VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘÍZENÍ POHYBU ROBOTA POMOCÍ RASPBERRY PI A KAMERY

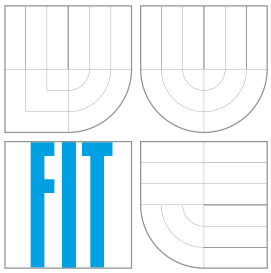
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV BRHEL

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘÍZENÍ POHYBU ROBOTA POMOCÍ RASPBERRY PI A KAMERY

MOTION CONTROLLING OF ROBOTIC CAR BY RASPBERRY PI AND CAMERA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV BRHEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá řízením robotického auta pomocí Raspberry Pi a kamery. Teoretická část práce popisuje jednotlivé kroky zpracování obrazu a také pravděpodobnostní plánování pro hledání cesty v prostředí. Zejména je rozebrán algoritmus RRT (rychle rostoucí náhodný strom), jehož varianta s dvěma stromy je později realizována a použita pro plánování trasy neholonomického robota v prostoru. Další část textu se věnuje návrhu řešení, ve kterém je detailně popsáno připojení Raspberry Pi k robotickému autu. Společně s kapitolou o implementaci se v práci nachází také vyhodnocení a testování. V závěru jsou popsány možnosti vylepšení a rozšíření realizovaného sytému.

Abstract

This Master's Thesis deals with the controlling of robotic car by Raspberry Pi and the camera. Theoretical part describes individual steps of image processing and probabilistic planning for searching path in the work space. In particular, algorithm RRT (Rapidly-exploring Random Tree) is discussed and the balanced bidirectional RRT is further introduced and used for nonholonomic planning in configuration space. Next chapter speaks about proposed solution and there is the accurate description of connection Raspberry Pi to the robotic car. Rest of the work provides look at implemetation details and evaluation. In the end, conclusion was given and some improvements were suggested.

Klíčová slova

zpracování obrazu, počítačové vidění, robotické auto, neholonomický robot, pravděpodobnostní plánování, RRT, Raspberry Pi, kamera

Keywords

image processing, computer vision, robotic car, nonholonomic robot, probabilistic planning, RRT, Raspberry Pi, camera

Citace

Miroslav Brhel: Řízení pohybu robota pomocí Raspberry Pi a kamery, diplomová práce, Brno, FIT VUT v Brně, 2015

Řízení pohybu robota pomocí Raspberry Pi a kamery

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. V práci jsem uvedl veškeré literární prameny a publikace, ze kterých jsem při její tvorbě čerpal.

.....
Miroslav Brhel
27. května 2015

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Jaroslavu Rozmanovi, Ph.D za poskytnuté rady a cenné připomínky. Také děkuji za jeho trpělivost a pochopení, které projevil při dokončování této diplomové práce.

Obsah

Kapitola 1

Úvod

Tento dokument popisuje diplomovou práci vytvořenou na Fakultě informačních technologií v Brně. Po krátkém úvodu a seznámení se zadáním práce následuje kapitola číslo 2, ve které je popsána oblast počítačového vidění, která je úzce spojena s robotikou nebo umělou inteligencí. Kapitola č. 3 teoreticky objasňuje problém pravděpodobnostního plánování při hledání cesty v prostoru. Návrh řešení se nachází v kapitole č. 4, na kterou navazuje popis samotné realizace a implementace projektu. Kapitola č. 6 se věnuje vyhodnocení celého systému. Poslední kapitola uzavírá celý text zhodnocením v podobě závěru.

Diplomová práce obsahuje také několik příloh. Nejzajímavější z nich sdružuje případové studie hledání cesty robota v prostředí. Jednotlivé experimenty jsou dokumentovány doprovodnými obrázky a tabulkami s výsledky.

1.1 Úvod do problematiky

Jak z názvu vypovídá, práce se bude zabývat řízením pohybu robotického autíčka. V dnešní době se oblast robotiky stále posunuje dopředu, nové a lepší technologie se objevují jak ve výzkumu, tak v komerčních produktech. Často se robotika pojí s umělou inteligencí a nejinak tomu bude i v této práci. Výsledkem celoročního snažení bude systém skládající se z několika částí. Ovládacím prvkem bude osobní počítač (notebook), na kterém poběží řídicí aplikace s grafickým uživatelským rozhraním.

Protože se jedná o řízení modelu autíčka pomocí kamery snímající scénu z ptačí perspektivy, bude ovládací program zobrazovat záběr ze vstupní kamery a bude také nabízet uživateli možnost zadat cíl, na který se robotické auto přesune. Práce uvažuje jako řízené autíčko *RC model HPI Racing E10 Toyota TruenoDrift*. Ovládací jednotkou vozítka představuje populární *Raspberry Pi*, což je levný počítač o velikosti kreditní karty, na kterém může běžet operační systém (například Linux). Toto zařízení umožní pohodlně ovládat autíčko a také komunikovat s řídicím počítačem.

Cílem práce bude sestavit výsledný systém z komponent představené v předcházejícím odstavci. Z nastíněného zadání lze vyčíst, že celý systém bude stát na dvou stěžejních částech: na zpracovaní obrazu ze vstupní videosekvence a na plánování a přesnému provedení nalezené trasy. Zadání přímo nespecifikuje jaké nástroje nebo programovací jazyky mají být použity pro vytvoření celého systému. Pro své výhody byly vybrány jazyky *C++* a *C*, jež budou tvořit základ systému, volba ostatních nástrojů je podrobněji popsána v kapitole o návrhu řešení.

Kapitola 2

Počítačové vidění

Lidé vnímají s určitou přirozeností trojrozměrný svět kolem sebe, a to zejména díky skvěle uzpůsobenému zraku a mozku. Obyčejný pohled na předměty v našem okolí nám poskytne okamžitou informaci o tvaru, velikosti, barvě nebo povrchu těchto objektů. Zdravé zrakové ústrojí společně s mozkem také lehce vnímá světla a stíny přítomné u pozorovaných objektů. Ze všech těchto vlastností si naše vnímání lehce vytvoří představu o probíhajících jevech v našem okolí.

V době velkého rozmachu počítačových technologií tak logicky vyvstává otázka, zda se na podobnou úroveň vnímání prostředí může také dostat i neživý stroj, jako je například počítač. Podobnými otázky a problémy se zabývá oblast informatiky nazývaná *počítačové vidění*. Tento poměrně mladý obor zahrnuje metody získávání, zpracování, analyzování a pochopení dat z reálného světa.

Možná není uplně na první pohled jasné, proč je potřeba se v této práci zabývat počítačovým viděním nebo zpracováním obrazu. Pravdou však je, že tato podoblast je velmi úzce spjata s robotikou, popřípadě umělou inteligencí. Aby mohl robot pracovat autonomně nebo se inteligentně rozhodovat, musí detailně poznat okolní prostředí, ve kterém se nachází. Robot může být vybaven různými typy senzorů (zvukovými, optickými, tlakovými a jinými), které mu umožňují pochopit okolí.

Podobně jako člověk získává nejvíce informací okem (až 80 %), tak i roboti nejlépe zmapují okolí pomocí kamer, jenž jim přímo nahradí zrakový orgán. Pouze snímky z kamer však samy o sobě neposkytnou žádnou informační hodnotu. Ta se musí ze získaných dat vyextrahovat, a zde se právě uplatní počítačové vidění zabývající se právě těmito problémy. Jak dále naložit ze získanými informacemi, zda se popřípadě naučit nějaké opakující se vzory, to už zase zkoumá obor umělé inteligence.

2.1 Zpracování obrazu

Jak už bylo řečeno obrazová informace obsahuje velkém množství užitečných dat, ne však všechna data jsou vždy prospěšná, dokonce se ve vstupních obrazech často objevují i nechtěná data, nazývaná šum. Ten vzniká při jakémkoliv měření v reálném světě, tedy i při zachycování reality pomocí kamery. Právě odstraněním nebo vyfiltrováním pouze užitečných dat ze vstupních obrázků se zabývá disciplína zvaná zpracování obrazu. V této práci se budeme věnovat pouze digitálnímu zpracování obrazu, protože vstupním prvkem budou digitální data. Optické nebo analogické zpracování není tedy uvažováno.

Většina technik z oblasti zpracování obrazu zachází s obrazem jako s dvoudimenzionál-

ním signálem a aplikuje tak na něho standardní operace, jenž se využívají při zpracování obecných signálů.

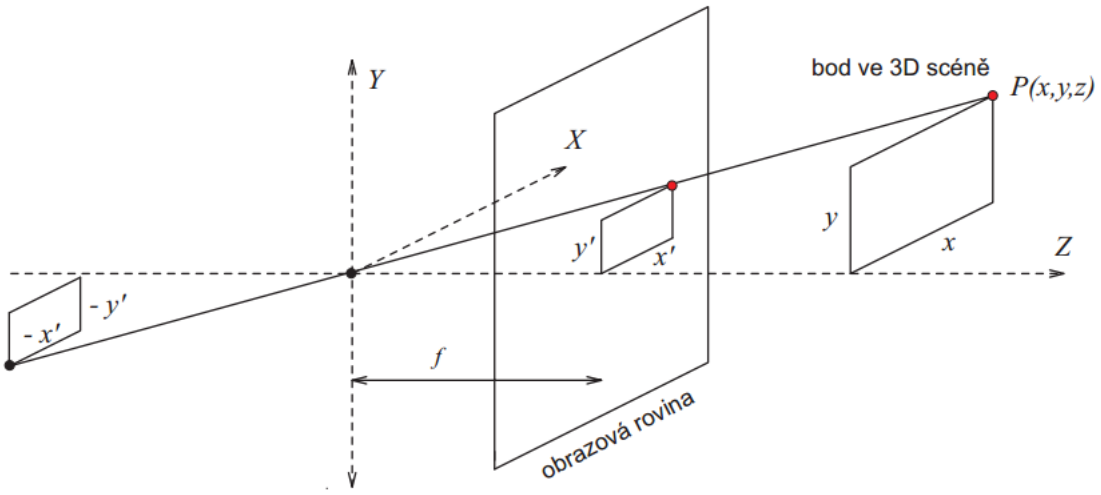
Exituje několik typů dělení postupu zpracování obrazu, podle literatury [11] je možné rozdělit proces do pěti základních kroků. Tyto kroky jsou podrobněji popsány v následujících kapitolách. První krok představuje *snímání a digitalizace* obrazu, na tento proces navazuje *předzpracování dat*. Třetí krok se označuje jako *segmantace obrazu*, předposlední se zabývá *popisem objektů* a závěrečný krok se jmenuje *klasifikace*.

2.2 Snímání a digitalizace obrazu

Předcházející podkapitola prozradila, že počátečním krokem procesu zpracování obrazu je snímání reálného světa. Podstata této operace spočívá v převodu optické veličiny na elektrický signál, jenž je spojitý v čase a úrovni. Podle typu použité kamery může být vstupní veličinou jas, rentgenové záření, ultrazvuk nebo třeba tepelné záření. Ve zbytku práce se bude uvažovat pouze nejčastější typ kamer, tedy těch které reagují na světelný jas.

2.2.1 Snímání obrazu

Obraz lze popsat matematicky dvourozměrnou spojitou funkcí $f(x, y)$, která se nazývá *obrazová funkce* a její parametry x a y představují souřadnice v rovině. Funkční hodnotou je již zmíněný jas. U jiných typ kamer může hodnota představovat odlišnou veličinu (např. teplo u termovize). Přestože reálný svět má tři dimenze, běžné kamery snímají obraz pouze jako dvourozměrný. Obrazová funkce $f(x, y)$ je tedy výsledkem *perspektivního zobrazení*, jenž je naznačeno na obrázku 2.1.



Obrázek 2.1: Perspektivní zobrazení pří snímání bodu v prostoru, převzato z [4].

Při tomto procesu dochází ke ztrátě velkého množství informací, pro potřeby v informatice, robotice nebo třeba zábavním průmyslu je tento typ kamer dostatečný a jsou tak masově rošířeny (díky nízkým nákladům) napříč všemi odvětvími. V dnešní době s rozvojem 3D televizorů jde dopředu i vývoj 3D kamer, nejsou však ještě tak rozšířeny, aby se

běžně objevovaly v oblasti robotiky. Princip těchto technologií není dále v práci rozvinut, neboť při realizaci projektu bude použita právě obyčejná statická kamera.

2.2.2 Digitalizace obrazu

Vstupní signál vzniklý snímáním reálného světa má spojitý charakter, protože s ním bude dále pracováno v prostředí digitálním (počítač), je nutné provést proces digitalizace, při kterém je vstupní spojitá veličina převedena do její diskrétní podoby. Obrazová funkce $f(x, y)$ nabývá reálných hodnot, které je potřeba pro další výpočty diskretizovat.

Proces digitalizace obrazu se skládá ze dvou částí: *vzorkování* a *kvantizace*. Vzorkování upravuje rozložení bodů v ploše a převádí obraz na matici $M \times N$ bodů. Diskrétní obraz f_s vznikne jako součin spojitého obrazu $f(x, y)$ a vzorkovací funkce $s(x, y)$ [11]. Druhá část digitalizace spočívá v kvantování spojité jasové úrovně každého vzorku do K intervalů, vznikne tak matice obsahující celá čísla. Proces kvantizace je vždy ztrátový a nevratný. Jednotlivé prvky ve vytvořené matici (rastru) se nazývají *pixely*[1].

2.3 Předzpracování dat

Cílem předzpracování je zejména zlepšení obrazu z hlediska jeho dalšího zpracování. Tento krok nezvyšuje informační hodnotu obrazu, pouze potlačuje nežádoucí jevy ve zpracovávaném obraze a naopak zdůrazňuje užitečnou informaci. Hlavní úkol předzpracování zpravidla spočívá v potlačení šumu, odstranění zkreslení a potlačení či zvýraznění rysů obrazu. Jak vyplývá z předcházejících řádků, vstupem i výstupem této části je samozřejmě obraz.

Existuje celá řada operací, jenž mohou být použity při přípravě vstupního obrazu pro následné zpracování. Je vždy však nutné vztahovat metody předzpracování k tomu, jaká informace bude nakonec z obrazu získána, a tak v důsledku budou vybrány jen ty operace, které povedou k zvýraznění hledané informace.

Podle [10] se operace předzpracování obrazu dělí do 4 hlavních kategorií:

- jasové transformace
- geometrické transformace
- lokální předzpracování
- restaurace obrazu

2.3.1 Jasové a geometrické transformace

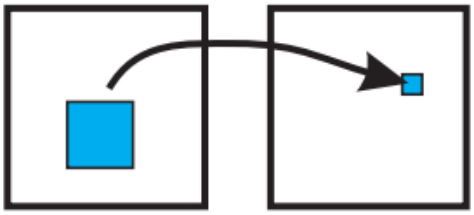
Vstupem i výstupem těchto operací je obraz stejných parametrů (rozlišení, bitová hloubka). Jasová transformace mění hodnotu obrazové funkce vstupního obrazu podle daného pravidla T [7]. Při této operaci nedochází k rozpoznávání nebo interpretaci objektů. Cílem transformace jasu představuje zvýšení kontrastu, potlačení zkreslení nebo například korekce nerovnoměrného osvětlení.

Jasové transformace se dělí do třech hlavních kategorií: globální, lokální a bodová. Již podle názvu lze usoudit, jakým způsobem bude vypočtena nová hodnota pixelu. V prvním případě se berou v potaz hodnoty z celého obrazu (znázorněno na obrázku 2.2.a). Do této

[1] Termín *pixel* vznikl zkrácením anglického spojení *picture element*, což v překladu znamená obrazový prvek.

skupiny se například řadí operace integrálního obrazu nebo Fouriérova transformace. Druhý způsob (obr. 2.2.b) výpočtu jasové hodnoty využívá lokálního okolí daného pixelu. Příkladem může být lokální filtrace (slouží k redukci šumu, vyhlazování obrazu) nebo zvýraznění rysů (hran). Poslední varianta vypočítává novou hodnotu pouze z téhož obrazového bodu (obr. 2.2.c). Tohoto způsobu výpočtu využívají techniky roztažení či vyrovnání histogramu nebo například jasová korekce.

a) globální jasová transformace



b) lokální jasová transformace

c) bodová jasová transformace

Obrázek 2.2: Grafické znázornění typů jasových transformací (vlevo vstup, vpravo výstup), převzato z [5].

Geometrické transformace na základě souřadnic ve vstupním obraze vypočítají souřadnice ve výstupním obraze. V počítačové grafice se často můžeme omezit na geometrické transformace v rovině (2D). Výpočty transformace lze však lehce zobecnit a použít tak geometrické transformace i pro operace v prostoru (3D).

Existují dva typy transformací. První provádí geometrickou transformaci bodů, kdy je zvolen souřadnicový systém a body geometrického objektu mění svou polohu právě vzhledem k systému souřadnic. Při druhém typu je geometrické transformaci podroben přímo souřadnicový systém. Tato operace se využívá například pro získání výhodnější pozice geometrického objektu pro matematické vyjádření operací s objektem. Mezi geometrické transformace se řadí operace jako posunutí, rotace kolem bodu, změna měřítka, zkosení nebo projekce.

2.3.2 Lokální předzpracování

Lokální předzpracování neboli filtrace obrazu využívá k výpočtu nové hodnoty pixelu malé okolí právě zpracovávaného obrazového bodu. Velikost okolí je určena podle vzdálenosti od zkoumaného pixelu a výsledek analýzy dané oblasti je zapsán do výstupního obrazu [6]. Metody lokálního předzpracování se mohou dělit z hlediska použití na *vyhlazování* a *hledání hran*. Jiné dělení je možné podle vlastností matematických rovnic, které jsou použity při výpočtu. Rovnice a následně metody mohou být *lineární* nebo *nelineární*.

Vyhlazování

Tento přístup lokálního předzpracování se zaměřuje na potlačení šumu v obrazu. Je potřeba si uvědomit, že metody vyhlazování jsou schopny odstranit pouze mírný šum v obrazu. Velmi zašumělé obrazy není možné jednoduše opravit a je často mnohem lepší pořídit vstupní data znovu. Pokud nelze provést nové pořízení obrazů a je alespoň znám způsob vzniku a charakter šumu, pak s pomocí těchto znalostí lze opravit vstupní data i s vysokým obsahem šumu.

Odstranění malého množství šumu lze docílit analýzou okolí daného pixelu. Při předpokladu, že sousední body mají stejnou nebo velmi podobnou hodnotu jasu, lze jednoduše určit, které obrazové body představují šum. Jsou to právě ty body, jejichž hodnota jasu se bude výrazně lišit od hodnot bodů ze svého blízkého okolí. Zašumělý bod je možné opravit *lineárním*, nebo *nelineárním* výpočtem.

Na lineární filtraci lze pohlédnout také jako na diskrétní konvoluci, přičemž konvoluční jádro definuje použité lokální okolí. Často se používá obdélníkové okolí $\mathcal{O}$ s lichým počtem řádků a sloupců, aby tak reprezentativní bod mohl ležet uprostřed konvoluční masky. Příspěvek jednotlivých pixelů v okolí $\mathcal{O}$ je vážen v lineární kombinaci koeficienty h podle vzorce:

$$g(x,y) = \sum_{(m,n)\in\mathcal{O}} \sum h(x-m, y-n) f(m,n) \tag{2.1}$$

Příkladem vyhlazování může být například obyčejné průměrování sousedních hodnot. Tohoto výsledku je možné dosáhnout použitím následující konvoluční masky:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

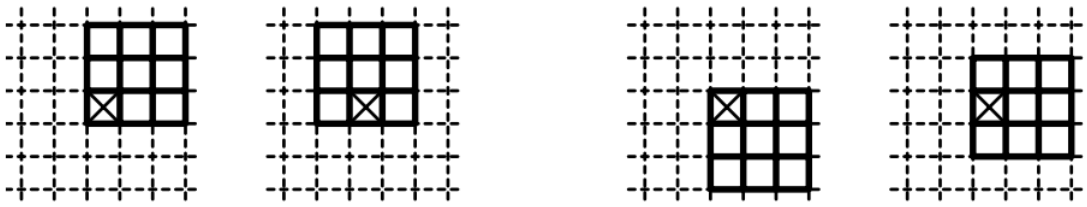
V některých případech se zvyšuje váha středového bodu masky nebo jeho sousedů. Takto upravené masky dokáží lépe vyhladit šum například s Gaussovským rozložením. Příklady často používaných masek:

$$h = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.3}$$

Hlavní nevýhoda použití obyčejného průměrování spočívá v rozmazávání hran ve vstupním obraze. Proto se průměrování většinou uplatňuje jako pomocná metoda pro výpočet střední hodnoty jasu a tento mezivýsledek je dále využit v prokročilejších nelineárních metodách [10].

Problém rozmazávání hran v obrazu částečně řeší nelineární metody vyhlazování, které se v analyzovaném okolí snaží najít část s konstantním jasem, do které patří reprezentativní pixel. Nová hodnota jasu daného bodu je vypočítána pouze z homogenní části okolí. Na tomto principu funguje například metoda *rotující masky*.

Na obrázku 2.3 je vyobrazena rotující maska o rozměru 3×3, která v okolí 5×5 vyhledává část s konstantním jasem. Rotující maska s takto nastavenými parametry může zaujmout celkově 9 různých poloh, ze kterých je zvolena ta s nejmenším rozptylem jasu.



Obrázek 2.3: Rotující maska o rozměrech 3×3 v okolí 5×5, zobrazeny 4 polohy z celkově 9 možných, převzato z [10].

Do skupiny nelineárních vyhlazovacích metod patří také například *filtrace mediánem*. Pro výpočet mediánu okolí daného pixelu postačí seřadit vzestupně hodnoty jasu v lokálním okolí a určit prostřední hodnotu, která se stává novou hodnotou zpracovávaného pixelu. Pro snadnější nalezení prostředního prvku posloupnosti se používají okolí s lichým počtem prvků, tedy o rozměrech 3×3, 5×5, atd.

Detekce hran

Metody této skupiny lokálního předzpracování obrazu jsou částečně v protikladu s předchozí skupinou, která se snažila náhlé změny v obrazu potlačit. Metody detekce hran se naopak snaží značné změny jasu v obrazu zvýraznit za účelem dalšího zpracování, například nalezení těchto hran. Místa v obrazu odpovídající významným hranám nesou více informace než jiná místa v obrazu, proto jsou častým předmětem zkoumání a hledání.

Hrana v obrazu je určena změnou hodnoty obrazové funkce $f(x, y)$. Gradient obrazové funkce je definován jako vektor ∇, jehož složky udávají směr největšího růstu funkce a strmost tohoto růstu [10]. Pro spojitou obrazovou funkci $f(x, y)$ jsou velikost gradientu $||\nabla f(x, y)||$ a směr gradientu ϕ určeny následujícími vztahy:

$$||\nabla f(x,y)|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.4}$$

$$\phi = arg\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{2.5}$$

Směr gradientu ϕ je úhel v radiánech mezi souřadnou osou a vektorem argumentu funkce.

Hranové detektory je možné rozdělit do tří hlavních skupin. Detektory z první kategorie hledají maximum prvních derivací obrazové funkce. V praxi se používají operátory, které lze vyjádřit jako masky pro konvoluci. Tyto operátory aproximují právě hledanou první derivaci obrazové funkce. Existuje několik operátorů, které se liší složitostí výpočtu a následnou přesností detekování hran. Patří zde například *Robertsův*, *Prewittové*, *Sobelův*

a jiné. Konvoluční masky jednotlivých operátorů a jejich podrobná specifikace může být nalezena v jiné literatuře, například v [10]. Do této skupiny se řadí také *Cannyho hranový detektor*, jenž se v jistém smyslu považuje za vyvrcholení snahy o nalezení nejlepšího hranového detektoru.

Druhá skupina hranových detektorů využívá druhé derivace obrazové funkce. Do této třídy patří méně operátorů, jenž aproximují druhou derivaci. Může to být například *Marr-Hildreth* nebo *LoG (Laplacian of Gaussian)* operátor [14]. Tyto metody vznikly, protože druhá derivace je výpočetně jednodušší a výsledek přesnější než hledání extrému první derivace. Operátory tohoto typu jsou také nativně invariantní vůči rotaci (na rozdíl od prvního typu). Do třetí skupiny patří detektory založené na lokální aproximaci obrazové funkce parametrickým modelem, například polynomem dvou proměnných.

2.4 Segmentace obrazu

Segmentace obrazu je jeden z nejdůležitějších kroků při analýze obrazu. Hlavní úkol představuje rozdělení obrazu na části, které mají silnou spojitost s objekty nebo oblastmi reálného světa obsažené ve vstupním obrazu. Výsledkem má být soubor vzájemně se nepřekrývajících regionů, které buď jednoznačně korespondují s objekty vstupního obrazu - jedná se o *kompletní segmentaci*. Pokud získané segmenty nesouhlasí přímo s předměty v obrazu, mluví se o tzv. *částečné segmentaci* [11].

Pro kompletní segmentaci je obecně nezbytná spolupráce s vyšší úrovní zpracování, v níž se využívá konkrétních znalostí řešeného problému. Existují však i úlohy, na kterých lze provést kompletní segmentaci pouze s nižší úrovní zpracovaní. Jedná se především o situace, kdy je obraz tvořen kontrastními objekty na neměnícím se pozadí (např. obraz s psaným textem). V takových případech je možno dosáhnout kvalitních výsledků pouze s globálními postupy bez bližší znalosti kontextu.

