Mendelova univerzita v Brně
Provozně ekonomická fakulta

Návrh asistenčního systému řízení pro drážní vozidla

Bakalářská práce

Vedoucí práce:
Ing. Marcel Vytečka

Jakub Jůza

Brno 2016

Rád bych poděkoval vedoucímu práce, Ing. Marcelu Vytečkovi, za jeho čas a ochotu, se kterou mi pomáhal při vypracovávání této práce. Zároveň bych chtěl poděkovat doc. Dr. Ing. Jiřímu Rybičkovi a Klubu modelářů železnic Brno 1 za umožnění práce v Laboratoři řízení kolejových vozidel a spolupráci při tvorbě programu.

Čestné prohlášení

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

V Brně dne 17. května 2016 .....

Abstract

Jůza, J. Proposal of an assistance system for railway vehicle control. Bachelor thesis. Brno, 2016.

This bachelor thesis deals with railway sign recognition in an image captured by a camera placed in a model of a railway vehicle and an adequate reaction of the railway vehicle to the sign that was found. Theoretical part of the thesis describes several image recognition libraries that could be used in solving this problem and some of the techniques the libraries employ in these tasks, a few programming languages that were considered for implementation of a demonstration program, followed by a description of some of the technologies available for networking and creating a graphical user interface in the programming language of choice. Practical part focuses on describing the process of creating the demonstration program, the training of railway sign detection classifiers and the results they achieved.

Abstrakt

Jůza, J. Návrh asistenčního systému řízení pro drážní vozidla. Bakalářská práce. Brno, 2016.

Tato bakalářská práce se zabývá rozpoznáváním železničního značení v obraze zachyceném kamerou umístěnou v modelu drážního vozidla a následné příslušné reakci vozidla na nalezenou značku. Teoretická část práce popisuje několik knihoven pro rozpoznávání objektů v obraze, které by mohly být použity k řešení tohoto problému a některé z technik, jež při těchto úkonech využívají, několik programovacích jazyků, které byly zvažovány pro implementaci demonstračního programu. Následuje popis některých technologií pro síťovou komunikaci a tvorbu grafického uživatelského rozhraní dostupných ve zvoleném programovacím jazyce. Praktická část se zaměřuje na popis procesu tvorby demonstračního programu, trénování klasifikátorů pro detekci železničního značení a výsledků, kterých tyto dosáhly.

Obsah

1 Úvod a cíl práce

1.1 Úvod

V posledních letech se dostává stále větší pozornosti automatizaci v prostředí dopravních prostředků. To je pochopitelné, neboť nejčastější příčinou nehod je stále lidský faktor a automatizace dopravy by snížila nároky kladené na osoby obsluhující dopravní prostředky. Stále častěji jsou v médiích zmiňovány autonomní automobily, které jsou vyvíjeny velkými světovými firmami jako je Google (https://www.google.com/selfdrivingcar/), Tesla (https://www.teslamotors.com/) a další, avšak vývoj probíhá i v České republice v podobě projektu Roboauto (http://roboauto.cz/). Tato práce pojednává o systému, který se, jak název práce napovídá, zaměřuje na asistenci řízení vozidel kolejových, čili vlaků. V této oblasti platí, že za množství nehod mohou nepozorní řidiči a lidé přecházející přes koleje, přesto by takový systém mohl dopomoci k redukci počtu železničních nehod a jsem přesvědčen, že jakákoli možnost snížení jejich množství stojí za to, aby se jí věnovala pozornost.

Systém je implementován v laboratoři řízení kolejových vozidel Mendelovy univerzity na modelu historické trati Kuřim–Veverská Bitýška. Obraz je zachycován pomocí kamery umístěné v modelu lokomotivy, odkud se přenáší v analogové podobě a následně je digitalizován pomocí USB Analog/Digital převodníku. Obraz je poté zpracován a vyhodnocen na počítači k tomu zvlášť určeném, neboť se jedná o vysoce výpočetně náročné úlohy. Pokud byla v obraze nalezena značka, systém určí odpovídající reakci na ni a zašle příkaz na server ovládající modelové kolejiště.

Pokud bude po úspěšné implementaci a následném otestování systému dosaženo dostatečné úspěšnosti při klasifikaci železničních značek, je možné dosažené poznatky využít k implementaci podobného systému fungujícího na reálné testovací železniční trati a ve vzdálenější budoucnosti i v běžném železničním provozu.

1.2 Cíl práce

Cílem práce je vybudovat systém, který by byl co nejspolehlivěji schopný detekovat železniční značky u modelu trati, správně je klasifikovat, vyhodnotit vhodnou reakci na značku a následně zaslat odpovídající příkaz na server ovládající modely vlaků.

2 Počítačové vidění

2.1 Úvod do počítačového vidění

Cílem počítačového vidění je vykonávání užitečných rozhodnutí o reálných fyzických objektech a scénách založených na zachycených snímcích. (Shapiro a Stockman, 2001) K rozhodování o těchto reálných objektech je však nejdříve nutné sestavit jejich popis nebo model ze zachyceného snímku. Z tohoto důvodu někteří experti tvrdí, že cílem počítačového vidění je spíše sestavování popisů a modelů objektů ze zachycených snímků.

Inteligentní stroje se schopností vidět a rozpoznávat objekty reálného světa byly již dlouhou dobu předmětem zájmu nejen Sci-Fi autorů a fanoušků, ale především vědců zabývajících se umělou inteligencí a informatikou obecně. Již Alan Turing, jeden ze zakladatelů oboru umělé inteligence, věřil, že lidé dají počítačům možnost vidění a rozpoznávání scén.

Počátky počítačového vidění jako vědní disciplíny sahají do přelomu 60. a 70. let, kdy si však vědci nedokázali představit, o jak komplexní problém se jedná. K ilustraci jejich představ o problému může posloužit tato příhoda z roku 1966. Martin Minsky na MIT zadal svému studentovi bakalářského studia, aby „strávil léto tím, že propojí kameru s počítačem a přiměje počítač, aby popsal, co vidí“ (Boden, 2006, s. 781). Později vědci pochopili, že se jedná o problém značně komplexnější. Avšak díky vyspělejším technologiím a pokroku nejen v matematických výpočtech, ale i dalších disciplínách spojených s počítačovým viděním se v posledních desetiletích vývoj v této vědní disciplíně posunul značně vpřed a nyní počítačové vidění nachází uplatnění ve všech možných oblastech lidského života.

2.2 Příklady využití počítačového vidění

Automatické rozpoznávání poznávacích značek

Tato aplikace počítačového vidění je postavena na technologii OCR neboli optické rozpoznávání znaků (z anglického Optical Character Recognition). Využití nachází především u různých policejních sborů, které automatické rozpoznávání poznávacích značek používají například v systému vybírání poplatků za placené silnice, k identifikaci kradených vozidel, zaznamenání SPZ při dopravních přestupcích atd.

Detekce a rozpoznání obličeje

Většina moderních digitálních fotoaparátů a chytrých mobilních telefonů je vybavena funkcí detekce obličejů pro automatické zaostřování. Rozpoznání obličeje pak nachází uplatnění v zabezpečení, kdy se musí osoba prokázat pomocí svého obličeje, např. technologie VeriFace, kterou u svých notebooků využívá společnost Lenovo (http://www.lenovo.com/).

Rozpoznávání gest

Rozpoznávání gest je dalším krokem k jednodušší interakci mezi počítačem a člověkem, neboť počítači pomocí matematických algoritmů umožňuje lépe rozumět řeči lidského těla. Tato technologie může být využita nejen k rozpoznávání emocí člověka z jeho gestikulace, ale například i k překladům znakové řeči.

2.3 Reprezentace digitálního obrazu

Jelikož je obraz reprezentován jako matice $m \times n$, kde oba rozměry dosahují řádů stovek až tisíců pixelů, bylo by výpočetně náročné a tedy i pomalé porovnávat obrazy pixel po pixelu. Matematici a vědci zabývající se počítačovým viděním tedy museli přijít na způsob reprezentace obrazu, který by umožnil zjednodušení a zrychlení výpočtů a operací nad obrazy. Způsobů, na které přišli je několik a v následujících odstavcích představím dva z nich, které patří k těm nejznámějším a jsou využívány při tvorbě kaskádového klasifikátoru v knihovně OpenCV.

Haar-like příznaky

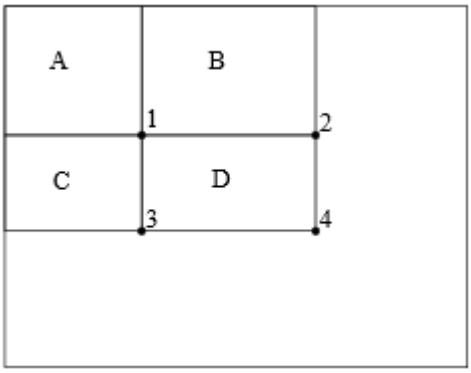
Tento způsob reprezentace digitálního obrazu používaný při detekci objektů v obraze byl poprvé publikován v roce 2001 dvojicí Viola a Jones, kteří navázali na práci, zveřejněnou v roce 1998 vědci C. Papageorgiou, M. Oren a T. Poggio. Motivací pro vyvinutí této techniky byla především detekce obličejů v obraze. Výstupem práce dvojice Viola, Jones byl první systém schopný detekovat obličeje v obraze v reálném čase, přičemž dosahoval úspěšnosti nejlepších detektorů obličeje té doby. Ostatní systémy však pracovaly s intenzitami obrazu jednotlivých pixelů apod., což činilo detekci vysoce výpočetně náročnou a byly tedy pomalé.

