ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra řídicí techniky

Bakalářská práce

Automatické předjíždění autonomního auta F1/10

Jan Lindauer
Kybernetika a robotika

Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lindauer** Jméno: **Jan** Osobní číslo: **486427**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Automatické předjíždění autonomního auta F1/10

Název bakalářské práce anglicky:

Automatic overtaking of an autonomous car F1/10

Pokyny pro vypracování:

1. Seznamte se s ROSem a projektem F1/10 (modelu autonomního auta).
2. Proveďte rešerši algoritmů vhodných pro předjíždění aut.
3. Na základě rešerše upravte gradientní plánovač prezentovaný v [1] tak, aby podporoval předjíždění.
4. Upravený plánovač otestujte na platformě F1/10.
5. Vše pečlivě zdokumentujte.

Seznam doporučené literatury:

[1] Záhora J. , Sojka M., Hanzálek Z. - Cone slalom with automated sports car – dosud nepublikováno
[2] Klapálek J. - Vyhýbání se dynamickým překážkám s autonomním autem F1/10 – diplomová práce ČVUT 2017
[3] Vajnar M. - Model formule pro soutěž autonomních aut F1/10 – diplomová práce ČVUT 2019

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Michal Sojka, Ph.D., katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **03.02.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce:
do konce letního semestru 2021/2022

Ing. Michal Sojka, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____
Datum převzetí zadání Podpis studenta

Poděkování / Prohlášení

Chtěl bych poděkovat Ing. Michalu Sojkovi, Ph.D. a Ing. Jiřímu Záhorovi za podnětné návrhy a odbornou pomoc během konzultací. Také bych rád poděkoval své rodině, přítelkyni a přátelům za podporu během studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2021

.....

Abstrakt / Abstract

Autonomní vozidla se postupně dostávají do běžného provozu. Vozidlo musí být schopno řešit řadu dopravních situací, které při jízdě mohou nastat. V některých situacích je potřeba použít předjížděcí manévr.

Cílem této práce bylo vytvořit předjížděcí algoritmus určený pro model závodního auta.

Pro plánování byl použit gradientní optimalizační plánovač cesty. Vstupem plánovače jsou průjezdní body, které byly voleny pomocí principu lokálního plánování. Pro sledování vytvořené trajektorie byly implementovány a porovnány tři algoritmy.

Funkčnost algoritmů byla otestována jak v simulaci, tak na fyzickém modelu auta. V simulaci algoritmy fungovaly správně. Při testech na fyzickém modelu auta se projevily chyby detekce a predikce kolize. V důsledku toho se předjíždění dařilo přibližně v polovině případů.

Aby bylo možné nasazení algoritmů v autonomním závodě, je potřeba použít lepší algoritmus pro detekci a predikci polohy předjížděného auta.

Klíčová slova: autonomní auto, gradientní plánovač, lokální plánovač, sledování cesty, F1/10, Bezierova křivka, ROS, gradientní optimalizace parametrů

Autonomous vehicles are gradually getting into normal operation. The vehicle must be able to deal with a number of traffic situations that may arise while driving. In some situations it is necessary to use an overtaking maneuver.

The aim of this work was to create an overtaking algorithm designed for a race car model.

A gradient path optimization planner was used for planning. The input of the planner is the via points, which were selected using the principle of local planning. For the task of trajectory following, three algorithms were implemented and compared.

The functionality of all implemented algorithms was tested both in simulation and on the physical model of the car. The algorithms worked correctly in the simulation. During the tests on the physical model of the car, the errors of collision detection and prediction became significant. As a result, overtaking was successful in about half of the cases.

In order to be able to deploy implemented algorithms in an autonomous race, it is necessary to use a better algorithm for detecting and predicting the position of an overtaken car.

Keywords: autonomous car, gradient planner, local planner, path following, F1/10, Bezier curve, ROS, gradient parameter optimization

Title translation: Automatic overtaking of an autonomous car F1/10

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Oblast autonomní mobility je neustále se vyvíjející obor. Autonomní vozidla se postupně dostávají do běžného života. Začínají se objevovat nejen v silničním provozu, ale vznikají i autonomní závody.

Jedním z takových závodů je i soutěž F1/10 [1–2]. Jedná se o závod autonomních modelů aut v měřítku 1:10, kterého se pravidelně účastní i tým z ČVUT. Závod probíhá v předem známém prostředí, ve kterém se může pohybovat i soupeřící auto. Pro nejlepší možné umístění závodního vozu je nutné vytvořit algoritmus zajišťující předjetí soupeřícího auta.

Problém vytvoření předjížděcího manévru lze rozdělit na několik částí. První částí je zajištění přesné detekce polohy předjížděného auta. Zároveň je potřeba také predikovat další pohyb předjížděného auta. Druhou částí je vytvoření trajektorie, která by se překážkám a soupeři vyhnula. *Trajektorie* je složena z *cesty*, definované posloupností *stavů*, a rychlostního profilu. Třetí částí je sledování vytvořené trajektorie. V této práci jsem se zaměřil zejména na plánování trajektorie. Pro testování plánovacích strategií však bylo potřeba vyřešit i ostatní podproblémy.

Bližší popis platformy auta a programů použitých pro detekci okolních objektů a lokalizaci jsem uvedl do kapitoly 2. V této kapitole popisuji také kinematický model pro popis pohybu auta.

Při plánování trajektorie používám gradientní plánovač cesty [3–4] (dále jen *gradientní plánovač*). Tento plánovač byl původně vyvinutý pro autonomní slalom s Porsche Panamera [3], sekundárním cílem této práce bylo plánovač otestovat i v jiné aplikaci. Vlastnosti gradientního plánovače a princip, na kterém je plánovač založen jsem popsal v kapitole 3.

Vstupem gradientního plánovače jsou požadované průjezdní body. Metodou jejich výběru se zabývám v kapitole 4. V této kapitole popisuji také použitý způsob pro predikci a detekci kolize na vytvořené trajektorii a způsob jakým z údajů o okolních překážkách získávám polohu a rychlost předjížděného auta.

Sledování vytvořené trajektorie se věnuji v kapitole 5.

Testováním a zhodnocením celkového řešení se zabývám v kapitole 6.

Kapitola 2
Kontext

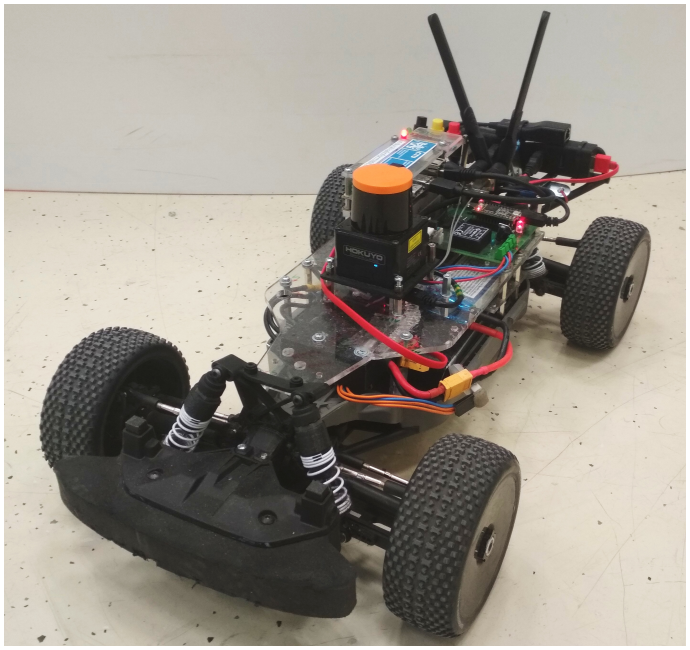
V této kapitole popisuji použitý model auta (2.1), hotová řešení pro lokalizaci auta (2.2) a detekci okolních objektů (2.3).

V posledních dvou sekcích popisuji kinematický model auta, který v práci dále používám. Model používám ve dvou souřadnicových soustavách – v kartézské soustavě souřadnic (2.4) a v souřadnicovém systému cesty (2.5).

2.1 Model auta

Použitý model závodního auta v měřítku 1:10 (obrázek 2.1) je uzpůsobený pro závod F1/10 [1–2]. Technické specifikace auta omezené pravidly soutěže jsou uvedeny v tabulce 2.1.

Řídící systém auta je implementován v prostředí ROSu (Robotic operating system [5]), programy jsou v ROSu organizovány do nodů [6], které mezi sebou komunikují.



Obrázek 2.1. Auto použité při testech

Funkčnost algoritmů jsem testoval také v simulátoru *Stage* [7].

2.2 Lokalizace

K lokalizaci auta je použit Google Cartographer [8]. Jedná se o systém Současné Lokalizace A Mapování (SLAM). Cartographer k určení polohy využívá data z IMU (Inertial

Tabulka 2.1. Technické specifikace

CPU	nVidia Jetson TX2 (Orbitty Carrier)
Platforma	Traxxas Slash 1:10 4WD VXL TQi TSM OBA RTR
Motor	Velineon 3500 (+ VESC)
Servo	Traxxas 2075R
Lidar	Hokuyo UST-10LX (2D)
OS	Ubuntu Linux 16.04 + *ROS* Kinetic

measurement unit) a z LIDARu. Na základě dat z LIDARu je vytvářena 2D mapa okolního prostoru, ve které je auto lokalizováno. Střední kvadratická odchylka lokalizace je během jízdy ≈ 10 cm [9].

2.3 Detekce objektů

K lokalizaci soupeřícího auta je využit algoritmus vyvinutý na Poznan University of technology [10]. Tento detektor jsem použil, jelikož byl na autě již nainstalován a naladěn [11].

Algoritmus není primárně určený k detekci auta. Je určený k detekci polohy a vektoru rychlosti kruhových překážek v okolí mobilního robota s pomocí LIDARu. Lepší detekce by mohlo být dosaženo použitím algoritmu přímo určeného k detekci auta v měřítku 1:10. Ten by mohl navíc zahrnovat kinematický model auta pro lepší odhad směru a rychlosti jízdy.

2.4 Kinematický model

Napříč touto prací je pro modelování pohybu auta[1] použit kinematický model [12] popsaný rovnicemi

$$\begin{aligned} \beta &= \tan^{-1}\left(\frac{L_r}{W}\tan(\delta)\right) \\ \dot{x} &= v\cos(\theta+\beta) \\ \dot{y} &= v\sin(\theta+\beta) \\ \dot{\theta} &= v\frac{\cos(\beta)}{W}\tan(\delta), \end{aligned} \tag{1}$$

význam jednotlivých veličin je uveden v tabulce 2.2 a ilustrován na obrázku 2.2.

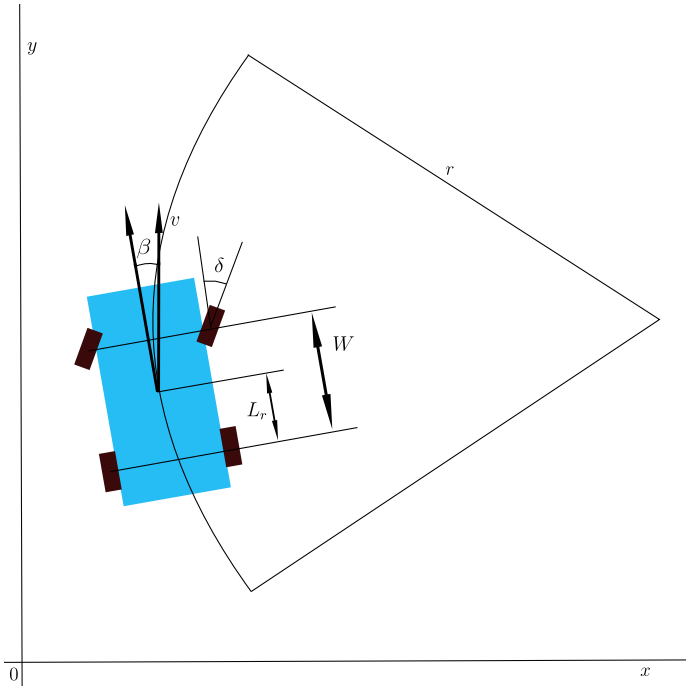
2.5 Souřadnicový systém cesty

Závod F1/10 probíhá v předem známém prostředí. Je tedy možné, buďto automaticky nebo manuálně, připravit cestu závodní dráhou (dále jen *globální cesta*) a v algoritmech budu předpokládat její apriorní znalost.