Při částečné segmentaci je výsledkem rozdělení obrazu do samostatných částí, které jsou homogenní vzhledem k určitým vlastnostem, jako jsou jas, barva, textura a tak podobně. Vytvořené oblasti, které jsou podobné v jistých rysech, se obecně mohou překrývat [11]. Na takto získaná data je potřeba použít postupy vyšší úrovně, jenž umožní najít výslednou segmentaci obrazu.

Mezi hlavní překážky dokonalé segmentace patří nejednoznačnost obrazových dat, šum v obrazu, nerovnoměrné osvětlení nebo jiné nedokonalosti. V této fázi tak není důležité získat přesnou segmentaci, naopak i částečná segmentace zajistí okamžitý přínos v podobě výrazné redukce objemu zpracovávaných dat, na které následně mohou být použity techniky vyšší úrovně pro celkové zpřesnění výsledku [10].

Metody segmentace mohou být rozděleny do několika základních skupin. Existuje mnoho způsobů dělení, v této práci však bude použita kategorizace uvedená v literatuře [10]. Hlaváč, Šonka a Boyle ve své knize představují rozdělení metod do skupin podle dominantní vlastnosti, které je při segmentaci využito. První kategorie využívá *globální znalosti* obrazu (nebo jeho části) reprezentované obvykle histogramem určitých vlastností. Další skupina zahrnuje metody založené na základě *detekce hran*. Třetí hlavní třídu představují metody orientující se na *detekci oblastí*.

2.4.1 Segmentace prahováním

Segmentace prahováním představuje nejjednodušší metodu segmentace obrazu, která je založena na hodnocení jasu každého pixelu. Tato metoda transformuje vstupní obraz f na

výstupní binární obraz g s prahem T. Ve výstupním obrazu pixely nabývají pouze binárních hodnot $\{1, 0\}$, kde 1 znamená, že daný obrazový bod náleží objektu (hodnota jasu je vyšší nebo rovna prahu). Naopak pixely s hodnotou 0 představují pozadí.

$$g(i,j) = \begin{cases} 1, & \text{pro} \quad f(i,j) \geq T \\ 0, & \text{pro} \quad f(i,j) < T \end{cases} \tag{2.6}$$

Pro prahování obrazu exituje celá řada variant, které se liší v typech prahu nebo způsobu jeho výpočtu. Základní metoda využívá jedné hodnoty prahu, která je použita globálně pro celý obraz. Tento přístup není vhodný pro obrazy s nerovnoměrným osvětlením.

Modifikací základního prahování je tzv. *poloprahování*, které pixelům mající vyšší nebo stejnou hodnotu prahu T ponechává originální hodnotu jasu, matematický předpis této techniky se zapisuje následovně:

$$g(i,j) = \begin{cases} f(i,j), & \text{pro} \quad f(i,j) \geq T \\ 0, & \text{pro} \quad f(i,j) < T \end{cases} \tag{2.7}$$

Jiným rozšířením je *adaptivní prahování (nebo také proměnné prahování)*. Při této technice představuje práh funkci polohy v obrazu, což znamená, že práh se určuje vždy pro část obrazu. V praxi je obraz rozdělen na několik předem daných oblastí, pro každou je nalezen práh (může být interpolován) a následně se provede prahování vždy s konkrétním prahem. Dalšími variantami mohou být procentní prahování nebo například vícestupňové prahování, které využívá více prahů pro rozdělení obrazu.

Zvolení vhodné hodnoty prahu má velký význam na výsledek celé operace. Práh může být zvolen experimentálně nebo také vypočítán automaticky. Při tomto výpočtu se často používá analýza histogramu. Za vhodný práh může být zvoleno lokální minimum mezi dvěma maximy, polovina vzdálenosti mezi maximy a jiné přístupy.

2.4.2 Segmentace na základě detekce hran

Tento typ segmentace využívá hranové detektory, jenž byly představeny v kapitole 2.3.2. Aplikací hranového filtru na vstupní obraz je získán výstupní obraz, který obsahuje pixely klasifikované do dvou skupin: body představující hrany v obrazu a body ležící mimo detekované hrany. Tato informace však pro segmentaci obrazu není dostačující, a tak je potřeba provést operaci, která spojí nalezené hrany do hranic.

Metody tohoto přístupu se musí vypořádat s problémem výskytu hran na místech, kde ve skutečnosti neexistuje hranice reálného objektu. "Falešné"hranice mohl do obrazu vložit přítomný šum, nepotřebné hrany pro segmentaci mohou být také detekovány na velmi členitém povrchu objektu v obrazu. Protichůdný problém při vytváření hranic představují chybějící hrany, které měly být v obrazu detekovány na místech, kde probíhá skutečná hranice objektu.

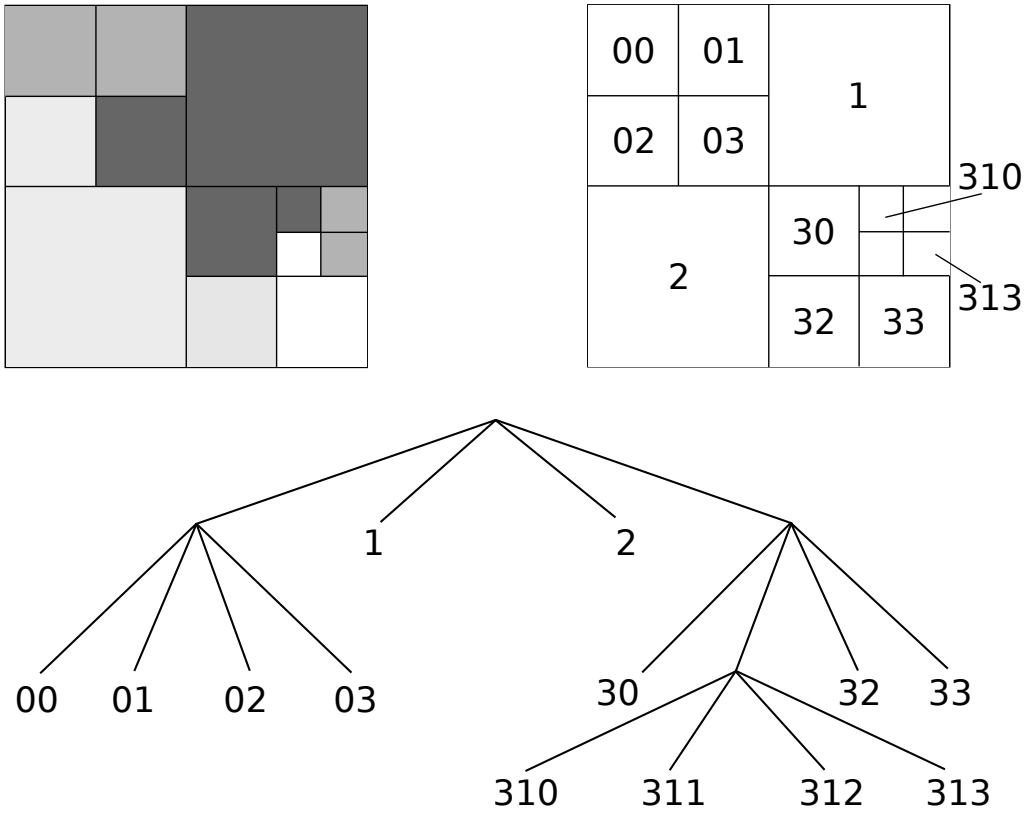
Segmentační metody spadající do této kategorie se musí vypořádat s výše popsanými problémy a vybrat správné hrany, popřípadě doplnit chybějící tak, aby vytvořené hranice oblastí co nejpřesněji kopírovaly skutečné hranice. Způsoby konstrukce hranic závisí na konkrétní implementaci metody, cílem však nadále zůstává kvalitní segmentace obrazu na základě detekce hran.

2.4.3 Segmentace na základě detekce oblastí

Metody tohoto typu segmentace fungují na principu shlukování sousedních pixelů, které mají stejné (podobné) hodnoty jako počáteční bod. Na rozdíl od metod z předcházejícího odstavce, algoritmy této skupiny nehledají hranice segmentů, nýbrž přímo hledané oblasti. Spojování do oblastí s podobnými vlastnostmi se může zakládat na porovnání jasu, barvy či textury. Skupina těchto metod si lépe poradí i s vyšším výskytem šumu v obrazu (na rozdíl od hranových detektorů).

Konstrukce segmentu může postupovat zdola nahoru. Tyto algoritmy iterativně zvětšují oblasti podobných bodů od počátečních pixelů – označované jako „semínka“ *(angl. seeds)*. Následně jsou odstraňovány slabé hrany mezi získanými segmenty, a tak dochází ke spojování do větších oblastí. Jednotlivé metody se liší způsobem vyběru počátečních bodů a také pravidly pro následné spojování oblastí.

Kromě segmentance narůstáním oblastí existují také algoritmy, které nejdříve vstupní obraz rozdělí na mnoho malých segmentů (předem dané struktury) a následně je spojí do větších oblastí na základě jejich podobnosti. Před začátkem samotné segmentace je potřeba definovat strukturu, která bude spravovat rozdělené oblasti. Často se využívá tzv. *quad-tree* stromová struktura, která je graficky znázorněna na obrázku 2.4. Pro konečné spojení segmentů je potřeba definovat kritérium, které se může lišit od kritéria použitého pro krok rozdělení.



Obrázek 2.4: Stromová struktura *quad-tree*, která se používá pro segmentaci metodou *split and merge*, převzato z [15].

2.5 Popis objektů

Následující oddíl se věnuje kroku popisu nalezených oblastí ve obraze. Metodám segmentace a získáním zajímavých oblastí z obrazu se věnuje předcházející kapitola. Popis oblastí je velmi důležitý krok z pohledu celkového porozumění obrazu. Aby rozpoznání nalezených oblastí mohlo být realizováno, například pomocí klasifikátoru, exaktní popis těchto částí je tak nezbytným mezikrokem.

Cílem popisu může být vypočítání *číselného vektoru příznaků*, nebo určení *nečíselného syntaktického popisu*, který by charakterizoval tvarové nebo jiné vlastnosti popisovaných oblastí [11]. Získání příznaků je však velmi obtížný proces, podobně jako definovat nějaký tvar. V lidském popisu jsou tvary slovně označovány jako kulaté, špičaté, dlouhé, s ostrými rohy apod. Počítače však s takovými termíny nedokáží úspěšně pracovat, a tak je potřeba nalézt nějakou matematickou reprezentaci.

Protože reálné objekty mohou nabývat rozmanitých tvarů, nepodařilo se prozatím v oblasti počítačového vidění nalézt obecnou metodologii, která by byla schopná obecně popsat různé objekty. V této problematice dokonce není ani přesně jasné, jaké vlastnosti jsou důležité pro samotný popis tvaru. Metody pro popis jsou tak vybírány až na základě konkrétního problému. Například popis výrazných zakřivení hranic oblastí může být využit pro rozpoznávání alfanumerických znaků, strojních součástí nebo křivek elektrokardiogramu (EKG) [10].

Po zvolení vhodných metod pro realizaci projektu by v následujících odstavcích měly být nastíněny alespoň základní principy těchto operací.

2.6 Klasifikace

Poslední krok přístupu zpracování obrazu, který byl představen na začátku této kapitoly, představuje použití klasifikačního algoritmu, jenž bude schopný třídit vstupní vzorky do předem zvolených tříd. Tato část zpracování obrazu je nezbytná pro porozumění počítače danému obrazu.

Rozpoznávání předmětů (klasifikace) spočívá v zařazování předmětů do tříd. Stroj, který tuto činnost vykonává, se nazývá *klasifikátor*. Počet tříd je určen předem – narozdíl od shlukování. Metody klasifikace mohou být rozděleny do dvou hlavních skupin [11]. První se nazývá rozpoznávání *příznakově* popsaných předmětů, ve kterém hraje velkou roli matematický popis příznaků v obrazu. Metody této skupiny využívají kvantitativního popisu předmětů. Klasifikaci provádí klasifikátor, natrénovaný vhodnou trénovací množinou.

Druhou skupinu představuje rozpoznávání *syntakticky* popsaných předmětů. Vstupní popis tohoto přístupu má kvalitativní charakter, jenž odráží strukturu předmětu. Elementární vlastnosti syntakticky popsaných objektů se nazývají *primitiva*. Množina všech primitiv se označuje jako *abeceda*. Dále jsou defnovány termíny jako *jazyk popisu*, nebo *gramatika*. Konečné rozpoznávání závisí na výsledku syntaktické analýzy. Předmět je zařazen do té třídy, jejíž gramatika dané slovo generuje.

Podle jiných autorů se k uvedeným dvěma skupinám přidává třetí, do které spadá klasifikace pomocí *neuronových síti*. Tato jistě významná podoblast umělé inteligence by mohla být teoreticky popsána na mnoha následujících stránách. Protože však v implementaci této práce nebyly použity neuronové sítě, nebudou dále podrobněji vysvětleny.

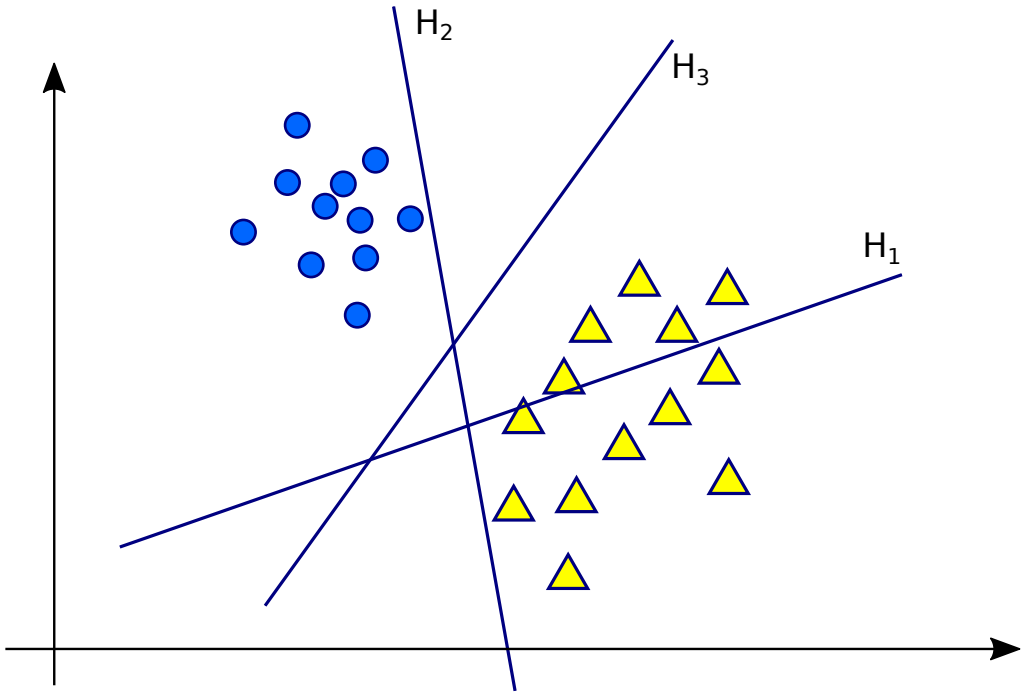
Bayesovský klasifikátor, *Perceptron*, *Support vector machines* nebo *rozhodovací stromy* patří mezi nejznámější klasifikátory spadající do první skupiny, jenž pro klasifikaci dat používají matematický popis příznaků. Následně bude popsán algoritmus *SVM*, který měl

být původně použit v praktické části. Výsledky testování ukázaly, že exituje vhodnější metoda (využití značek na kapotě auta) k rozpoznání robotického auta, a tak klasfikátor *SVM* nakonec v práci uplatnění nenašel.

2.6.1 Klasifikátor SVM

Za zkratkou *SVM* se ukrývá anglický název *Support Vector Machines*, který se do češtiny převážně nepřekládá. Jedná se o algoritmus, který používá „podpůrné“ vektory pro popis nadrovin, jenž rozdělují vstupní vzorky do jednotlivých tříd. Klasifikace pomocí SVM byla představena v roce 1962 ruským vědcem Vladimirem Vapnikem. Své uplatnění velmi často nachází při řešení problému kategorizace textu, analýze digitálního obrazu nebo v oblasti bioinformatiky.

Výpočetní algoritmus hledá nejlépe oddělující nadrovinu vstupních dat. Jako kritérium pro porovnání vhodnosti nadrovin je brána vzdálenost mezi nejbližšími vzorky obou tříd a dané nadroviny. Metoda *SVM* zvolí dělící nadrovinu s maximálním odstupem (anglicky *maximal margin*) od klasifikovaných vzorků. Na obrázku 2.5 lze vidět příklad dělení vstupních vzorků pomocí nadrovin. H_1 nerozděluje data do dvou tříd, H_2 odděluje data úspěšně, pouze však s malým odstupem mezi třídami. Nadrovina H_3 rozděluje data do dvou tříd lépe než H_2.



Obrázek 2.5: Rozdělení vstupních dat pomocí nadrovin H_1, H_2 a H_3, algoritmus SVM zvolí nadrovinu s maximálním odstupem – H_3, převzato z [12].

Další důležitou součástí algoritmu *SVM* představuje jádrová transformace prostoru příznaků, která mapuje původní prostor do vyšší dimenze, a tak umožňuje klasifikovat i lineárně neseparovatelné problémy. Ze své podstaty je algoritmus *SVM* binární (tedy klasifikuje data do dvou tříd), po provedení několika rozšíření je však možné dosáhnout klasifikace i do více tříd.

Kapitola 3

Pravděpodobnostní plánování

Tato kapitola se zabývá studiem plánovacích algoritmů, které se používají v oblasti robotiky při hledání cest. Text se bude převážně věnovat pravděpodobnostním algoritmům pro plánování, jelikož podle zadání prace právě tímto typem algoritmů má být robot navigován a řízen v prostoru.

Na úvod lze konstatovat, že existuje celá řada jiných plánovacích algoritmů, které se mohou uplatňovat v robotice. Pod názvem *neinformované* plánovací metody se slučují algoritmy, které při svém výpočtu nemají kompletní znalost o prohledávaném prostředí. Do této skupiny se řadí algoritmus *Bug1*, *Bug2* nebo například *Tangent Bug* [8]. Podrobnější popis těchto metod není v práci uveden.

Druhou skupinu představují *informované* metody, které při svém výpočtu mají kompletní povědomí o prostředí – vědí o překážkách, stejně tak znají informaci o startovním a cílovém bodě. Při použití těchto algoritmů je nejdříve vypočtena celá cesta založená na dané mapě, poté se teprve robot začne pohybovat po nalezené trase směrem k cíli [8]. Tento typ metod lze dále dělit do několika kategorií. Metody *rozkladu na buňky* používají pro svůj výpočet diskretizaci prostoru do mřížky. Do této kategorie patří *klasické hledání cesty v grafu* nebo metoda *potenciálových polí*. Další kategorií mohou být tzv. *mapy cest (Roadmaps)*, do které patří metody jako *graf viditelnosti* nebo *Voroného diagram*. Konečně poslední skupinou *informovaných* metod jsou právě *pravděpodobnostní algoritmy*, které budou blíže představeny následujících kapitolách [2].

3.1 Reprezentace prostředí robota

Aby bylo vůbec možné použít plánovací algoritmy pro výpočet a nalezení cesty, je potřeba reálné prostředí robota nejprve definovat matematickým popisem. V robotice bylo zavedeno několik pojmů, které usnadňují a napomáhají vytvoření modelu reálného světa.

3.1.1 Pracovní prostor

Robot se pohybuje a koná činnost v nějakém prostředí. Toto prostředí se nazývá pracovní prostor a značí se $\mathcal{W}$. Pracovní prostor $\mathcal{W}$ lze reprezentovat jako N-rozměrný euklidovský prostor $\mathbb{R}^N$, kde $N = 2, 3$ (dvojrozměrný nebo trojrozměrný prostor). Jak bylo popsáno v kapitole 2.2.1, při snímání prostoru kamerou se perspektivním zobrazením převádí obraz pouze do 2D podoby. Protože je však scéna snímána z ptačí perspektivy, není ztráta třetí

dimenze zásadní problém. Ve zbytku práce tak bude uvažováno zjednodušení:

$$\mathcal{W} = \mathbb{R}^2 \tag{3.1}$$

V pracovním prostředí se může vyskytovat množina statických překážek $\mathcal{O}$. Označme pohyblivé auto v prostoru jako $\mathcal{A}$. Nechť platí $\mathcal{O} \subset \mathcal{W}$, podobně bude platit $\mathcal{A} \subset \mathcal{W}$. Pro matematický popis pozice robota v prostoru byl zaveden termín *konfigurační prostor*.

3.1.2 Konfigurační prostor

Podle [9] je konfigurační prostor definován jako množina všech transformací robota. Tento koncept umožňuje reprezentovat robota jedním bodem v prostoru, což znamená jednodušší vstupní informaci pro plánovací algoritmy. Nechť konfigurační prostor (C-space) je označen jako $\mathcal{C}$, dále označme konfiguraci robota jako q, která jednoznačně označuje pozici robota.

Konfiguraci lze reprezentovat jako m rozměrný vektor, jehož složky jednoznačně určují pozici a orientaci robota v pracovním prostoru. Velikost vektoru je nazývána *stupněm volnosti robota*. Oblast prostoru $\mathcal{W}$, kterou zabírá robot $\mathcal{A}$ nacházející se v konfiguraci q se označuje jako $\mathcal{A}(q)$ [1] [13].

Pro robotické autíčko v 2D prostoru bude konfigurace definována jako

$$q = (x, y, \theta) \tag{3.2}$$

kde x a y jsou souřadnice referenčního bodu robota, θ představuje úhel natočení auta vzhledem k počátku prostoru $\mathcal{W}$ [1].

V konfiguračním prostoru lze definovat podmožinu $\mathcal{C}_{free}$, která bude obahovat pouze přípustné konfigurace systému. Nazvěme ji volný konfigurační prostor, pro něhož platí $\mathcal{C}_{free} \subseteq \mathcal{C}$ a je definován:

$$\mathcal{C}_{free} = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} = 0\} \tag{3.3}$$

V opozici k $\mathcal{C}_{free}$ stojí konfigurační prostor $\mathcal{C}_{obs}$ obsahující konfigurace, ve kterých dochází ke kolizi mezi robotem a překážkou. Definice je následující:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} \neq 0\} \tag{3.4}$$

a platí:

$$\mathcal{C}_{obs} = \mathcal{C} / \mathcal{C}_{free} \tag{3.5}$$

Pro úspěšné plánování je potřeba ještě stanovit počáteční konfiguraci robota q_I a také cílovou q_G. Obě takto dané konfigurace musí náležet do množiny volného konfiguračního prosotoru [1]. Na obrázku 3.1 je zobrazen příklad konfiguračního prostoru.

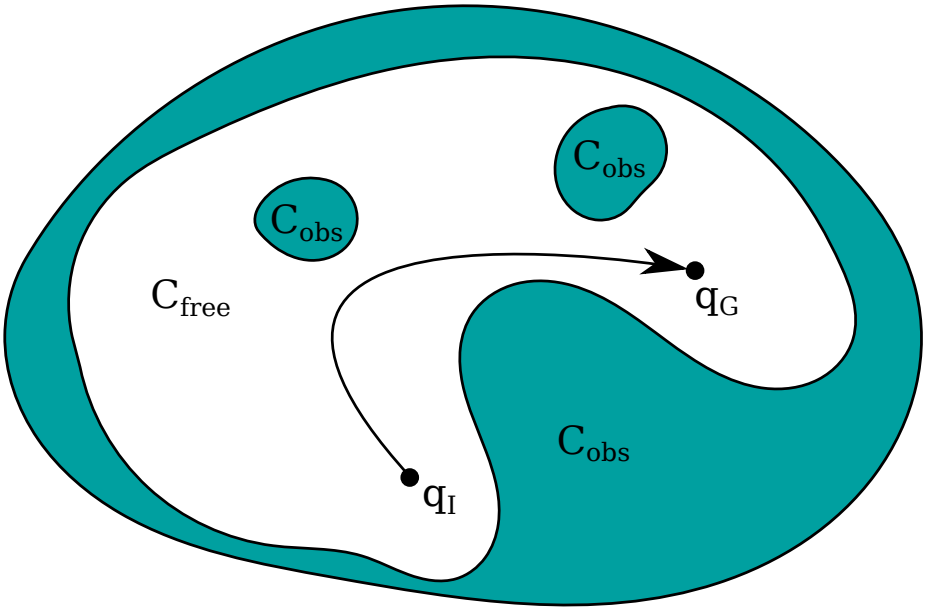
Metrika

Každý plánovací algoritmus vyžaduje funkci, která měří vzdálenost mezi dvěma konfiguracemi v prostoru $\mathcal{C}$. Ve většině případů tato nutnost vede k uvedení termínu *metrika prostoru*, pomocí které může být určena vzdálenost mezi body v konfiguračním prostoru [9].

Definujme *metriku prostoru* jako (X, ρ), kde X je topologický prostor a ρ je zobrazení:

$$\rho : X \times X \to \mathbb{R} \tag{3.6}$$

a pro jakékoliv $a, b, c \in X$ platí následující axiomy:



Obrázek 3.1: Grafické znázornění konfiguračního prostoru, tak jak je popsán v kapitole 3.1.2, převzato z [9].