Jméno této metody klasifikace obrazu je odvozeno od Haarovy vlnky. Této funkce a složitějších funkcí od ní odvozených se využívá k vytvoření tzv. integrálního obrazu. V tomto integrálním obrazu ma každý bod o souřadnicích x, y určitou hodnotu – sumu intenzity bodů směrem nahoru a vlevo od daného bodu, která se spočítá pomocí rovnice 1.

$$I_{\sum}(x,y) = \sum_{\substack{x^{'}<=x \\ y^{'}<=y}} i(x^{'},y^{'}) \tag{1}$$

Tato technika má za následek to, že po vypočítání hodnot pro všechny body v obraze je možné jakýkoliv Haar-like příznak vypočítat v konstantním čase bez ohledu na jeho pozici, či měřítko. Při porovnávání se používá obdélníkových příznaků, jejichž hodnota lze, jak již bylo zmíněno, spočítat v konstantním čase za využití čtyř bodů. Výpočet sumy obdélníka D je znázorněn v rovnici 2 za využití ilustrace na obrázku 1.

$$D = 4 + 1 - 2 - 3 \tag{2}$$



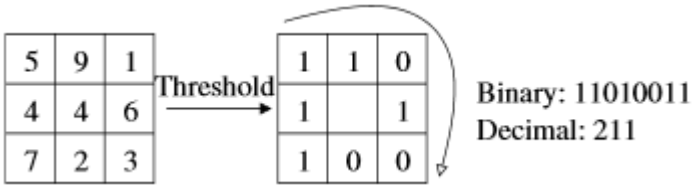
Obrázek 1: Znázornění výpočtu hodnoty obdélníku (Viola, Jones, 2001)

Pro výpočet rozdílu dvou obdélníků je tedy třeba osm bodů (šest pokud se jedná o obdélníky sousedící).

Metoda je založena na třech druzích příznaků, které využívají dva, tři a čtyři obdélníky. U prvního druhu se jedná o rozdíl dvou stejně velkých, stejně orientovaných a horizontálně nebo vertikálně sousedících obdélníků. U tří obdélníků se jedná o součet dvou vnějších obdélníků odečtený od obdélníku prostředního. Poslední příznak se vypočítá jako rozdíl dvou diagonálních párů obdélníků (Viola, Jones, 2001) (Papageorgiou, Oren, Poggio, 1998).

Local Binary Patterns

LBP je založeno na metodě zvané Texture Spectrum model, která byla poprvé navržena v roce 1990 čínskými vědci Dong-Chen He a Li Wang (He, Wang, 1990). Samotné LBP bylo popsáno v roce 1996 trojicí Ojala T., Pietikäinen M., Harwood D. z finské University of Oulu. Původní verze funguje tak, že oblast o rozměrech 3×3 ohodnotí hodnotou prostředního pixelu (Ojala, Pietikäinen, Harwood, 1996). Získání této hodnoty je znázorněno na Obrázku 2.



Obrázek 2: Získání hodnoty prostředního pixelu (Ahonen, Hadid, Pietikäinen, 2006)

To, zda pixelu přiřadíme hodnotu 0 nebo 1 záleží na porovnání s prostředním pixelem, kdy jsou pixely s vyšší a stejnou intenzitou šedi jako prostřední ohodnoceny jedničkou a pixely s menší nulou. Z těchto jedniček a nul se následně sestaví binární číslo, které představuje hodnotu bloku pixelů (Mäenpää, 2003) (Vijayalakshmi,

Subbiah Bharathi, 2011). Další způsob pro výpočet hodnoty v dekadické soustavě vidíme naznačen v rovnici 3.

$$\begin{aligned} LBP &= \sum_{i=1}^{8} E_i \ \cdot 2^{i-1} \\ E_i \ &= \begin{cases} 1 & \text{pokud } V_i \ \geq V_0 \\ 0 & \text{pokud } V_i \ < V_0 \end{cases} \end{aligned} \tag{3}$$

Tuto metodu stejná skupina vědců v roce 2002 rozvinula přidáním možnosti volby velikosti bloků. Parametry, které si volíme jsou počet pixelů P a poloměr oblasti R; g_c je prostřední bod. Nejčastěji volenými hodnotami parametru P jsou P = 8 a P = 16. U této verze se využívá vzorec popsaný v rovnici 4.

$$\begin{aligned} LBP_{P,R} &= \sum_{p=0}^{P-1} s(g_p - g_c) \cdot 2^p \\ s(g_p - g_c) \ &= \begin{cases} 1 & \text{pokud } (g_p - g_c) \ \geq 0 \\ 0 & \text{pokud } (g_p - g_c) \ < 0 \end{cases} \end{aligned} \tag{4}$$

Dalším rozšířením jsou tzv. uniformní (jednotné) vzory. Binární číslo je uniformní, pokud se v něm nevyskytuje více přechodů z 0 na 1 (nebo z 1 na 0) nežli 2. Například číslo 00011000 tuto podmínku tedy splňuje, ale číslo 10100000 již nikoli. Matematicky lze počet přechodů zapsat pomocí rovnice 5.

$$U(G_P) = |s(g_{P-1} - g_c) - s(g_0 - g_c)| + \sum_{p=1}^{P-1} |s(g_p - g_c) - s(g_{p-1} - g_c)| \tag{5}$$

Tato metoda se uchytila poté, co Ojala a spol. zjistili, že až 90% vzorů v obraze je uniformních. Tohoto se dále využívá tak, že se všechny vzory, které nejsou uniformní označí jako 1 vzor.(Ojala, Pietikäinen, Mäenpää, 2002) To znamená, že pokud jako P zvolíme 8, dostáváme 256 různých označení, avšak z nich je jen 58 uniformních, z čehož nám u této metody vyplyne 59 možných označení oblasti a další snížení výpočetní náročnosti.

2.4 Kaskádový klasifikátor

Princip klasifikátoru

Pro detekci objektů v obraze OpenCV využívá kaskádový klasifikátor. Ten byl vyvinut dvojicí Viola a Jones v roce 2001 a použit pro detekci obličejů. Klasifikátor se označuje jako kaskádový, protože je potřeba jej aplikovat v několika krocích, jelikož

každý jednotlivý krok má úspěšnost jen o málo vyšší než náhodné hádání. Každý klasifikátor má N kroků a platí následující postup. Na kroku M (M < N) se ověří, zda detekovaný objekt odpovídá objektu hledanému. Pokud ne, detekovaný objekt není objektem hledaným, pokud ano, klasifikátor přejde ke kroku M+1. Postup se opakuje, dokud se nestane, že detekovaný objekt neodpovídá objektu hledanému nebo detekovaný objekt neprojde N-tým krokem, kdy se o něm prohlásí, že je objektem hledaným. Důležitými vlastnostmi kaskádového klasifikátoru jsou robustnost (nízký výskyt chyby 1. a 2. druhu) a vysoká rychlost, díky které se mu dostalo pozornosti i na úkor jiných algoritmů. (Viola, Jones, 2001)

Učení klasifikátoru

Pro učení klasifikátoru využívá OpenCV techniku zvanou boosting, což je skupina algoritmů používaných při tzv. supervised learningu neboli učení s učitelem, což je jedna z metod strojového učení. Myšlenka boostingu spočívá ve vytvoření silného klasifikátoru kombinací několika slabých klasifikátorů, pro něž je podmínkou chybovost nižší než $0,5$ a nejsou tedy příliš výpočetně náročné. Kombinací těchto slabých klasifikátorů však dostáváme silný klasifikátor, který často předčí výsledky jiných přístupů jakými jsou např. Support Vector Machine nebo neuronové sítě (Kearns, Valiant, 1989). Nejvýznamnější a nejpoužívanější formou boostingu je AdaBoost (Adaptive Boosting), který je schopen upravit pozdější slabé klasifikátory podle klasifikátorů předchozích. Nevýhodou AdaBoostu je citlivost na data, kdy špatně označená data mohou mít velký vliv na výsledný klasifikátor. Na druhou stranu jeho výhodou je nižší náchylnost na přeučení oproti ostatním modifikacím boostingu. Přeučení znamená, že klasifikátor je sice schopný dobrých výsledků na tréninkové množině dat, avšak na jiných datech jich zdaleka nedosahuje a byl tedy naučen příliš na míru tréninkovým datům. AdaBoost má opět několik variant, např. Real AdaBoost, Gentle AdaBoost, LogitBoost, Discrete AdaBoost a další, z nichž OpenCV pro tvorbu klasifikátoru nejčastěji využívá variantu Gentle AdaBoost. Postup při učení klasifikátoru za pomocí AdaBoostu je následující:

- Nejprve je všem vzorkům přiřazena stejná váha
- Proběhne učení slabého klasifikátoru na tréninkových datech
- Je spočtena směrodatná odchylka a škálovací faktor klasifikátoru
- Nesprávně klasifikovaným vzorkům je přiřazena vyšší váha
- Váhy vzorků jsou normalizovány
- Probíhá trénink dalšího slabého klasifikátoru

(Freund, Schapire, 1997)

2.5 Tvorba klasifikátoru z pohledu uživatele

Pro tvorbu klasifikátoru se používá k tomu určená utilita, která je součástí knihovny OpenCV. Při vytváření klasifikátoru je základem dostatečně velký soubor pozitivních a negativních vzorků. Pozitivní vzorky je možné vytvořit pomocí jiné utility, kterou opět poskytuje OpenCV, ta z jednoho obrázku transformací vytvoří požadované množství vzorků. Druhou možností je pořídit dostatečné množství opravdových fotografií objektu. První z možností je možné použít, pokud je hledaný objekt opravdu jednoznačně odlišitelný (např. logo OpenCV), většinou se však preferuje přístup druhý, pomocí kterého lze obecně získat klasifikátor spolehlivější. Doporučené množství pozitivních vzorků se liší od několika set (u jednodušších objektů, jako jsou dopravní značky) po tisíce (u komplexnějších objektů, např. lidské obličeje). U množství negativních vzorků neexistuje jednotné doporučení. Rainer Lienhart při tvorbě kaskády, kdy představoval rozšíření původního algoritmu vyvinutého dvojicí Viola a Jones použil poměr pozitivní:negativní vzorky 5:3 (Lienhart, Maydt, 2002), v komunitě OpenCV však převažuje názor, že by na 1 pozitivní vzorek měly připadat 2 negativní. Jednotný recept zde tedy neexistuje a platí, že se ideální poměr pozitivní:negativní liší od případu k případu.

3 Softwarové knihovny pro zpracování obrazu

3.1 ccv

Knihovna ccv se vyznačuje svou jednoduchostí, neboť zahrnuje implementaci pouze několika algoritmů z kategorie počítačového vidění a zpracování obrazu. Těchto několik algoritmů je však implementovaných kvalitně v jazyce C. K tvorbě knihovny se její autor rozhodl z toho důvodu, že v době, kdy práci na knihovně započal (rok 2010) existovaly buď kvalitní a ozkoušené implementace algoritmů starých, nebo nové algoritmy s implementací v lepším případě v MATLABu, a tedy v praxi naprosto nepoužitelné. Knihovna je dobrou volbou pro někoho, kdo potřebuje pouze vybrané algoritmy a ocení tedy její „lightweight“ provedení ve srovnání např. s knihovnou OpenCV, avšak musí počítat s omezenou uživatelskou přívětivostí a absencí širší komunity. Nespornou výhodou knihovny je její přenositelnost, neboť pro většinu funkcionality je zapotřebí jen plnohodnotného C překladače a jedinou komponentou mající nějakou závislost jsou konvoluční neuronové sítě se závislostí v podobě knihovny BLAS. http://libccv.org/

3.2 TINA

TINA je knihovnou, která má nejen zajímavý název (TINA Is No Acronym), ale i dlouhou historii sahající do roku 1986, kdy její vývoj započal na britské University of Sheffield, odkud se pak v roce 1998 přesunul na University of Manchester. Vývoj knihovny byl po čas její existence financován různými institucemi – kromě jiných i EU, která knihovnu zařadila do programu pro podporu Open Source software. Obsah knihovny je rozdělen na 2 části: tina-libs obsahující samotné algoritmy pro zpracování obrazu v podobě portabilního kódu v jazyce C a tina-tools – tato část staví na balíku tina-libs a umožňuje tvorbu aplikací využívajících knihovnu TINA. Kód, představující balík tina-tools je opět převážně v jazyce C, ale místy už se jedná o kód platformě závislý. Obsahem knihovny jsou nejen algoritmy pro zpracování obrazu a strojové vidění, ale i numerické a statistické funkce, či funkce pro lékařskou informatiku. http://www.tina-vision.net/

3.3 OpenCV

Základní informace

OpenCV neboli Open source Computer Vision je pravděpodobně nejpoužívanější knihovna pro počítačové vidění. Původně byla vyvinuta společností Intel, ale od roku 2012 je spravována neziskovou organizací OpenCV.org. Knihovna je dostupná na většině populárních operačních systémů včetně Linux, OS X, Windows, iOS, Android a dalších. První implementace byla realizována v jazyce C, avšak většina funkcí v novějších verzích knihovny (2.0 a dále) je napsána v jazyce C++.

Kromě již zmíněných jazyků dnes knihovna OpenCV nabízí interface pro další programovací jazyky jako Java, Python a další. Dále existují wrappery například pro .NET (Emgu CV), Ruby, Perl. Poslední verzí OpenCV je verze 3.1, která oproti verzím 2.4.x mimo jiné opravuje několik set nahlášených chyb, zlepšuje využití technologií OpenCL, CUDA a NEON a díky spolupráci s firmou Intel a lepšímu zapojení knihovny Integrated Performance Primitives (https://software.intel.com/en-us/intel-ipp) zvyšuje rychlost některých algoritmů až o 70%. (Garcia, Suarez, Aranda, Tercero, Gracia, Enano, 2015)

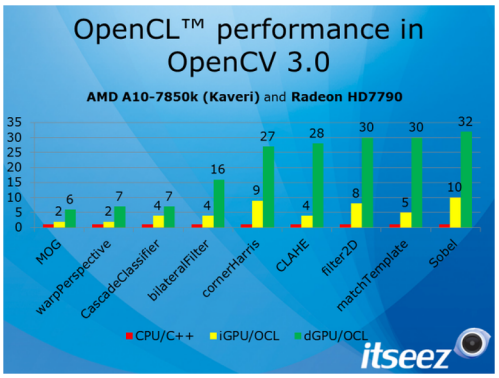
Rozpoznání objektů v obraze je pouze jednou z oblastí, na které se OpenCV zaměřuje. Dalšími moduly jsou například: trackování pohybu, extrakce 3D modelu objektu, sledování pohybu očí a další. (http://opencv.org/)

Rozpoznání objektů v obraze

Nynější verze OpenCV (3.1) dává na výběr ze 2 různých způsobů pro reprezentaci digitálního obrazu. Každá z technik má své klady a zápory a vyplatí se tedy vyzkoušet obě, jelikož jejich výsledky se liší v závislosti na datech. Metody, z kterých nám OpenCV dává na výběr jsou Haar-like příznaky a Local Binary Patterns. Ke klasifikaci objektů v obraze poté používá klasifikátor popsaný v sekci 2.4 v kapitole 2.

Využití OpenCL

OpenCL je volně dostupným standardem pro multi-platformní paralelní programování, který podporuje využití CPU, GPU, DSP apod. (Khronos) Možnost přenést výpočet na GPU přináší u mnoha algoritmů značné zrychlení, avšak je zapotřebí vlastnit grafickou kartu s podporou OpenCL. Dosažené zrychlení u několika algoritmů je možno vidět na Obrázku 3.

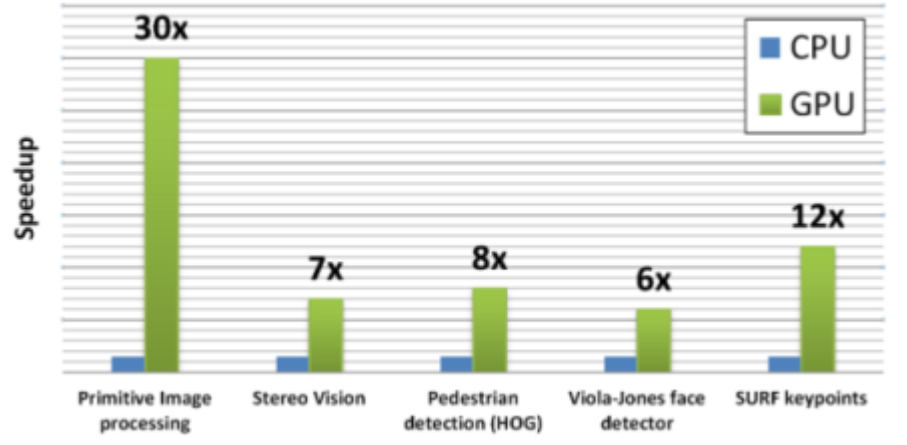


Obrázek 3: Rychlost algoritmů s využitím OpenCL (OpenCV)

Verze 3.0 knihovny OpenCV s sebou přinesla novinku v podobě tzv. Transparent API (zkráceně T-API), jehož implementace byla podporována firmami AMD a Intel. Toto API uživateli umožňuje napsat kód, který může běžet na CPU, ale pokud je v počítači dostupná grafická karta s podporou OpenCL, bude zpracování algoritmu přeneseno na GPU. V teoretické rovině tato slova sice zní krásně, avšak v praxi tato novinka nepřináší jen radosti. Může nastat situace, kdy má programátor pocit, že název API, tedy „transparentní“, je značně ironický, neboť nynější stav API způsobuje, že programátor ztrácí kontrolu nad tím, zda kód poběží na CPU nebo GPU. Mohlo by se zdát, že pokud je k dispozici GPU, měly by výpočty probíhat na něm, protože proběhnou rychleji, avšak pokud máme výkonný CPU a pouze málo výkonnou integrovanou grafickou kartu s podporou OpenCL, proběhnou naše výpočty pomaleji, a to někdy i méně než poloviční rychlostí. OpenCV sice nabízí způsoby, jak si vynutit zpracování dat na CPU, ale ty ne vždy fungují a poté nezbývá než znovu zkompilovat celou knihovnu bez podpory OpenCL, což je časově relativně náročný úkon. Ve verzi 3.1 ne zcela fungující možnost volby mezi CPU a GPU sice stále přetrvává, ale fakt, že se jedná o Open Source knihovnu by mohl přispět k brzkému vyřešení tohoto problému, což by odstranilo vadu na kráse této jinak velmi dobré novinky.