Často je výhodné použít souřadnicový systém globální cesty[2] [13]. Polohové souřadnice $[s,d]$ v souřadnicovém systému cesty udávají podélnou pozici na cestě a laterální vzdálenost od cesty (se znaménkem). Souřadnicový systém je znázorněn na obrázku 2.3.

[1] pokud není uvedeno jinak

[2] v anglické literatuře je souřadnicový systém cesty označován jako Frenet frame



Obrázek 2.2. Kinematický model

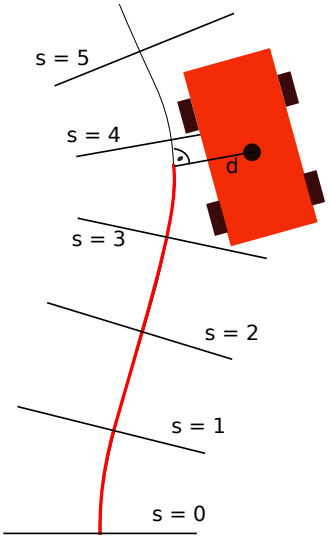
Tabulka 2.2. Vysvětlení použitých veličin

$[x, y]$	polohové souřadnice referenčního bodu na autě
θ	orientace auta
δ	úhel natočení kol
W	vzdálenost mezi nápravami
L_r	vzdálenost mezi bodem $[x, y]$ a zadní nápravou
β	slip angle
r	poloměr křivosti zatáčky
c	$= \begin{cases} 1/r & \text{kola zatočená doprava,} \\ -1/r & \text{kola natočená doleva.} \end{cases}$, křivost
ε	časová změna křivosti

Matematický popis kinematického modelu v tomto souřadnicovém systému, pokud pro zjednodušení zvolím referenční bod ve středu zadní nápravy ($L_r = 0$, $\beta = 0$) je [13]

$$\begin{aligned} \dot{s} &= v\frac{\cos(\theta_r)}{1 - dc_p} \\ \dot{d} &= v\sin(\theta_r) \\ \dot{\theta}_r &= v\left(\frac{\tan(\delta)}{W} - \frac{c_p\cos\theta_r}{1 - dc_p}\right), \end{aligned} \tag{2}$$

kde θ_p je orientace a c_p křivost cesty, jejíž souřadnicový systém je použit, rozdíl $\theta - \theta_p$ je označen θ_r.



Obrázek 2.3. Souřadnicový systém cesty

Kapitola 3
Gradientní plánovač

V této kapitole je blíže představen gradientní plánovač. Nejprve je v sekci 3.1 popsán kinematický model auta použitý v plánovači. Funkční princip plánovače a jeho odvození je uvedeno v části 3.2. Z funkčního principu vyplývají některé vlastnosti plánovače. Ty jsou shrnuty v sekci 3.3. Vyvstávají dva hlavní problémy.

Prvním problémem je velká výpočetní náročnost, té jsem věnoval sekci 3.4.

Druhým problémem je, že v plánovači je použit předpoklad konstantní rychlosti ve směru osy x použitého souřadnicového systému. V důsledku toho plánovač není vhodné použít v zatáčkách. Možné řešení tohoto problému jsem uvedl v poslední sekci této kapitoly (3.5).

3.1 Kinematický model pro gradientní plánovač

V gradientním plánovači je použit kinematický model auta popsaný v sekci 2.4, který je dále zjednodušen. Užitím přibližných vztahů (platných pro malé úhly)

$$\begin{aligned} \tan\delta \approx \delta \approx cW \\ \beta \approx cL_r \\ \cos(\beta) \approx 1 \end{aligned} \tag{1}$$

je možné psát

$$\begin{aligned} \dot{x} &= v\cos(\theta + cL_r) \\ \dot{y} &= v\sin(\theta + cL_r) \\ \dot{\theta} &= vc \\ \dot{c} &= \varepsilon \end{aligned} \tag{2}$$

nebo při užití diskrétního časového kroku Δt

$$\begin{aligned} x_{k+1} &= x_k + v_k\cos(\theta_k + c_kL_r)\Delta t \\ y_{k+1} &= y_k + v_k\sin(\theta_k + c_kL_r)\Delta t \\ \theta_{k+1} &= \theta_k + v_kc_k\Delta t \\ c_{k+1} &= c_k + \varepsilon_k\Delta t \\ \varepsilon_{k+1} &= \frac{c_k - c_{k-1}}{\Delta t} \end{aligned} \tag{3}$$

Transformací z globálních souřadnic do souřadnicového systému auta a předpokladem konstantní rychlosti v_x ve směru osy x se první rovnice v soustavě (3) zjednoduší na

$$x_{k+1} = x_k + v_x\Delta t$$

3.2 Princip

Princip plánovače byl navržen v projektu autonomního slalomu s autem Porsche Panamera [3] a vychází z algoritmů popsaných v [4]. Gradientní plánovač využívá zjednodušený diskrétní kinematický model (3). Předpokladem konstantní rychlosti ve směru osy x je možné první rovnici z plánování vynechat. Pro reprezentaci stavu $\boldsymbol{x}$ a akce $\boldsymbol{u}$ pak můžeme zavést vektory

$$\boldsymbol{x} = \begin{pmatrix} y \\ \theta \\ c \end{pmatrix}, \qquad \boldsymbol{u} = (\,\varepsilon\,)$$

Plánovač řeší optimalizační úlohu hledání cesty mezi dvěma zadanými stavy určenými vektory $\boldsymbol{x_s}$, $\boldsymbol{x_f}$ vedoucí skrz průjezdní body $[x_{j,flex}, y_{j,flex}]$, kde $j \subseteq \{1, \ldots, N-1\}$, N je počet diskrétních kroků tvořících cestu. Úlohu vytvoření optimální cesty[1] je možné napsat jako

$$\begin{aligned} &\min\, L(\boldsymbol{X}) \\ &\text{za podmínky } \boldsymbol{G}(\boldsymbol{X}) = \boldsymbol{0}, \end{aligned} \tag{4}$$

kde

$$\boldsymbol{G}(\boldsymbol{x_0} \ldots \boldsymbol{x_N}, \boldsymbol{u_0} \ldots \boldsymbol{u_N}) = \begin{pmatrix} y_{k+1} - y_k - v_k \sin(\theta_k + c_k L_r)\Delta t \\ \theta_{k+1} - \theta_k - v_k c_k \Delta t \\ c_{k+1} - c_k - \varepsilon_k \Delta t \end{pmatrix}\Bigg|_{k=1,\ldots,N-1} \\ \begin{pmatrix} \boldsymbol{x_s} - \boldsymbol{x_0} \\ \boldsymbol{x_f} - \boldsymbol{x_N} \\ y_{j,flex} - y_j\,, \forall j \end{pmatrix}$$

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x_0} \\ \vdots \\ \boldsymbol{x_N} \\ \boldsymbol{u_0} \\ \vdots \\ \boldsymbol{u_{N-1}} \end{pmatrix}$$

a

$$L(\boldsymbol{X}) = \sum_{i=1}^{4N} w_i X_i^2\,, \quad w_i \geq 0,$$

kde w_i jsou váhy a N je počet diskrétních stavů mezi průjezdními body[2]. Váhy plánovače použité v této práci jsou $w_{3i+2} = 1$, $w_{i \geq 3N} = 0.1$, $w_{other} = 0$, úloha (4) tedy minimalizuje součet kvadrátů křivosti c sečtený se součtem kvadrátů derivací křivosti ε váženým konstantou 0.1.

Jedná se o úlohu s nelineárními omezeními, pro její řešení je tedy možné použít metodu Lagrangeových multiplikátorů. Pro lokální extrém musí platit[3]

[1] optimální ve smyslu minimalizace ztrátové funkce $L(\boldsymbol{X})$

[2] plánovač je spuštěn dvakrát, poprvé s N vypočteným ze vzdálenosti mezi průjezdními body a podruhé s hodnotou upravenou na základě výsledku první iterace

[3] platí jen za předpokladu, že $\boldsymbol{X}$ je regulární bod, tj. $\text{rank}(\boldsymbol{G}'(\boldsymbol{X})) = m$, kde m je počet řádků matice $\boldsymbol{G}$, (tedy počtu omezení)

$$\begin{aligned} \nabla_{\boldsymbol{X}}(L(\boldsymbol{X}) + \lambda^T \boldsymbol{G}(\boldsymbol{X})) &= \boldsymbol{0} \\ \boldsymbol{G}(\boldsymbol{X}) &= \boldsymbol{0}, \end{aligned} \tag{5}$$

kde $\nabla_{\boldsymbol{X}}$ značí transponovanou parciální derivaci podle $\boldsymbol{X}$.

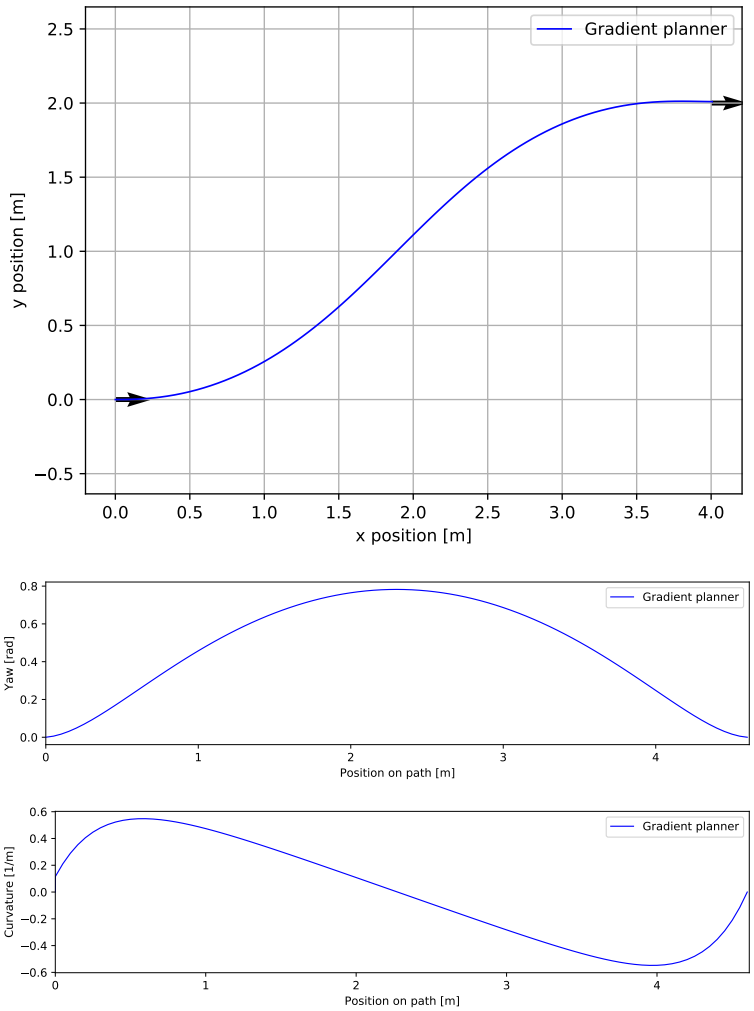
Tuto nelineární soustavu rovnic lze řešit Newtonovou iterační metodou pro řešení soustavy rovnic. Jeden krok Newtonovy metody pro hledání řešení obecné soustavy $\boldsymbol{F}(\boldsymbol{z}) = \boldsymbol{0}$ je