1. **Nezápornost:** $\rho(a, b) \geq 0$
2. **Totožnost:** $\rho(a, b) = 0 \Leftrightarrow x = y$
3. **Symetrie:** $\rho(a, b) = \rho(b, a)$
4. **Trojúhelníková nerovnost:** $\rho(a, b) + \rho(b, c) \geq \rho(a, c)$

Euklidovská metrika

Metrika splňující tyto axiomy umožňuje určit vzálenost v libolném prostoru a její volba závisí na konkrétní aplikaci. Výpočet cest v tomto projektu je situován do *euklidovského prostoru*, pro který se používá *euklidovská metrika*, která je v N-rozměrném prostoru definována takto:

$$\rho(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2} \tag{3.7}$$

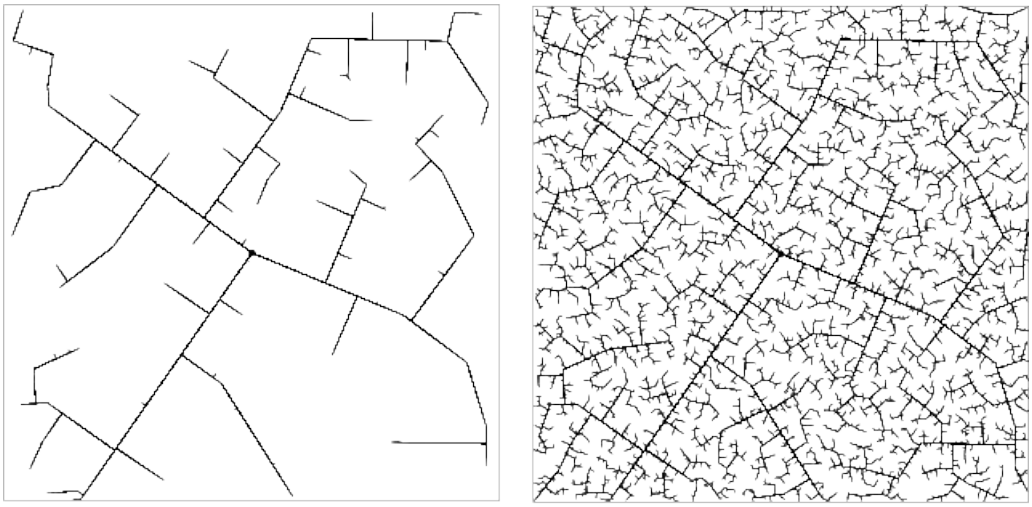
kde $a = (a_1, a_2, \ldots, a_n) \in \mathbb{R}_n$ a $b = (b_1, b_2, \ldots, b_n) \in \mathbb{R}_n$

3.2 Algoritmus RRT

Rapidly-exploring Random Tree (dále jen RRT) je datová struktura a algoritmus navržený pro efektivní prohledávání vícedimenzionálních prostorů. Termín se RRT se do češtiny nepřekládá, pro vysvětlení by bylo možno algoritmus popsat jako rychle prohledávací náhodný strom. RRT poprvé představili Steven M. LaValle a James Kuffner, velmi podrobné informace lze získat v knize *Planning Algorithms*, kterou sepsal právě Steven LaValle.

Algoritmus RRT se velmi hodí pro problémy hledání cesty v prostředí, které zahrnuje překážky a jiné omezující podmínky (kinetické a dynamické). Jedná se o metodu jednodotazovou, kdy se nevytváří graf celého prostoru ale pouze strom, jenž spojuje výchozí a cílovou

pozici. Metoda funguje tak, že z výchozí pozice q_I strom expanduje do neprozkoumaného prostoru k náhodně vygenerované konstrukci z konfiguračního prostoru $\mathcal{C}$. Algoritmus je tedy inkrementálního charakteru, což je možno pozorovat na obrazku 3.2.

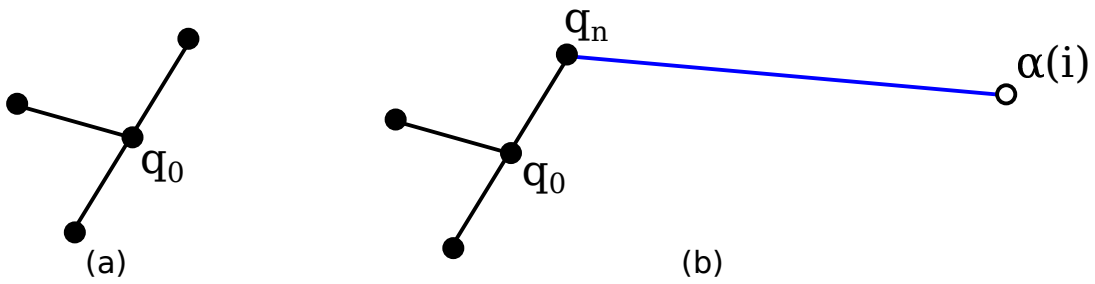


Obrázek 3.2: RRT algoritmus po 45 iteracích (vlevo), po 2345 iteracích (vpravo), převzato z [9].

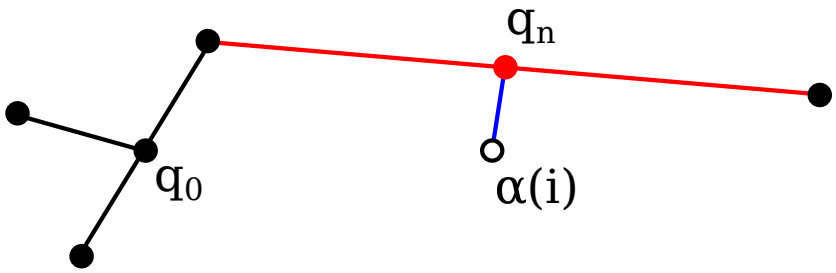
3.2.1 Základní varianta

Základní forma algoritmu RRT pro konstrukci stromu vyžaduje jako vstup pouze množinu s konfiguracemi. Tato varianta algoritmu však není schopna detekovat překážky, pro potřeby této práce tak není dostačující. Protože z ní ale vycházejí rozšířené varianty schopné detekce překážek, základní algoritmus je zapsán v 3.2.1.

Algoritmus iterativně spojuje náhodný vzorek $\alpha(i)$ a nejbližší bod stromu (grafu) G. Připojení k nejbližšímu bodu q_n je ukázáno na obrázku 3.3. RRT je topologický graf $G(V, E)$ sestavený z vrcholů V a hran E, $S \subset C_{free}$ představuje množinu bodů dosažitelných grafem G. V případě kdy nejbližší konfigurace q_n je někde na hraně stromu (ne na jeho vrcholu), hrana je rozdělena na dvě části a do grafu jsou přidány dva nové vrcholy q_n a $\alpha(i)$ společně s hranou mezi nimi. Tento speciální proces je zobrazen na obrázku 3.4.



Obrázek 3.3: Spojení náhodného vzorku $\alpha(i)$ s nejbližším bodem grafu q_n, převzato z [9].



Obrázek 3.4: Speciální spojení kdy nejbližší bod není vrchol grafu. Hrana je rozdělena na dvě části, převzato z [9].

Algoritmus 1 Základní varianta RRT

```
1: G.init(q_0);
2: for i = 1 to k do
3:     G.add_vertex(α(i));
4:     q_n ← NEAREST(S(G), α(i));
5:     G.add_edge(q_n, α(i));
6: end for
```

3.2.2 Varianta s jedním stromem

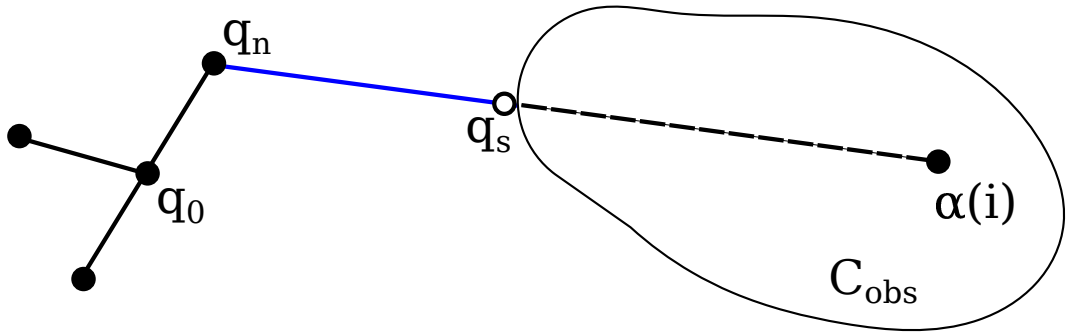
Poměrně efektivní plánovač cesty může být vytvořen pomocí algoritmu, které je popsán pseudokódem v 3.2.2. Tato varianta si poradí s překážkami v prostoru, což lze porozovat v popisu algoritmu na řádku 4 a 5. Metoda *STOPPING-CONFIGURATION* vrací bod q_s, který je vytvořen na hraně mezi vzorkem $\alpha(i)$ a nejbližším bodem q_n, pokud náhodně vybraný vzorek leží v prostoru překážky. Graficky je tento postup znázorněn na obrázku 3.5. Nový bod q_s získán pomocí algoritmu, který se stará o detekci kolizí [9].

Algoritmus 2 RRT s detekcí překážek

```
1: G.init(q_0);
2: for i = 1 to k do
3:     q_n ← NEAREST(S, α(i));
4:     q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5:     if q_s ≠ q_n then
6:         G.add_vertex(q_s);
7:         G.add_edge(q_n, q_s);
8:     end if
9: end for
```

3.2.3 Varianta s dvěma stromy

Mnohem lepších výsledků lze dosáhnout použitím dvou stromů. První prohledává prostor z vychozí konfigurace q_I a naopak druhý strom vychází z cílové pozice q_G. Metoda se snaží oba rostoucí stromy spojit a zároveň udržet jejich rapidní růst. To může být dosaženo například variantou *vyvážování*, která zajistí stejnou velikost pro oba rostoucí stromy [9]. Algoritmus 3.2.3 popisuje celý postup pseudokódem.



Obrázek 3.5: Připojení nového bodu q_s a hrany ke stromu, pokud náhodně vygenerovaný vzorek $\alpha(i)$ leží v prostoru překážky, převzato z [9].

Graf G se skládá ze dvou stromů T_a a T_b původně startující z q_I a q_G. Po několika iteracích algoritmu může dojít k přehození obou stromů, a tak T_a nemusí vždy obsahovat q_I. V každé iteraci T_a roste podobně jako v základní variantě. Pokud je nový vrchol q_s přidán do stromu T_a, potom je také učiněn pokus o přidání nového vrcholu do stromu T_b (v pseudokódu řádky 10-12). Pro toto přidání se místo náhodně vygenerovaného vzorku $\alpha(i)$ použije raději vrchol q_s, což způsobuje růst stromu T_b směrem k T_a. Pokud se podaří spojit oba stromy (testovací podmínka na řádku 14), algoritmus končí a vrátí řešení.

Vyvažování obou stromů vykonávají řádky 18-19 v pseudokódu. Tento důležitý krok se uplatní v situacích, kdy jeden strom má problém s prohledáváním a růstem, jelikož v jeho okolí je mnoho překážek. Nový průzkum má tak na starosti právě menší strom. Metod jak definovat menší strom existuje více, jednoduché řešení je použít strom s menším celkovým počtem vrcholů [9].

3.3 Plánování pohybu robotického auta

V této kapitole bude představen koncept plánování cesty pro roboty, jenž se pohybují na stejném principu jako běžné automobily. Robotická autíčka se řadí mezi tzv. *neholonomické* systémy. Opakem jsou systémy *holonomické*, které mohou být v klasické mechanice definovány jako systémy, jejichž všechna omezení jsou holonomická, tedy jsou vyjádřitelná jako funkce:

$$f(x_1, x_2, \ldots, x_n, t) = 0 \tag{3.8}$$

Holonomická omezení závisí pouze na souřadnicích x_j a času t, naopak nezávisí na hybnosti nebo rychlosti systému. Příkladem takového systému může být například kyvadlo.

Omezení, jenž nelze vyjádřit ve formě 3.8, se označují na neholonomická. Neholonomické systémy v robotice lze také rozpoznat podle stupňů volnosti daného systému. Pokud je počet říditelných stupňů volnosti menší než počet potřebný pro provedení libovolného pohybu v daném prostoru, potom takový systém patří mezi neholonomické. Typickým příkladem je auto, které pohyb do stran může provádět pouze s omezením maximálního natočení předních kol.

Algoritmus 3 RRT s dvěma stromy

```
1: T_a.init(q_I); T_b.init(q_G);
2: for i = 1 to k do
3:     q_n ← NEAREST(S_a, α(i));
4:     q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5:     if q_s ≠ q_n then
6:         T_a.add_vertex(q_s);
7:         T_a.add_edge(q_n, q_s);
8:         q'_n ← NEAREST(S_b, q_s);
9:         q'_s ← STOPPING-CONFIGURATION(q'_n, q_s));
10:        if q'_s ≠ q'_n then
11:            T_b.add_vertex(q'_n);
12:            T_b.add_edge(q'_n, q'_s);
13:        end if
14:        if q'_s = q_s then
15:            return SOLUTION;
16:        end if
17:    end if
18:    if |T_b| > |T_a| then
19:        SWAP(T_a, T_b);
20:    end if
21: end for
22: return FAILURE;
```

3.3.1 Neholonomický robot

Na řádcích výše byl vysvětlen termín *neholonomičnosti*, nyní je potřeba uvažovat omezující podmínky takového systému a matematicky popsat rovnice přechodu mezi konfiguracemi.

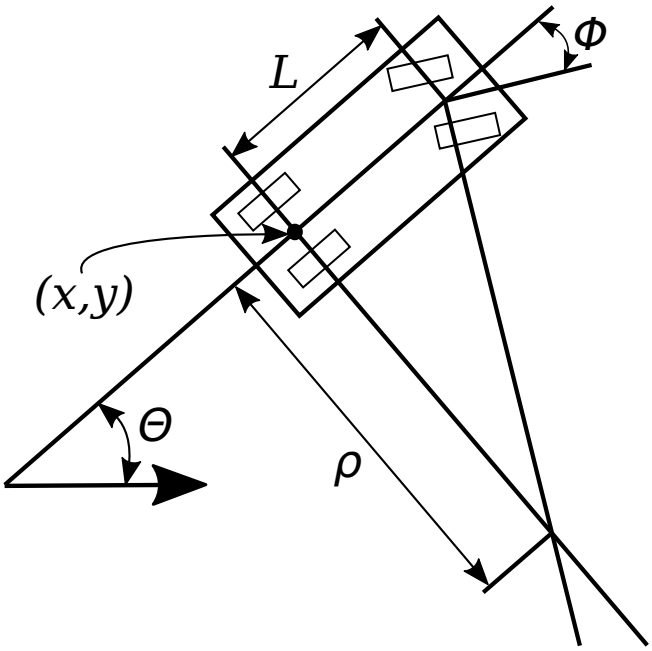
Nákres robotického autíčka a jeho parametry jsou zobrazeny na obrázku 3.6. Jeho konfigurační prostor (více v kapitole 3.1.2) je dán předpisem $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. Konfigurace pro robota ve 2D prostoru je definována jako $q = (x, y, \theta)$, kde x a y jsou souřadnice referenčního bodu, který se nachází uprostřed osy zadních kol (viz obrázek 3.6). Znak θ představuje úhel mezi hlavní osou robota a počátkem souřadného systému.

Hodnota ω představuje velikost úhlu natočení předních kol, který může být negativní pro zatáčení doleva a pozitivní pro zatáčení doprava. Pro praktické autíčka existuje také maximální hodnota natočení kol. Vzdálenost mezi zadní a přední osou kol je označena jako L. Pokud se úhel zatáčení nemění, auto se pohybuje po kruhové trajektorii s poloměrem ρ.

Mějme vektor pohybu auta $u = (u_s, u_\phi)$, který obsahuje složku rychlosti u_s a složku natočení předních kol u_ϕ. Rovnice přechodu mezi konfiguracemi robota mohou být vyjádřeny následovně:

$$\begin{aligned} \dot{x} &= u_s \cos(\theta) \\ \dot{y} &= u_s \sin(\theta) \\ \dot{\theta} &= \frac{u_s}{L} \tan(u_\phi) \end{aligned} \tag{3.9}$$

Aby rovnice přechodu byly kompletní, je potřeba specifikovat hodnoty, kterých mohou nabývat složky u_s a u_{phi}. Pro zjednodušení lze uvažovat hodnoty pro rychlost pouze z množiny $\{-1, 0, 1\}$, kde negativní hodnota představuje pohyb auta vzad, pozitivní znamená pohyb vpřed a 0 značí, že robot se nepohybuje.



Obrázek 3.6: Znázornění robotického autíčka se třemi stupni volnosti, převzato z [9].

Poslední rovnice v 3.9 slouží pro výpočet nového úhlu natočení auta. Lze z ní pozorovat, že hodnoty $\pi/2$ a $-\pi/2$ pro natočení předních kol jsou problematické, jelikož je následně počítán *tangens* daných hodnot. Podmínka $|u_\phi| \leq \phi_{max} < \pi/2$ ošetří problematické hodnoty a ϕ_{max} bude představovat maximální natočení předních kol robotického autíčka. Protože reálná provedení aut neumožňují zatočit koly pod úhlem $\pi/2$, navržená podmínka nijak neomezuje reálné vstupní hodnoty. Odvození rovnic 3.9 a podrobnější matematický popis lze nalézt v knize [9].

3.3.2 Výpočet nového uzlu v RRT

Pravděpodobnostní algoritmy RRT pracují na principu konstrukce stromu v konfiguračním prostoru. Po vygenerování nové náhodné konfigurace q, je pro ni vyhledán v existujícím stromu nejbližší uzel vn. K uzlu vn je připojen nový uzel qn, který je nejblíže k původní náhodné konfiguraci q. Takto strom expanduje až k cílové pozici [22].

Výpočet nejbližšího uzlu je optimalizační problém, jenž bude dále popsán rovnicemi, ve kterých se bude objevovat struktura konfigurace se třemi stupni volnosti. Použití této struktury zpřehlední celý popis, protože nebudou použity indexy, ale konkrétní jména proměnných (výsledný popis bude lehce připomínat „pseudokód“).

Mějme strukturu `q = (x,y,theta)`, která popisuje konkrétní konfiguraci robota v prostředí. Vysvětlení jednotlivých složek se nachází v kapitole 3.1.2. Přístup ke složkám struktury uvažujme podobně jako v jazyce C přes operátor ”.”(tečky), např. q.x přistupuje na složku x konfigurace q.

Výpočet vzdálenosti mezi dvěma konfiguracemi je dána podle vztahu:

$$dist(qn, q) = \sqrt{(qn.x - q.x)^2 + (qn.y - q.y)^2 + A * minTheta^2} \tag{3.10}$$

$$\begin{aligned} minTheta = min[&(qn.theta - q.theta),\\ &(qn.theta - q.theta + 2 * PI),\\ &(qn.theta - q.theta - 2 * PI)] \end{aligned} \tag{3.11}$$

Výpočet nové konfigurace se provede podle:

$$qn.x = vn.x + v * dt * \cos(phi) * \sin(vn.theta)$$

$$qn.y = vn.x + v * dt * \cos(phi) * \cos(vn.theta)$$

$$qn.theta = vn.theta + v * dt * \sin(phi)$$

kde qn je označení pro novou konfiguraci, vn pro výchozí konfiguraci a dále:

- dt je krok posunu auta
- v je rychlost, nabývá hodnot z množiny $\{1, 0, -1\}$
- phi je úhel natočení předních kol auta

Optimalizační problém výpočtu nové nejbližší konfigurace lze vyřešit derivací vztahu 3.10 podle phi a následně položit derivaci rovno nule.

Pro rychlost $v = 1$:

$$\frac{\partial(dist(qn, q))}{\partial(phi)} = 0 \tag{3.12}$$

$$a * \cos(phi) + b * \sin(phi) + c * \cos(phi) * \sin(phi) = 0 \tag{3.13}$$

Pro rychlost $v = -1$:

$$\frac{\partial(dist(qn, q))}{\partial(phi)} = 0 \tag{3.14}$$

$$a * \cos(phi) + b * \sin(phi) - c * \cos(phi) * \sin(phi) = 0 \tag{3.15}$$

kde

- $a = \frac{A}{L} * minTheta$
- $b = (vn.x - q.x) * \cos(vn.theta) + (vn.y - q.y) * \sin(vn.theta)$
- $c = \frac{A}{L * L} - 1$
- L je vzdálenost mezi přední a zadní nápravou auta
- A je určitá konstanta

Pro rychlost $v = 0$ není třeba rovnice řešit, neboť nedojde k žádnému posunu.

Jak je uvedeno například v [22], při použití speciálního případu:

$$A = L * L$$

následně dostaneme:

$$c = 0$$

potom může být hodnota natočení kol phi vypočtena přímo z rovnice 3.13 (popřípadě z 3.15) jako:

$$phi = \arctan\left(\frac{a}{b}\right)$$

kde

- $a = \frac{A}{L} * minTheta$
- $b = (vn.x - q.x) * \cos(vn.theta) + (vn.y - q.y) * \sin(vn.theta)$

Provedenením výše popsaného výpočtu dostáváme dva optimální pohybové vektory $inp1 = (1, phi)$ a $inp2 = (-1, phi)$. Aplikací těchto vektorů na konfiguraci vn vzniknou dvě nové konfigurace, ze kterých je vybrána ta bližší k původní náhodné konfiguraci q. Takto vybrána konfigurace představuje nově hledanou konfiguraci qn. Odvození rovnic bylo vytvořeno za pomoci [22].

Kapitola 4

Návrh řešení

V této kapitole je navržen způsob řešení, který bude později realizován a implementován (více v kapitole č. 5). Na samotném začátku této kapitoly se nachází analýza zadání práce, na kterou naváže popis hardwarového zapojení a fyzické realizace projektu. Postupně bude představen počítač Raspberry Pi a jeho propojení k autíčku. Krátký odstavec se také zabývá problematikou snímání scény a výběru kamery.

Druhá část kapitoly se věnuje softwarové části diplomové práce, kterou tvoří především dva programy. První běží na osobním počítači uživatele, druhý potom na počítači Raspberry Pi. Mezi těmito dvěma systémy probíhá komunikace, jejíž popis se nachází v závěru této kapitoly.

4.1 Analýza zadání

Jak z názvu diplomové práce vypovídá, cílem je řízení pohybu robotického autíčka pomocí kamery a počítače Raspberry Pi. Zadání dále specifikuje, aby program pro extérní počítač ve snímané scéně detekoval polohu auta a překážek a následně naplánoval cestu robota v prostoru pomocí pravděpodobnostního algoritmu. Aplikace také musí umožnit uživateli pohodlně zadat cílovou polohu, na kterou se má robotické autíčko přesunout. Práce musí také obsahovat vyhodnocení úspěšnosti detekce objektů a přesnost plánování a navigace.

Ze zadání je dané, jaký typ auta má být použit a také, že autíčko má být řízeno počítačem Raspberry Pi. Tyto požadavky jsou samozřejmě respektovány. Na uživatelskou aplikaci je kladen pouze nárok na pohodlné zadání cílové polohy. To bude splněno vytvořením ovládacího programu s grafickým uživatelským rozhraním, které je v dnešní době standardem pro interakci uživatele s počítačem.

4.2 Hardwarové komponenty

Tato podkapitola se zabývá fyzickým vybavením projektu, tedy zejména ovládaným robotem (autem), prostředkem pro jeho řízení nebo použitou kamerou.

4.2.1 Robotické autíčko

Zadání práce určuje, že robot použitý v tomto projektu bude mít podobu auta. Na Fakultě informačních technologií Vysokého učení technického v Brně působí výzkumná skupina

robotiky *Robo@FIT*[1], která má k dispozici různé druhy robotických zařízení. Model auta *Toyota Trueno Drift*[2] (obrázek 4.1) od firmy HPI Racing je jedním z nich a právě toto zařízení bylo zvoleno pro řešení této práce.

Model auta v základním vybavení je v pojízdném stavu, není tak potřeba řešit napájení nebo jiné technické detaily. Autíčko se primárně ovládá pomocí dálkového ovladače, ten však v této práci nebude použit. Místo něho se k autu připojí malý počítač *Raspberry Pi*, který obstará řízení motorů a bezdrátovou komunikaci s ovládací aplikací.

Obrázek 4.1: Model auta HPI Racing E10 Toyota Trueno Drift, převzato z [17].

Autíčko je schopné se pohybovat oběma směry, tedy vpřed i vzad. Pohon auta zajišťuje elektrický motor společně s modulem *ESC*[3], což je anglická zkratka pro jednotku řízení elektrických motorů. Modul *ESC* umožňuje motor ovládat jako klasické servo. Například pomocí PWM pulsů, které svou šířkou jsou schopny ovlivňovat rychlost a směr otáček motoru. Zatáčení předních kol zajišťuje již normální servomotor.

