chyba [%]
3	3.76	+25.3
9	10.37	+15.2
12	12.37	+3.1
15	16.45	+9.7
24	26.91	+12.1
39	43.6	+11.8
55	59.11	+7.4

Tabulka 4: Výběr výsledků testování přesnosti metody odhadu vzdálenosti

Jak je vidět z tabulky 4, kromě vzdáleností menších než 10 metrů, chybovost nepřesahuje 15 %. Chyby jsou odůvodněné tím, že i ve statickém obraze nelze přesně určit polohu referenčního objektu a horizontu. Tato chyba však není kritická a metoda je použitelná k účelu řešení daného problému.

6 Závěr

Zadáním mé diplomové práce bylo navrhnout a vytvořit aplikaci pro operační systém Android, která by čistě vizuální cestou odhadovala vzdálenost k předcházejícím autům v reálném provoze. Má práce byla zaměřená na několik základních aspektů: nalezení vhodného algoritmu detekce aut v provozu, návrh metody odhadu vzdálenosti vizuální cestou a návrh uživatelského rozhraní pro mobilní aplikaci.

Provedl jsem analýzu algoritmů, které již existují a navrhl jsem vlastní metodu detekce aut. Navržená metoda funguje na základě koncepce spojení metod sledování významných bodů v obraze a kaskádového klasifikátoru. Natrénovaný kaskádový klasifikátor projde obrazem několikrát, aby našel oblasti na obraze, které mohou být identifikovány jako auta. Na výstup klasifikátoru se aplikuje verifikátor, který potvrzuje nebo zamítá rozhodnutí klasifikátoru. Verifikace probíhá dle dvou metod: hledání symetrie (auto je zezadu do jisté míry symetrické vůči vertikální ose) a hledání horizontálních hran (předpokládá se, že na autě je zezadu několik horizontálních hran: registrační značka, horní a dolní hrana skla, atd.). Poté se pomocí funkcí knihovny OpenCV naleznou v obraze významné body, ze kterých se vyberou pouze ty, které se s největší pravděpodobností vyskytují na hledaném objektu (body, které jsou blíže k centru). Tyto body se sledují a shluky bodů, které se nerozpadají jsou považovány za potencionálně hledané objekty. V případě, že shluk významných bodů se překrývá s oblastí, o které klasifikátor rozhodl, že se v ní nachází auto, pak je v této oblasti auto. Navržená metoda se zakládá na spolehlivosti klasifikátoru.

Odhad vzdálenosti vizuální cestou je založen na zákonech perspektivy. Pokud známe vlastnosti kamery, víme v jaké výšce je nainstalována a dokážeme odhadnout, kde se sledované vozidlo střetává s vozovkou, pak můžeme s velkou přesností odhadnout vzdálenost.

V průběhu práce se ukázalo jako nezbytné umět určovat polohu horizontu v obraze. Je to nutné pro kvalitní odhad vzdálenosti, poněvadž vzdálenost od objektu je určena počtem pixelů mezi horizontem a místem kontaktu objektu se zemí. Navrhl jsem algoritmus odvození polohy horizontu z vodorovného značení na vozovce. Pokud tyto vodorovné linie prodloužíme, tak se překříží v místě, kde je poloha horizontu.

Detektory aut a horizontu byli realizovány na platformě Windows a testovány na plnohodnotném počítači (notebooku). Rychlost detekce bohužel nedovolila následné nasázení těchto detektorů na mobilní platformu.

V další fázi projektu byla navržena mobilní aplikace na platformě Android. Nejdříve byl realizován kalibrační mód aplikace, který měl za úkol zpřesnit výsledky odhadu vzdálenosti. Princip se zakládá na měření skutečné vzdálenosti k objektu pomocí lokalizačních služeb. V průběhu testování se ukázalo, že lokalizační služby Android nedokážou poskytnou dostatečně přesná data a kalibrace v tomto duchu tím pádem přestává mít smysl.

Dále bylo navrženo uživatelské rozhraní aplikace při normálním běhu. Při spuštění aplikace se tedy spustí demonstrační režim, který na videu a výstupech z detektorů aut

a horizontu ukáže, jak by vypadala aplikace za běhu v případě, že by se detektory také implementovaly na mobilní platformě.

V závěrečné fázi práce byly provedeny experimenty přesnosti navržené metody odhadu vzdálenosti. Testy ukazují, že metoda je použitelná a pro vzdálenosti větší 10 metrů má chybu menší než 15 %.

Reference

[1] STEIN Gideon P.; MANO O.; SHASHUA A., *Vision-based ACC with a Single Camera: Bounds on Range and Range Rate Accuracy*, 2003.

[2] KLÍMA Jindřich, *BINOKULÁRNÍ VIDĚNÍ A VÝROBA ANAGLYFŮ*, 2003.

[3] Oční klinika NeoVize, *Zrak a řízení motorových vozidel*, `http://www.lekari-online.cz/ocni-lekarstvi/novinky/zrak-a-rizeni-motorovych-vozidel`, 2009.

[4] GARDNER Rags, *Digital SLR Lens Properties and Terminology*, `http://www.rags-int-inc.com/PhotoTechStuff/Lens101/`, 2003.