$$\boldsymbol{z}_{k+1} = \boldsymbol{z}_k - \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{z}_k}(\boldsymbol{z}_k)^{-1} \boldsymbol{F}(\boldsymbol{z}_k),$$

dosazením levých stran rovnic (5) za $\boldsymbol{F}(\boldsymbol{z})$ a $\boldsymbol{X}$ za $\boldsymbol{z}$ dostaneme rovnici

$$\begin{pmatrix} \boldsymbol{X}_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}_k \\ \boldsymbol{0} \end{pmatrix} - \begin{pmatrix} \nabla^2_{\boldsymbol{X}_k}(L(\boldsymbol{X}_k) + \lambda^T \boldsymbol{G}(\boldsymbol{X}_k)) & \nabla \boldsymbol{G}(\boldsymbol{X}_k) \\ \nabla \boldsymbol{G}^T(\boldsymbol{X}_k) & \boldsymbol{0} \end{pmatrix}^{-1} \begin{pmatrix} \nabla L(\boldsymbol{X}_k) \\ \boldsymbol{G}(\boldsymbol{X}_k) \end{pmatrix}$$

Pro vyřešení je použita *C++* knihovna *Eigen*. Podrobnější odvození je uvedeno v [3–4]. Příklad cesty vytvořené touto metodou je na obrázku 3.1.

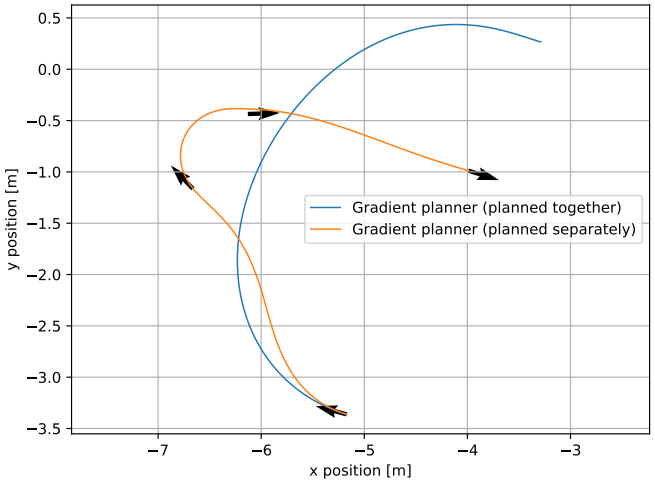


Obrázek 3.1. Příklad cesty vytvořené gradientním plánovačem. Zadaný počáteční stav měl v tomto případě souřadnice $[0, 0]$, nastavený koncový stav měl souřadnice $[4, 2]$, oba stavy měly nastavenou požadovanou křivost cesty a úhel udávající směr jízdy na nulové hodnoty, nebyly zvoleny žádné další průjezdní body.

3.3 Vlastnosti

Z použitého matematického modelu vyplývají některé vlastnosti plánovače:

- Plánovač respektuje kinematický model auta a díky tomu by měl generovat snadno sledovatelnou cestu.
- Plánovač je vhodné použít pouze při rychlostech, kdy můžeme zanedbat dynamické vlastnosti auta.
- Díky vynechání podélné souřadnice x z plánování a předpokladu konstantní rychlosti v tomto směru není možné generovat cestu, která by auto příliš ($\geq \frac{\pi}{2}$ rad) otočila od orientace v počátečním stavu $\boldsymbol{x_0}$. Příklad výsledku plánování pokud tato podmínka není dodržena je na obrázku 3.2. Výsledek plánování se výrazně zlepší (viz obrázek 3.2) rozdělením plánování na více částí, pak je totiž použit pro každý plán vlastní souřadnicový systém.
- Před spuštěním algoritmu je potřeba odhadnout počet kroků N, který bude cestu tvořit. Při nepřesném odhadu cesta neskončí přesně v zadaném koncovém bodě.



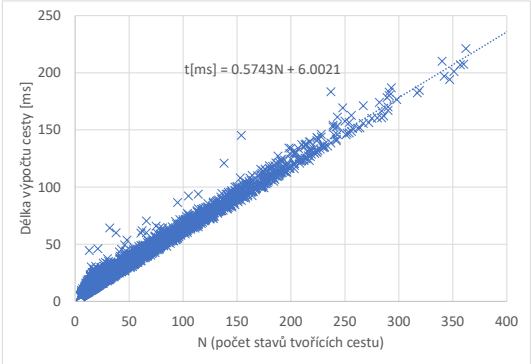
Obrázek 3.2. Cesta vytvořená gradientním plánovačem v zatáčce. Průjezdní body jsou vyznačeny šipkami. Modře označená cesta byla vytvořena přímým použitím gradientního plánovače. Oranžově označená cesta byla vytvořena mezi každými dvěma po sobě jdoucími průjezdními body zvlášť.

3.4 Výpočetní náročnost

Délka výpočtu je závislá na počtu kroků N. Pro cesty ve tvaru přímky je $N = \frac{s}{v\Delta t}$, kde s je Euklidovská vzdálenost mezi počátečním a koncovým bodem, Δt je časový krok plánovače a v je rychlost auta. Pro zakřivenější cesty (s většími součty absolutních hodnot křivostí) je $N > \frac{s}{v\Delta t}$.

Změřil jsem délku výpočtu cest s počátečním stavem o souřadnicích [0,0], křivosti $c = 0$ a orientaci $\theta = 0$ a náhodně vybranými koncovými stavy tak, aby vzdálenost s byla v intervalu [1,4] m. Časový krok plánovače jsem zvolil $\Delta t = 0.1s$ a rychlost $v = 2.5m/s$. Počty kroků N vytvořených cest a odpovídající délky výpočtu jsem vynesl do grafu na obrázku 3.3. Měření jsem prováděl přímo na procesoru auta (Nvidia TX2).

Ukazuje se, že délka výpočtu je přibližně přímo úměrná počtu kroků N (střední kvadratická odchylka lineární regrese je 3.7 ms). Průměrná délka výpočtu během tohoto



Obrázek 3.3. Závislost doby výpočtu cesty gradientním plánovačem na počtu stavů N tvořících plán

experimentu byla 31 ms. V případě, že je potřeba vytvářet více cest najednou (řádově více než 10), je taková délka výpočtu již příliš dlouhá.

3.5 Plánování v souřadnicovém systému cesty

Gradientní plánovač předpokládá rovnoměrný pohyb ve směru x-ové souřadnice. Proto plánování selhává v zatáčkách. To by se dalo vylepšit použitím souřadnicového systému globální cesty. Předpokladem by pak nebyl rovnoměrný pohyb ve směru x-ové osy, ale rovnoměrný pohyb ve směru podélné souřadnice s.

V případě autonomního závodu je možné použít souřadnicový systém globální cesty. V případě, že není známá globální cesta, může být pro první odhad cesty použita například polynomiální křivka (Bezierova křivka). V této sekci navrhuji úpravu gradientního plánovače aby toto podporoval.

Pro odvození použiji kinematický model v souřadnicovém systému cesty (2). Postupuji stejně jako při odvození vztahů pro gradientní plánovač v kartézských souřadnicích. Použitím zjednodušujících vztahů (1), předpokladem konstantní rychlosti v_s ve směru s a použitím diskrétního časového kroku Δt dostanu

$$\begin{aligned} s_{k+1} &= s_k + v_s \Delta t \\ d_{k+1} &= d_k + v_k \sin(\theta_{r,k}) \Delta t \\ \theta_{r,k+1} &= \theta_{r,k} + v_k \left(c_k - \frac{c_{p,k} \cos(\theta_{r,k})}{1 - d_k c_{p,k}} \right) \Delta t \\ c_{k+1} &= c_k + \varepsilon_k \Delta t. \end{aligned} \tag{6}$$

Matice omezení $\boldsymbol{G}(\boldsymbol{X})$ je pak

$$\boldsymbol{G}(\boldsymbol{X}) = \begin{pmatrix} d_{k+1} - d_k - v_k \sin(\theta_{r,k}) \Delta t \\ \theta_{r,k+1} - \theta_{r,k} - v_k \left(c_k - \frac{c_{p,k} \cos(\theta_{r,k})}{1 - d_k c_{p,k}} \right) \Delta t \\ c_{k+1} - c_k - \varepsilon_k \Delta t \end{pmatrix}_{k=1,\dots,N-1}, \\ \boldsymbol{x_s} - \boldsymbol{x_0} \\ \boldsymbol{x_f} - \boldsymbol{x_N} \\ d_{j,flex} - d_j \ , \forall j \end{pmatrix}$$

přičemž vektor stavu $\boldsymbol{x} = [d, \theta_r, c]$. Dále je možné aplikovat stejný postup jako u neupravené verze gradientního plánovače.

Tuto úpravu jsem bohužel nestihl implementovat, mohla by však zlepšit plánování v zatáčkách.

Kapitola 4
Plánování předjížděcí trajektorie

Tato kapitola je zaměřena na plánování trajektorie. První sekce (4.1) je věnována výběru metody pro plánování cesty. Vybraná metoda je blíže popsána v sekci 4.1. Aby bylo možné trajektorii v reálném čase bylo potřeba snížit výpočetní náročnost plánování. Proto jsem navrhl způsob jak s pomocí Bezierových křivek cestu nejprve odhadnout bez použití výpočetně velmi náročného gradientního plánovače. Tento způsob jsem popsal v sekci 4.3.

Pro vytvoření trajektorie je potřeba k cestě přidat rychlostní profil, způsob kterým rychlostní profil vytvářím je popsán v 4.4.

Při plánování cesty je potřeba mít možnost detekovat a predikovat kolizi. Pro predikci kolize se soupeřícím autem je navíc potřeba detekovat a predikovat jeho polohu. Tyto problémy jsem řešil v sekci 4.5.

V případě, že druhé auto není možné předjet je potřeba snížit rychlost pro dodržení dostatečného odstupu. Regulátor rychlosti, který jsem navrhl pro tento případ popisuji v sekci 4.6

4.1 Výběr metody

V literatuře [14–15] je popsáno velké množství algoritmů pro tvorbu předjížděcích a objížděcích manévrů. Mezi nejčastěji zmiňované patří RRT*[1], MPC, Optimal Control[2], Potential fields, lattice search a lokální plánovače.

Gradientní plánovač je výhodné použít v kombinaci s metodou, která cestu sestavuje z delších úseků definovaných průjezdními body. To umožňuje Lokální plánovač a Lattice search, které popíši dále.