Pulsně šířková modulace

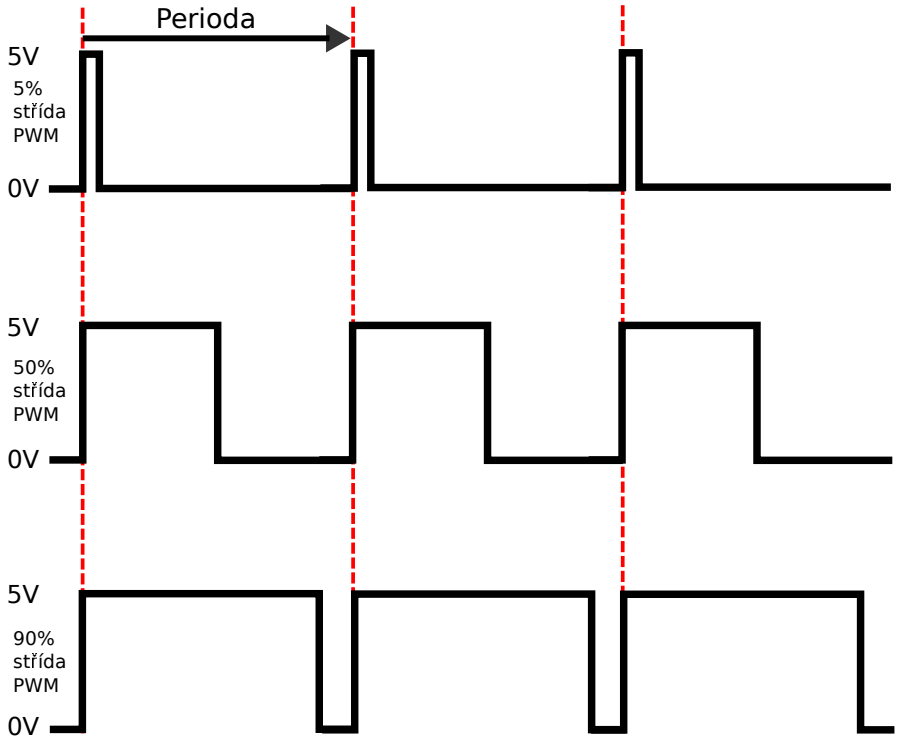
Pulsně šířková modulace *(anglicky Pulse-width modulation)* – dále jen PWM – je typ diskrétní modulace pro přenos analogového signálu pomocí dvouhodnotového signálu. Ačkoli tato technika modulace může být použita pro kódování informace při jejím přenosu, hlavní využití PWM spočívá v řízení velikosti napětí nebo proudu, jenž jsou dodávány do elektrických zařízení.

Přenosový signál nese informaci o přenášené hodnotě, která může nabývat hodnot zapnuto/vypnuto (logická 1/logická 0). Jde o signál s konstantní periodou, ve kterém se mění *střída*, což představuje poměr mezi délkou impulzu (logická 1) a délkou mezery (logická 0) v jedné periodě (viz obrázek 4.2). Střída se často vyjadřuje procentuálně, kde 100 % představuje poměr 1 : 0 *(délka impulzu:délka mezery)*, 50 % znamená poměr 1 : 1, atd. Střída se v zahraniční literatuře označuje termínem *Duty Cycle*.

[1] http://www.fit.vutbr.cz/research/groups/robo/

[2] http://www.modelarina.cz/rc-auto-hpi-racing-e10-toyota-trueno-drift-4x4-m110-rtr-p-44.html

[3] Electronic Speed Control



Obrázek 4.2: Princip pulsně šířkové modulace

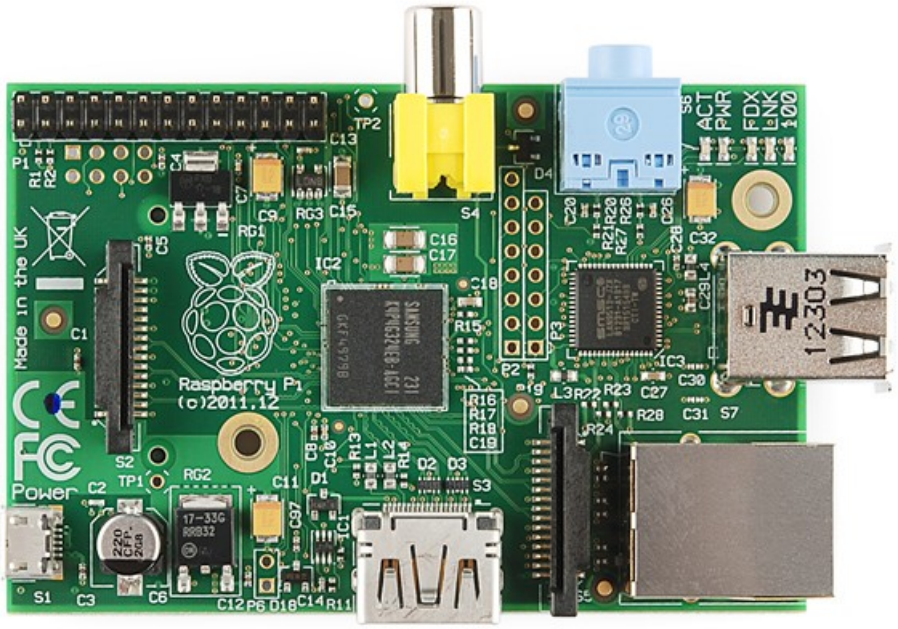
4.2.2 Raspberry Pi

Raspberry Pi[4] je velmi levný (verze B stojí 35 amerických dolarů) počítač o velikosti kreditní karty, který se dočkal již několika revizí a nových verzí. V této práci bude použit počítač první verze, model B, revize 1.0 (zobrazen na obrázku 4.3), který na své desce obsahuje CPU o frekvenci 700 MHz, RAM o velikosti 512 MB, dva USB porty nebo třeba piny pro obecné použití (GPIO). Pro spolehlivý běh deska vyžaduje napájení 5 V. Úložiště počítače je řešeno pomocí paměťových karet typu SD, pro které je připraven slot ve spodní části zařízení. Díky velké popularitě mezi počítačovými nadšenci existuje několik operačních systémů, které mohou na počítači běžet. Často se jedná o „ořezané“ verze oblíbených linuxových distribucí. Zařízení dokonce zvládne spustit grafické rozhraní operačního systému a přehrát například film ve vysokém rozlišení, pro účely této práce však grafické prostředí nebude potřeba zapínat. Celá specifikace zařízení je jednoduše dohledatelná na internetu.[5]

Tento jednodeskový počítač se umístí přímo na robotické autíčko a pomocí GPIO spojí se servomotory, které bude možno následně ovládat. Protože auto musí být mobilní a nesmí být limitováno žádným připojeným kabelem, komunikace mezi osobním počítačem uživatele (notebookem) a Raspberry Pi bude probíhat pomocí bezdrátové technologie *Wi-Fi*. Počítač Raspberry Pi lze však v základní výbavě připojit do počítačové sítě pouze drátově. Řešení tohoto problému spočívá v použití bezdrátového *Wi-Fi* adaptéru do USB (viz obrázek 4.4).

Pro napájení Raspberry Pi je připraveno 6 nabíjecích baterií typu AA s hodnotou $1,2\,V$. To v součtu představuje více než požadovaných $5\,V$, pro usměrnění na vhodnou hodnotu se

[4] http://www.raspberrypi.org/help/what-is-a-raspberry-pi/
[5] https://www.raspberrypi.org/



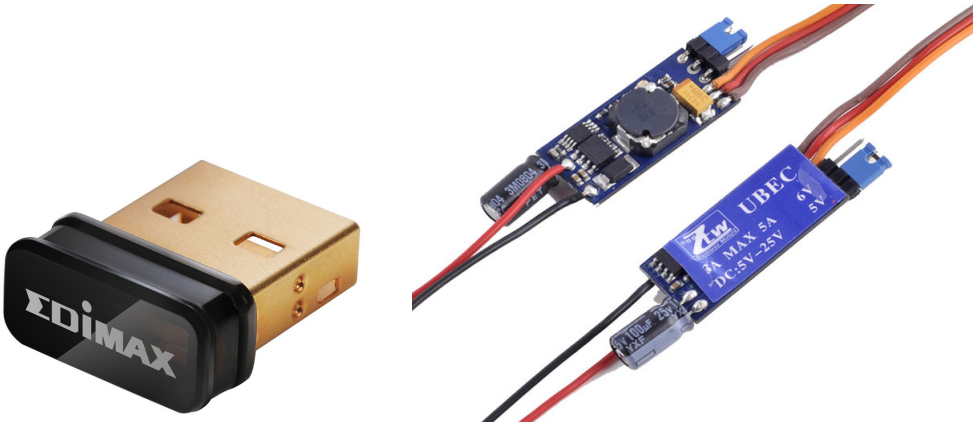
Obrázek 4.3: Počítač Raspberry Pi verze B, převzato z [21].

využívá zařízení *ZTW 3A 5-6v UBEC* (zobrazeno na obrázku 4.4), které bude nastaveno tak, aby pracovalo v režimu 5 *V*/3 *A*. Dodávané napětí na vstup Raspberry Pi se bude pohybovat maximálně na hodnotě 5 *V*, čímž nedojde k poškození počítače. Raspberry Pi je poměrně náročné na odběr proudu a je také citlivé na jakékoliv výkyvy napětí, a tak použití spínaného stabilizátoru UBEC se jeví jako ideální řešení.

4.2.3 Videokamera

Další zařízení potřebné pro funkčnost celého systému je videokamera, která se bude starat o snímání scény z ptačí perspektivy. Použití kamery se vyžaduje přímo ze zadání práce, proto se pouze rozhodovalo, jaký typ kamery bude využit. Nakonec byla vybrána běžně dostupná webová kamera. V práci se uvažuje základní nebo alespoň minimální osvětlení snímaného prostoru, aby mohla být uskutečněna v požadované míře detekce objektů ve videu. Noční nebo neosvětlené scény tak nejsou brány vůbec v potaz.

Výsledný systém by měl být schopný pracovat se všemi běžnými webovými kamerami, pro účely implementace a testování byla použita kamera od značky *Creative*, konkrétně produkt *Creative Live! Cam Sync HD* (viz obrázek 4.5), která svými parametry vyhovuje řešení problému. Toto zachytávací zařízení dokáže zaznamenávat video v HD, tedy v rozlišení 1280 × 720 obrazových bodů. Maximální rychlost snímání obrazu dosahuje hodnoty 30 snímků za vteřinu. Kamera se připojuje do počítače standardně přes masově rozšířené rozhraní USB.



Obrázek 4.4: Bezdrátový Wi-Fi adaptér do USB (vlevo), zařízení UBEC pro usměrnění napětí na $5\,V$ (vpravo), převzato z [18] a [19].

Obrázek 4.5: Kamera Creative Live! Cam Sync HD, převzato z [20].

4.3 Softwarové vybavení

Ovládání fyzické realizace projektu zajišťují zejména dva programy, které fungují v režimu *klient-server*. Program běžící na Raspberry Pi (server) čeká na připojení uživatelské aplikace (klienta), načež začne zpracovávat řídicí příkazy přicházející z programu klienta.

4.3.1 Program pro Raspberry Pi

Na tento program je kladeno několik zásadních požadavků. První představuje schopnost komunikovat pomocí protokolu TCP[6]. Více informací a důvod výběru právě tohoto protokolu se nachází v podkapitole 4.4. Protože na počítači Raspberry Pi běží operační systém založený na linuxovém jádru, lze pro realizaci TCP připojení využít mechanismu síťových *socketů*, který rovněž umožní vytvořit program vystupující v roli *klienta*.

Další požadavek se váže k ovládání elektrických servomotorů modelu autíčka pomocí PWM pulsů. Díky masovému rozšíření platformy Raspberry Pi existuje několik knihoven pro práci s piny obecného použití (GPIO). Černý konektor GPIO lze vidět vlevo nahoře

[6]TCP - Transmission Control Protocol

na obrázku 4.3. Pro tento projekt byla zvolena knihovna *pigpio*[7] napsaná v jazyce *C*, jež umožňuje jednoduchým způsobem ovládat piny obecného použití a generovat na ně softwarové PWM pulsy. Použití konkrétních funkcí z knihovny je popsáno v podkapitole 5.1.3.

Protože oba stěžejní požadavky lze efektivně řešit v jazyce *C* a realizovaný program nebude velkého rozsahu, bude pro implementaci použit právě jazyk vyvinutý *Kenem Thompsonem* a *Dennisem Ritchiem*.

4.3.2 Uživatelská aplikace

Pro uživatele asi nejzajímavější část celého projektu představuje aplikace, který poběží na osobním počítači (PC), popřípadě notebooku. Požadavky pro vytvářený program kladou důraz na jeho ovladatelnost, intuitivnost a jednoduchost, aby tak i neznalý uživatel byl po krátkém seznámení schopen ovládat celý systém.

Samozřejmostí je grafické uživatelské rozhraní, jehož hrubý návrh lze vidět na obrázku 4.6. Největší část prostoru by měl zabírat „přehrávač“, jenž bude zobrazovat videosekvenci se snímanou scénou. Program by měl zpracovávat vstupní záběr z webové kamery a detekovat v něm objekty dvojího druhu – ovládané auto a překážky. Výsledky detekce se následně barevně vyznačí ve snímaném obraze. Zadávání cílové pozice autíčka se váže na události myši. Přesun kurzoru myši posune se zamýšlenou cílovou polohou, tažením myši při stisknutém pravém tlačítku bude rotovat koncovou pozici auta. Levý klik myši zvolí aktuální polohu jako cílovou. Jakákoliv změna musí být ihned vykreslována do „prohlížeče“ tak, aby celá práce zadávání výsledné pozice nebyla matoucí. V hlavní části programu by měla být také vykreslena naplánovaná a vykonávaná cesta. Poměr stran zobrazování videozáběru musí odpovídat skutečnému poměru tak, aby nedocházelo k zkreslení. Velikost by se také měla poměrně měnit v závislosti na změně velikosti celého okna aplikace.

V pravé části aplikace se nachází panel s možnostmi nastavení celého programu. První skupina ovládacích prvků sdružuje možnosti TCP připojení k autíčku (Raspberry Pi). Uživateli musí být umožněno vložit IP adresu a port, na kterém běží server. Operace připojení nebo odpojení by měly být realizovány pomocí tlačítek. Další ovládací prvky by měly uživateli dopřát parametrizaci detekování objektů. Tato část je závislá na implementovaném chování detekce, jenž je popsáno v podkapitole 5.2.2.

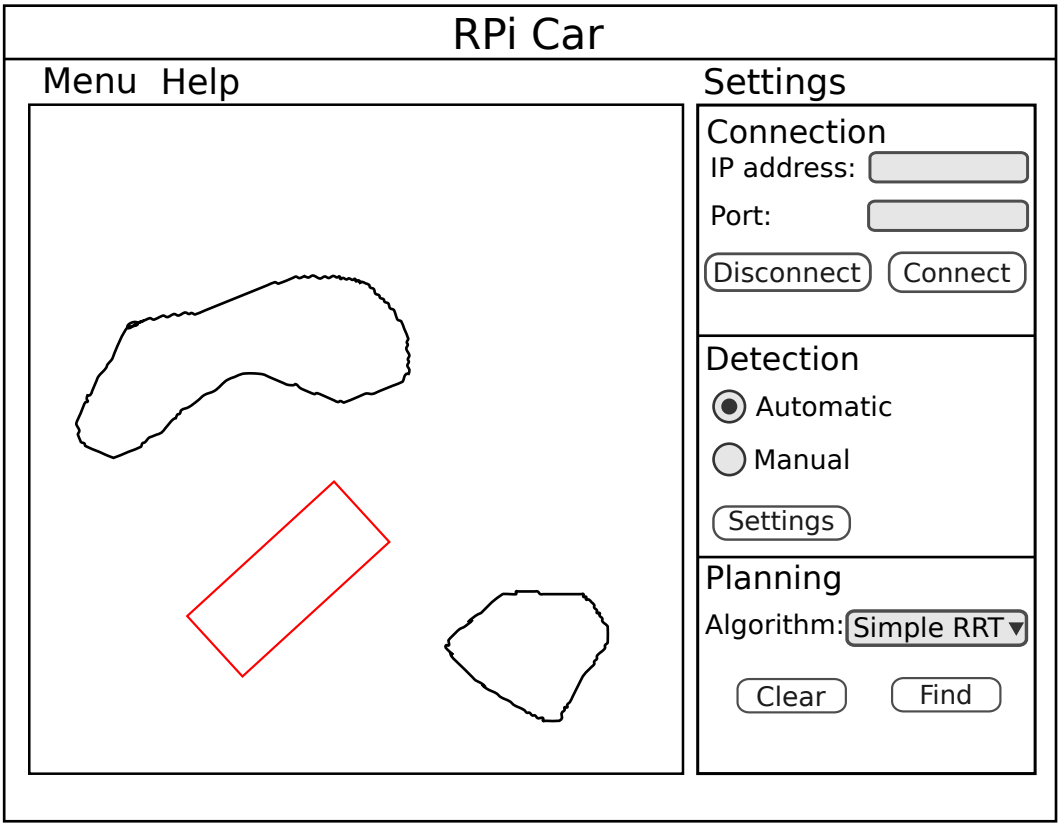
Zbytek nastavení představují ovládací prvky pro plánování a navigaci robotického autíčka. Pomocí rolovací nabídky může uživatel vybrat požadovaný algoritmus pro hledání cesty v prostoru. Tlačítky mohou být spuštěny operace hledání cesty, její následné smazání nebo zahájení (zastavení) navigace auta.

Způsob vytvoření popsané aplikace nebylo zadáním nijak limitováno, volba nástrojů tak čistě závisí na uvážení řešitele. Nebyl specifikován ani cílový operační systém uživatele. Aplikace však bude napsána tak, aby byla schopna běžet na OS *Linux* a *Microsoft Windows*. Jelikož při řízení autíčka bude velkou roli hrát celkové zpoždění systému, byl pro implementaci řídicí aplikace zvolen programovací jazyk *C++*, především díky svému výkonu, efektivitě a flexibilitě. Pro řešení problému detekce se velmi dobře hodí knihovna *OpenCV*[8], která nabízí aplikační rozhraní pro *C++* a splňuje podmínku přenositelnosti. Vytvoření programu s grafickým uživatelským rozhraním představuje již poměrně obsáhlý úkol a nevyužití objektově orientovaného přístupu by bylo nešťastným řešením. Z dostupných knihoven, jenž umožňují vytvořit GUI, byl vybrán multiplatformní *framework Qt*[9].

[7] http://abyz.co.uk/rpi/pigpio/

[8] knihovna Open Source Computer Vision, http://opencv.org

[9] http://www.qt.io



Obrázek 4.6: Prvotní návrh grafického uživatelského rozhraní hlavní aplikace

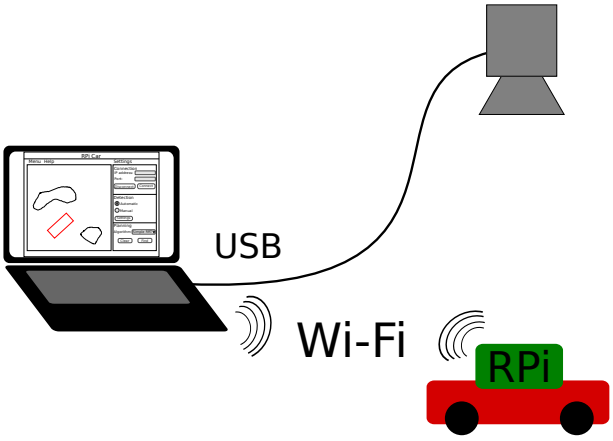
4.4 Komunikace mezi systémy

V předcházejících odstavcích byl představen návrh jednotlivých komponent. Aby výsledný systém byl funkční, je potřeba ustanovit způsob komunikace mezi subsystémy; její primitivní nákres je možné zhlédnout na obrázku 4.7.

Připojení mezi videokamerou a ovládacím počítačem bude je řešeno drátově pomocí rozhraní USB (standardní způsob připojení webových kamer). Moderní operační systémy by měly správně rozpoznat nově připojená extérní zařízení a pomocí technologie *plug-and-play* automaticky nainstalovat ovladače a zajistit spolehlivý provoz tohoto zařízení. V případě selhání tohoto kroku bude nutné nainstalovat ovladače pro videokameru manuálně.

Komunikace mezi osobním počítačem a Raspberry Pi by mohla být realizována některou z dostupných bezdrátových technologií. Pro tento projekt byla nakonec zvolena bezdrátová komunikace pomocí *Wi-Fi*, a to z důvodu dostupnosti této technologie na dnešních zařízeních výpočetní techniky. Notebooky jsou běžné vybaveny *Wi-Fi* adaptérem pro bezdrátovou komunikaci. Počítač Raspberry Pi musel být rozšířen extérním adaptérem připojitelným do USB (viz obrázek 4.4). Pro komunikaci skrze počítačovou síť byla zvolena architektura *klient-server*, jenž je objasněna dále v textu.

Následně je potřeba uvážit jakým způsobem bude komunikace probíhat, a tak v podkapitole 4.4.2 bude popsán protokol TCP, který bude využit pro komunikaci systémů přes počítačovou síť. Mezi hlavní důvody zvolení právě tohoto protokolu pro realizaci komunikace mezi řídicím programem a ovládaným autíčkem patří samotné vlastnosti protokolu



Obrázek 4.7: Ilustrace způsobu komunikace mezi řídicím programem běžícím na osobním počítači, kamerou a Raspberry Pi ovládající autíčko.

TCP. Garance doručení dat a také zachování pořadí těchto dat představují nutné předpoklady pro správné ovládání autíčka. Posílanými daty budou řídicí signály pro motory, které se jistě nemohou ztrácet nebo měnit své pořadí.

4.4.1 Architektura klient-server

Klient-server model je distribuovaná výpočetní struktura, která rozděluje zátěž mezi poskytovale služeb nazývající se *server* a žadatele služeb (*klienty*). Často klienti a servery komunikují přes počítačovou síť (každý běžící na vlastním hardwaru), existují však také varianty, kdy *server* i *klient* běží na stejném systému. *Klient* zahajuje komunikaci se *serverem* a žádá od něj obsah nebo nějakou funkci. *Server* čekající na připojení jednoho nebo více klientů (v závislosti na implementaci) zpracovává jeho požadavky a následně na ně odpovídá. Příkladem této architektury může být webový prohlížeč, email, tiskový server, atd.

V této práci bude použit model, kdy program běžící na Raspberry Pi bude představovat *server*, jenž čeká na připojení pouze jednoho *klienta*, což bude program s grafickým uživatelským rozhraním, který poběží na osobním počítači uživatele.

4.4.2 Transmission Control Protocol

Tímto anglickým termínem (dále jen TCP) je označován protokol transportní vrstvy v sadě protokolů *TCP/IP*[10], jenž se velmi často používá v síti *Internet*[11]. TCP protokol dopravuje data mezi konkrétními aplikacemi, protokol *IP* se stará o doručení dat mezi počítači. Použitím protokolu TCP mohou komunikující aplikace vytvořit mezi sebou spojení, přes které lze obousměrně přenášet data. Protokol TCP garantuje spolehlivé doručování se zachováním pořadí přenášených datových segmentů. Pro případ ztracení nebo poškození jsou segmenty číslovány a jsou opatřeny kontrolním součtem. Ochrana dat však slouží pouze proti technickým problémům, nikoli proti cíleným útokům. Pro takovou ochranu je potřeba použít některý zabezpečovací protokol [3].

[10] sada protokolů pro komunikaci v počítačové síti
[11] globální systém propojených počítačových sítích

Protože protokol TCP komunikuje na vrstvě mezi aplikacemi, musí je nějakým způsobem rozlišit. Tato identifikace se děje pomocí *čísla portu*, jenž je jednoznačně přiřazováno komunikujícím aplikacím. Porty mohou být číslovány hodnotou v rozsahu od 0 do 65535. Podle této informace operační systém pozná, které aplikaci má přenášený segment doručit. Podrobnější informace včetně diagramů TCP segmentů mohou být nalezeny například v knize [3].

Kapitola 5

Implementace

Následující řádky této kapitoly se věnují samotné implementaci a realizaci praktické části diplomové práce. Nejedná se o přepis zdrojových kódů, hlavním cílem je seznámení čtenáře s použitými technologiemi a se způsoby řešení daných problémů. Kapitola se dělí na dvě stěžejní podkapitoly, první se zabývá sestavením robotického autíčka, na kterou navazuje text zabývající se tvorbou řídicí aplikace běžící na počítači uživatele.

5.1 Sestavení robotického autíčka

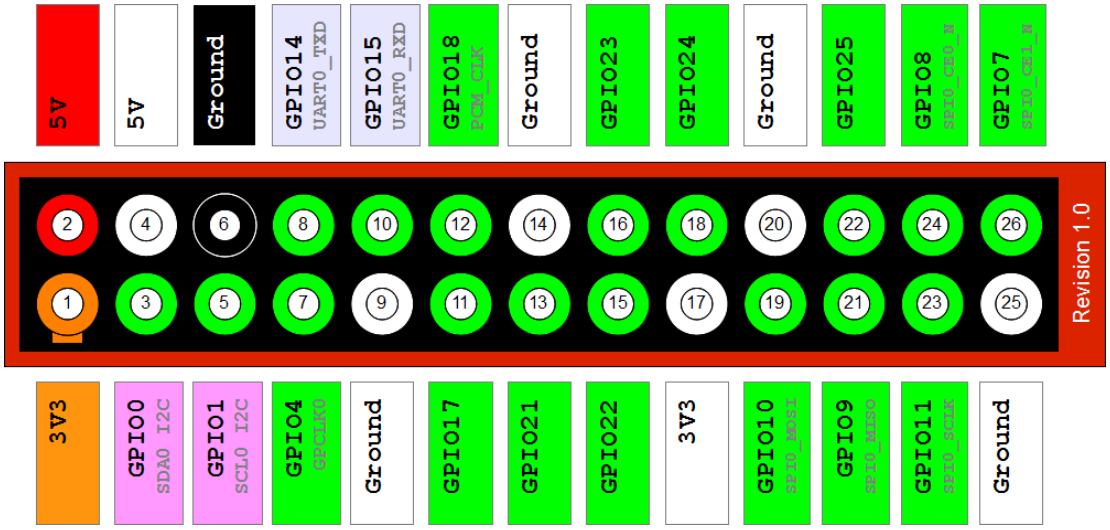
Podle návrhu popsaného v kapitole 4.2 má být řízený robot vytvořen z RC modelu auta *Toyota Trueno Drift* (na obrázku 4.1), jenž bude řídit počítač Raspberry Pi. Autíčko v pojízdném stavu potřebuje v podstatě jednu úpravu, a to připojení dvou motorků (pohon a zatáčení) k nové řídicí jednotce. Původně byly motory připojené k přijímači *HPI RF-1*, jenž komunikoval s dálkovým ovladačem a podle přijímaných signálů řídil oba motory. Autíčko obsahuje akumulátor o hodnotě $7,2\,V$, která dodává napájení celému RC modelu. Pro zapnutí nebo vypnutí robotického auta slouží dvoupolohový vypínač.