[5] MROVLJE Jernej, VRANČIĆ Damir, *Distance measuring based on stereoscopic pictures*, 2008.

[6] DOLCOURT Jessica, *Camera megapixels: Why more isn't always better*, `http://www.cnet.com/news/camera-megapixels-why-more-isnt-always-better-smartphones-unlocked/`, 2012.

[7] Camera Image Sensor, *Senzor Size*, `http://cameraimagesensor.com/size/`, 2015.

[8] iFixit, *iFixit Store*, `https://www.ifixit.com`, 2015.

[9] VIOLA P., JONES M. J., *Rapid Object Detection using a Boosted Cascade of Simple Features*, 2001.

[10] Opencv dev team, *Cascade Classification*, `http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html`, 2014.

[11] STANISLAV Dušek, *Určení parametrů pohybu ze snímků kamery*, 2009.

[12] JIANBO Shi, CARLO Tomasi, *Good Features to Track*, 1994.

[13] JEAN-YVES Bouguet, *Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm*, 2001.

[14] CHRIS Harris, MIKE Stephens, *A Combined Corner and Edge Detector*, 1988.

[15] GENE H. Golub, HENK A. van der Vorst, *Eigenvalue computation in the 20th century*, 2000.

[16] National Geographic Society, *Horizon*, `http://education.nationalgeographic.com/education/encyclopedia/horizon/?ar_a=1`.

[17] Opencv dev team, *Canny Edge Detector*, `http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`, 2015.

[18] ZOLTÁN Fegyver, *Understanding the Hough transform*, http://matlabtricks.com/post-39/understanding-the-hough-transform, 2014.

[19] Opencv dev team, *Hough Line Transform*, http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html#id1, 2015.

[20] MCDERMOTT, J., WEISS, Y., ADELSON, E.H., *Beyond junctions: Nonlocal form contraints on motion interpretation.*, http://web.mit.edu/persci/demos/Motion&Form/demos/one-square/one-square.html, 2001.

[21] BLATNÝ Petr, *MOBILNÍ SYSTÉM PRO PODPORU CESTOVÁNÍ*, http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=15979&file=t, 2014.

[22] DONG Ho Han, *Android, at a glance*, http://www.cubrid.org/blog/dev-platform/android-at-a-glance/, 2011.

[23] KOUDELKA Jan, *NÁSTROJ PRO SPRÁVU PROJEKTŮ S VYUŽITÍM TABLETŮ PLATFORMY ANDROID*, http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=16847&file=t, 2014.

[24] *App Manifest*, http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[25] *App Components*, http://developer.android.com/guide/components/index.html.

[26] NASA, *THE EGM96 GEOID UNDULATION WITH RESPECT TO THE WGS84 ELLIPSOID*, http://cddis.nasa.gov/926/egm96/doc/S11.HTML, 2004.

[27] Intel Corporation, *Intel® Core™ i7-4710HQ Processor*, http://ark.intel.com/products/78930/Intel-Core-i7-4710HQ-Processor-6M-Cache-up-to-3_50-GHz.

[28] Pronto Marketing, *Processors: Computer vs mobile*, http://www.techadvisory.org/2013/12/processors-computer-vs-mobile/, 2013.

[29] BAUER Christine, *On the (In-)Accuracy of GPS Measures of Smartphones: A Study of Running Tracking Applications*, http://www.researchgate.net/profile/Christine_Bauer/publication/259190145_On_the_(In-)Accuracy_of_GPS_Measures_of_Smartphones_A_Study_of_Running_Tracking_Applications/links/00b4952a7a7f01b848000000.pdf, 2013.

[30] KÉZROVÁ Eva, *Bezpečná vzdálenost a plynulá jízda v kolonách - jezděte po dálnici správně*, http://www.rozhlas.cz/zelenavlna/motozurnal/_zprava/bezpecna-vzdalenost-a-plynula-jizda-v-kolonach-jezdete-po-dalnici-spravne--1368665, 2014.

[31] MLČOCH Zbyněk, MUDr., *Bezpečná jízda vozidlem, odhad vzdálenosti mezi vozidly*, http://www.zbynekmlcoch.cz/informace/texty/automobily-motocykly/bezpecna-jizda-vozidlem-odhad-vzdalenosti-mezi-vozidly, 2008.

[32] LEE David C., *Boosted Classifier for Car Detection*, ECE Department, Carnegie Mellon University, U.S.A.

[33] Prodigy Productions, LLC, *Lane Detection with OpenCV and C#*, http://www.prodigyproductionsllc.com/articles/programming/lane-detection-with-opencv-and-c/, 2013.

[34] BOURDONNAYE A. de La, DOSKOČIL R., KŘIVÁNEK V., ŠTEFEK A., *Practical Experience with Distance Measurement Based on Single Visual Camera*, 2012.

[35] ATKINS Bob, *Field of View - Rectilinear and Fishye Lenses*, http://www.bobatkins.com/photography/technical/field_of_view.html, 2014.

[36] http://www.devicespecifications.com, *Samsung Galaxy Note 3 N9000 - Specifications*, http://www.devicespecifications.com/en/model/9de0289b.

7 Seznam příloh

Příloha A: DVD disk s aplikaci a zdrojovými kódy.