4.1.1 Lokální plánovač

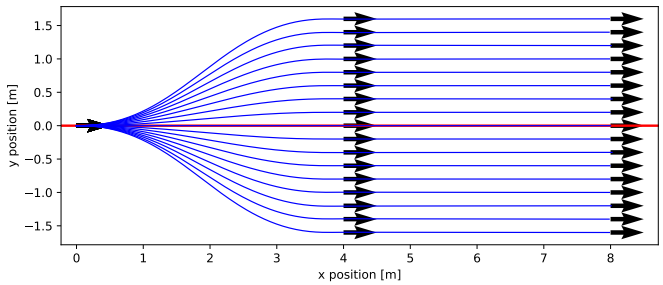
Lokální plánovač [14, 16] při plánování vytvoří n různých cest s počátečním stavem $\boldsymbol{x_0}$ odpovídajícím aktuálnímu stavu, ve kterém se auto nachází a koncovými stavy umístěnými dále ve směru cesty v různých laterální vzdálenostech od cesty. Následně se na základě vybrané ztrátové funkce zvolí nejlepší z nich, ta se následné použije. Vizuální ukázka plánování je na obrázku 4.1.

4.1.2 Lattice search

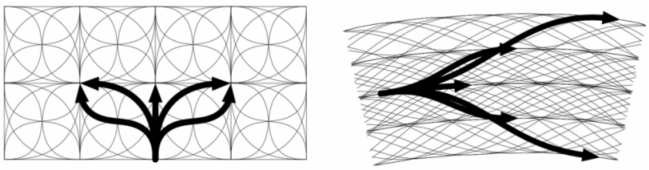
Lattice search hledá nejlepší cestu v tzv. lattice. Lattice je diskrétní prostor stavů, z nichž některé jsou spojené cestami (obrázek 4.2). Úloha nalezení nejlepší cesty je tedy převedena na úlohu prohledávání grafu. K prohledávání může být použit například algoritmus A* [17].

[1] RRT* utváří cestu z velmi krátkých úseků, tyto úseky mohou být přímo vypočteny z diskrétního modelu auta (3), kombinace s gradientním plánovačem tedy není výhodná

[2] do této kategorie lze zařadit samotný gradientní plánovač



Obrázek 4.1. Vytváření cesty pomocí lokálního plánovače, auto je v pozici znázorněné černou šipkou vlevo, šipka znázorňuje směr jízdy, červená křivka znázorňuje globální cestu. Je vytvořeno $n = 17$ cest (znázorněny modře) s různými koncovými stavy, následně je jedna vybrána a použita



Obrázek 4.2. Princip lattice search – formulace jako problém prohledávání grafu [18]

4.1.3 Vybraná metoda

V této práci jsem použil princip lokálního plánovače. Hlavním důvodem, proč jsem zvolil lokální plánovač cesty, je velká výpočetní náročnost gradientního plánovače. Lokální plánovač oproti lattice search vyžaduje vytvoření pouze bezprostředních cest. Cesta vytvořená principem lokálního plánovače je navíc plynulejší.

Zároveň je v prostředí autonomního závodu obtížné předpovídat pohyb protivníků, proto je nutné naplánovanou cestu dynamicky přizpůsobovat aktuální situaci. U delších plánů vytvořených s pomocí lattice search tak hrozí, že v důsledku rychlých změn prostředí bude možné sledovat pouze část vytvořené cesty.

4.2 Lokální plánovač

4.2.1 Implementace

Cílem je naplánovat n různých cest a nejlepší (ve smyslu minimalizace zvolené ztrátové funkce) z nich následně použít. Použitím gradientního plánovače se tato úloha zjednoduší. Vstupem gradientního plánovače je posloupnost (konfigurace) průjezdních bodů, úloha se tedy změní na úlohu vytvoření n různých posloupností (konfigurací) průjezdních bodů, kterými bude možné naplánovat cesty.

Pro plánování cesty s více[1] průjezdními body nepoužívám možnost přímého zadání více průjezdních bodů na vstup gradientního plánovače. Při plánování by pak byl použit stejný souřadnicový systém pro celou cestu, čímž by se zhoršily vlastnosti plánovače v zatáčkách (viz sekce 3.3). Plánování jsem rozdělil na plánování cest mezi každými dvěma po sobě jdoucími průjezdními body zvlášť.

Cesty jsem chtěl vytvořit tak, aby respektovaly koncept „jízdních pruhů“ (viz obrázek 4.2 vpravo). Plánovanou cestu jsem si tak pomyslně rozdělil do dvou částí:

[1] s více než 2 – startovní a koncový stav

- přejezd do vybraného pruhu – část mezi aktuální polohou auta a prvním průjezdním bodem[1]
- jízda v podélném pruhu – část od prvního průjezdního bodu dále

Pro volbu průjezdních bodů jsem s výhodou využil souřadnicový systém globální cesty. Označme aktuální pozici auta $[s_0,d_0]$. Konfiguraci průjezdních bodů jsem zvolil vždy tak, že všechny průjezdní body mají stejnou laterální souřadnicovou složku d_i a jsou rozmístěny v pravidelných intervalech podél globální cesty, tj. podélná složka j-tého bodu posloupnosti je $s_0 + j\Delta s$.

U každého průjezdního bodu je nutné specifikovat také směr jízdy θ_j a křivost c_j. Na cestě zvolme referenční stav $\boldsymbol{x_{ref}} = [x_{ref}, y_{ref}, \theta_{ref}, c_{ref}]$ s nejmenší Euklidovskou vzdáleností k průjezdnímu bodu. Směr jízdy je zvolen stejný jako je směr referenčního stavu, $\theta_j = \theta_{ref}$. Pro křivost cesty jsem odvodil přizpůsobující vzorec

$$c_j = \frac{1}{\frac{1}{c_{ref}} - d_i}, \qquad \frac{1}{c_{ref}} - d_i \neq 0, \quad c_{ref} \neq 0.$$

Ukázka naplánovaných cest je na obrázku 4.1.

Vzhledem k velké výpočetní náročnosti není možné vytvářet cesty skrz všechny konfigurace průjezdních bodů přímo gradientním plánovačem (viz 3.4). Proto je tvar cesty průjezdními body nejprve odhadnut s využitím Bezierovy křivky třetího stupně (viz 4.3). Odhady cest jsou ohodnoceny pomocí ztrátové funkce popsané v sekci 4.2.2 a seřazeny podle výsledků.

Konfigurace průjezdních bodů, pro kterou měl odhad cesty Bezierovou křivkou nejnižší výsledek ztrátové funkce je pak použita pro vytvoření cesty gradientním plánovačem. Pokud je výsledná cesta neprůjezdná, použije se konfigurace s druhým nejlepším výsledkem ztrátové funkce, se třetím, atd.

Ukázka průběhu plánování je demonstrována v následujícím pseudokódu

```
candidates = []
for lateral_distance in range(-1.5,1.5):
      waypoints = Choose_control_points(lateral_distance)
      path = Compute_bezier_curve(waypoints)
      if (not is_feasable(path)) continue
      loss = Loss_function(path)
      candidates.append({loss, waypoints})
sort_by_losses(candidates)


for waypoints in candidates:
       path = gradient_planner(waypoints)
       if (is_feasable(path)) return path
print("No path found")
```

Ve výsledné implementaci jsem pro tvorbu každé cesty použil 4 průjezdní body (včetně startovního a koncového, tzn. $j \in \{0, \ldots 3\}$), které jsou od sebe v souřadnicovém systému cesty vzdáleny o $\Delta s = 2.5$ m.

Cesta je přeplánována pokud aktuální cestu není možné sledovat více než $L_{LOOK\ FORWARD} = 1 + \frac{v}{2}$ metrů, kde v je aktuální rychlost. Cestu není možné sledovat pokud je na ní detekována kolize nebo cesta končí.

[1] sem patří i zařazení zpět po dokončení předjížděcího manévru

4.2.2 Ztrátová funkce

Prvním požadavkem na vytvořenou cestu je, že při jejím sledování nedojde ke kolizi. To je možné otestovat metodou popsanou v části 4.5. Stejným způsobem lze také zjistit vzdálenost od překážky. Cestu, na které byla detekována kolize z výběru nevyřadím úplně, ale budu ji výrazně penalizovat. Cestu vyřadím pouze v případě, že by ke kolizi mělo dojít za méně než vzdálenost $L_{LOOK\ FORWARD}$.

Z nevyřazených cest se vybere cesta minimalizující ztrátovou funkci. Ztrátovou funkci jsem navrhl tak, aby zohledňovala průměrnou křivost cesty, délku cesty, vzdálenost od nejbližší překážky, délku průjezdné části cesty a vážený průměr vzdáleností všech stavů tvořících cestu od překážek. Výpočet hodnoty ztrátové funkce je proveden podle vzorce

$$L(p) = L_c(p) + L_{\mathrm{L}}(p) + L_{\mathrm{GAP}}(p) - W_{\mathrm{NEAR}} \min_{i \in \{0,\dots N-1\}} d(\boldsymbol{s_i}) + W_{LEN} l(p),$$

kde

$$L_c(p) = W_c \frac{1}{c_{MAX} N} \sum_{i=0}^{N-1} c_i, \qquad L_c(p) \in [0, W_c]$$

$$L_{\mathrm{L}}(p) = \begin{cases} 0 & \text{, pokud je cesta průjezdná celá} \\ W_{PEN} + W_L \left(1 - \frac{N_{feasable}}{N}\right) & \text{, jinak} \end{cases}$$

$$L_{\mathrm{GAP}}(p) = -W_{\mathrm{GAP}} \frac{1}{N} \sum_{i=0}^{N-1} d(\boldsymbol{s_i}) \frac{i}{N},$$

p je cesta, $\boldsymbol{s_i}$ je i-tý stav na cestě, $d(\boldsymbol{s})$ je vzdálenost auta ve stavu $\boldsymbol{s}$ od nejbližší překážky. W_c, W_ε, W_{GAP}, W_{NEAR}, W_{PEN} a W_{L} jsou nastavitelné váhy. Konstanty c_{MAX} a ε_{MAX} slouží k normování[1].

4.3 Aproximace Bezierovou křivkou

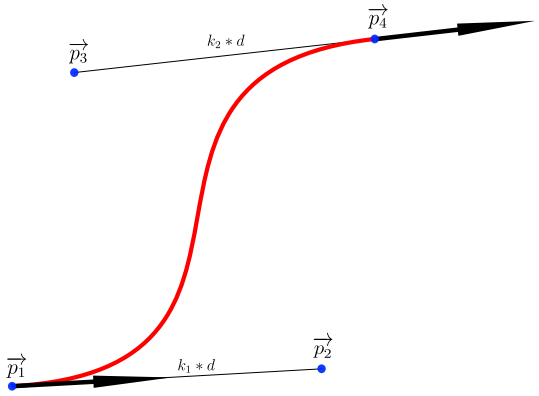
Pro snížení výpočetní náročnosti je možné cestu nejprve odhadnout bez použití gradientního plánovače. K tomu používám Bezierovy křivky třetího stupně [16]. Křivka je definována čtyřmi body – počátečním, koncovým a dvěma body upravující tvar křivky. První nastavitelný řídící bod jsem zvolil na přímce procházející počátečním bodem ve směru požadované orientace, druhý nastavitelný bod na přímce procházející koncovým bodem ve směru požadované orientace. Volba řídících bodů je znázorněna na obrázku 4.3 a).