5.1.1 Zapojení Raspberry Pi

Připojení Raspberry Pi k napájení a oběma motorům je realizováno pomocí malého nepájivého pole a barevných vodičů o průměru převážně $0,5\,mm$. Jako zdroj napětí pro Raspberry Pi by bylo možné využít baterii autíčka a připojit čip k vodiči jednotky *ESC*, který dodává požadovaných $5\,V$. Toto napětí však není dostatečně stabilní pro bezproblémový běh Raspberry Pi, protože dochází ke kolísání napětí v závislosti na zátěži elektrického motoru pohánějící auto, což následně způsobuje restartování Raspberrry Pi. Tento zdroj napětí je však dostatečný k napájení servomotorku pro zatáčení, a tak *červené* žíly vodičů vycházející z obou motorů jsou spojeny. Stejně tak jsou spojeny *černé* vodiče vedoucí k motorům označované jako *zem* ($0\,V$). Zbývající drátky *(bílé)* obou třížilových vodičů se používají pro ovládání motorů a jsou přímo připojeny k pinům obecného použití na Raspberry Pi.

Číslování GPIO pinů se liší podle verze a revize Raspberry Pi a může být lehce matoucí. Tento projekt počítá s připojením servomotorku pro zatáčení k pinu označeným číslem 21 (13. pin fyzicky na konektoru) a *ESC* modulu k pinu č. 17 (fyzicky 11. pin)[1]. Na obrázku č. 5.1 je přehledně zobrazeno číslování konektoru GPIO.

[1] číslování pinů podle *Broadcom*, více informací viz http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/



Obrázek 5.1: Náhled na číslování GPIO konektoru na Raspberry Pi, model B, revize 1, převzato z [16].

Napájení Raspberry Pi je řešeno samostatně pomocí šesti „tužkových“ baterií typu AA. Ke zdroji napětí je zapojen dvoupolohový vypínač a následně spínaný stabilizátor napětí *UBEC*, který snižuje napětí na požadovaných $5\,V$. Zdroj vstupního napětí je připojen na piny 4 a 6 (pořadí fyzicky na konektoru). Celé schéma zapojení je možné zhlédnout na obrázku č. 5.2.

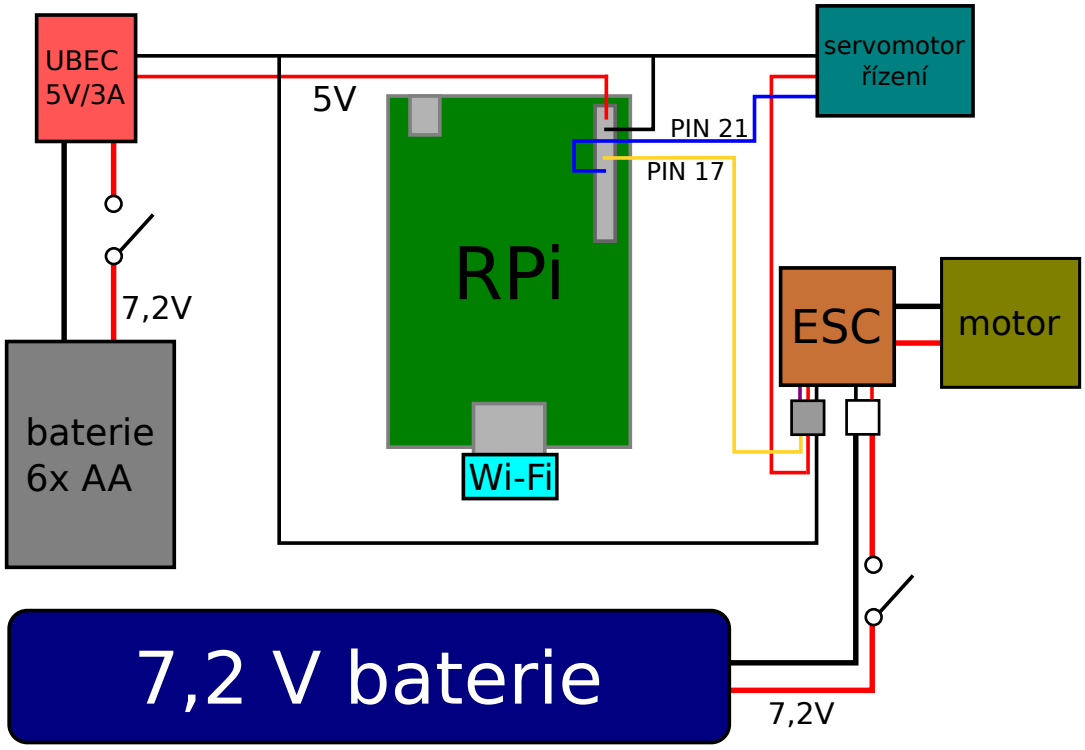
5.1.2 Příprava Raspberry Pi

Pro úspěšné spuštění počítače Raspberry Pi je potřeba zajistit operační systém, který řídí celý hardware. Tento jednodeskový počítač, jenž vyvíjí britská nadace *Raspberry Pi Foundantion*, má na své spodní straně slot pro SD paměťové karty, které slouží jako výchozí úložiště. Na kartu je nutné nakopírovat operační systém linuxového typu a zasunout ji do příslušného slotu. Po připojení napájení za předpokladu, že nenastaly žádné potíže, počítač Raspberry Pi začne zavádět a spouštět operační systém, což je indikováno blikáním malých LED diod na desce. Jednoduchou výměnou několika karet s různými verzemi systémů lze rychle uzpůsobit počítač pro vícero použití (multimediální centrum, webový server, řídicí jednotka . . .).

Pro účely tohoto projektu byla použita karta od výrobce *Kingston* o velikosti $4\,GB$, na kterou byl nakopírován operační systém *Raspbian*. Tento doporučovaný systém se zakládá na OS *Debian*[2] verze 7.6 s označením *wheezy*.

Správný běh počítače lze kontrolovat vizuálně, a to připojením monitoru přes rozhraní HDMI, nebo přihlášením přes protokol *SSH*[3], to však Raspberry Pi musí být připojeno do počítačové sítě. To bylo dosaženo připojením *Wi-Fi* adaptéru do portu USB a následnou konfigurací souboru `/etc/network/interfaces`, do kterého byly přidány řádky pro připojení k bezdrátové síti. Celý konfigurační soubor si lze prohlédnout ve výpisu 5.1.

[2] svobodný operační systém, více na https://www.debian.org/
[3] Secure Shell – zabezpečený komunikační protokol



Obrázek 5.2: Schéma zapojení Raspberry Pi a motorů

Výpis kódu 5.1: Konfigurační soubor `/etc/network/interfaces`

```
1  auto lo
2  iface lo inet loopback
3  iface eth0 inet dhcp
4
5  auto wlan0
6  allow-hotplug wlan0
7  iface wlan0 inet dhcp
8  wpa-ssid "nazev_site"
9  wpa-psk "heslo_site"
10 iface default inet dhcp
```

Další konfigurace počítače Raspberry Pi lze dosáhnout pomocí příkazu `raspi-config` (s právy administrátora). Při řešení tohoto projektu není potřeba spouštět grafické rozhraní operačního systému, a tak bylo zakázano právě přes zmíněný konfigurační nástroj.

5.1.3 Program pro Raspberry Pi

Vývoj programu určeného pro Raspberry Pi probíhal přímo na této platformě pomocí připojení přes protokol *SSH*. Díky kvalitnímu textového editoru *Vim* nebylo potřeba spouštět ani grafické rozhraní. Po dobu vývoje byl počítač připojen do počítačové sítě pomocí kabelu a napájení bylo realizováno standardním *micro USB* konektorem. Překladač *gcc* by

měl být nainstalovaný v základní verzi operačního systému *Raspbian*, a tak pro samotný vývoj programu v jazyce *C* nebylo potřeba dalších nástrojů.

Hlavní část zdrojového kódu se nachází v souboru `rpi-car.c`, ke kterému náleží hlavičkový soubor `rpi-car.h`. Pro síťovou komunikaci se využívá aplikační rozhraní knihovny *BSD sockets*[4]. Pro vytvoření programu v roli serveru, který v nekonečné smyčce čeká na připojení klienta, se používají funkce `getaddrinfo()`, `socket()`, `setsockopt()`, `bind()` a `accept()`. Podrobnější vysvětlení chování jednotlivých funkcí lze dohledat v manuálových stránkách. Po ustálení připojení s klientem jsou zprávy přijímány pomocí funkce `recv()` a následně odesílány pomocí `write()`. Program naslouchá na portu číslo 1234, tato hodnota je pevně definovaná na začátku zdrojového kódu.

Program očekává řídicí příkazy v přesně stanoveném formátu, jehož popis se nachází v podkapitole 5.2.5. Pro kontrolu a zpracování těchto příkazů se využívá vytvořená funkce `parseCommand()`. Po úspěšném provedení řídicího příkazu se hlavnímu programu běžícímu na PC posílá zpráva *"Done;"*, čímž je potvrzeno vykonání akce.

Podle přijatých příkazů jsou následně ovládány motory robotického autíčka. Pro tuto operaci se využívá knihovna *pigpio*[5], která umožňuje softwarově generovat PWM pulsy (více v odstavci 4.2.1) na libovolné piny konektoru GPIO. Knihovna je dostupná v podobě archivu se zdrojovými kódy, jenž jsou nachystány pro překlad a instalaci pomocí příkazů `make` a `make install`. Nejprve musí být zavolána inicializační funkce `gpioInitialise()` a následně mohou být jednotlivé piny nastaveny pomocí volání funkce `gpioSetMode(pin, PI_OUTPUT)`, kde `pin` představuje číslo pinu (pro řízení číslo 21, pro pohon číslo 17). Konstanta `PI_OUTPUT` zajistí, že pin se bude chovat jako výstupní, tedy že se na něj bude zapisovat hodnota.

Běžné servomotorky mohou být ovládány PWM pulsy s periodou $50\,Hz$ ($20\,ms$). Při nastavení *střídy* signálu na hodnotu $1,5\,ms$ jsou motorky v klidové pozici. Hranice představují hodnoty $1\,ms$ a $2\,ms$. Servomotor, jenž se stará o zatáčení auta, při těchto hodnotách dosahuje největšího natočení doleva, respektive doprava. Nastavené hraniční hodnoty pro pohonný motor znamenají maximální výkon směrem vpřed nebo vzad. Zápis hodnot na jednotlivé piny se děje pomocí funkce `gpioServo(cislo_pinu, hodnota)`.

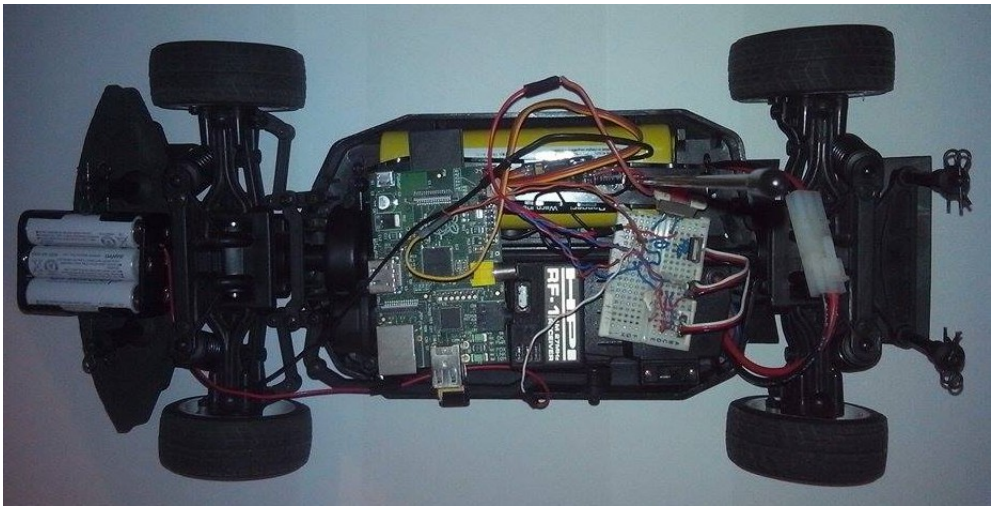
Knihovna *pigpio* poskytuje i jiný způsob ovládání pinů, a to skrze funkce `gpioPWM()`, `gpioSetPWMrange()` nebo `gpioSetPWMfrequency`. Více informací lze získat na internetových stránkách této knihovny[6]. Voláním funkce `gpioTerminate()` dojde k čistému způsobu ukončení práce s GPIO. Výsledný program musí běžet s právy administrátora, které jsou vyžadovány právě při práci s piny obecného použití.

Automatické spuštění

Aby došlo k okamžitému spuštění řídicího programu po naběhnutí operačního systému, je potřeba vytvořit spouštěcí skript, který se nakopíruje do adresáře `/etc/init.d/`. V tomto projektu byl vytvořen skript `/etc/init.d/rpi-car.sh`, jenž po startu systému zajistí start programu `/usr/sbin/rpi-car`, což je kopie vytvořeného spustitelného programu. Skript v adresáři `/etc/init.d/` umožňuje pohodlně program spouštět, zastavovat, nebo kontrolovat. Všechny akce lze vykonat pomocí příkazu:

```
service rpi-car.sh [start|stop|restart|status]
```

[4] BSD sockety – Berkeley sockety
[5] http://abyz.co.uk/rpi/pigpio/
[6] http://abyz.co.uk/rpi/pigpio/cif.html



Obrázek 5.3: Reálné připojení Raspberry Pi k autíčku pomocí nepájivého pole

5.2 Program pro PC

Kromě základních požadavků na aplikaci vycházející ze zadání práce byl přidán jeden doplňující, za to však poměrně důležitý. Aplikace pro uživatele by měla být napsána tak, aby byla přenositelná mezi různými operačními systémy. Z tohoto důvodu se výsledná aplikace skládá pouze z multiplatformních nástrojů, což umožní běh minimálně na operačních systémech *Linux* a *MS Windows*. Základ tvoří programovací jazyk *C++* spolu s knihovny *Qt* a *OpenCV*.

Protože se již jedná o program většího rozsahu, pro implementaci je využito objektově orientovaného přístupu, kde každá třída je zapsána v zdrojovém a hlavičkovém souboru. Jednotlivé zdrojové kódy jsou dále strukturovány do podadresářů podle svého účelu.

5.2.1 Grafické uživatelské rozhraní

Vytvořené grafické rozhraní je založeno na volně dostupné (pro nekomerční použití) knihovně *Qt (verze 5.4.1)*[7], jenž nabízí kvalitní způsob tvorby aplikací s GUI[8]. Důkazem mohou být programy jako *Google Earth*, *Skype* nebo *VLC media player*, které jsou postaveny právě na této knihovně. Napsaná je v jazyce *C++* a momentálně je pod záštitou společnosti *Qt Company*. Kromě aplikačního rozhraní a kvalitní dokumentace knihovny společnost nabízí také vývojové prostředí *Qt Creator*[9], ve kterém pomocí nástroje *Qt Designer* lze jednoduše tvořit vzhled aplikace pomocí „přetahování“ jednotlivých grafických prvků.

Podle požadavků v kapitole 4.3.2, jenž pojednává o návrhu grafického rozhraní aplikace a způsobu jejího ovládání, je celý systém zapsán v několika zdrojových souborech. Třída `MainWindow` se stará o chování hlavního okna aplikace a nachází se v souborech `mainwindow.cc`, `mainwindow.h` a `mainwindow.ui`. Soubory s koncovkou `.ui` generuje aplikace *Qt Designer* ve formátu *XML* a definují vzhled konkrétní části vyvíjeného programu. Implementace tříd `DetectionSettings`, `PlanningSettings` a `SettingsWidget` je zapsána

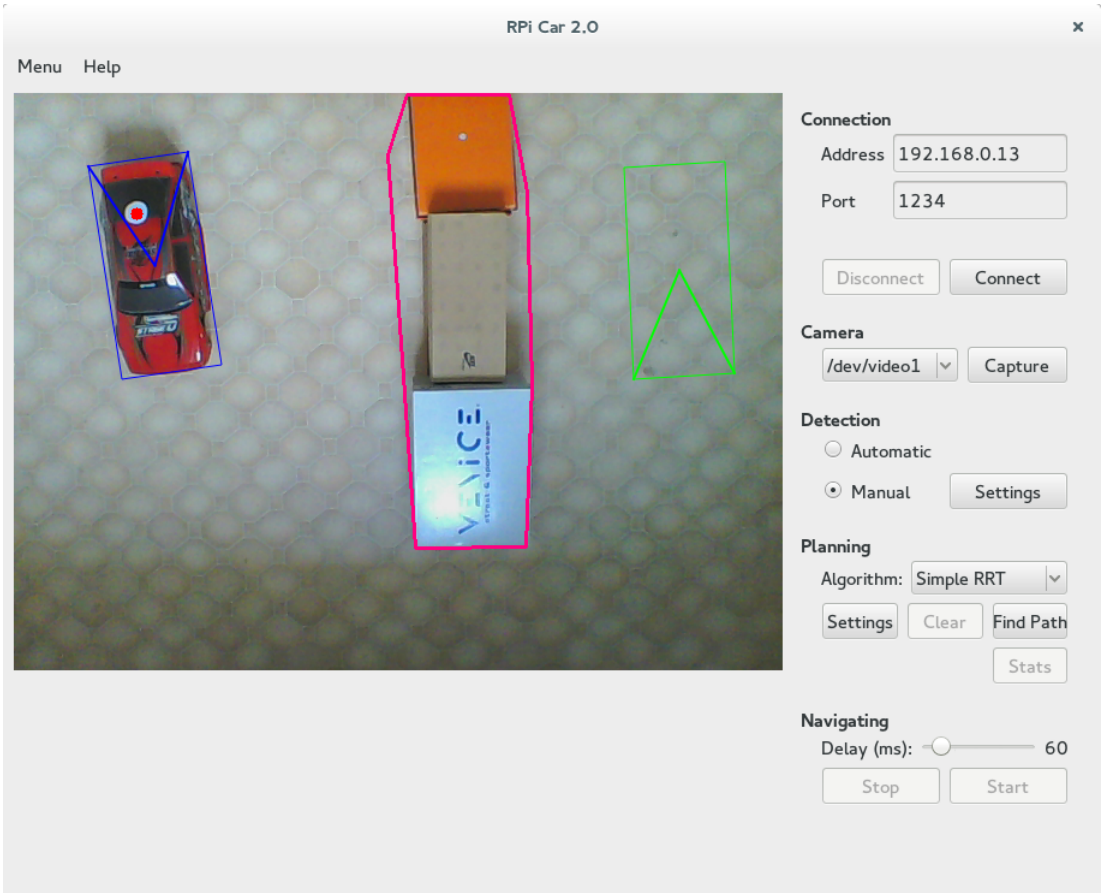
[7] http://www.qt.io/

[8] anglická zkratka pro grafické uživatelské rozhraní

[9] ke stažení na https://www.qt.io/download-open-source/

ve zdrojových kódech obdobně. Všechny tyto třídy se starají o zobrazení grafických prvků starajících se o různá nastavení. Zajímavější třída může být `ControllerWidget`, ve které se propojuje grafické ovládání s jádrem programu, které se stará o detekci v obraze, plánování cesty nebo navigaci auta. Třída je opět zapsána ve třech souborech: `controllerwidget.cc`, `controllerwidget.h` a `controllerwidget.ui`.

V této prakticé části není potřeba zdrojové kódy detailněji rozebírat, často se jedná pouze o správné použití tříd a metod z aplikačního rozhraní knihovny *Qt*. Náhled na vytvořenou aplikaci je možné nalézt na obrázku 5.4. Program umožňuje spustit v módu hledání cesty ve virtuálním prostoru, toto chování je zachyceno na obrázku 5.5.

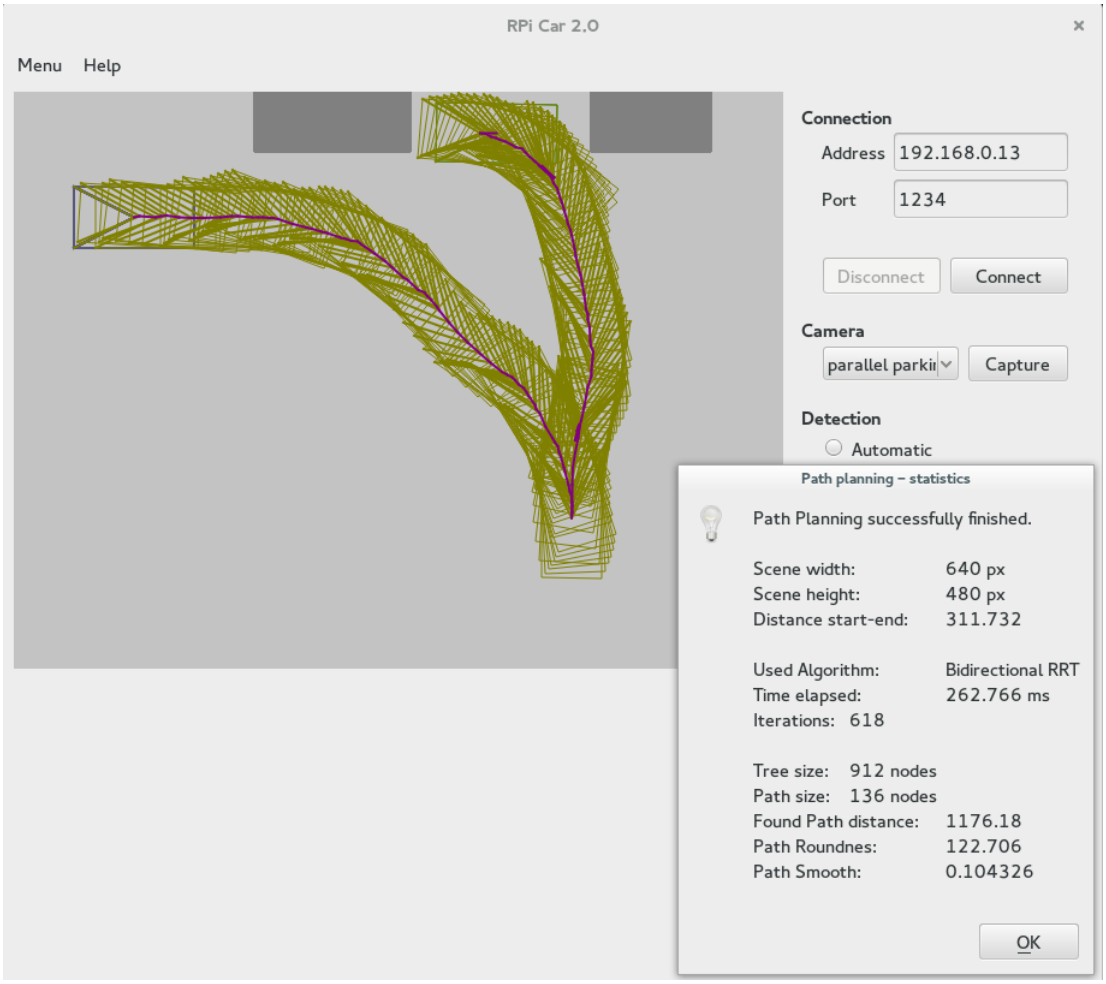


Obrázek 5.4: Náhled na grafické uživatelské rozhraní aplikace běžící v prostředí operačního systému *Linux*.

5.2.2 Detekce objektů v obrazu

O detekci objektů v obraze se stará třída `Detector`, zapsaná v souborech `detector.cc` a `detector.h`. Protože při samotném detekování probíhá neustále výpočetní proces, je potřeba metody této třídy spouštět ve vlastním vlákně, aby tak nebylo brzděno grafické rozhraní aplikace. Toto se děje implementací třídy `Detector` jako potomek třídy `QThread` z knihovny *Qt*.

Rozpoznávání autíčka a překážek ve snímané scéně je realizováno pomocí volně do-



Obrázek 5.5: Aplikace v módu s hledáním cesty ve virtuálním prostředí

stupné, populární knihovny *OpenCV*, která je napsaná v jazycích *C/C++*. Dokáže navíc využít výhod hardwarové akcelerace pro maximální výkon.

V aplikaci jsou dostupné dva módy detekce, automatická a manualní. V ručním módu detekce auta uživatel může upravovat hodnoty, které se uplatňují při segmentaci obrazu na základě HSV barev. Má také možnost upravit práh pro *Cannyho* hranový detektor, který se používá pro hledání překážek ve scéně. Automatická detekce nepotřebuje žádná speciální nastavení a měla by s určitou úspěšností fungovat pro dobré světelné podmínky.

Rozpoznání, zda-li se jedná o robotické autíčko nebo překážku, lze provést několika způsoby. Nejprve bylo navrhováno řešení s použitím klasifikátoru *SVM*, který by byl natrénován pozitivními a negativními vzorky. Nakonec však byla použita klasifikace na základě červené barvy auta společně s nalezením bílé kruhové značky, která se nachází nad zadní nápravou auta.