Využití CUDA

CUDA je platformou pro paralelní výpočty probíhající na GPU, která byla vyvinuta společností NVIDIA v roce 2003 a odhalena veřejnosti o tři roky později. Umožňuje vývojářům psát vysokoúrovňový kód pro GPU, který navíc často výkonnostně předčí kód ručně napsány v ASM. Abychom mohli využít možností CUDA, je nutné vlastnit grafickou kartu značky NVIDIA s podporou CUDA. (NVIDIA) Obrázek 4 poté nabízí výkonnostní porovnání mezi CPU a CUDA GPU.



Obrázek 4: Rychlost algoritmů s využitím CUDA (OpenCV)

U nové verze OpenCV sice u platformy CUDA nepřišli vývojáři knihovny s API, které by uživatelům usnadňovalo práci, jako tomu je u OpenCL, avšak jak jsem uvedl v předchozí sekci, větší pohodlí při vývoji nám někdy může v konečném důsledku způsobit více starostí a práce. Pro práci s algoritmy je v OpenCV modul nazvaný GPU a práce s ním se mírně liší od práce s moduly standardními. Na implementaci tohoto modulu se podíleli vývojáři ze společnost NVIDIA se zaměřením na rychlost a maximální využití možností platformy CUDA a algoritmy z tohoto modulu dosahují vyšší rychlosti nežli ty využívající OpenCL. Soustředění se na rychlost výpočtu však má i své stinné stránky, neboť například algoritmus pro rozpoznání objektů v obraze dosahuje při použití stejné kaskády klasifikátorů značně horších výsledků než jeho CPU implementace. Je tedy nutné počítat s tím, že u některých algoritmů může být získaná vyšší rychlost kompenzována nižší kvalitou výsledků. Nejlepším přístupem je pak vyzkoušet obě možnosti a na základě dosažených výsledků učinit konečné rozhodnutí.

4 Programovací jazyk

Při volbě programovacího jazyka vhodného k implementaci systému je nutné zvolit jazyk takový, který by byl dostatečně portabilní, rychlý a existovaly pro něj vhodné a kvalitní knihovny pro zpracování obrazu a rozpoznání objektů. Tyto předpoklady splňují Java, C a C++ a v této kapitole si jednotlivé jazyky blíže představíme.

4.1 Java

Historie

Historie jazyka Java sahá do konce roku 1990, kdy na něm začala pracovat skupina inženýrů společnosti Sun vedená Jamesem Goslingem. Hlavní pohnutkou k vytvoření Javy byla víra, že budoucnost IT leží ve spojení stolních počítačů a přenosných digitálních zařízení a zároveň frustrace ze stavu C a C++ API, které společnost Sun využívala. Cílem bylo tedy vyvinout jazyk, který by se v některých ohledech inspiroval u úspěšných jazyků C a C++ a zároveň by kladl nižší nároky a zodpovědnost na programátora. (Evans, Flanagan, 2014)

Java dnes

Z historického hlediska byla Java považována za pomalý jazyk, což dnes díky technologiím jako je např. Just-In-Time compilation (JIT) již neplatí. Zastánci jazyka rádi tvrdí, že v mnoha případech v rychlosti předčí C a C++, avšak pokud se na tyto případy podíváme blíže, většinou zjistíme, že se jedná o výjimky, kdy navíc autoři zcela porušují paradigmata pro psaní kódu v jazyce Java a užívají velice agresivních optimalizací. Java Virtual Machine (JVM), na které programy napsané v Javě běží, zaručuje, že všude, kde máme k dispozici JVM v dané verzi, nám naše programy poběží. Tato vlastnost Javy je jednou z klíčových a vztahuje se k ní slogan „write once, run everywhere“. Toto však s sebou nese i fakt, že i když náš program zrovna není spuštěný, JVM samo o sobě běží dál a zabírá systémové prostředky, což může být u některých systémů problémové. Další vlastností Javy, která je tentokráte bezesporu výhodou je nepřeberné množství knihoven, které usnadňují spoustu úkonů a šetří programátorovi čas. Jen je třeba (stejně jako u jiných programovacích jazyků) dbát na to, abychom využívali knihovny ověřené, které jsou implementovány efektivně a bez chyb.

4.2 C

Historie

Programovací jazyk C vyvinul v roce 1972 v Bell Labs Dennis Ritchie při práci na operačním systému Unix. Při tvorbě C vycházel Dennis Ritchie především z jazyků BCPL a B, od kterého se odvíjí i název jazyka C. V první polovině 70. let byla hlavní

výhodou jazyka C jeho přenositelnost mezi systémy a zároveň schopnost zachovat rychlost programů v něm napsaných. Z tohoto důvodu je dodnes C přezdíváno jako „portabilní assembler“. V roce 1978 byla dvojicí Dennis Ritchie a Brian Kernighan vydána kniha The C Programming Language, která se zároveň stala prvním standardem jazyka C a dnes platí za ekvivalent Bible tohoto jazyka. Raná vydání této knihy jsou velice žádaným sběratelským artiklem a často se na ni odkazují i jiné programovací jazyky. Ačkoli od vzniku „Céčka“ uplynulo již více než 40 let, stále se v něm píše nový kód a spousta programátorů na něj nedá dopustit. (Prinz, Kirch-Prinz, 2009)

Jazyk C dnes

Programovací jazyk C ušel od 70. let dlouhou cestu a standardy C89, C99 a C11 přidaly spoustu nové funkcionality při zachování zpětné kompatibility. Standardy C99 a C11 však nejsou podporovány všemi kompilátory a je tedy nutné si dopředu ověřit, jaký standard překladače pro naši cílovou platformu podporují. Zvláštní odnoží je Embedded C, které v sobě zahrnuje funkcionalitu určenou speciálně pro embedded systémy. Trumfem jazyka je, že existuje C kompilátor pro takřka všechny představitelné platformy a jeho vlastnost, že v něm programátor dokáže v abstraktním jazyce (vetšinou nezávislém na hardware) vyjádřit svoje myšlenky a přitom má maximální kontrolu nad tím, co se v důsledku jeho kódu odehraje. Tento přístup však předpokládá, že programátor si je zcela vědom, co jeho kód způsobí (a nezpůsobí!) a má za následek, že zejména začínající programátoři se při psaní svých programů dopouštějí chyb. Chyby se však vyskytnou i u zkušenějších programátorů, následkem mohou potom být slabá místa náchylná na útoky z venčí (např. buffer overflow). Tyto slabiny se vyskytují především v kódu, který se zaměřuje na dosažení co nejvyšší rychlosti a nejsou tedy ohlídány všechny možné chybové situace.

4.3 C++

Historie

Jazyk C++ vychází z jazyka C, kde se operátor `++` používá k navýšení hodnoty proměnné. Ideou tedy byla nová, větší a lepší iterace jazyka C. Vývoj C++ započal v roce 1979 Bjarne Stroustrup a kromě jazyka C se inspiroval jazykem Simula, konkrétně jeho iterací Simula 67. Původní název jazyka byl C with Classes a měl tedy rozšířit C o možnosti OOP, kterými se Bjarne Stroustrup inspiroval u jazyka Simula. Dalším cílem bylo vyvinout silně typovaný jazyk, neboť slabá typová kontrola u jazyka C dovolující implicitní konverzi mezi nesouvisejícími datovými typy byla zdrojem mnoha chyb. V roce 1983 byl jazyk přejmenován na C++ a zároveň byla přidána nová klíčová funkcionalita jako virtuální funkce, přetěžování funkcí a reference. V dalších letech přišly věci jako templates neboli šablony umožňující generické programování, vícenásobné dědění a další. Velkým problémem pro jazyk byla dlouhodobá absence uznávaného mezinárodního standardu, který se objevil až

v roce 1998 a zároveň znamenal zahrnutí Standard Template Library do jádra jazyka. Dalším standard z roku 2003 znamenal především opravu chyb vyskytujících se ve standardu předchozím. Významným bodem ve vývoji jazyka bylo zveřejnění C++11, které přineslo mnoho nových vlastností, kterým se věnuji v následující sekci. C++11 bylo rozšířeno standardem z roku 2014 a nyní je již za dveřmi C++17. (Cplusplus.com) Velmi důležité je, že i přes všechny novinky, které přineslo posledních několik let, zůstává jazyk zpětně kompatibilní. Tento fakt je takřka nutností, neboť existují miliardy řádků kódu, který byl napsaný před C++11, či dokonce C++98 a Bjarne Stroustrup slibuje, že zpětná kompatibilita bude zachována i pro příštích několik desetiletí.