Pro výpočet křivky jsem použil rovnici [16]

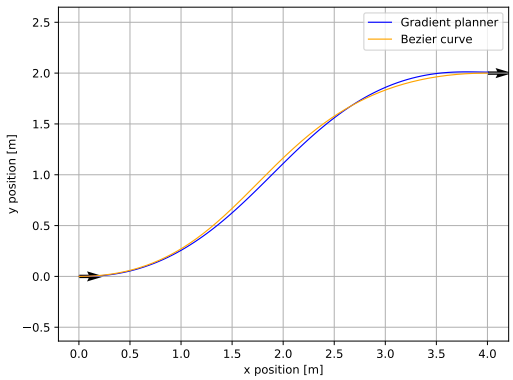
$$\boldsymbol{z}(t) = (1-t)^3 \boldsymbol{p_1} + (1-t)^2 t \boldsymbol{p_2} + (1-t) t^2 \boldsymbol{p_3} + t^3 \boldsymbol{p_4}, t \in [0,1]$$

Srovnání Bezierovy křivky s křivkou vytvořenou gradientním plánovačem je na obrázcích 4.3 b) - d). Rozdíl je znatelný zejména na koncích cest. Průjezdní body pro gradientní plánovač jsou totiž definovány také křivostí cesty. Bezierova křivka tento údaj nepoužívá. Díky tomu při napojování cest vznikají nespojitosti v křivosti cesty. To by se dalo vylepšit použitím Bezierovy křivky vyššího stupně.

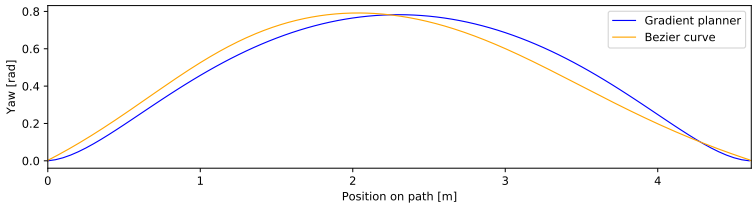
[1] maximální hodnotu křivosti c_{MAX} lze vypočítat z maximálního úhlu natočení kol podle přibližného vztahu (1), maximální hodnota derivace křivosti ε_{MAX} lze s využitím stejného vztahu vypočítat z maximální rychlosti otáčení kol (v rad/s)



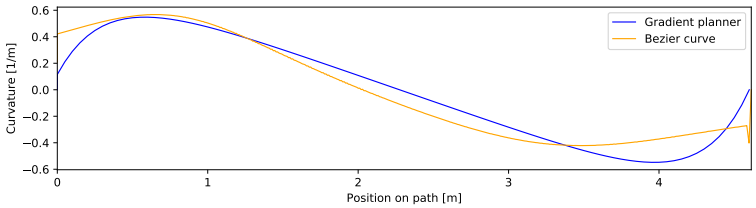
a)Volba řídících bodů



b)Srovnání tvaru cest



c) Graf úhlu natočení auta



d) Graf křivosti cesty

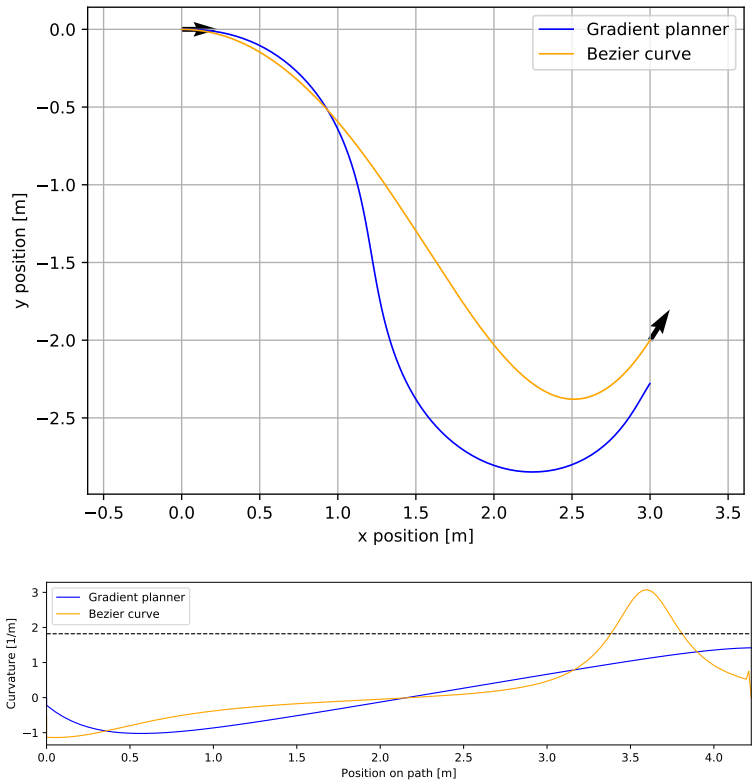
Obrázek 4.3. Ukázka volby řídících bodů Bezierovy křivky je na obrázku a). Srovnání parametrů cesty vytvořené gradietním plánovačem a bezierovou křivkou jsou na obrázcích b) - d)

Existují situace, kdy cesta vytvořená Bezierovými křivkami je vytvořena tak, že není možné ji projet kvůli příliš velkému zakřivení. Příklad, kdy je výhodnější použít gradientní plánovač je na obrázku 4.4. Na stejném obrázku je vidět také jeden z problémů gradientního plánovače. Gradientní plánovač se často do koncového bodu netrefí přesně, kvůli nepřesnému odhadu počtu kroků N. V praxi to však většinou není problém. Než auto dojede na konec cesty, je často vytvořena nová cesta.

4.4 Rychlostní profil

Pro vytvoření trajektorie je k cestě potřeba přidat informaci o rychlostech. U rychlostního profilu v závodě požaduji maximální možnou rychlost. Navíc přidávám omezení na maximální laterální zrychlení $a_{lat,max}$ pro snížení rizika smyku. Maximální laterální zrychlení jsem zvolil $2\frac{m}{s^2}$. Na základě těchto požadavků můžu pro každý stav tvořící cestu nezávisle určit rychlost v_i

$$v_i = \sqrt{\frac{a_{lat,max}}{|c_i|}} \tag{1}$$
$$v_{min} \leq v_i \leq v_{max},$$



Obrázek 4.4. Příklad, kdy je lepší použít gradientní plánovač. Cesta vytvořená Bezierovou křivkou je neprůjezdná, křivost cesty přesahuje maximální křivost cesty, kterou je auto schopno projet ($c > 1.82\frac{1}{m}$, na obrázku vyznačeno přerušovanou čarou).

kde v_{min} a v_{max} jsou horní a dolní limit pro rychlost.

Auto je však schopné dosáhnout podélného zrychlení pouze v intervalu $[a_{min}, a_{max}]$. Pro zrychlení mezi každými dvěma sousedními stavy platí vzorec

$$a_i = \frac{{v_{i+1}}^2 - {v_i}^2}{2s}, \tag{2}$$

kde s je vzdálenost mezi stavy.

Pro přizpůsobení rychlostního profilu tomuto omezení jsem použil dvouiterační algoritmus [3, 19]. V prvním kroku algoritmus prochází cestu ve směru jízdy. Rychlost je při této iteraci omezena pomocí vzorce

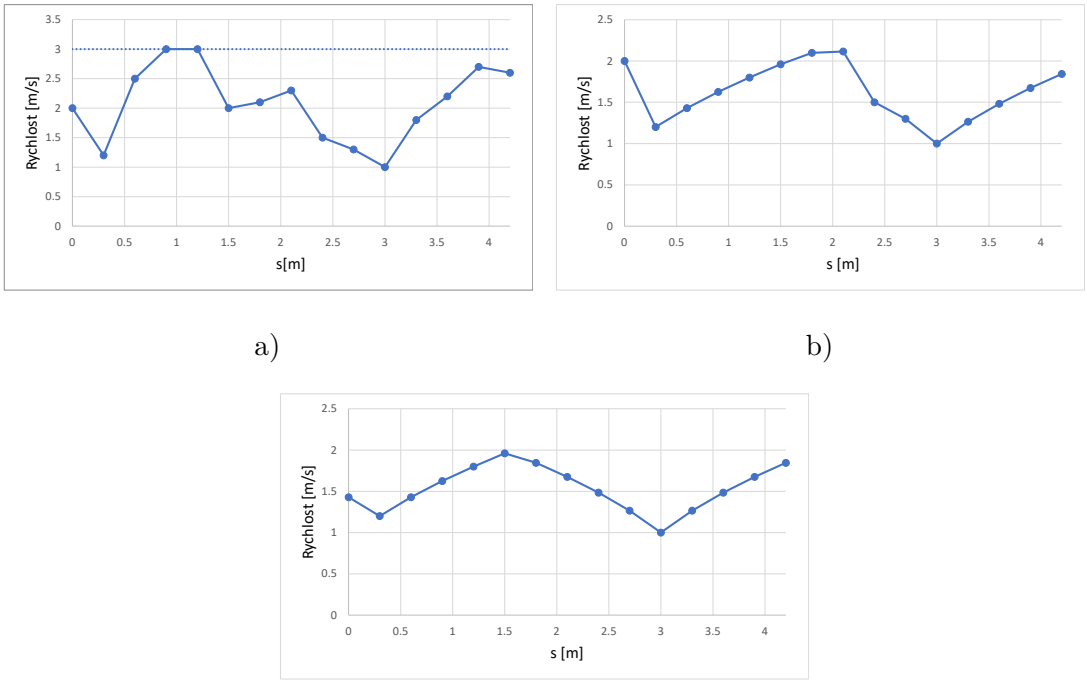
$$v_i = \min\left(v_i, \sqrt{v_{i-1}^2 + 2a_{max}s}\right)$$

Tím je zajištěno, že je dodržena podmínka $a_i \leq a_{max}$.

Druhá iterace algoritmu prochází cestu v opačném směru (proti směru jízdy). Rychlost je při ní upravena pomocí vzorce

$$v_i = \min\left(v_i, \sqrt{v_{i+1}^2 - 2a_{min}s}\right)$$

Tím je zajištěno, že je dodržena podmínka $a_i \geq a_{min}$. Ukázka funkce algoritmu je na Obrázku 4.5.



a) b)

c)

Obrázek 4.5. Demonstrace funkce dvouiteračního algoritmu pro přizpůsobení rychlostního profilu. Na obrázku a) je příklad rychlostního profilu před použitím algoritmu. Na obrázku b) je výsledek po první iteraci algoritmu. Na obrázku c) je výsledek po druhé (zpětné) iteraci algoritmu. Výsledný rychlostní profil splňuje $a_{min} \leq a_i \leq a_{max}, \forall i$

4.5 Predikce kolize

U vytvořené trajektorie je nutné zjistit, zda při jejím sledování nedojde ke kolizi s předjížděným autem nebo se statickou překážkou. Test kolize provádím v každém stavu tvořícím trajektorii $\boldsymbol{s_i}, i \in \{0, \ldots N-1\}$.[1] Způsob jakým detekuji zda ve stavu $\boldsymbol{s_i}$ došlo ke kolizi jsem popsal v podsekci 4.5.1.

Polohu soupeře získávám pomocí programu pro detekci objektů (2.3). Způsob získávání polohy auta z údajů o okolních objektech jsem popsal v podsekci 4.5.2. Změřil jsem také přesnost detekce polohy soupeře (4.5.3). Jednoduchý způsob pro predikci polohy soupeře jsem popsal v podsekci 4.5.4.

Ostatní překážky detekuji přímo s pomocí LIDARu a považuji je za statické bodové překážky[2].

4.5.1 Detekce kolize

Mějme stav na trajektorii. Chceme zjistit zda auto v tomto stavu koliduje s bodovou překážkou o souřadnicích $\boldsymbol{z} = [x\ y]^T$. Auto aproximuji obdélníkem o rozměrech $(l + 2 \times \text{margin}) \times (w + 2 \times \text{margin})$, kde l a w je délka a šířka auta (obrázek 4.6 žlutý obdélník).