Zdrojový kód v souborech `detectedcar.cc` a `detectedcar.h` definuje třídu popisující vlastnosti detekovaného autíčka, které jsou dále používány pro plánování trasy. Mezi tyto vlastnosti patří například referenční bod (ve středu zadní nápravy auta), úhel natočení autíčka, délka, šířka apod. Jak je vidět v levé části obrázku 5.4, robotické autíčko je vyznačeno ohraničujícím obdélníkem. Tento zjednodušující předpoklad lze uplatnit, protože

prakticky nijak neomezuje následné plánování nebo navigaci auta. Během detekování a plánování cesty se ve zpracovávané videosekvenci může objevovat několik prvků, jejich seznam je uveden v tabulce 5.1.

prvek	zobrazení	spouštěcí akce
detekované auto	modrý obdélník	automaticky
referenční bod (střed zadní nápravy)	modrý bod	automaticky
cílová poloha auta	červený obdélník	najetí kursorem
rotace cílové polohy auta	červený obdélník	pravý klik a tažení myši
zvolená cílová poloha auta	zelený obdélník	levý klik
detekovaná překážka	zelený mnohoúhelník	automaticky
naplánovaná cesta	žluté obdélníky	tlačítko „Find path“

Tabulka 5.1: Přehled prvků objevujících se ve zpracovávané videosekvenci

5.2.3 Plánovací modul

Realizace části práce, která se stará o uložení konfigurace prostředí, se nachází ve třídě `Space`. Informace o rozložení objektů ve scéně se získávají od výše zmíněného detektoru. Samotné algoritmy hledající cestu v konfiguračním prostoru jsou zapsány ve třídě `Planner`. Podle teoretického rozboru v kapitole č. 3 byla implementována jednoduchá varianta RRT algoritmu s jedním stromem. V aplikaci je pojmenována jako *Simple RRT*. Verze se dvěma stromy a vyvažováním (*Bidirectional RRT*) je implementována také ve třídě `Planner`. Zdrojové a hlavičkové soubory se nazývají vždy podle názvu dané třídy. Pomocný modul `objects.cc` obsahuje třídy pro popis základních objektů ve scéně (auto a překážka).

Implementace obou algoritmů je téměř shodná s pseudokódy v kapitole 3.2, a tak není potřeba dalšího komentáře. Růst stromu a generování nových uzlů vyžaduje výpočty optimálních pohybových vektorů pro konrétní konfiguraci v prostoru. Podrobnější matematický popis lze nalézt v kapitole 3.3.2.

5.2.4 Navigace auta

Tento modul celého systému má na starosti provedení jízdy robotického autíčka po naplánované trase. Nalezená cesta pomocí plánovacích algoritmů existuje ve formě spojených uzlů stromu. Jednotlivé uzly jsou ve zdrojovém kódu reprezentovány třídou `Node`, která uchovává informace o poloze daného uzlu v konfiguračním prostoru a také hodnoty optimálního pohybového vektoru, jehož provedením se auto dostane do následující konfigurace.

Samotná navigace tak spočívá ve zpracování všech uzlů cesty a vygenerování řídicích příkazů pro robotické autíčko. Konstrukce řídicích příkazů je popsána v kapitole 5.2.5. Program běžící na Raspberry Pi dokáže rozpoznat přijaté příkazy a ovládat motorky pro pohon a řízení. V práci je implementován naivní způsob řízení autíčka, kdy na základě experimentů byla stanovena hodnota zpoždění mezi prováděním jednotlivých příkazů.

Systém navigace robota by potřeboval zpřesnit a vylepšit. Jeden ze způsobů zlepšení provádění jízdy auta by mohl spočívat v implementaci zpětné vazby skrze kameru, pomocí které by se kontrolovala aktuální poloha robota a podle odchylky od naplánované trasy by docházelo k posílání zpřesňujících navigačních příkazů.

5.2.5 Síťová komunikace

O propojení aplikace s robotickým autíčkem přes počítačovou síť se stará třída `Client`, jenž využívá pro navázání spojení třídu `QTcpSocket` z knihovny *Qt*. Aplikace vystupuje v síťové komunikaci v roli klienta a zahajuje připojení na zadanou *IP adresu* a *číslo portu*. Při řešení tohoto projektu se nepoužívá statická *IP adresa* pro robotické autíčko (Raspberry Pi), naopak adresa je dynamicky přidělována v závislosti na nastavení *DHCP serveru*[10], jenž se používá v počítačových sítích pro automatickou konfiguraci připojených zařízení. Pro připojení aplikace k autíčku je potřeba zjistit aktuálně přidělenou *IP adresu* Raspberry Pi, což může být dosaženo několika způsoby: webové rozhraní routeru, nástroji pro analýzu sítí, apod.

Číslo portu je dáno implementací programu na Raspberry Pi a je pevně nastaveno na hodnotu 1234 (definováno ve zdrojovém souboru `rpi-car.c`). Zdrojový kód třídy `Client` se nachází v souborech `client.cc` a `client.h`.

Řídicí příkazy

Analogicky k lidské řeči je potřeba i pro neživé přístroje definovat význam zpráv a příkazů, které si mezi sebou při komunikaci budou vyměňovat. Pro účely tohoto projektu byl zvolen následující formát zpráv. Klientská aplikace řídí motory autíčka (serveru) zprávou o délce 7 znaků ve formátu:

$$C;DDDD.$$

- znak C představuje pokyn pro pohon auta a může nabývat hodnot:
  - F – pohyb dopředu
  - B – pohyb dozadu
  - S – zastavení pohybu
- znak středníku ; je povinným oddělovačem
- přesně 4 znaky D znamenají celé číslo z intervalu $\langle 1200, 1800 \rangle$, které udává střídu PWM pulsu ovládající natočení kol
- tečka . je povinným ukončovacím znakem

Raspberry Pi umí zprávy tohoto typu zpracovat a následně vykonat. Jakkékoliv porušení výše zapsaných pravidel vede k zahození zprávy a přeskočení příkazu. Klientská aplikace zprávy vytváří a odesílá skrze své komunikační rozhraní. Několik názorných řídicích příkazů je zapsáno v tabulce 5.2. Program běžící na Raspberry Pi posílá po každém provedení příkazu zprávu ve formátu "*Done;*", čímž potvrzuje splnění akce.

příkaz	**význam**
F;1500.	jízda dopředu, žádné natočení kol
F;1650.	jízda dopředu, natočení kol doleva na polovinu max. výchylky
B;1800.	jízda dozadu, max. natočení doleva
S;1200.	pohyb zastaven, max. natočení doprava

Tabulka 5.2: Příklady příkazů pro řídicí program na Raspberry Pi a jejich význam

[10] Dynamic Host Configuration Protocol

Kapitola 6

Testování a vyhodnocení

Následující řádky této kapitoly se věnují testování a zejména vyhodnocení celého systému. Nejdříve je popsána evaluace detekce objektů v obraze, za kterou následuje zhodnocení plánovacího algoritmu. Předposlední část se zabývá vyhodnocením jízdy auta po naplánované trase. V závěru se nachází slovní popis způsobu testovaní systému v průběhu jeho vývoje.

6.1 Vyhodnocení detekce objektů v obraze

V zadaní práce se píše, že otestování úspěšnosti detekce robota a překážek má proběhnout v závislosti na barvě podlahy, barvě překážek nebo barevných značek umístěných na robotovi. Celá práce se omezuje na případ, kdy se na středu zadní nápravy robotického autíčka nachází barevná značka, konkrétně bílý bod kruhového tvaru, který je důležitým referenčním bodem při plánovaní trasy auta. Toto omezení není v rozporu se zadaním práce, vyhodnocení barevných značek však nebude dále provedeno. Výběr vhodného tvaru a barvy značky proběhl na základě mnoha experimentů, které však nebyly dokumentovány. Původně uvažovaný žlutý kříž byl nakonec nahrazen právě bílým bodem, protože dosahoval lepších výsledků. Obě značky lze vizuálně porovnat na obrázku 6.1.

Obrázek 6.1: Značky na kapotě auta určující střed zadní nápravy, nakonec byla použita varianta s bílým bodem (vlevo).

6.1.1 Princip detekce

Samotné rozpoznání objektů ve scéně se dá rozdělit do dvou kroků. První představuje detekci tvaru objektu, na kterou navazuje binární klasifikace, zda se jedná o auto nebo překážku. Vyhodnocení detekce oblastí objektů není zcela triviální a pro úspěšnou realizaci by bylo potřeba detailní popis referenční scény, podle které by se určovalo, do jaké míry detekce funguje. Kontrola hledaných hranic objektů byla provedena pouze vizuálně. Je potřeba zmínit, že detekce tvaru auta je poměrně přesná, neboť se používá segmentace podle barev po převodu obrazu do *HSV* modelu barev. Detekce nebude příliš fungovat, pokud se bude autíčko nacházet na červené podlaze, v ostatních případech lze hovořit o spolehlivé metodě detekce, dokonce i při změně osvětlení.

Překážky nelze detekovat na základě barev, protože jejich barva není předem známá. Princip detekce překážek tak spočívá v použití *Cannyho* hranového detekoru, který nalezne hranice objektů. Tato metoda trpí při větším výskytu šumu v obraze nebo při změnách jasu a stínů.

Rozdělení objektů na překážky a auto je v následujících odstavcích vyhodnoceno pomocí standardních metod pro hodnocení binárních klasifikátorů. Evaluace automatického módu detekce (v aplikaci je dostupná ještě manuální) probíhala pomocí dvou experimentů. Při detekování objektů bylo uloženo 50 konkrétních snímků videa, ve kterých byly barevně vyznačeny nalezené předměty. Při každém experimentu došlo ke čtyřem opakováním, při kterých se měnila poloha robotického autíčka. Celkem bylo tedy vygenerováno 200 snímků, které byly manuálně procházeny a ohodnoceny.

Vyhodnocení detekce objektů proběhne formou dvou experimentů, které se liší typem podlahy a také barvou, počtem a tvarem překážek. Matematické vyjádření úspěšnosti klasifikace objektů spočívá ve výpočtu následujících metrik. Některé názvy byly ponechány v angličtině, protože se pro ně české překlady příliš nepoužívají.

- *True positive (TP) – hit* = správně klasifikováno jako pozitivní
- *False positive (FP) – false alarm* = chybně klasifikováno jako pozitivní
- *True negative (TN)* = správně klasifikováno jako negativní
- *False negative (FN) – miss* = chybně klasifikováno jako negativní
- *True positive rate (TPR) – Recall* = míra skutečně pozitivních
  $TPR = \frac{TP}{TP + FN}$
- *False negative rate (FNR)* = míra falešně pozitivních
  $FNR = \frac{FN}{TP + FN}$
- *Precision*
  míra správně pozitivně klasifikovaných ku všem pozitivním vzorkům
  $P = \frac{TP}{TP + FP}$

6.1.2 Experiment č. 1

Jak je vidět na obrázku 6.2, pro první experiment byla použita světlá podlaha. Ve scéně bylo přítomné robotické autíčko společně se čtyřmi různými překážkami. Pro nasvícení prostoru bylo použito umělého osvětlení (běžná žárovka). Kontrast mezi všemi objekty byl poměrně výrazný, dá se proto říci, že se jedná o ideální podmínky pro detekci.

Vyhodnocení klasifikace auta

Z tabulky 6.1 lze vyčíst, že ani jednou nenastala situace *False Positive*, tedy že by nějaká reálná překážka byla označena jako autíčko. Toto se následně projeví v průměrované hodnotě *Precision*, která je shodná s hodnotou *Recall*.

Ve $13,5\,\%$ případů se však stalo, že objekt byl chybně klasifikován jako překážka, i když se ve skutečnosti jednalo o auto – *False negative rate*. Opačná hodnota představuje, jak často bylo autíčko správně detekováno, při experimentu č. 1 detekce probíhala s úspěšností $86,5\,\%$. Výsledky jsou souhrně zobrazeny v tabulce 6.2.

	True	**False**
Positive	173	0
Negative	0	27

Tabulka 6.1: Vyhodnocení klasifikace auta pro 200 náhodných snímků

True positive rate (Recall)	$0,865$
False negative rate	$0,135$
Precision	$0,865$

Tabulka 6.2: Vyhodnocení klasifikace auta, průměrné hodnoty pro 200 snímků

Vyhodnocení klasifikace překážek

Na stejném principu probíhalo vyhodocení detekce překážek v prostoru. Je třeba mít na paměti, že tabulky 6.3 a 6.4 obsahují výsledky evaluace klasifikace překážek. Samotná detekce tvarů překážek není popsána žádnou matematickou hodnotou. Lze však vidět, že pokud se podaří detekce hran objektu, tak následné zařazení do správné skupiny (mezi překážky) se děje již s 95% přesností.

	True	**False**
Positive	758	31
Negative	0	42

Tabulka 6.3: Exp. 1: Vyhodnocení klasifikace překážek pro 200 náhodných snímků

True positive rate (Recall)	$0,948$
False negative rate	$0,052$
Precision	$0,968$

Tabulka 6.4: Exp. 1: Vyhodnocení klasifikace překážek, průměrné hodnoty pro 200 snímků

6.1.3 Experiment č. 2

Na obrázku 6.3 si lze prohlédnout scénu zachycenou při provádění experimentu č. 2. Je patrné, že byla použita tmavší podlaha než v prvním případě. Rozmístění a tvar překážek se také odlišuje. Druhý experiment byl prováděn při horším osvětlení, což se projevilo na jeho výsledcích.

Vyhodnocení klasifikace auta

Jak je vidět v tabulce č. 6.5, při experimentu č. 2 nastala situace *False Positive*, a to celkem šestkrát ze 200 vzorků. Tato hodnota určuje stav, kdy červená překážka ve scéně byla chybně označena za hledané autíčko. V tomto prováděném experimentu také nastávalo daleko častěji *False Negative*, tedy situaci kdy reálný robot ve scéně byl chybně označen jako překážka. Podle tabulky 6.6 úspěšnost klasifikátoru při daných podmínkách dosahoval hodnoty asi 79 %.

	True	**False**
Positive	157	6
Negative	0	43

Tabulka 6.5: Exp. 2: Vyhodnocení klasifikace auta pro 200 náhodných snímků

True positive rate (Recall)	0, 785
False negative rate	0, 215
Precision	0, 785

Tabulka 6.6: Exp. 2: Vyhodnocení klasifikace auta, průměrné hodnoty pro 200 snímků

Vyhodnocení klasifikace překážek

Klasifikace překážek zaznamenala také zhoršení výsledků, podobně jako rozpoznávání autíčka. Opět se projevilo horší osvětlení scény, zejména při detekci červené překážky na tmavém podkladu. Obálka kolem objektu nebyla přesně vyznačena a reálná překážka byla zobrazována jako více malých objektů, což se v tabulce 6.7 projevilo ve vyšším počtu hodnoty *False Negative*. Stav, kdy auto ve scéně je označeno jako překážka, představuje hodnota *False Positive*. V tabulce 6.8 jsou uvedeny průměrné hodnoty některých metrik pro 200 testovacích vzorků.

	True	**False**
Positive	904	42
Negative	0	96

Tabulka 6.7: Exp. 2: Vyhodnocení klasifikace překážek pro 200 náhodných snímků

True positive rate (Recall)	0, 904
False negative rate	0, 096
Precision	0, 956

Tabulka 6.8: Exp. 2: Vyhodnocení klasifikace překážek, průměrné hodnoty pro 200 snímků

Obrázek 6.2: Zobrazení scény při experimentu č. 1

Obrázek 6.3: Zobrazení scény při experimentu č. 2

6.2 Vyhodnocení plánovacího algoritmu

V této podkapitole se nachází zhodnocení vytvořeného plánovacího algoritmu. Aplikace obsauje dva pravděpodobnostní algoritmy. Jednoduchá varianta RRT s jedním stromem je v programu označena jako *Simple RRT*. Přestože metoda dokáže naplánovat cestu i mezi překážkami, její využití je vhodnější pouze pro nepříliš složité konfigurační prostředí. Druhým implementovaným algoritmem je vyvažovaný RRT s dvěma stromy (aplikaci nazvaný *Bidirectional RRT*). Podrobnější popis obou algoritmů se nachází v kapitole 3.

Varianta *Simple RRT* dosahuje v porovnání s *Bidirectional RRT* lepších výsledků pouze při hledání přímých krátkých cest. Naopak *Bidirectional RRT* najde řešení rychleji a lépe pro složitější problémy, při kterých první varianta selhává.

6.2.1 Metriky

Následující metriky byly vybrány pro vyhodnocení plánovacích algoritmů:

- t – doba výpočtu algoritmu (v celé praci uvedeno v milisekundách)
- n – počet uzlů cesty – uzel cesty představuje konkrétní polohu auta, kterou se má projet (žluté obdélníky)
- d – délka naplánované cesty (v pixelech) – součet vzdáleností referenčních bodů jednotlivých uzlů cesty (spojnice bodů vyznačena v názorných obrázcích fialově)

$$d = \sum_{i=1}^{n-1} dist(m_i, m_{i+1})$$

kde $dist(m_i, m_{i+1})$ je vzdálenost mezi referenčními body pro dva na sebe navazující uzly m_i a m_{i+1}; n je celkový počet uzlů

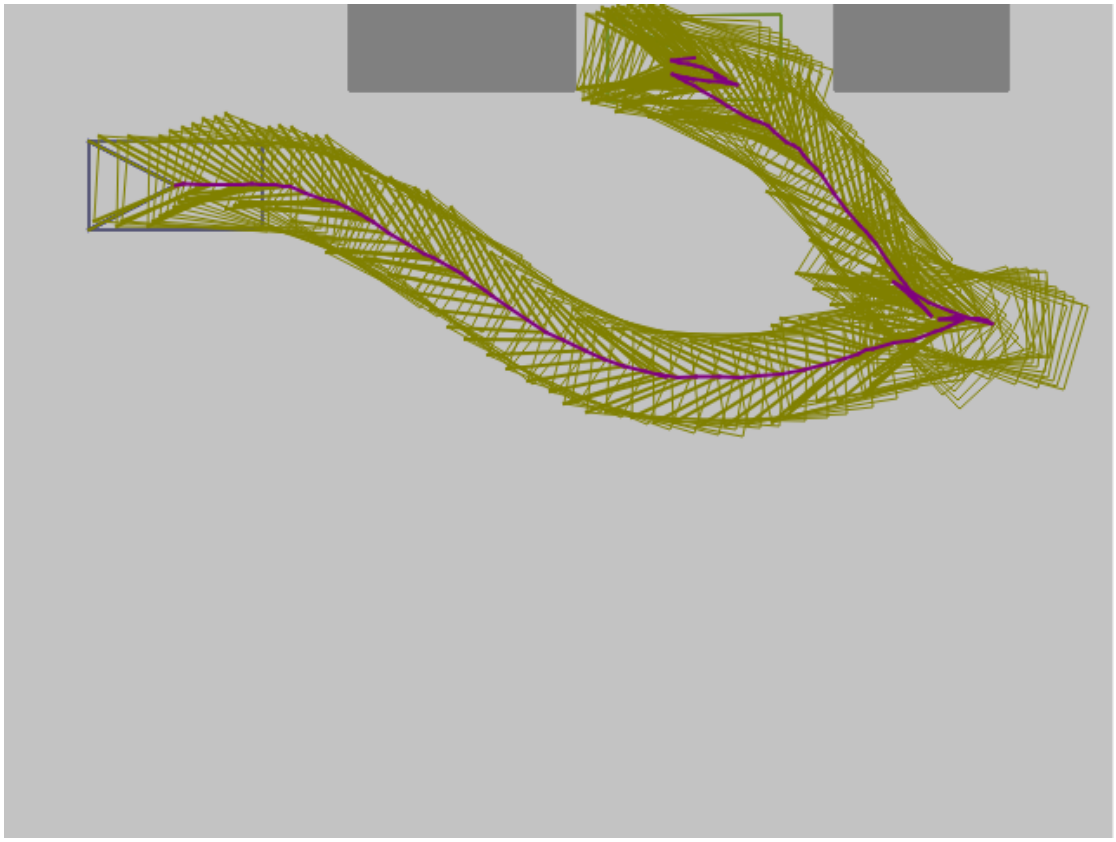
6.2.2 Paralelní parkování ve virtuálním prostředí

V této kapitole je ukázán výsledek plánování cesty pro robotické autíčko při problému paralelního parkování ve virtuálním prostředí. Pro nalezení trasy byl vybrán algoritmus *Bidirectional RRT*, vstupní parametry algoritmu a vlastnosti autíčka jsou zapsány v tabulce 6.9

Hodnota *Max.* ϕ představuje maximální natočení předních kol autíčka, hodnota *dt* udává „krok“ pohonu auta. *Ukončující vzdálenost* se používá k ukončení výpočtu algoritmu, tedy jaká vzdálenost od koncového (hledaného) místa je uspokojivá pro skončení výpočtu. *„Vzdušná“ vzdálenost start-cíl* udává přímou vzdálenost referenčních bodů mezi počáteční a koncovou polohou, pro uživatele má pouze informační účel.

Algoritmus	*Bidirectional RRT*
Rozměry scény	$640 \times 480\,px$
„Vzdušná“ vzdálenost start-cíl	$311,73\,px$
Max. ϕ	$25\,^\circ$
Krok (dt)	$10\,px$
Ukončující vzdálenost	$5\,px$

Tabulka 6.9: Vlastnosti algoritmu a zkoumané scény



Obrázek 6.4: Ukázka plánování cesty pro problém paralelního parkování (výřez z aplikace); počátek cesty vlevo, cílová pozice mezi šedými překážkami

Hledání cesty bylo 100krát opakováno se stejným nastavením a jedno řešení lze pozorovat na obrázku 6.4. Zobrazená cesta byla vybrána „pseudo-náhodně“ (zvoleno bylo jedno z lepších řešení). Nejedná se ani o nejkratší, ani o nejdelší cestu. Tabulka 6.10 ukazuje výsledky algoritmu získané po 100 pokusech.

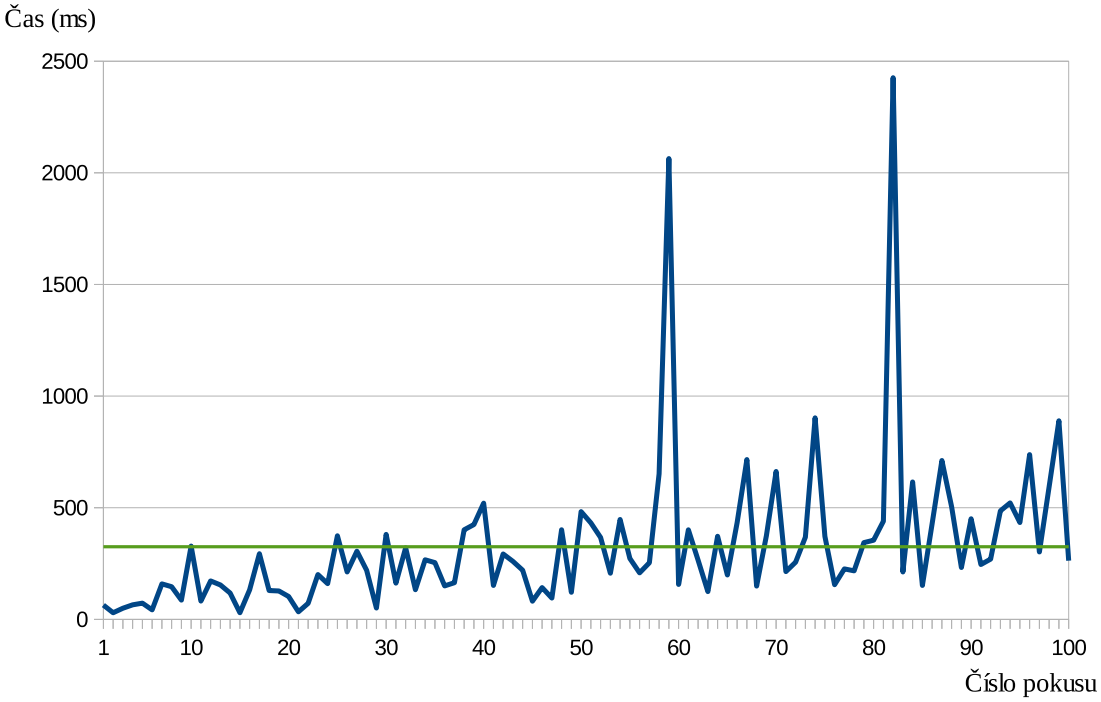
	Doba běhu (ms)	**Délka cesty (px)**	**Počet uzlů cesty**
Průměr	$325,15$	$1245,87$	$140,64$
Minimum	$29,37$	$476,5$	52
Maximum	$2426,1$	$2449,44$	268

Počet nenalezených řešení	2

Tabulka 6.10: Výsledné hodnoty metrik pro 100 výpočtů algoritmu

Získané výsledné hodnoty pro hledání cesty v prostředí při paralelním parkování jsou také vyneseny do grafické podoby. Graf 6.5 zobrazuje dobu trvání algoritmu pro jednotlivá měření. Dva vysoké „zuby“ představují dva pokusy, ve kterých algoritmus nedokázal nalézt cestu. Zelená čára v grafu představuje průměrnou dobu trvání pro 100 pokusů o hodnotě asi $325,15\,ms$.