C++ si za dobu své existence vysloužilo spoustu kritiky, často i od významných osobností softwarového světa jako je Linus Torvalds, který se například nechal slyšet, že dokud bude naživu, nebude operační systém Linux napsán v C++. Důvodů k nevraživosti k tomuto jazyku bývá jeho komplexnost, velmi složitá a nejasná chybová hlášení a další, ale nejlépe je, myslím si, shrnul sám autor jazyka tímto citátem: „C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.“ Linus Torvalds není zdaleka jediným kritikem jazyka, těch se najde v oblasti programování spousta a nutno dodat, že velká část kritiky je oprávněna, avšak dosud se neobjevil programovací jazyk, který by dokázal nahradit C++ a s novinkami, které přináší C++11 a další iterace, se většina této kritiky stává zastaralou. Nakonec bych se ke kritice jazyka vyjádřil dalším citátem, jehož autorem je opět Bjarne Stroustrup. „There are only two kinds of languages: the ones people complain about and the ones nobody uses.“(Stroustrup, 2013)

Moderní C++

C++11, někdy označováno jako C++0x, přineslo celou řadu dlouho očekávaných novinek, z nichž spousta dříve vyžadovala použití externí knihovny (zejména Boost - http://www.boost.org/). Část funkcionality pak byla úplně nová, např. přesouvací konstruktor nebo mazání funkcí, které bylo dříve simulováno nedokonalým způsobem. Tato verze C++ změnila jazyk takovým způsobem, že o ní Bjarne Stroustrup prohlásil, že se jedná v podstatě o nový jazyk. Tyto změny se odráží na výrazné změně stylu, jakým by se mělo v C++ programovat. To má za následek, že přechod na tuto verzi je velmi pomalým procesem a ve velké části firem se stále používá C++03. Problém je pak na školách a univerzitách, kde spousta učitelů učí C++ tak, jak se jej učili oni a studentům jsou tak mnohdy vštěpovány zastaralé zásady, od kterých je nyní odrazováno. Příkladem za všechny jsou pointery, kdy se dnes výrazně odrazuje od tzv. raw pointerů (syntax `type* var;`) a namísto nich je doporučováno užití smart pointerů (syntax `std::unique_ptr<type> var;`), které si hlídají, zda je paměť, na kterou ukazují stále odkazována a v případě, že není, tak ji automaticky vymažou (toto platí pro proměnné, u argumentů funkce, který není vlastníkem paměti, je raw pointer v pořádku). S tímto doporučením je spojeno přísné vyhýbání se operátorům `new` a `delete`. Na závěr následuje výčet několika

nejpodstatnějších novinek: smart pointery, podpora multithreadingu, modul chrono pro přesné měření času, automatická dedukce datového typu (pro správné použití je třeba znát její pravidla), lambda výrazy, jednotná inicializace proměnných pomocí složených závorek (syntax `std::vector<std::string> v{"abc","mno","xyz"};`), explicitní mazání funkcí, rozsahový for cyklus, nullptr – opravdu typ pointer, již se nejedná o integer, dále pak přidání nových algoritmů do standardní knihovny a mnoho další nové funkcionality. (Meyers, 2014) (Stroustrup, 2013)

4.4 Síťová komunikace v C++

Boost Asio

Boost je nejspíše nejvyužívanější skupinou knihoven (obsahuje jich několik desítek), co se týče jazyka C++. Mnoho vývojářů ji vnímá jako nutné rozšíření standardní knihovny, která si klade za cíl být minimalistická a nemůže tedy ani zdaleka obsáhnout všechny potřeby C++ programátorů. Boost zahrnuje knihovny pro práci s časem, algoritmy, síťovou komunikací, multithreadingem, matematickými funkcemi a mnohé další. Významným faktem je i to, že velká část knihoven je tzv. header-only a není je tedy třeba kompilovat, stačí pouze připojit požadovaný hlavičkový soubor. S příchodem C++11 se poněkud omezila potřeba zahrnovat boost do projektů, neboť část nejčastěji využívané funkcionality nabízené v rámci knihoven Boost byla implementována jako součást standardní knihovny.

Knihovna Asio (ASynchronous Input Output) byla původně založena jako knihovna primárně zaměřená na networking, ale s postupem času se její funkcionalita rozrostla o další vstupně/výstupní operace. Část zaměřená na síťovou komunikaci je založena na Berkeley sockets API, stejně jako většina knihoven zaměřených na networking, a to i v jiných programovacích jazycích. Výhodou využití Asio je portabilita a určitá míra abstrakce, kdy Asio vykonává některé rutinní operace za nás. (http://www.boost.org)

Windows socket API

Windows socket API – většinou zkracováno na Winsock (případně WSA) je API specifikující síťovou komunikaci programů na platformě Windows. Historie Winsock započala v roce 1991 v důsledku omezených možností síťového programovaní v MS-DOS a raných verzích Windows, kdy Microsoft vůbec nepodporoval rodinu protokolů TCP/IP – na toto reagovalo hned několik výrobců svými vlastními řešeními, avšak tato řešení byla nesourodá a každá z firem používala své vlastní API. API Winsock je založeno na Berkeley sockets API a bylo vytvářeno s myšlenkou jednoduchého portování aplikací mezi Unix systémy a Windows, avšak toto se u složitějších aplikací stejně ukázalo být nelehkým úkolem. Programování za využití Winsock je relativně nízkoúrovňové, jelikož voláme přímo systémové funkce a je tedy třeba postupovat opatrně, neboť tento přístup nám nechává volnou ruku, čímž zase vzniká více prostoru pro programátorské chyby. (MSDN WSA)

4.5 Grafické uživatelské rozhraní

Qt

Qt je jedním z nejpoužívanějších frameworků pro vývoj multiplatformních programů v C++, lze jej však využít i v jiných programovacích jazycích. Fakt, že se jedná o framework mimo jiné znamená, že oproti knihovně je zde vyšší vstupní bariéra a je tedy třeba větší časové investice, abychom pronikli do způsobu vývoje aplikací v Qt a i následně daleko více ovlivní náš styl programování. Na druhou stranu Qt nabízí velmi širokou funkcionalitu a pokud se s frameworkem naučíme pracovat, tak nám především u větších projektů ušetří spoustu času. Nejčastěji se vývojáři k využití Qt uchylují při tvorbě grafických uživatelských rozhraní, avšak v rámci frameworku je k dispozici i práce s textovými řetězci, vlákny a další užitečná funkcionalita, která je navíc přenositelná mezi různými platformami. Qt má však i své mouchy – mnoha lidem vadí, že grafická uživatelská rozhraní napsaná v Qt mají vzhled odlišný od nativního pro daný operační systém, což může následně narušit uživatelský zážitek. Další nejčastěji zmiňované problémy jsou:

- Potřeba kompilace a nastavení prostředí
- Prakticky nutnost používání IDE Qt Creator
- Zesložiťuje sestavování programů
- Pouze LGPL licence
- Má za následek velmi velké binární soubory

První verze Qt byla vydána v roce 1995 firmou Trolltech, kterou později koupila Nokia. Od Nokie poté Qt koupila společnost Digia, která za frameworkem stojí dodnes. Qt se může chlubit rozsáhlou komunitou uživatelů, která se od doby prvního vydání značně rozrostla a nyní se skládá z 1 milionu uživatelů po celém světě. Pro usnadnění práce s frameworkem bylo vyvinuto i IDE zvané Qt Creator, které zahrnuje Qt Designer – nástroj pro návrh grafických uživatelských rozhraní. Kromě uvedených nástrojů Qt poskytuje skriptovací jazyk QML na bázi JavaScriptu k urychlení vývoje aplikací. (https://www.qt.io/)

Uživatelské rozhraní ve WinAPI

Windows API slouží k interakci uživatelských programů s operačním systémem Windows (vlákna, vstupně/výstupní operace atd.). WinAPI je napsáno v jazyce C a je určeno primárně pro použití při programování v jazycích C a C++, jeho pochopení však může být přínosem i pro programátory pracující s vysokoúrovňovějšími jazyky jako je C#. Tvorba GUI je s využitím WinAPI ve srovnání s Qt sice náročnější, ale u programů menšího rozsahu se nejedná o příliš značný rozdíl. Přirozenou nevýhodou WinAPI je jeho vázanost na operační systémy Windows, čímž se vylučuje přenositelnost programů, avšak pokud neřešíme portabilitu a stačí nám, když pro-

gram poběží na platformě Windows, máme oproti Qt výhodu nativního vzhledu, což je jedna z věcí, jejíž absence mnoho lidí odrazuje od využití Qt pro tvorbu grafických uživatelských rozhraní. Další nespornou výhodou WinAPI je, že pro tvorbu programů nepotřebujeme instalovat žádné frameworky a knihovny, neboť všechny jsou již součásti systému Windows. (MSDN WinAPI)

5 Vlastní práce

5.1 Laboratoř řízení kolejových vozidel

Laboratoř je situovaná v areálu Mendelovy univerzity pod budovou X v prostorách bývalého skladu papíru. Laboratoř vznikla v roce 2013 na základě spolupráce Mendelovy univerzity a Klubu modelářů železnic Brno 1 a vyskytují se v ní modely v měřítku H0 a TT. (Rybička, 2014) Velkou výhodou práce v laboratoři je přítomnost konstantního osvětlení, které je v běžných podmínkách nedosažitelné. Část laboratoře je možno vidět na obrázku 5.