Auto s překážkou koliduje, pokud je jeho vzdálenost od přímky podélné se směrem jízdy a procházející středem auta (a_l) menší než polovina šířky obdélníku (d_w) a zároveň

[1] zde předpokládám, že vzdálenost mezi jednotlivými sousedními stavy na trajektorii je dostatečně krátká (menší než $\approx 0.1m$), aby kolize byla detekována i v případě, že by k ní došlo mezi těmito stavy

[2] z LIDARových dat však nejprve odstraním body příslušející druhému autu (aby nebylo zároveň považováno za statickou překážku)

je jeho vzdálenost od přímky kolmé ke směru jízdy procházející středem auta menší než polovina délky obdélníku (d_l). Tedy z rovnice pro vzdálenost bodu od přímky plyne, že bod s autem koliduje pokud jsou splněny nerovnosti

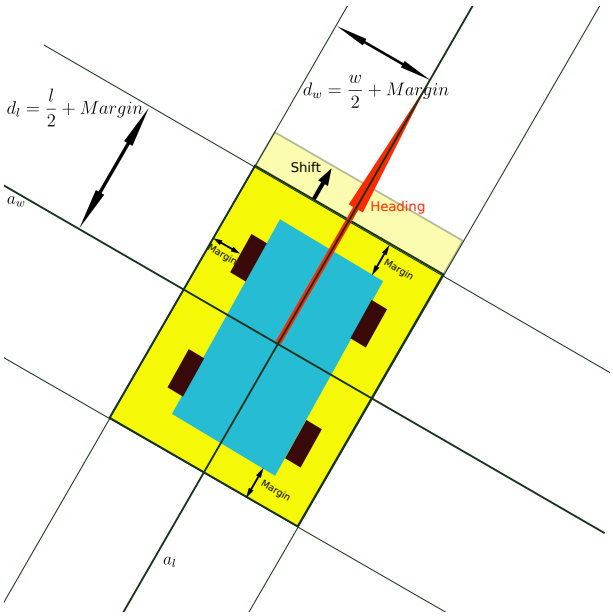
$$\begin{aligned} |\boldsymbol{v_l}^T \boldsymbol{z} - \boldsymbol{v_l}^T \boldsymbol{p_i}| &< \frac{d_w}{2} \\ |\boldsymbol{v_w}^T \boldsymbol{z} - \boldsymbol{v_w}^T \boldsymbol{p_i}| &< \frac{d_l}{2}, \end{aligned} \tag{3}$$

kde $\boldsymbol{v_w}$ je jednotkový vektor ve směru jízdy (ve směru přímky a_l), $\boldsymbol{v_l}$ je jednotkový vektor kolmý na směr jízdy (ve směru přímky a_w) a $\boldsymbol{p_i}$ je poloha středu auta. Je-li orientace auta definovaná pomocí úhlu natočení θ je možné vektory $\boldsymbol{v_l}$ a $\boldsymbol{v_w}$ určit podle vztahů

$$\boldsymbol{v_l} = \begin{pmatrix} \sin\theta \\ -\cos\theta \end{pmatrix} \qquad \boldsymbol{v_w} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

Pro statické překážky je popsaný postup aplikován přímo, pro každou překážku zaznamenanou LIDARem a každý stav auta na testované trajektorii.

Stejný postup lze použít i pro detekci kolize s autem. Auta je opět možné reprezentovat obdélníky o rozměrech $2d_l \times 2d_w$. Aby se obdélníky nepřekrývaly (kolize), nesmí ani jeden z obdélníků mít žádný ze svých rohů uvnitř druhého obdélníku. To je možné ověřit postupným aplikováním vzorců (3) pro každou kombinaci roh-auto.



Obrázek 4.6. Aproximace auta obdélníkem

4.5.2 Získání polohy soupeře

Pro detekci polohy druhého auta používám ROS node vytvořený na Poznan University of technology (viz sekce 2.3). Výstupem nodu je množina všech detekovaných objektů s údajem o rychlosti, poloze a směru pohybu. Postup, kterým z této množiny získávám polohu a rychlost auta jsem popsal v následujícím pseudokódu:

```
if (time_now - last_opponents_state.time > TIMEOUT):
    last_opponents_state = None     //data are too old
new_opponents_state = None

for obstacle in all_detected_obstacles:
    if (obstacle.speed < MIN_SPEED):
        continue
    if (new_opponents_state == None):
        criterion = true
    else if (last_opponents_state == None):
        criterion = (obstacle.speed > new_opponents_state.speed)
    else:
        //obstacle.shift is distance between
        //last detected car position and obstacle
        obstacle.shift = distance(obstacle, last_opponents_state)
        criterion = (obstacle.shift < new_opponents_state.shift)
    if (criterion):
        new_opponents_state = obstacle
        last_opponents_state.time = time_now
```

Dobu TIMEOUT, po které je „zapomenuta“ poloha soupeřícího auta jsem zvolil 2 sekundy. Minimální hodnotu rychlosti druhého auta MIN_SPEED jsem zvolil 0.5 m/s, to umožní vybrat z množiny všech detekovaných překážek nepohyblivé překážky. Často je však detekována nenulová rychlost i u nehybných překážek. Tyto překážky se však v množině detekovaných překážek objeví vždy jen na krátkou dobu. Přidáním kritéria minimální vzdálenosti od poslední detekované polohy auta se podařilo většinu takových překážek z množiny odstranit.

4.5.3 Přesnost detekce stavu soupeře

Přesnost lokalizace druhého auta jsem změřil v simulátoru Stage. Měření přímo na autě nebylo možné, protože nebyla známá přesná poloha druhého auta. Simulátor Stage umožňuje kromě simulace pohybu aut (na základě kinematického modelu), také simulovat LIDARová data. Toho jsem využil v tomto měření.

Reálná měření jsou navíc zašuměna díky nepřesnosti LIDARu (v laboratorních podmínkách je přesnost použitého LIDARu ±40 mm [20]) a nedokonalým odrazovým plochám.

Měření jsem prováděl při jízdě za detekovaným autem. Vzdálenost mezi auty se pohybovala mezi 0.5 a 3 metry. Rychlost aut byla 1 m/s.

Ukázka průběhu chyby detekce polohy, rychlosti a směru jízdy jsou znázorněny na obrázku 4.7. Z opakovaných měření jsem vypočítal také střední kvadratické odchylky (RMSE) detekovaných údajů. Výsledky jsou uvedeny v tabulce 4.1.

Při předjíždění je potřeba chybu detekce uvažovat zvýšením minimální vzdálenosti od auta. Vzhledem k velkým odchylkám směru jízdy jsem se rozhodl při predikci dalšího pohybu předpokládat pohyb podélný se směrem cesty.

V reálných testech často auto není detekováno vůbec nebo je detekováno více pohyblivých překážek. V důsledku toho pak může dojít ke kolizi.

Tabulka 4.1. Chyby detekce auta

	RMSE
Detekce polohy	0.32 m
Detekce směru jízdy	0.45 rad
Detekce rychlosti	0.17 m/s

4.5.4 Predikce polohy soupeře

Při testování kolize s autem je potřeba předpovědět polohu předjížděného auta za čas t, kdy bude předjíždějící auto v testovaném stavu $\boldsymbol{s_i}$. O pohybu soupeře jsem zavedl dva zjednodušující předpoklady:

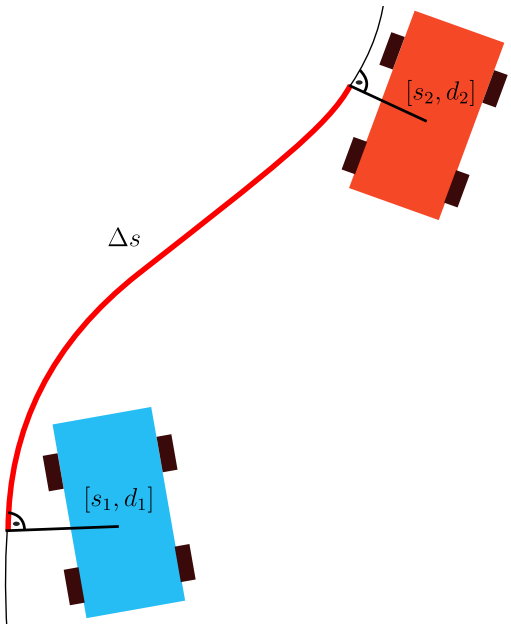
- směr jízdy je v každém okamžiku podélný se směrem globální cesty (údaj z detektoru auta jsem nepoužil kvůli velké nepřesnosti určení směru jízdy, viz sekce 4.5.3)
- auto se pohybuje konstantní rychlostí.

4.6 Přizpůsobení rychlosti soupeři

V případě, že druhé auto není možné předjet[1], plánovač dočasně přizpůsobí rychlost druhému autu tak, aby se udržel nastavený odstup s_{GAP}. Rychlost je přizpůsobována podle vztahu

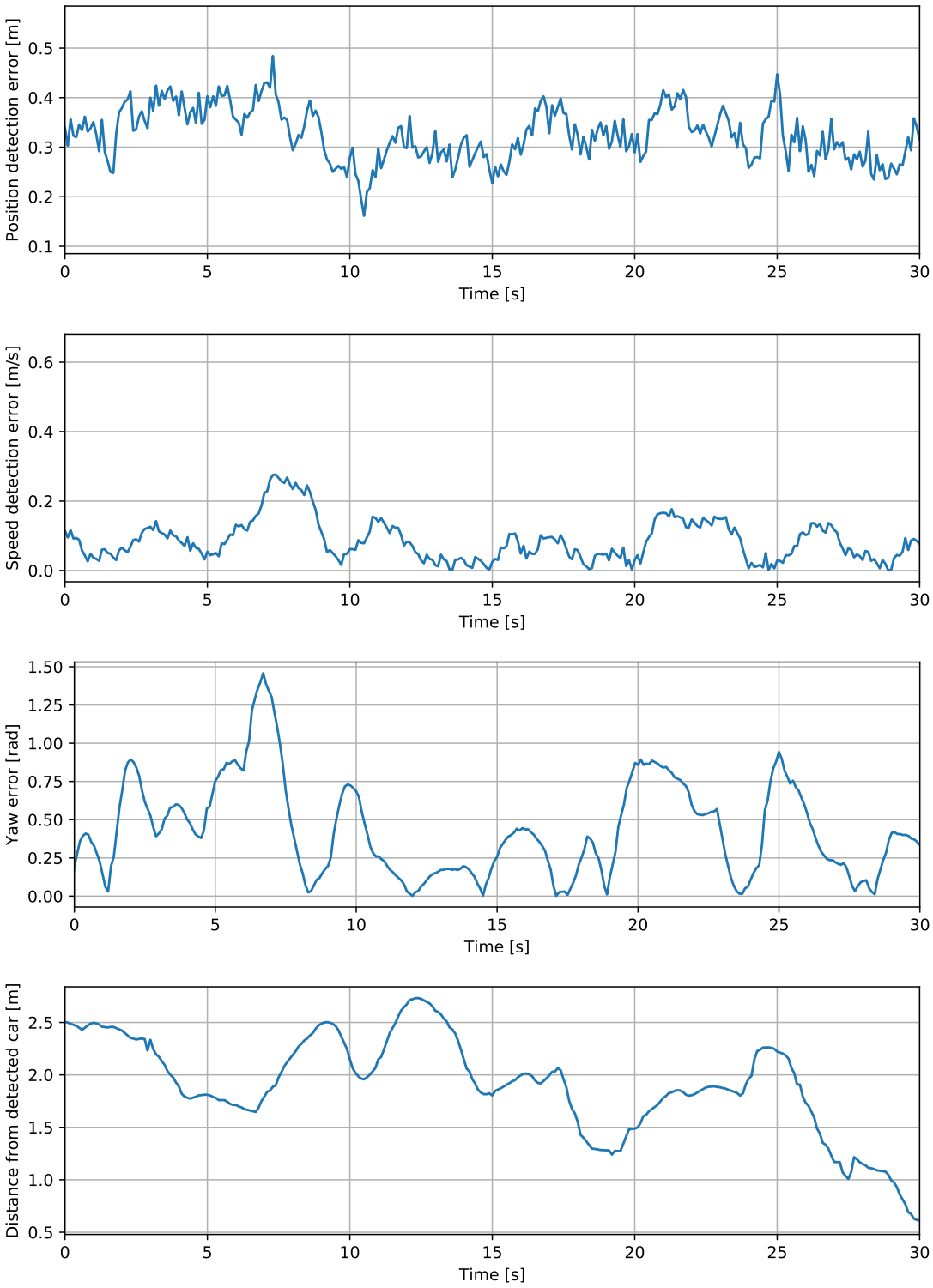
$$v = v_{car2}(1 + K_{GAP}(\Delta s - s_{GAP})), \qquad v > 0,$$

kde K_{GAP} je konstanta, v_{car2} je rychlost druhého auta, Δs je podélná vzdálenost od druhého auta v souřadnicovém systému globální cesty (viz obrázek 4.8).