Zbylé dva grafy mají podobný charakter, protože oba zobrazují určitým způsobem na-



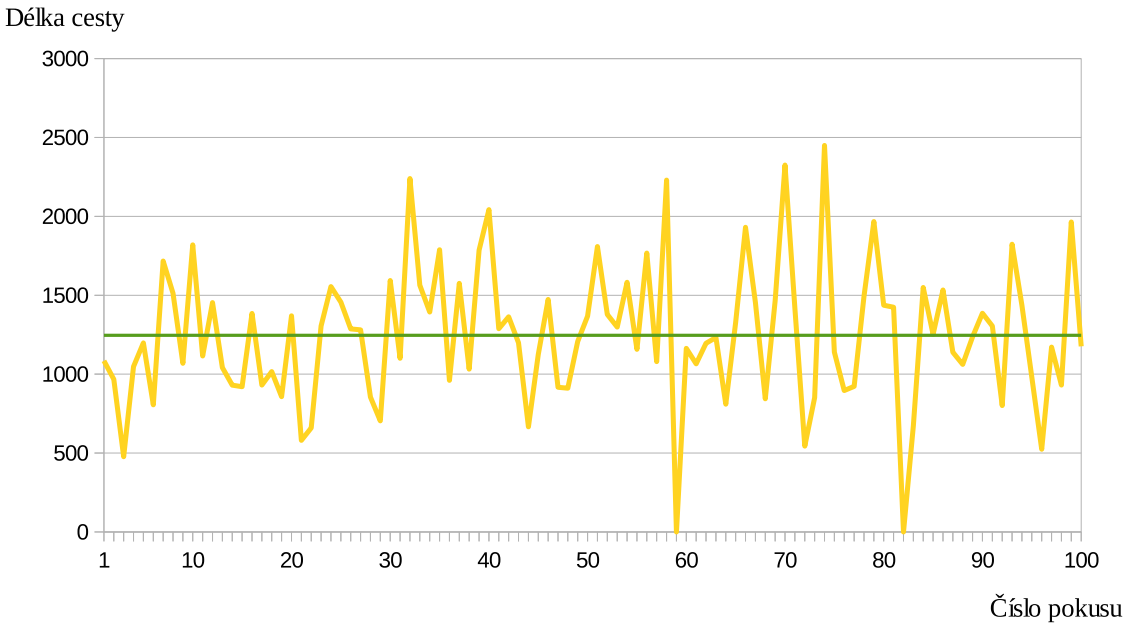
Obrázek 6.5: Vynesení *doby běhu algoritmu* do grafu pro 100 náhodných spuštění, zelená linka představuje průměrnou hodnotu.

lezenou cestu. V grafu 6.6 jsou vyneseny hodnoty délky nalezených cest, graf 6.7 obsahuje informaci o počtu uzlů výsledných cest. Lze pozorovat, že hodnoty délek cesty jsou asi $9,15$krát vyšší v porovnání s hodnotami počtů uzlů cesty. Tento poměr je dán přibližně „krokem“ pohonu auta, který byl nastaven na hodnotu 10 (viz tabulka 6.9). V obou grafech se dvakrát vyskytuje hodnota 0, což představuje neúspěšné hledání cesty. Zelená linka určuje průměrnou hodnotu všech pokusů.

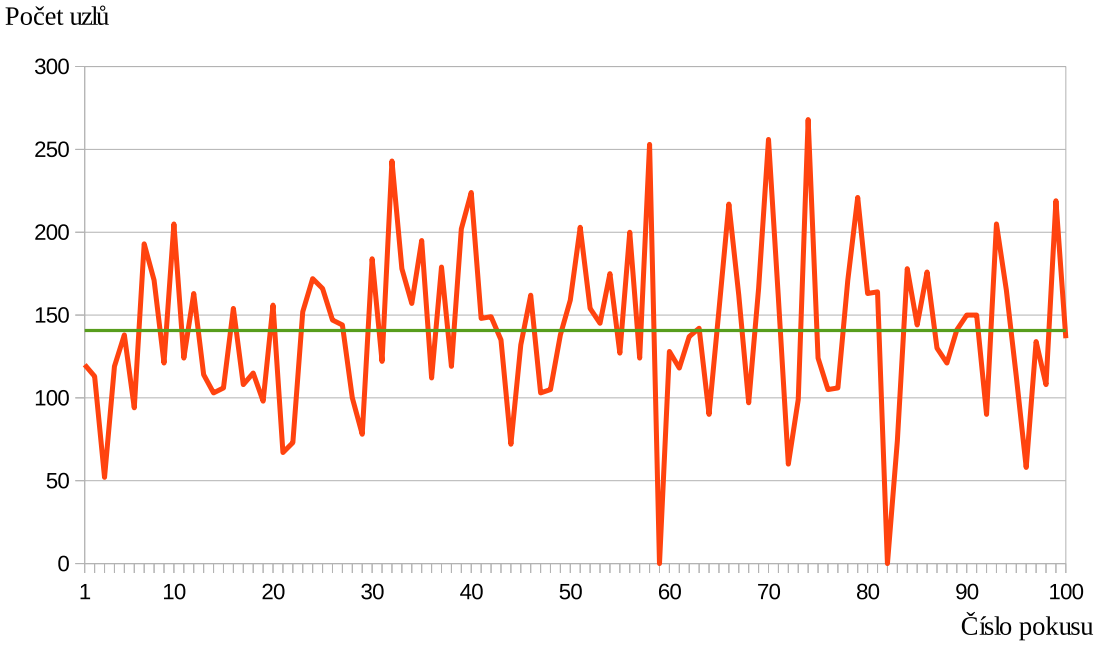
6.2.3 Více případových studií

Vyhodnocení plánovacích algoritmů probíhalo podobným způsobem jako v předcházející podkapitole, a to jak pro virtuální problémy, tak i pro reálné prostředí. Protože se jedná převážně o obrázky a výsledné tabulky, byla tato část zařazena do přílohy C diplomové práce, která se nachází v samotném závěru.

Postupně je ve virtuálním prostředí ukázán problém *paralelního parkování* (viz C.1) a *otočení auta a couvání mezi překážky* (C.2). Z reálných problémů bylo pro demonstraci vybráno řešení *objetí dlouhé překážky* (C.3) a nalezení cesty *mezi překážkami s následným couváním* (C.4). Všechny experimenty používají algoritmus *Bidirectional RRT*, protože se více hodí při hledání cesty ve složitějším prostředí.



Obrázek 6.6: Vynesení *délky naplánované cesty* do grafu pro 100 náhodných spuštění, zelená linka představuje průměrnou hodnotu.



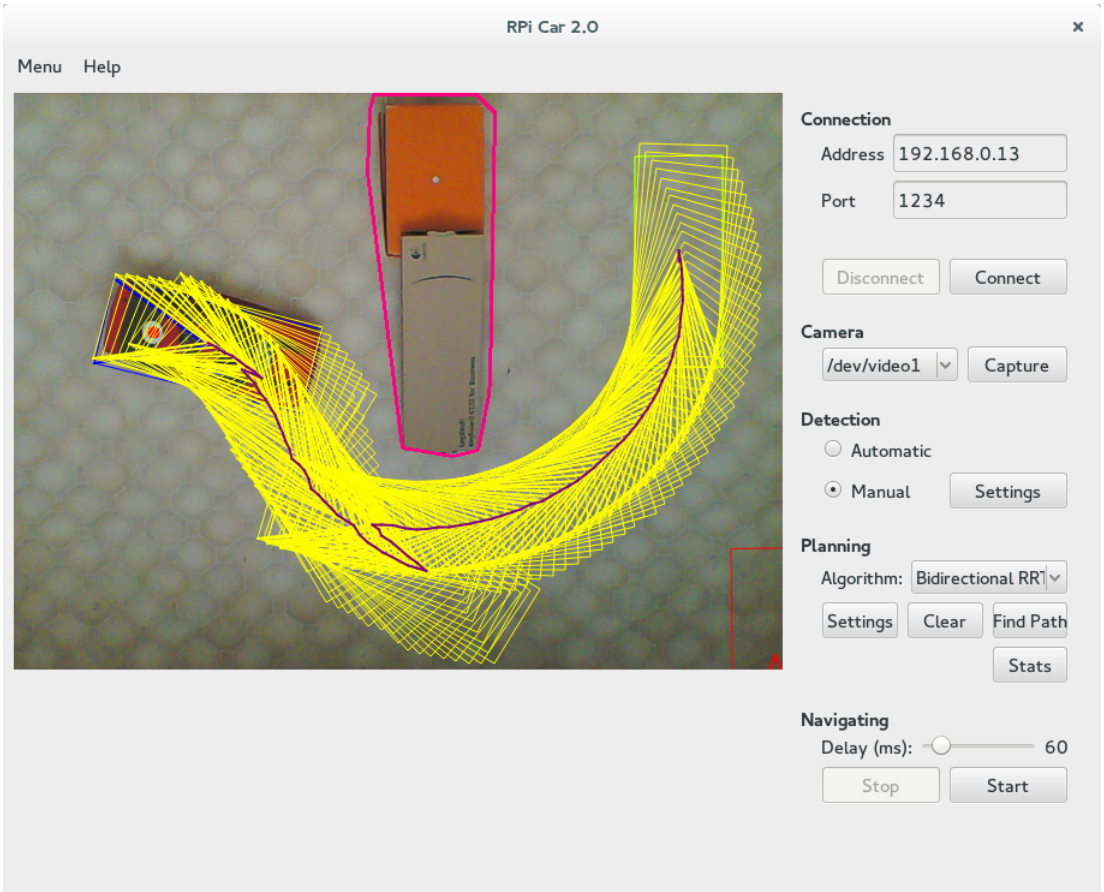
Obrázek 6.7: Vynesení *počtů uzlů cesty* do grafu pro 100 náhodných spuštění, zelená linka představuje průměrnou hodnotu.

6.3 Vyhodnocení navigace auta

Způsob jakým dochází k přesunu robotického auta po naplánované trase je popsán v kapitole 5.2.4. Zvolená metoda patří mezi primitivní a poměrně naivní. Sekvenční vykonávání pohybových příkazů totiž nijak nevyhodnocuje nedostatky, které mohly nastat v průběhu jízdy. Protočení kol na místě, smyk, nebo zaseknutí o překážku mohou výrazným způsobem ovlivnit výslednou pozici auta. Rychlost pohybu a výkon elektrického motoru závisí na stavu a nabití akumulátoru. Všechny tyto vstupní proměnné a omezení tak sehrají při jízdě auta svou roli.

Aby výsledný přesun auta byl co nejpřesnější, je potřeba kvalitně nakonfigurovat systém provádění řídicích příkazů. Tento jednoduchý mechanismus však nebude patrně nikdy dokonalý. Pro vyšší přesnost by bylo vhodné rozšířit navigaci o zpětnou vazbu, která by podle aktuální odchylky od naplánované trasy generovala nové zpřesňující příkazy.

Pro práci bylo původně uvažováno použití metriky vzdálenosti mezi naplánovaným a reálně dosaženým cílem. Tím by došlo k vyhodnocení navigace robotického autíčka pomocí matematického vztahu. Z časových důvodů však provedená měření nebyla zdokumentována.



Obrázek 6.8: Zobrazení naplánované cesty v reálném prostředí, auto je připraveno pro zahájení navigace

6.4 Způsob testování

Testování celého systému lze rozdělit na dvě hlavní etapy. První probíhala současně s vývojem jednotlivých modulů, kdy například pomocí ladících výpisů docházelo k okamžitému hledání chyb v programu. Pro řešení závažnějších potíží byl k dispozici „debugger“ *GDB*, který je vestavěn do vývojového prostředí *Qt Creator*. Pro odhalování chyb s uvolňováním paměti bylo využito nástroje *valgrind*, jenž se také nachází ve zmiňováném prostředí.

Druhá část testování proběhla po skončení implemntace a měla za úkol najít dosud neznámé chyby a neočekávané „pády“ programu. Několik chyb bylo také odhaleno při vyhodnocování jednotlivých částí systému. Testování byl také podroben program běžící na Raspberry Pi.

Kapitola 7

Závěr

V rámci diplomové práce byly nejprve prostudovány principy počítačového vidění a metody zpracování obrazu. Byly také představeny segmentační techniky využitelné pro detekci objektů v obraze. Teoretická část práce se dále věnovala pravděpodobnostnímu plánování, zejména variantám algoritmu RRT. Cíl práce spočíval v řízení robotického autíčka pomocí Raspberry Pi a kamery, a tak další část textu se zabývala pojmu neholonomický robot.

V kapitole o návrhu řešení se nachází detailní popis připojení počítače Raspberry Pi k motorům auta. Dále následuje představení konceptu řídicí aplikace, pomocí které se ovládá celý systém. Na kapitolu o samotné realizaci projektu navazuje text s vyhodnocením implementovaných modulů. Evaluace detekce a klasifikace objektů ve videu je zapsána v podobě dvou experimentů. Plánovací algoritmus je vyhodnocen na základě uvedených metrik. Vše je doplněno názornými obrázky a grafy. Na úplném konci práce se jako příloha C nacházejí čtyři případové studie hledání cesty v prostředí.

Výsledný systém se skládá z fungujícího robotického auta, které čeká na řídicí pokyny hlavní aplikace, jenž je postavena na třech stěžejních modulech. Detekce a rozpoznávání objektů ve snímané scéně je na obstojné úrovni. Navazující plánovací algoritmus se dokáže vypořádat i s poměrně složitým prostředím a funguje spolehlivě. Třetí část v podobě navigace auta na základě nalezené cesty není dostatečně kvalitní a přesnost jízdy auta je velmi malá.

7.1 Návrhy vylepšení a rozšíření práce

Výsledný systém je možné vylepšit několika způsoby. Detekce objektů je náchylná na změnu osvětlení nebo výskyt šumu v obraze. Zvýšení úspěšnosti hledání auta a překážek ve scéně by tak mohlo být prvním rozšířením diplomové práce.

Nalezení cesty do cíle funguje poměrně spolehlivě, výsledné trasy však často jsou zbytečně dlouhé a komplikované. Optimalizace plánovacího algoritmu, která by vedla k minimálním délkám nalezených cest, by mohla představovat zajímavý problém. Provádění jízdy autíčka by mohlo být rozšířeno o implementaci zpětné vazby pomocí kamery, která by kontrolovala jeho aktuální pozici. Podle odchylky od plánované cesty by mohlo docházet k zpřesnění jízdy například na základě *fuzzy* řízení.

Literatura

[1] Choset, H.; Lynch, K. M.; Hutchinson, S.; aj.: *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005, 603 s., ISBN 0-262-03327-5.

[2] Dlouhý, M.: Pravděpodobnostní plánování [online]. 2003-12-05 [cit. 2015-01-17]. Dostupné z: http://robotika.cz/guide/probplan/cs

[3] Dostálek, L.; Kabelová, A.: *Velký průvodce protokoly TCP/IP a systémem DNS*. Praha: Computer Press, 2002, 542 s., ISBN 80-7226-675-6.

[4] Hlaváč, V.: Digitální obraz, základní pojmy [online]. [cit. 2015-01-15]. Dostupné z: http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/014DigitalImageCz.pdf

[5] Hlaváč, V.: Jasové a geometrické transformace [online]. [cit. 2015-05-15]. Dostupné z: http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/18BrightGeomTxCz.pdf

[6] Hlaváč, V.: Předzpracování obrazu v lokálním okolí [online]. [cit. 2015-05-15]. Dostupné z: https://cw.fel.cvut.cz/wiki/_media/courses/a4m33dzo/21imagpreproc_.pdf

[7] Horák, K.: Jasové transformace [online]. [cit. 2015-05-15]. Dostupné z: http://midas.uamt.feec.vutbr.cz/ZVS/lectures-pdf/04_Jasove_transformace.pdf

[8] Knispel, L.: *Advanced Robot Path Planning (RRT)*. Diplomová práce, Vysoké učení technické, Fakulta strojního inženýrství, 2012 [cit. 2015-01-15]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=57322

[9] LaValle, S. M.: *Planning Algorithms [online]*. Cambridge, U.K.: Cambridge University Press, 2006. Dostupné z: http://planning.cs.uiuc.edu/

[10] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. Toronto: Thomson, 2008, 829 s., ISBN 978-0-495-08252-1.

[11] Sonka, M.; Hlaváč, V.: *Počítačové vidění*. Praha: Grada, 1992, 272 s., ISBN 80-85424-67-3.

[12] Thomé, A. C. G.: *SVM Classifiers – Concepts and Applications to Character Recognition*. 2012, ISBN 978-953-51-0823-8. Dostupné z: http://www.intechopen.com/books/export/citation/BibTex/advances-in-character-recognition/svm-classifiers-concepts-and-applications-to-character-recognition

[13] Tzafestas, S. G.: *Introduction to Mobile Robot Control*. Burlington: Elsevier Science, 2013, ISBN 9780124170490.

[14] WWW stránky: 55:148 Dig. Image Proc. Chapter 4, Part 3 [online]. 2000-08-31 [cit. 2015-05-09]. Dostupné z: http://user.engineering.uiowa.edu/~dip/LECTURE/PreProcessing3.html

[15] WWW stránky: 55:148 Dig. Image Proc. Chapter 5, Part 3 [online]. 2003-11-06 [cit. 2015-05-15]. Dostupné z: http://user.engineering.uiowa.edu/~dip/lecture/segmentation3.html

[16] WWW stránky: Simple Guide to the RPi GPIO Header and Pins. 2012-06-09 [cit. 2015-05-20]. Dostupné z: http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/

[17] WWW stránky: E10 DRIFT Toyota Sprinter Trueno AE86 RTR [online]. [cit. 2015-01-17]. Dostupné z: http://www.carson-modelsport.cz/zbozi/24907/E10-DRIFT-Toyota-Sprinter-Trueno-AE86-RTR.htm

[18] WWW stránky: Edimax EW-7811UN 150Mbps Wireless Nano USB Adapter [online]. [cit. 2015-01-17]. Dostupné z: http://www.cmsdistribution.com/networking/edimax-wi-fi-150mbps-mini-usb

[19] WWW stránky: ZTW 3A 5-6v Switch Mode UBEC [online]. [cit. 2015-01-17]. Dostupné z: http://www.hobbyhangar.co.nz/voltage-regulators/switch-mode-ubec-p-5331.html

[20] WWW stránky: Live! Cam Sync HD - Kamery - Creative Labs. [cit. 2015-05-17]. Dostupné z: http://cs.creative.com/p/web-cameras/live-cam-sync-hd

[21] WWW stránky: Raspberry Pi - Model B. [cit. 2015-05-17]. Dostupné z: https://www.sparkfun.com/products/11546

[22] WWW stránky: Nonholonomic Mobile Robot Motion Planning. [cit. 2015-05-22]. Dostupné z: http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu/

Příloha A

Obsah CD

- src/pc-app/ – zdrojové kódy hlavní aplikace
- src/rpi/ – zdrojové kódy programu pro RPi
- src/Makefile – hlavní Makefile
- tex/ – zdrojové kódy technické zprávy
- projekt.pdf – technická zpráva v elektronické podobě
- readme.txt – manuál k diplomové práci

Příloha B

Manual

```
#########################################################################
# Manual k diplomove praci:
# Rizeni pohybu robota pomoci Raspberry Pi a kamery
# Autor: Miroslav Brhel, xbrhel02, 27.5.2015
# FIT VUT v Brne
#########################################################################
```

Adresarova struktura	
src/Makefile	hlavni Makefile
src/pc-app/	adresar s hlavni aplikaci
src/rpi/	adresar s programem pro RPi
readme.txt	tento dokument
tex/	adresar s textem DP

Instalace Raspberry Pi
Zdrojove kody: src/rpi/
1. Nakopirovat OS Linux na SD kartu (Raspbian) 2. Zapojit napajeni a prislusenstvi k RPi (viz text DP) 3. Pripojit RPi k robotickemu autu (schema v DP) 4. Spustit a provest konfiguraci RPi (viz text DP) 5. Instalace knihovny pigpio 6. Preklad ridiciho programu (pomoci Makefile) 7. Kopie inicializacniho skriptu (viz text DP) 8. Restartovnani RPi, nasledne by automaticky melo dojit ke spravnemu spusteni programu

```
| Instalace hlavni aplikace na pc (verze Linux i Windows)
|
| Zdrojove kody: src/pc-app/
|
| 1. Nutna knihovna Qt (ve verzi 5.4.1)
| 2. Nutna knihovna OpenCV (verze 2.4.9)
| 3. Preklad pomoci Makefile nebo v QtCreatoru (.pro soubor)
| 4. Spusteni aplikace
```

```
| Postup prace s aplikaci
|
| 1. Zvol IP adresu a port Raspberry Pi a pripoj se
| 2. Vyber pozadovanou kameru z nabidky a spust nahravani
|    (nebo zvol virtualni prostredi z nabidky kamer)
| 3. Vyber typ detekce: automaticka nebo manualni
| 4. Zvol cilovou pozici levym klikem mysi do sceny
| 5. Rotace polohy funguje pres prave tlacitko mysi
| 6. Zvol algoritmus a nastav jeho paramety (vlevo)
| 7. Spust planovani cesty (lze opakovane)
|    (k dispozici je tlacitko pro smazani cesty)
| 8. Spust navigaci nebo ji pozastav
```

Příloha C

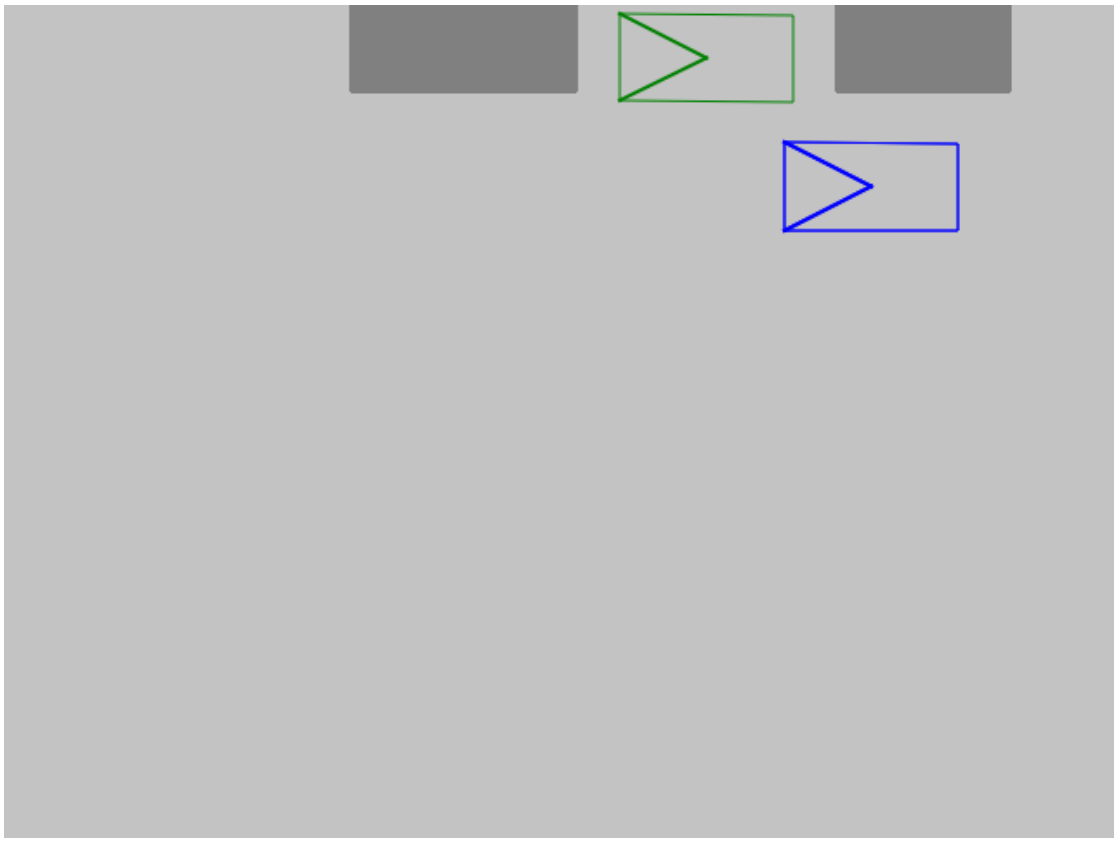
Ukázky hledání cest v prostoru

Na následujících stránkách budou ukázány 4 případové studie hledání cesty v prostoru pro robotické autíčko. První dva případy jsou z virtuálního prostředí implementované aplikace, a vyhodnocení proběhlo po 100 pokusech vyhledání cesty. Dosažené výsledky se nachází za obrazovými ukázkami daného problému. Poslední dva případy jsou ukázkou plánovacího algoritmu v praxi. V obou případech byla vybraná náhodná cesta. Výsledky se týkají pouze konkrétní cesty.