Obrázek 5: Laboratoř řízení kolejových vozidel

5.2 Získávání obrazu kolejiště

Model kolejového vozidla

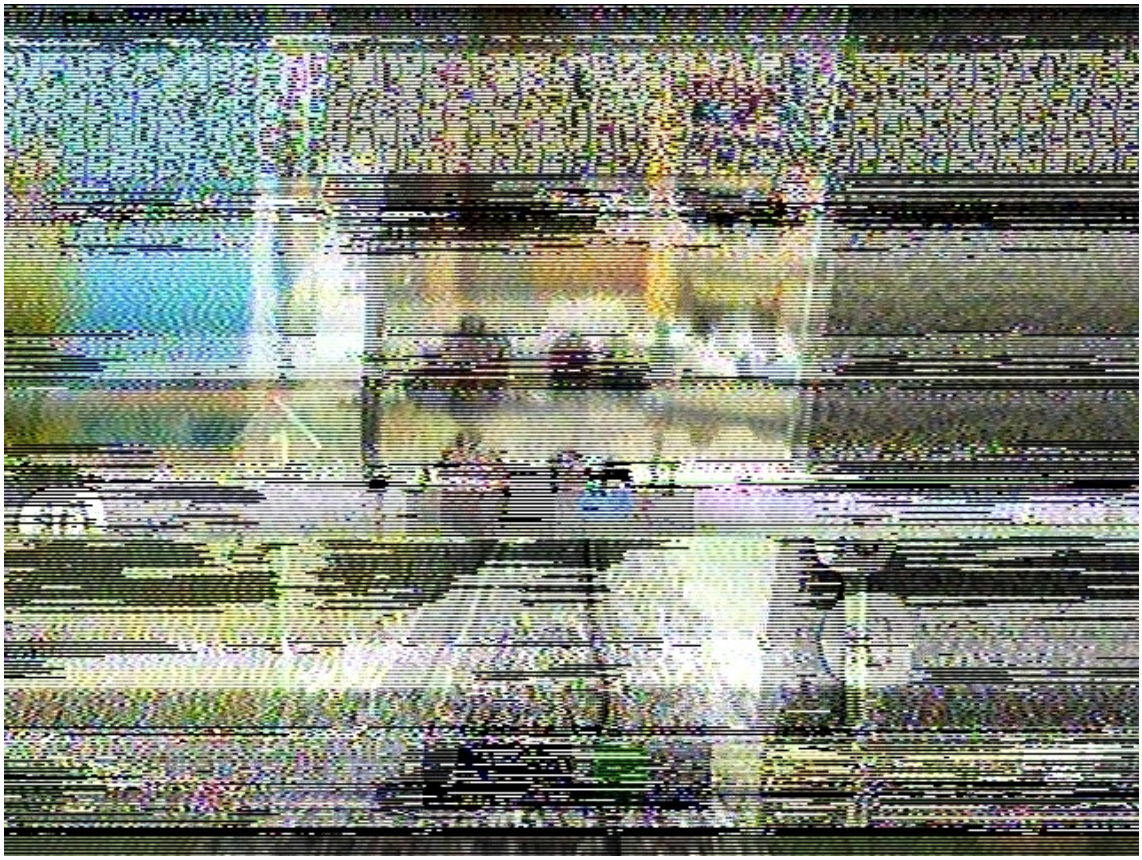
Obraz kolejiště je získáván pomocí kamery integrované v modelu lokomotivy odkud je ve formátu $640 \times 480px$ odesílán v analogové podobě. Signál je následně přijat a pomocí A/D převodníku digitalizován. Poté je signál za použití programu DVDriver převeden na proud videa, který již zpracovává náš program. (Vytečka, 2014) Podoba modelu lokomotivy je zachycena na obrázku 6.

Obrázek 6: Model lokomotivy s kamerou uvnitř

Problémy při získávání obrazu

Při získávání obrazu z modelu lokomotivy je nutno počítat s potížemi, které toto s sebou nese. První z nich je samotná kvalita obrazu, která je dána použitou technikou, jež je zase omezena rozměry modelu lokomotivy, které neumožňují použití větší kamery. Druhým problémem je šum, který se do obrazu dostává při rádiovém přenosu. Tento je ještě umocněn, pokud se v dráze přenosu vyskytují překážky či pohybují lidé. Šum je problémem především při větších rychlostech modelu a v zatáč-

kách, kdy máme v kombinaci s rychlostí detekce značek v obraze k dispozici pouze jeden až dva snímky, na kterých je možno značku dobře vidět. V důsledku šumu potom mohou být i tyto snímky nepoužitelné a značku se tedy nepodaří detekovat, což je patrné z obrázku 7.



Obrázek 7: Snímek znehodnocený šumem

5.3 Program k demonstraci výsledků

Požadavky na program

Požadavky na program, který by byl schopný demonstrace dosažených výsledků byly, aby nabízel nějaké jednoduché grafické uživatelské rozhraní a byl schopný síťové komunikace se serverem v laboratoři.

Programovací jazyk

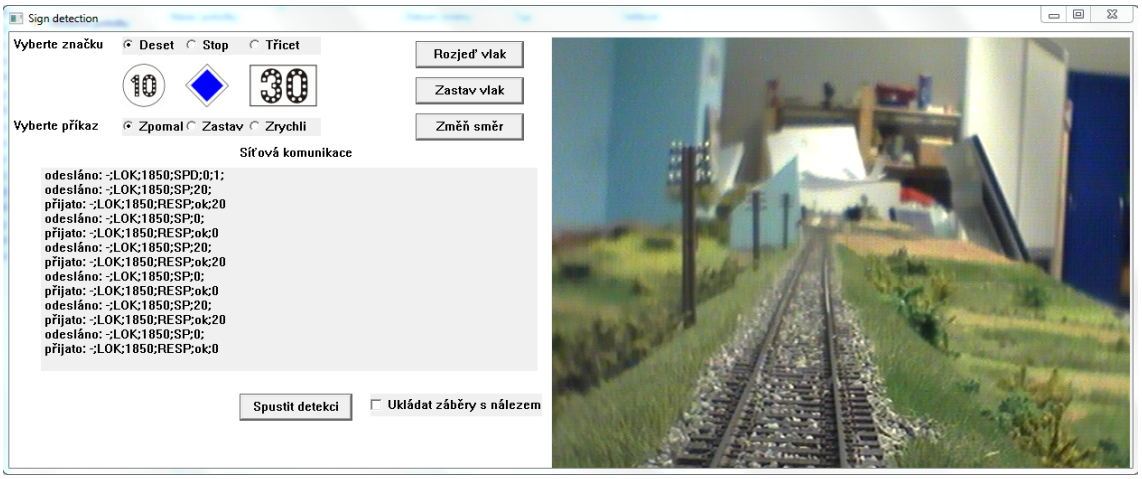
Z programů popsaných v teoretické části práce jsem se rozhodl pro použití C++, což je zároveň implementační jazyk knihovny OpenCV a z prostředí tohoto jazyka se

s knihovnou nejlépe pracuje. Zároveň je nutné si přiznat, že ve spojení s laboratorním PC nelze uvažovat o přebytku výkonu, a tak je C++ vhodnější než Java. Vývoj v C++ je většinou příjemnější nežli v C (zvláště pokud na programu pracujeme sami a nemusíme se bát skrytých bugů, které se v C++ hůře odhalují) a eventuální zrychlení a snížení paměťové náročnosti by bylo zanedbatelné. Navíc je většina časově kritických funkcí volána z knihovny OpenCV a tyto funkce jsou většinou maximálně optimalizované.

Grafické uživatelské rozhraní

Pro implementaci grafického uživatelského rozhraní, jsem se rozhodl využít WinAPI. WinAPI jsem zvolil, protože jsem do programu nechtěl zanášet další závislosti např. v podobě knihovny Qt. Jedná se o poměrně jednoduché GUI (Graphical User Interface) a programování za využití WinAPI s sebou tedy neneslo žádné výraznější potíže. Musím však přiznat, že pokud by se jednalo o program vyžadující složitější grafické uživatelské rozhraní, pravděpodobně bych zvolil jinou alternativu, jelikož u WinAPI je nutné pamatovat opravdu na všechny úkony a postarat se o ně sám, což s sebou u rozsáhlejších projektů nese značné zpomalení vývoje, a i když programy poté běží rychleji, většinou se tato časová investice nevyplatí.

V grafickém rozhraní lze vybrat, kterou ze značek má program detekovat a jak má vlak na její detekci zareagovat. Dále je možné manuálně zastavit či rozjet vlak. V GUI je též k vidění komunikace mezi klientem a serverem a levou část okna zabírá obraz přenášený z lokomotivy, jak je vidět na obrázku 8.