Obrázek 4.8. Vzdálenost mezi auty v souřadnicovém systému cesty je délka znázorněné červené křivky (Δs)

[1] dráha je příliš úzká, rychlost protivníka je příliš velká, výskyt jiné překážky atp.



Obrázek 4.7. Přesnost detekce polohy auta, první graf znázorňuje vzdálenost mezi detekovanou polohou auta a reálnou polohou, druhý graf znázorňuje absolutní hodnotu rozdílu detekované rychlosti a reálné rychlosti, třetí graf ukazuje průběh chyby odhadu směru jízdy, na posledním grafu je vynesena vzdálenost mezi auty během experimentu

Kapitola 5
Sledování trajektorie

V této kapitole jsem řešil problém sledování vytvořené trajektorie. Otestoval jsem tři různé sledovací algoritmy. Jejich princip je vysvětlen v sekci 5.2. Význam proměnných používaných v těchto algoritmech jsem popsal v sekci 5.1. Strukturu řídící smyčky auta jsem popsal v sekci 5.3.

Pro naladění parametrů regulátorů jsem použil gradientní metodu. Tu jsem blíže popsal v sekci 5.4. Odchylku sledování všech tří sledovacích algoritmů jsem otestoval a porovnal v sekci 5.5.

5.1 Testované proměnné

Referenčním stavem je v této kapitole míněn takový stav na trajektorii, jehož Euklidovská vzdálenost od referenčního bodu na autě je nejkratší. Referenční bod na autě je pro každý ze tří použitých sledovacích algoritmů jinde. V případě algoritmu Stanley je to střed přední nápravy, u regulátoru z autonomního slalomu je to střed auta, pro look forward sledovač je to střed zadní nápravy.

Ve sledovacích algoritmech jsou použity následující proměnné

- odchylka úhlu: $\theta_e = \theta_o - \theta_r$, kde θ_o je orientace auta a θ_r je orientace v referenčním stavu
- vzdálenost od trajektorie d_e: Euklidovská vzdálenost referenčního bodu na autě od referenčního stavu na trajektorii
- referenční natočení kol δ_{ref} vypočítané z křivosti trajektorie v referenčním stavu s využitím aproximujícího vztahu (1)

$$\tan(\delta_{ref}) \approx cW \Rightarrow \delta_{ref} \approx \operatorname{atan}(cW)$$

- v je rychlost auta

5.2 Použité algoritmy

5.2.1 Stanley

Sledovač trajektorie původně prezentovaný v [21] určuje natočení kol podle vztahu

$$\delta(t) = \theta_e(t) + \operatorname{atan}\left(\frac{K d_e(t)}{K_{soft} + v(t)}\right), \tag{1}$$

kde K a K_{soft} jsou nastavitelné parametry. Sledovač je možné dále upravovat pro zlepšení dynamických vlastností [21]. Po prvních testech jsem však použil čistou variantu (1) s vlastními úpravami.

Podle srovnání [22–23] je tento sledovač v některých situacích i přes svou jednoduchost srovnatelný se složitějšími algoritmy.

Kromě čisté verze Stanleyho sledovače jsem vyzkoušel přičíst ke vztahu referenční natočení kol δ_{ref}. Výsledný vztah pro natočení kol je tedy

$$\delta(t) = \delta_{ref}(t) + \theta_e(t) + \operatorname{atan}\left(\frac{K d_e(t)}{K_{soft} + v(t)}\right) \tag{2}$$

5.2.2 Look forward

Metoda představená v [24] je založená na kinematickém modelu auta (viz rovnice 2.4). Budeme-li reprezentovat polohu auta pomocí souřadnic středu zadní nápravy, zjednoduší se rovnice 2.4 na

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \frac{v}{W}\tan(\delta) \end{aligned} \tag{3}$$

změnu polohy auta za čas Δt můžeme aproximovat diskrétním modelem jako

$$\begin{aligned} \Delta x_k = x_{k+1} - x_k &= v_k\cos(\theta_{k,ez})\Delta t \\ \Delta y_k = y_{k+1} - y_k &= v_k\sin(\theta_{k,ez})\Delta t \\ \Delta\theta_k = \theta_{k+1} - \theta_k &= \frac{v_k}{W}\tan(\delta_k)\Delta t, \end{aligned} \tag{4}$$

kde $[x_k,y_k,\theta_k]$ je aktuální poloha auta a $[x_{k+1},y_{k+1},\theta_{k+1}]$ je poloha auta za čas Δt. $\theta_{k,ez}$ je aproximace úhlu natočení auta během kroku, můžeme ji určit vydělením prvních dvou rovnic soustavy (4):

$$\frac{\Delta x_k}{\Delta y_k} = \frac{\cos(\theta_{k,ez})}{\sin(\theta_{k,ez})} \Rightarrow \theta_{k,ez} = \operatorname{atan2}(\Delta y_k, \Delta x_k)$$

Ze soustavy rovnic (4) můžeme vyjádřit v_k a δ_k

$$v_k = \frac{\Delta p_k}{\Delta t} \tag{5}$$

$$\delta_k = \operatorname{atan}\left(\frac{W\Delta\theta_k}{\Delta p_k}\right), \tag{6}$$

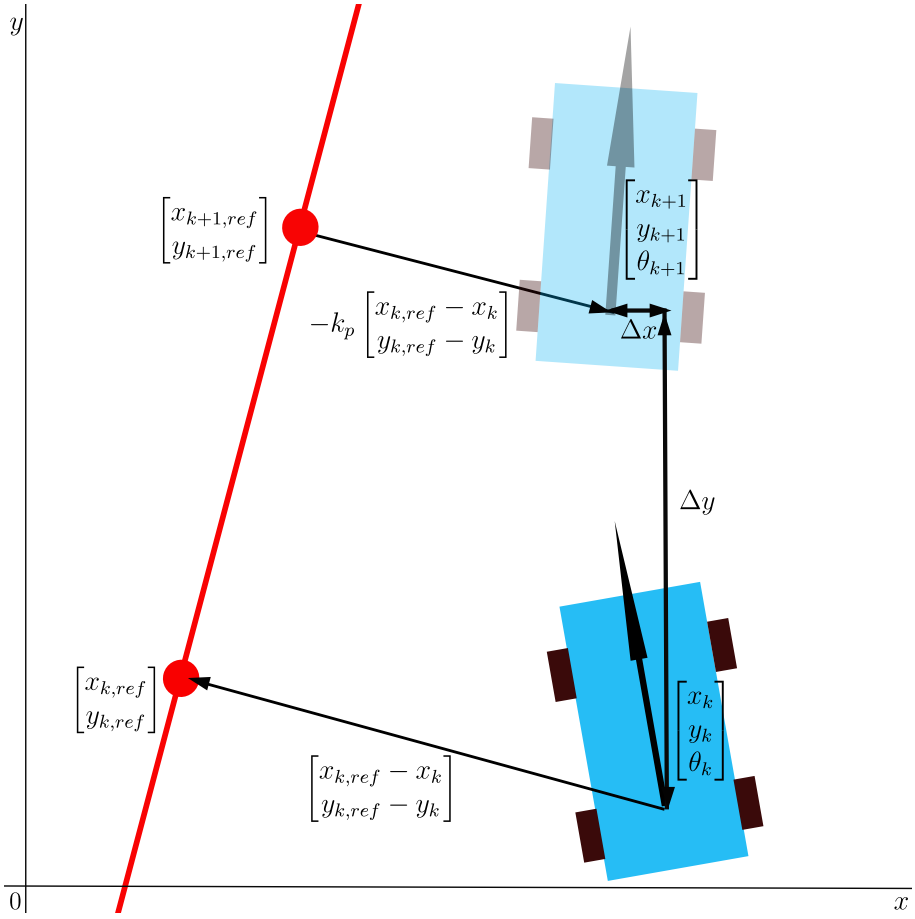
kde Δp_k je vzdálenost mezi body $[x_k,y_k]$ a $[x_{k+1},y_{k+1}]$.

Nyní máme vztah pro výpočet potřebné rychlosti v_k a úhlu natočení kol δ_k pro dosažení vybraného stavu $[x_{k+1},y_{k+1},\theta_{k+1}]$ za čas Δt.

Na trajektorii zvolíme referenční stav o souřadnicích $[x_{k,ref},y_{k,ref},\theta_{k,ref}]$. Dále zvolíme následující referenční stav $[x_{k+1,ref},y_{k+1,ref},\theta_{k+1,ref}]$ o němž předpokládáme, že bude referenčním stavem v příštím kroku. Hodnoty Δx_k, Δy_k a $\Delta\theta_k$ zvolíme

$$\begin{aligned} \Delta x_k &= x_{k+1,ref} - k_p(x_{k,ref} - x_k) - x_k \\ \Delta y_k &= y_{k+1,ref} - k_p(y_{k,ref} - y_k) - y_k \\ \Delta\theta_k &= \theta_{k+1,ez} - k_\theta(\theta_{k,ez} - \theta_k) - \theta_k, \end{aligned} \tag{7}$$

kde $0 < k_p < 1$ a $0 < k_\theta < 1$ jsou nastavitelné parametry. Princip sledovače je znázorněn na obrázku 5.1



Obrázek 5.1. Look forward sledovač trajektorie – určení polohy $[x_{k+1}, y_{k+1}, \theta_{k+1}]$, sledovaná trajektorie je označena červenou křivkou

5.2.3 Sledovač z autonomního slalomu

Regulátor byl navržen v [25] a byl také použitý v autonomním slalomu [3] s gradientním plánovačem. S modifikací navrženou v [3] je natočení kol řízeno podle

$$\delta(t) = \delta_{ref}(t) + A_1(t) + A_2(t) + A_3(t),$$

kde

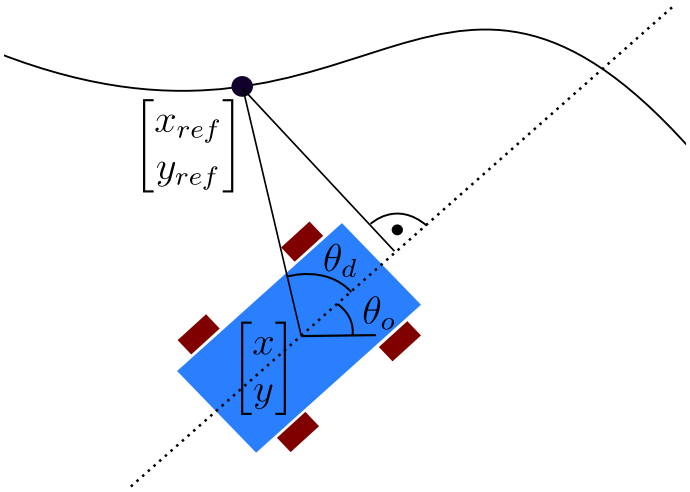
$$\begin{aligned} A_1(t) &= K_\theta \theta_e(t) \\ A_2(t) &= K_{lat} \frac{\sin(\theta_e(t))}{\theta_e(t)} y_e(t) \\ A_3(t) &= K_{fut}(\delta_{ref}(t + t_{fut}) - \delta_{ref}(t)), \end{aligned} \tag{8}$$

kde K_θ, K_{lat}, K_{fut} a t_{fut} jsou nastavitelné parametry, y_e je vzdálenost referenčního bodu na trajektorii od přímky zarovnané se směrem jízdy a procházející středem auta (obrázek 5.2).