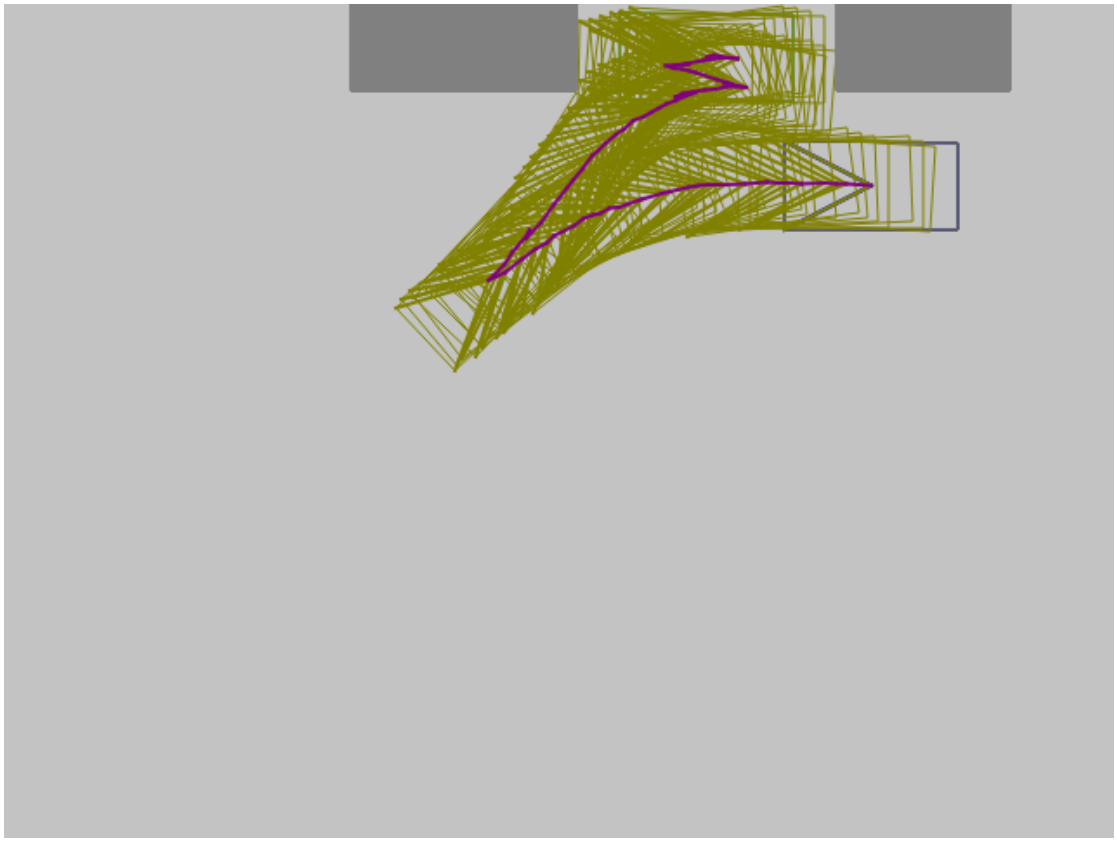
C.1 Paralelní parkování ve virtuálním prostředí

Algoritmus	*Bidirectional RRT*
Rozměry scény	$640 \times 480\,px$
„Vzdušná“ vzdálenost start-cíl	$120,42\,px$
Max. ϕ	$25\,°$
Krok (dt)	$10\,px$
Ukončující vzdálenost	$5\,px$

Tabulka C.1: Vlastnosti algoritmu a zkoumané scény



Obrázek C.1: Problém paralelního parkování auta z počátečního místa (modře) do cílového místa (zeleně)



Obrázek C.2: Ukázka nalezené cesty pro problém paralelního parkování auta

	Doba běhu (ms)	**Délka cesty (px)**	**Počet uzlů cesty**
Průměr	1112, 39	1283, 33	142, 2
Minimum	39, 75	387, 227	45
Maximum	3062, 18	2388, 08	268

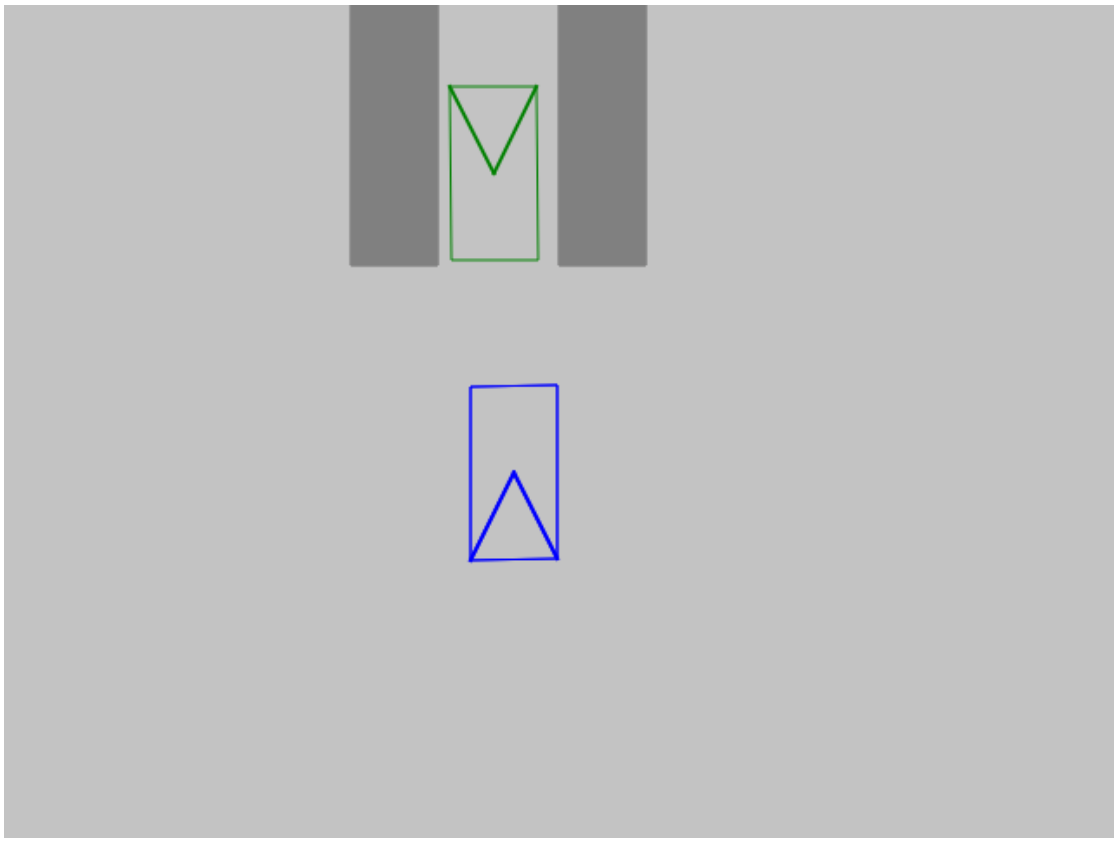
Počet nenalezených řešení	25

Tabulka C.2: Výsledné hodnoty pro 100 náhodných spuštění

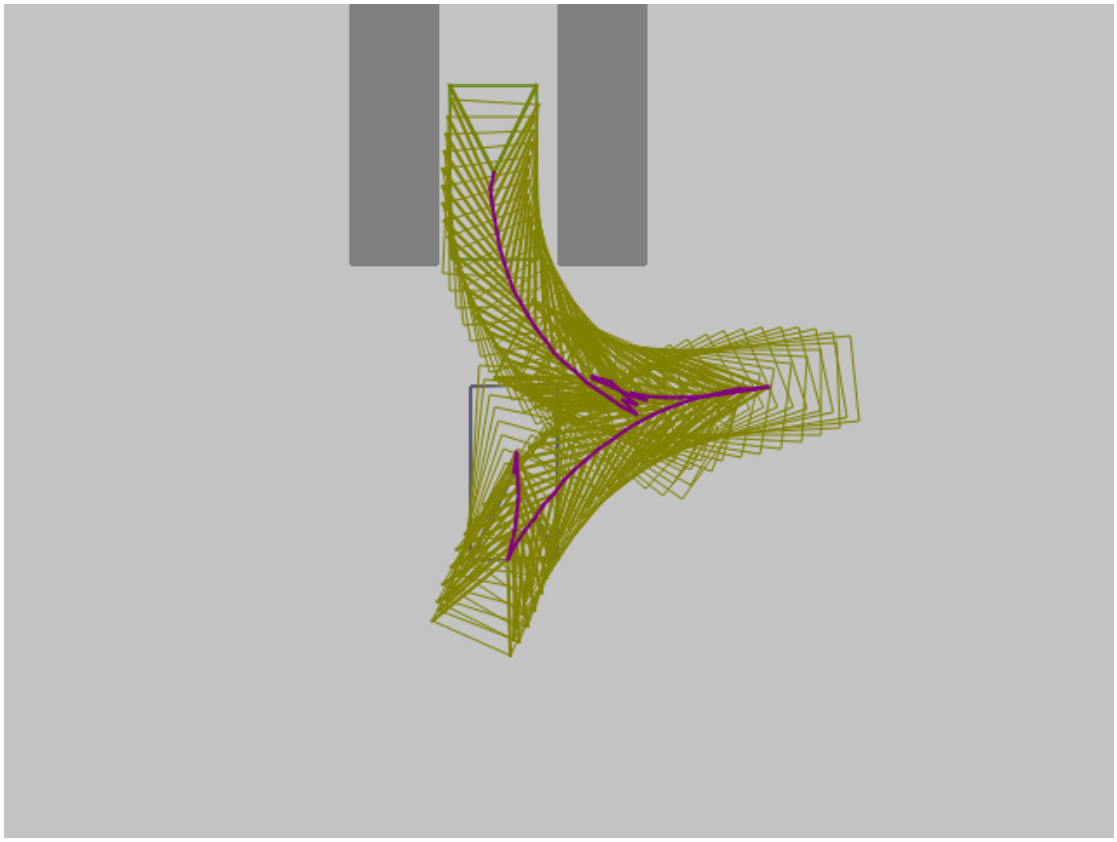
C.2 Otočení a couvání mezi překážky ve virtuálním prostředí

Algoritmus	*Bidirectional RRT*
Rozměry scény	$640 \times 480\,px$
„Vzdušná“ vzdálenost start-cíl	$233,26\,px$
Max. ϕ	$25\,°$
Krok (dt)	$10\,px$
Ukončující vzdálenost	$5\,px$

Tabulka C.3: Vlastnosti algoritmu a zkoumané scény



Obrázek C.3: Problém otočení a couvání auta mezi překážky z počátečního místa (modře) do cílového místa (zeleně)



Obrázek C.4: Ukázka nalezené cesty pro problém couvání mezi překážky

	Doba běhu (ms)	**Délka cesty (px)**	**Počet uzlů cesty**
Průměr	239, 15	984, 76	110, 95
Minimum	12, 55	559, 62	64
Maximum	1029, 48	1709, 24	193

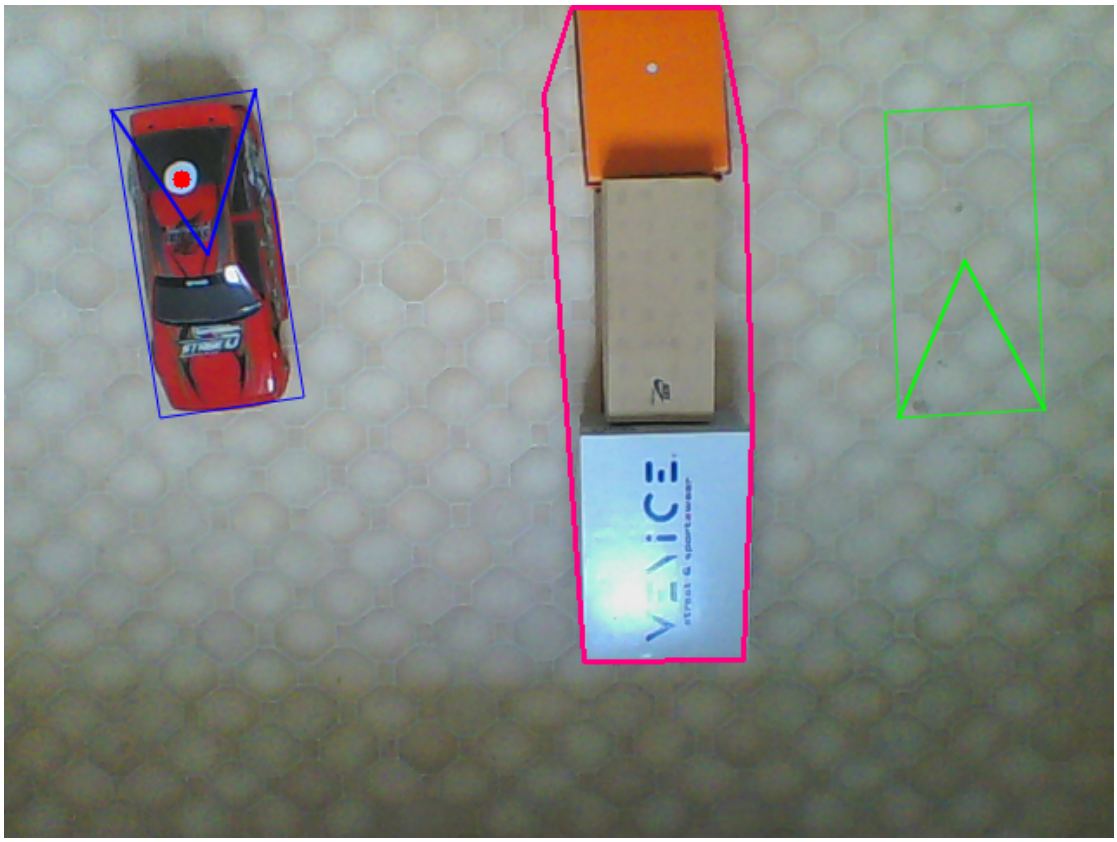
Počet nenalezených řešení	0

Tabulka C.4: Výsledné hodnoty pro 100 náhodných spuštění

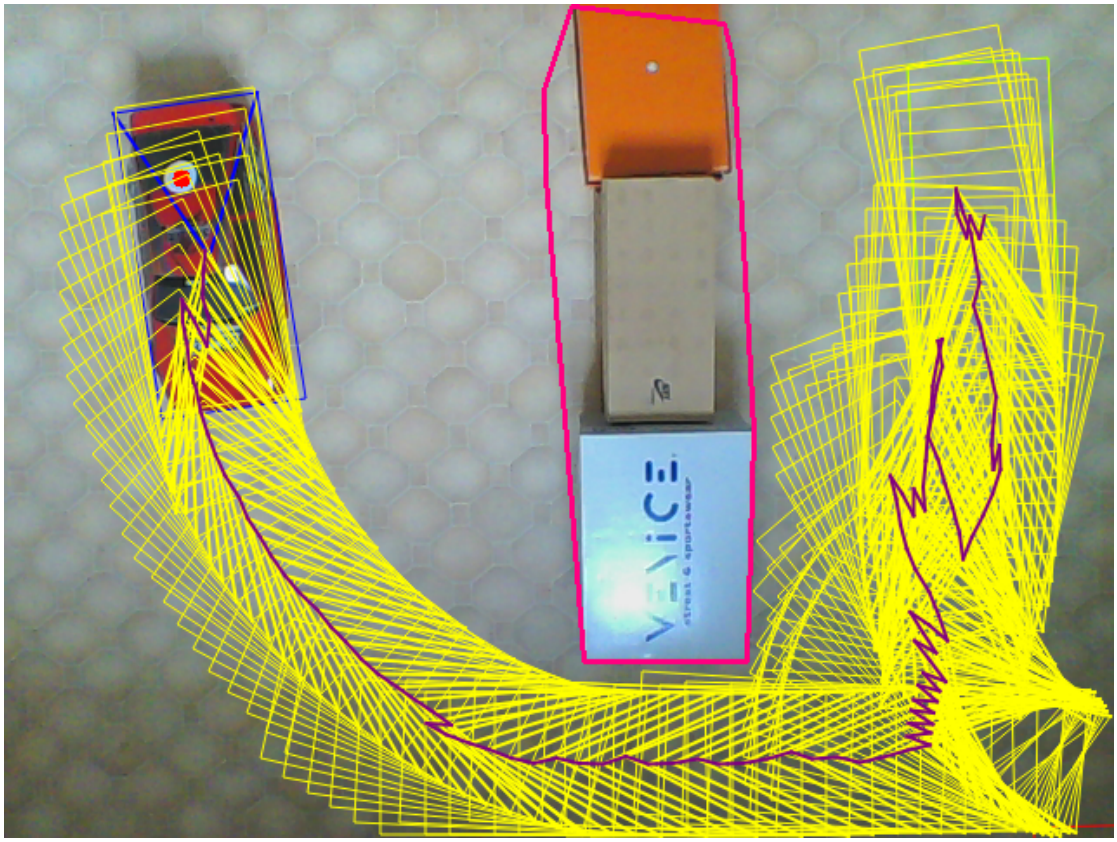
C.3 Objetí dlouhé překážky v reálném prostředí

Algoritmus	*Bidirectional RRT*
Rozměry scény	$640 \times 480\,px$
„Vzdušná“ vzdálenost start-cíl	$443,4\,px$
Max. ϕ	$25\,°$
Krok (dt)	$15\,px$
Ukončující vzdálenost	$5\,px$
Doba běhu	$648,1\,ms$
Počet uzlů cesty	123
Délka nalezené cesty	$1641,94\,px$

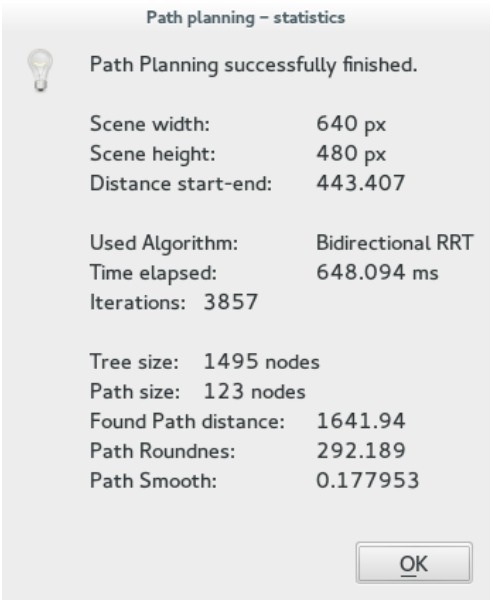
Tabulka C.5: Vlastnosti algoritmu a zkoumané scény



Obrázek C.5: Problém objetí dlouhé překážky v reálném prostředí z počátečního místa (modře) do cílového místa (zeleně)



Obrázek C.6: Ukázka nalezené cesty pro problém objetí dlouhé překážky

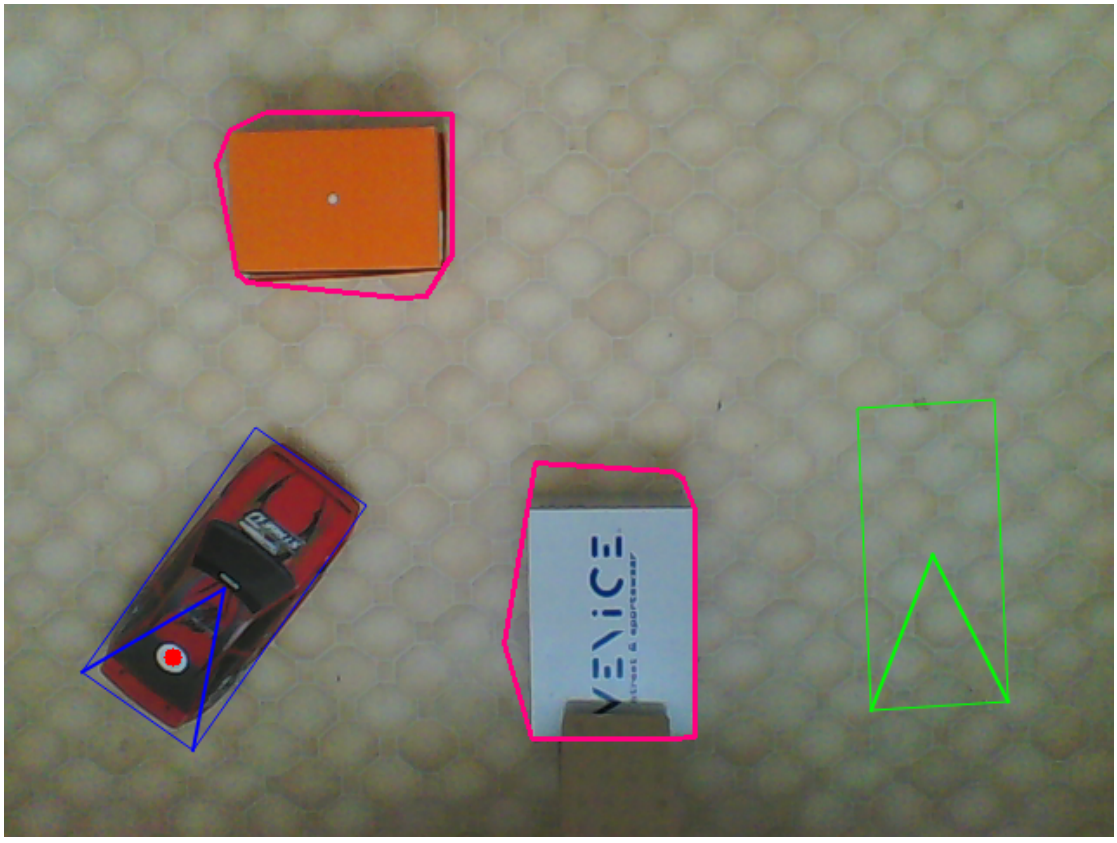


Obrázek C.7: Zobrazení dialogového okna s výsledky plánovacího algoritmu

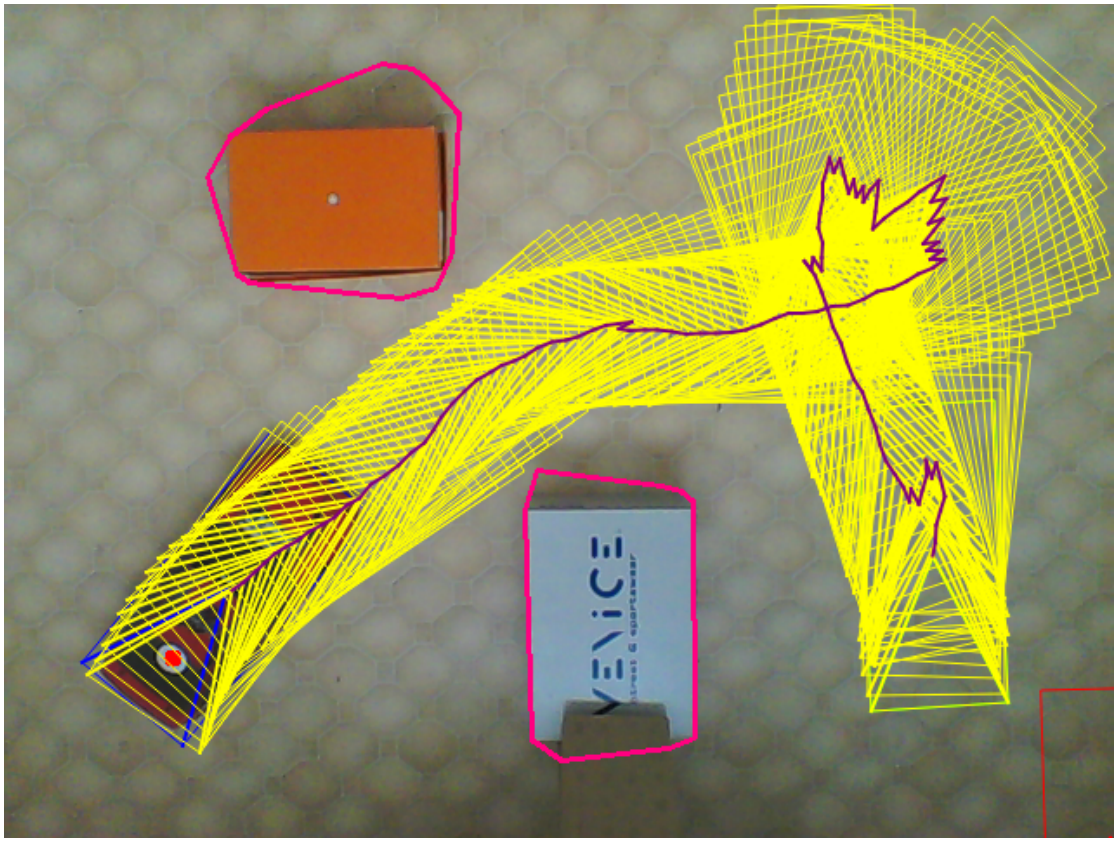
C.4 Jízda mezi překážkami s následným couváním v reálném prostředí

Algoritmus	*Bidirectional RRT*
Rozměry scény	$640 \times 480\,px$
„Vzdušná“ vzdálenost start-cíl	$434,04\,px$
Max. ϕ	$35\,^{\circ}$
Krok (dt)	$10\,px$
Ukončující vzdálenost	$10\,px$
Doba běhu	$119,8\,ms$
Počet uzlů cesty	129
Délka nalezené cesty	$1105,34\,px$

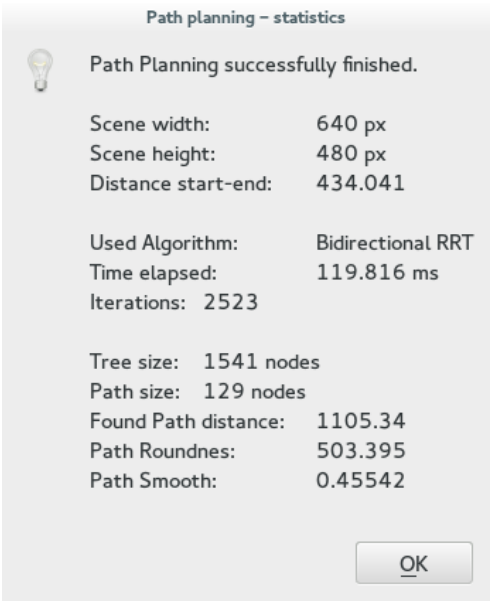
Tabulka C.6: Vlastnosti algoritmu a zkoumané scény



Obrázek C.8: Problém nalezení cesty mezi překážkami z počátečního místa (modře) s couváním do cílového místa (zeleně)



Obrázek C.9: Ukázka nalezené cesty mezi překážkami s následným couváním



Obrázek C.10: Zobrazení dialogového okna s výsledky plánovacího algoritmu