Obrázek 8: Grafické uživatelské rozhraní programu

Síťová komunikace

Při volbě řešení jsem uvažoval podobně minimalisticky jako při tvorbě GUI a rozhodl jsem se pro využití Windows Socket API, zkráceně Winsock. Opět se jedná

o čistě Windows řešení bez potřeby užití externích knihoven, i když by u síťové komunikace nemuselo jít o tak významnou závislost. I zde platí, že programátor si musí dávat pozor, zda udělal vše co měl, jinak by mohlo dojít k chybám. Například je nutné před přijímáním dat vynulovat buffer. Pokud bychom tak neučinili, mohly by v bufferu být různé hodnoty, jež na tomto místě v paměti byly dříve a které by měly za následek chybu v přijatých datech. Při programování za využití ať už WinAPI nebo Winsock je nedocenitelnou pomůckou Microsoft Developers Network (https://msdn.microsoft.com/), kde se nachází nejrůznější návody a doporučené postupy.

5.4 Klasifikátor

Trénování klasifikátoru

Pro trénování klasifikátoru bylo využito snímků značek vyříznutých z fotografií pořízených kamerou umístěnou v modelu lokomotivy, neboť takto je možno klasifikátor trénovat přímo na podobu značky, v jaké se vyskytuje v obraze. Tento přístup již ze své podstaty přináší lepší výsledky než použití umělých snímků značky stažených z internetu. Jako negativní příklady posloužily také fotografie trati, které neobsahovaly danou značku. Pro každou značku bylo použito zhruba sto pozitivních a o několik více než dvojnásobek negativních příkladů. Zpočátku byl vyzkoušen i přístup využívající utility pro tvorbu pozitivních příkladů, která je poskytovaná v rámci OpenCV, ale tato metoda nepřinesla uspokojující výsledky. Z parametrů zadávaných při učení klasifikátoru stojí za zmínku `minHitRate = 0,999` a `maxFalseAlarmRate = 0,3`. Parametr `numStages` specifikující počet kroků, ve kterých je klasifikátor aplikován byl pro značky `30` a `10` roven osmi a pro značku `Stop` pěti. Jako `featureType` bylo nejdříve zvoleno LBP z důvodu rychlejšího učení, avšak u všech značek byly výsledky velmi neuspokojivé. Při použití Haar příznaků byla doba učení nesrovnatelně delší, ale výsledky byly o mnoho lepší a rychlost detekce srovnatelná.

Předzpracování obrazu před detekcí

Rozlišení obrazu přenášeného z kamery je $640 \times 480px$. Toto je pro detekci objektů v obraze příliš velké, neboť když klasifikátor musí procházet takto velký obraz, dochází k výraznému zpomalení detekce. Jsou tedy na výběr dvě možnosti: zmenšit rozlišení nebo vyříznout část obrazu, ve které se dá očekávat výskyt značek – tato oblast má rozměry $310 \times 229px$, plocha snímku je tedy více než čtyřikrát menší. Druhá z možností s sebou kromě zvýšení rychlosti detekce nese další výhodu – omezení výskytu chyby prvního typu (značka byla nalezena tam, kde neměla). Porovnání celého snímku a vyříznuté části je na obrázku 9.

Obrázek 9: Vlevo – celý snímek, vpravo – vyříznutá část

Dalším z úkonů, který je v rámci předzpracování obrazu doporučován je jeho převedení na obraz černobílý. OpenCV funkce pro detekci objektů v obraze si tuto skutečnost však kontroluje a případně barvy obrazu převede sama. Tento úkon má tedy vliv pouze v případě, že je obraz používán pro několik různých klasifikátorů a obraz se nemusí převádět na černobílý při volání detekční funkce u každého klasifikátoru. Nakonec se ve velké části literatury a návodů ke klasifikátoru doporučuje vyrovnat histogram obrazu, avšak v případě této práce bylo dosaženo lepších výsledků (a vyšší výpočetní rychlosti), pokud byla tato operace vynechána.

5.5 Dosažené výsledky

Úspěšnost jednotlivých klasifikátorů byla testována na množině 2314 snímků kolejiště pořízených pomocí kamery v modelu lokomotivy. Jednotlivé značky jsou vyobrazeny na obrázcích 10, 11 a 12.

Značka `10`

Při detekci značky `10` byly vybírány parametry pro detekční funkci tak, aby byla minimalizována chyba prvního typu i za cenu zvýšení procenta snímků, ve kterých značka detekována být měla a nebyla (chyba druhého druhu), neboť chyba druhého druhu byla stále dostatečně nízká, aby klasifikátor značku zachytil při každém průjezdu kolem ní. Toto má za následek, že pokud byla značka detekována, je vysoká pravděpodobnost, že se o značku opravdu jedná. Parametry použité pro detekci značky `10` byly `scaleFactor = 1,03` a `minNeighbors = 3`.

Obrázek 10: Značka `10`

Značka `30`

U této značky bylo dosaženo nejhorších výsledků ze tří značek vybraných pro tuto práci. Při pohledu na následující obrázek je vidět, že značka dosti splývá s pozadím. Tato značka byla také jediná, kterou nebylo možno přemístit, ale při umístění tmavého pozadí za značku se úspěšnost detekce výrazně zlepšila. Pokud však bylo ponecháno původní pozadí za značkou nebo bylo použito pozadí světlé, stalo se nalezení značky v obraze o mnoho složitější a klasifikátor tedy nemohl být příliš přísný, což mělo přirozeně za následek vyšší výskyt chyby prvního typu, například si klasifikátor v několika případech spletl značku `30` se značkou `10`. Parametry použité pro detekci značky `30` byly `scaleFactor = 1,02` a `minNeighbors = 2`.

Obrázek 11: Značka `30`

Značka `Stop`

Značka `Stop` měla ze všech značek výsledky zdaleka nejlepší. Úspěšnost detekce je z velké části dána výrazným tvarem značky a zároveň použitou kombinací barev, která má význam i po převodu barvy na černobílou, neboť se jedná o tmavý střed se světlým okrajem. Parametry použité pro detekci značky `30` byly `scaleFactor = 1,06` a `minNeighbors = 8`.

Obrázek 12: Značka `Stop`

Dosažená úspěšnost

Úspěšnost detekce je hodnocena pomocí 2 vlastností: precision (absence chyby prvního typu) a recall (absence chyby druhého druhu). Jednotlivé vlastnosti se vypočítají

podle vzorců 6 a 7, kde
`Tp` = Počet případů, kdy značka byla nalezena a opravdu nalezena být měla
`Fp` = Počet případů, kdy značka byla nalezena a nalezena být neměla
`Fn` = Počet případů, kdy značka nalezena nebyla a nalezena být měla

$$Precision = \frac{Tp}{Tp + Fp} \tag{6}$$

$$Precision = \frac{Tp}{Tp + Fn} \tag{7}$$

V tabulce 1 je prezentována precision a recall jednotlivých značek. Z celkové množiny snímků necelá stovka obsahovala značku `10`, značka `Stop` byla také zhruba na sto snímcích a značka `30` byla obsažena v přibližně šedesáti fotografiích.

Značka `10`	
Precision	0,964
Recall	0,769
Značka `30`	
Precision	0,745
Recall	0,673
Značka `Stop`	
Precision	0,991
Recall	0,955

Tabulka 1: Úspěšnost detekce značek

6 Diskuze

Nejen výsledky dosažené pro značku `Stop`, ale i pro značku `10` jsou velmi dobré, jenže je třeba mít na paměti, že jich bylo dosaženo v omezené testovací sadě dat. Je velmi pravděpodobné, že v situaci, kdy bude model lokomotivy jezdit po trati, budou výsledky o poznání horší, a to především zvýšený výskyt chyby prvního typu. Pokud by se mělo uvažovat ještě dále, například k aplikaci programu na reálné trati, je nutné brát v potaz, že výsledky byly dosaženy v laboratorních podmínkách, což se týká především osvětlení značek. V reálné situaci může nastat mnoho problémů: bude svítit slunce do kamery, značka nebude dostatečně osvětlená, atd.

Z hlediska budoucí práce je možné přidat detekci dalších značek a u některých typů značek jako je značka `30` zdokonalit úspěšnost detekce přidáním pozadí za značku, aby se tato stala výraznější. Dále by úspěšnosti detekce výrazně pomohlo použití dokonalejší techniky. Pokud by se lepší kamera s vyšším rozlišením nevešla do modelu lokomotivy, mohla by být například umístěna na samostatný model vagonu, který by model lokomotivy tlačil před sebou. Další překážkou, jejíž odstranění by zvýšilo spolehlivost programu, je rušení signálu, který přenáší obraz z kamery. Jelikož použitý klasifikátor dokáže detekovat jednu značku, je třeba snímek znovu procházet pro každý typ značky, což je výpočetně velmi náročné na CPU. Počítač v laboratoři je značně zastaralý a momentálně je možné rozpoznávání jen jedné až dvou značek zároveň, pokud má být zachována přijatelná rychlost detekce. Zejména pokud má být zvýšen počet rozpoznatelných značek, bylo by třeba pořídit výkonnější CPU. Užitečným a z hlediska výzkumu významným rozšířením programu by bylo přidání detekce překážek na trati, například za využití canny edge detektoru pro detekování kolejí a následné přerušení linií.

Má-li být podobný program ve vzdálenější budoucnosti nasazen v reálném provozu, nebylo by vhodné postavit systém pouze na kameře, neboť v mnoha situacích na ni nelze spoléhat. Vhodná by mohla být kombinace s vysílači umístěnými na značkách, podle kterých by se systém také řídil. Dále by nebylo od věci mít v počítači umístěném v lokomotivě mapu železničních tratí, která by obsahovala pravidla provozu pro jednotlivé úseky a pomocí GPS lokalizovat lokomotivu na mapě. Ani jedna z technologií není tzv. stoprocentní, ale jejich kombinace by mohla vytvořit systém, který by byl dostatečně robustní, aby jej mohlo být využito jako asistence řízení drážních vozidel v reálném provozu.

7 Závěr

Cílem práce bylo vytvoření systému schopného co nejspolehlivější detekce a klasifikace železničního značení v obraze a následná reakce na něj. K tomuto úkolu bylo třeba zvolit vhodné nástroje pro detekci objektů v obraze, síťové komunikaci, ale také vhodný implementační jazyk a framework či knihovnu pro tvorbu grafického uživatelského rozhraní demonstračního programu.

Při bližším pohledu na volně dostupné knihovny pro detekci objektů v obraze bylo zjištěno, že momentálně knihovna OpenCV nemá nejen v tomto oboru, ale i ve zpracování obrazu celkově konkurenci, pokud nebereme v potaz okrajové případy a značná část dalších populárních knihoven pro práci s obrazem, především v jazycích jako je C#, Java nebo Python, je postavena na OpenCV. Ostatní knihovny na tomto poli předčí nejen rozsahem funkcionality, ale i kvalitou implementace algoritmů, která se odráží na jejich rychlosti a nelze opomenout ani velkou komunitu okolo OpenCV, která může být nápomocna každému, kdo při práci s knihovnou narazí na problémy. Pro implementaci byl zvolen jazyk C++, a to z několika důvodů: rychlost, jednoduché propojení programu s OpenCV, ale v neposlední řadě i osobní preference. Jako prostředek pro síťovou komunikaci a tvorbu uživatelského rozhraní byly zvoleny standardní nástroje dostupné v rámci operačního systému Windows, které nevyžadovaly zařazení dalších frameworků či knihoven do projektu. Pokud by však byly vzneseny požadavky na přenositelnost programu na platformu Linux, bylo by třeba zvolit řešení jiná.

Klasifikátor byl vytvořen pro tři různé železniční značky a testovací výsledky pro dvě z nich dopadly velmi příznivě – robustnost klasifikátorů je dostatečná pro potenciální nasazení na modelové trati v laboratoři řízení kolejových vozidel Mendelovy univerzity. Pro značku `30` by bylo třeba upravit pozadí za značkou pro její zvýraznění a dosažení lepších výsledků při její detekci. Hlavním omezujícím faktorem byla kvalita pořizovaného obrazu a ještě následné zhoršení této kvality při jeho přenosu z modelu lokomotivy.

Je spousta možností, jak systém dále rozvíjet a některé z nich jsou uvedeny v kapitole Diskuze. Při dostatečné funkcionalitě a robustnosti systému je možné, a to i když se ukáže neproveditelné adaptovat jej na reálné trati, využít jej pro zvýšení zájmu nejen o laboratoř řízení kolejových vozidel, ale i o samotnou Mendelovu univerzitu především mezi příznivci automatizace a podobných oborů. Dále se při jeho rozvoji mohou realizovat již stávající studenti univerzity, a to nejen co se software týče, protože i vylepšení přenosových technologií a elektroniky používané v rámci modelu trati by výrazně pomohlo ke zdokonalení systému.

8 Reference

Ahonen, T., Hadid, A., Pietikäinen, M. *Face Description with Local Binary Patterns: Application to Face Recognition*. Pattern Analysis and Machine Intelligence, IEEE Transactions on 28(12), IEEE Computer Society, 2006, 2037-2041. ISSN 0162-8828.

Boden, M. *Mind as a machine: A History of Cognitive Science*. Oxford, Spojené království: Clarendon Press, 2006. 1 708 s. ISBN 0-19-954316-X.

Evans, B.J., Flanagan, D. *Java in a Nutshell, 6th Edition*. O'Reilly Media, 2014, 418 s. 1-4493-7081-0.

Freund, Y., Schapire, R.E. *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. Journal of Computer and System Sciences, Orlando, USA: Academic Press, Inc., 1997, 119-139. ISBN 0-19-954316-X.

Garcia, G.B., Suarez, O.D., Aranda, J.L.E., Tercero, J.S., Gracia, I.S., Enano, N.V. *Learning Image Processing with OpenCV*. Packt Publishing, 2015. 223 s. ISBN 1-78328-766-7.

He, DC., Wang, L. *Texture Unit, Texture Spectrum, And Texture Analysis*. Geoscience and Remote Sensing, IEEE Computer Society, 1990, 509-512. ISSN 0196-2892.

Kearns, M., Valiant, L.G. *Cryptographic limitations on learning Boolean formulae and finite automata*. Proceedings of the twenty-first annual ACM symposium on Theory of computing, New York, USA: ACM, 1989, 433-444. ISSN 0196-2892.

Khronos *OpenCL – The open standard for parallel programming of heterogeneous systems*. Khronos. [online]. 2016 [cit. 2016-03-05]. Dostupné z: https://www.khronos.org/opencl/.

Lienhart, R., Maydt, J. *An Extended Set of Haar-like Features for Rapid Object Detection*. IEEE ICIP 2002, 2002, 900-903. ISBN 0-7803-7622-6.

Mäenpää, T. *The Local Binary Pattern Approach to Texture Analysis – Extensions and Applications*. Oulu, Finsko: University of Oulu, 2003. 80 s. ISBN 951-42-7076-2.

Meyers, S. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, 2014. 336 s. ISBN 1-4919-0399-6.

Microsoft *Windows API Index*. Microsoft. [online]. 2016 [cit. 2016-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516.

Microsoft *Windows Sockets 2*. Microsoft. [online]. 2016 [cit. 2016-04-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ms740673.

NVIDIA *Parallel Programming and Computing Platform | CUDA*. NVIDIA. [online]. 2016 [cit. 2016-03-05]. Dostupné z: http://www.nvidia.com/object/cuda_home_new.html.

Ojala, T., Pietikäinen, M. and Harwood, D. *A Comparative Study of Texture Measures with Classification Based on Feature Distributions*. Pattern Recognition (The Journal of the Pattern Recognition Society) 19(3), Pattern Recognition Society, Elsevier, 1996, 51-59. ISSN 0031-3203.

Ojala, T., Pietikäinen, M. and Mäenpää, T. *Multiresolution Gray-scale and Rotation Invariant Texture Classification with Local Binary Patterns*. Pattern Analysis and Machine Intelligence, IEEE Transactions on 24(7), IEEE Computer Society, 2002, 971-987. ISSN 0162-8828.

OpenCV *CUDA | OpenCV*. OpenCV. [online]. 2016 [cit. 2016-03-05]. Dostupné z: http://opencv.org/platforms.html.

Papageorgiou, C.P., Oren, M., Poggio, T. *A General Framework for Object Detection*. Computer Vision, 1998. Sixth International Conference on, Bombay, Indie: IEEE Computer Society, 1998, 555-562. ISBN 81-7319-221-9.

Prinz, P., Kirch-Prinz, U. *C Pocket Reference*. O'Reilly Media, 2009. 144 s. ISBN 0-596-10396-4.

Rybička, J. *Laboratoř řízení kolejových vozidel*. [online]. 2014 [cit. 2016-03-18]. Dostupné z: https://akela.mendelu.cz/ rybicka/prez/lrkv/prezLuhac14.pdf.

Stroustrup, B. *The C++ Programming Language (4th Edition)*. Addison-Wesley, 2013. 1363 s. ISBN 0-13-352285-7.

Shapiro, L., Stockman, G. *Computer Vision*. New Jersey, USA: Prentice Hall, 2001. 608 s. ISBN 0-13-030796-3.

TheRabbitologist *History of C++* . cplusplus.com. [online]. 2012 [cit. 2016-03-06]. Dostupné z: http://www.cplusplus.com/info/history/ .

Vijayalakshmi, B., Subbiah Bharathi, V. *A Novel Approach to Texture Classification using Statistical Feature*. Surapet, Indie: Velammal College of Management and Computer Studies, 2011 .

Viola, P., Jones, M. *Rapid Object Detection Using a Boosted Cascade of Simple Features*. Computer Society Conference on Computer Vision and Pattern Recognition, Hawai, USA: IEEE Computer Society, 2001, 511-518. ISBN 0-7695-1272-0.

Vytečka, M. *The System For Automatic Train Control Simulation.* AD ALTA: Journal of Interdisciplinary Research, Hradec Králové, Česká republika: MAGNANIMITAS Assn., 2014, 67-69. ISSN 1804-7890.