Pro výpočet y_e platí

$$y_e = |d_e|.sin(\theta_d) = |d_e|.\sin(\text{atan2}\,(y_{ref} - y, x_{ref} - x) - \theta_o)$$

K tomuto sledovači budu dále referovat jako „sledovač z autonomního slalomu“.

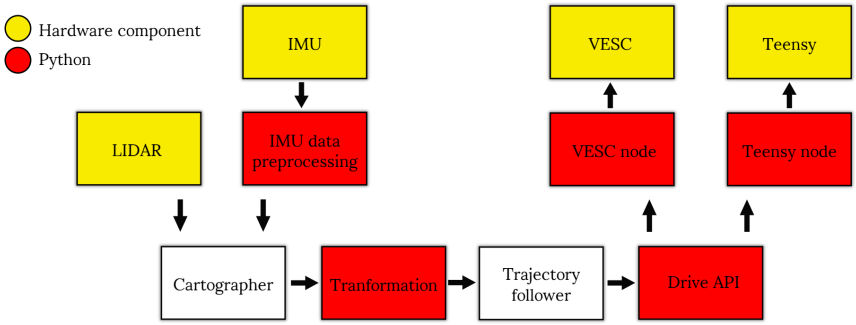


Obrázek 5.2. Geometrie sledovače z autonomního slalomu

5.3 Řídící smyčka

Struktura řídící smyčky auta je zobrazena na obrázku 5.3. V řídící smyčce je použito velké množství ROS nodů. Při komunikaci mezi nimi dochází k nepravidelným zpožděním, což zvyšuje chybu sledování. Problematické je, že téměř všechny základní systémy auta běží v Pythonu, který je nevhodný pro realtime řídící aplikace. V Pythonu běží zpracování dat z IMU, komunikace s regulátorem rychlosti (VESC), transformace mezi souřadnicovými systémemy, Drive API a komunikace se servem natáčejícím kola.

Ve snaze o minimalizaci zpoždění jsem transformace souřadnicového systému a funkce Drive API zakomponoval přímo do ROS nodu implementujícím sledovací algoritmy.



Obrázek 5.3. Struktura řídící smyčky auta v ROSu

5.4 Nastavení parametrů

Pro nastavení řídících konstant jednotlivých regulátorů jsem použil gradientní optimalizátor. Ten v pravidelných intervalech (s frekvencí 10 Hz) vzorkoval vzdálenost od trajektorie $d_{e,i}$ (i značí číslo vzorku) a odchylku úhlu $\theta_{e,i}$. Ztrátovou funkci jsem zvolil tak, aby zahrnovala průměrnou odchylku úhlu a průměrnou vzdálenost od trajektorie. Pro její výpočet jsem použil vzorec

$$\frac{2}{3N}\sum_{i=0}^{N} d_{e,i} + \frac{1}{3N}\sum_{i=0}^{N} \theta_{e,i}. \tag{9}$$

Zaveďme vektor parametrů $\boldsymbol{p} = (p_0\ p_1\ \ldots)^T$, do kterého umístíme všechny laděné parametry. Pro numerický odhad gradientu ztrátové funkce podle vektoru parametrů jsem zvolil metodu středních diferencí (Central differences). Metoda počítá gradient po složkách. Pro každou složku je hodnota parciální derivace odhadnuta ze tří vzorků. Vzorec pro odhad parciální derivace ztrátové funkce $L(\boldsymbol{p})$ podle parametru p_j metodou středních diferencí je

$$\frac{\partial L(\boldsymbol{p})}{\partial p_j} = \frac{L(\boldsymbol{p} + \frac{1}{2}\varepsilon \boldsymbol{l_j}) - L(\boldsymbol{p} - \frac{1}{2}\varepsilon \boldsymbol{l_j})}{\varepsilon},$$

$\boldsymbol{l_j}$ značí vektor nul s jedničkou na j-té pozici, ε je volitelná konstanta, která určuje o kolik se při výpočtu gradientu upraví parametr p_j.

5.4.1 Implementace

Optimalizátor jsem naprogramoval ve samostatném ROS nodu s využitím veřejně dostupné C++ knihovny Gradient Descent [26].

Knihovna vyžaduje objekt s přetíženým operátorem (), jehož návratovou hodnotou je hodnota ztrátové funkce. Funkci přetěžující operátor jsem naimplementoval tak, že nejprve pošle nové nastavení parametrů sledovači (ve formě pole přes ROS).

Parametry jsou zasílány a přijímány ve formě procent z počáteční hodnoty parametru. Výhodou procentuálního nastavení parametrů je, že jsou při inicializaci všechny procentuální parametry normovány na stejnou hodnotu (100 %). Node implementující gradientní optimalizátor také díky tomu může optimalizovat libovolné parametry, bez ohledu na jejich velikost.

Po odeslání nových parametrů se začnou vzorkovat odchylky od trajektorie. Vzorkování je ukončeno po uražení nastavené vzdálenosti. Gradientnímu optimalizátoru je vrácen výsledek ztrátové funkce (9).

5.5 Výsledky

Pro zhodnocení kvality sledování jsem všechny plánovače otestoval na několika trajektoriích s různými rychlostmi a křivostmi. U každého sledovače jsem měřil vzdálenost od trajektorie a odchylku úhlu a vynesl je do grafů (viz obrázky 5.4, 5.5, 5.6).

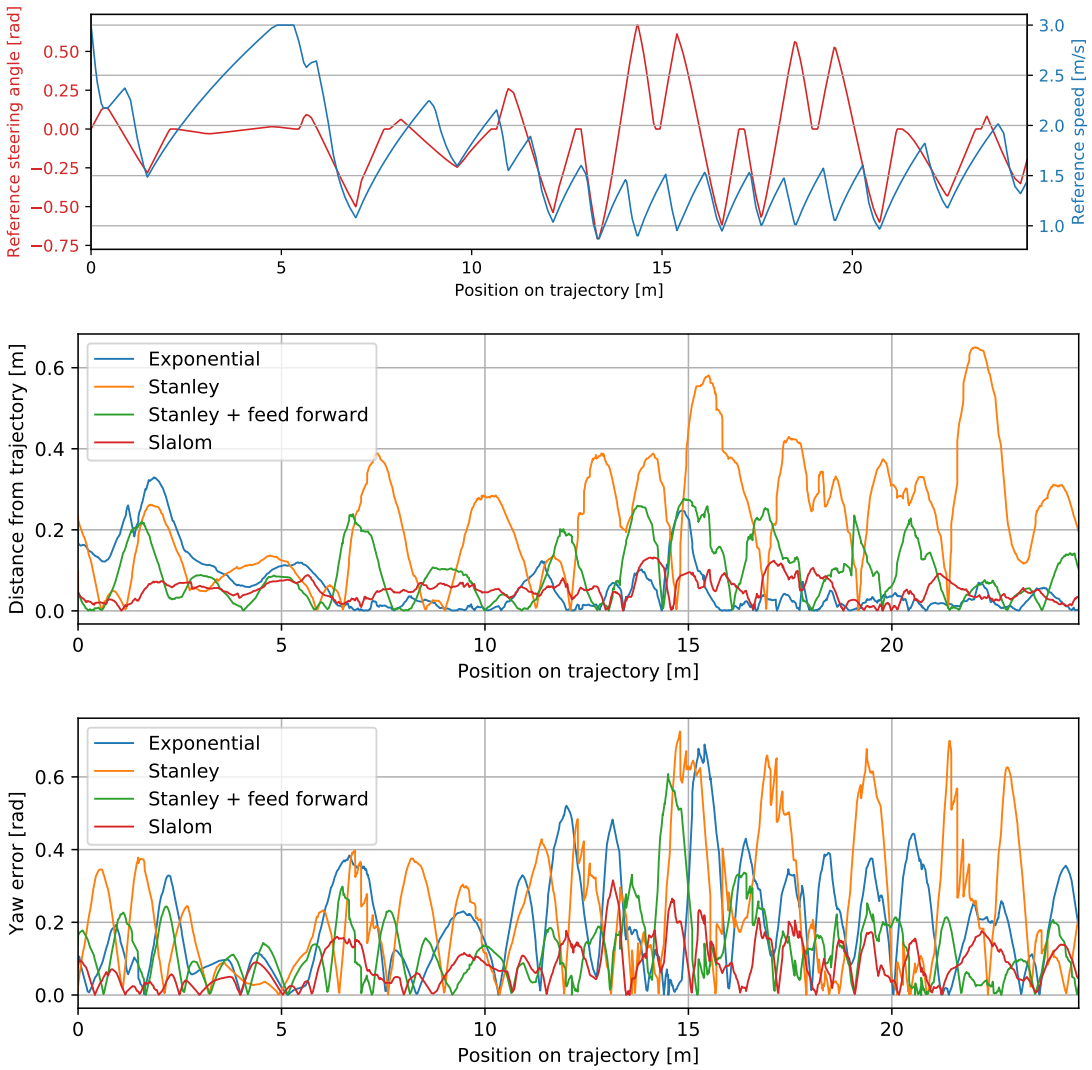
Výstupem každého měření jsou tři grafy. První graf vždy znázorňuje rychlostní profil trajektorie a graf referenčního natočení kol. Tyto údaje jsou součástí trajektorie.

Druhé dva grafy znázorňují chyby sledování jednotlivých sledovačů. Vzdálenost od trajektorie v průběhu sledování je v druhém grafu. Odchylka úhlu je ve třetím grafu. V těchto grafech je chyba každého sledovače označena vlastní barvou – Stanley: oranžová, Stanley s přičteným referenčním úhlem natočení kol: zelená, Look forward: modrá, sledovač z autonomního slalomu: červená.

Nejhorších výsledků dosahuje Stanley a to zejména při sledování trajektorie s ostrými zatáčkami (obrázek 5.4). Vzdálenost od trajektorie dosahuje hodnot větších než 0.6 metru. Odchylky Stanleyho (oranžová) se výrazně sníží přidáním referenčního natočení kol do řídícího vzorce (zelená). I s touto úpravou se však odchylky pohybují v hodnotách do 0.3 metru, na méně zakřivených úsecích do 0.15 metru.

Look forward sledovač dosahuje dobrých výsledků na částech trajektorie s malými změnami křivosti. Nezvládá však rychlé změny křivosti, s odchylkami přesahujícími 0.3 metru.

Z testovaných sledovačů dosahuje nejmenší odchylky sledovač z autonomního slalomu. Odchylka od trajektorie se pohybuje do 15 cm. Při testování předjížděcího algoritmu jsem se rozhodl použít tento regulátor.

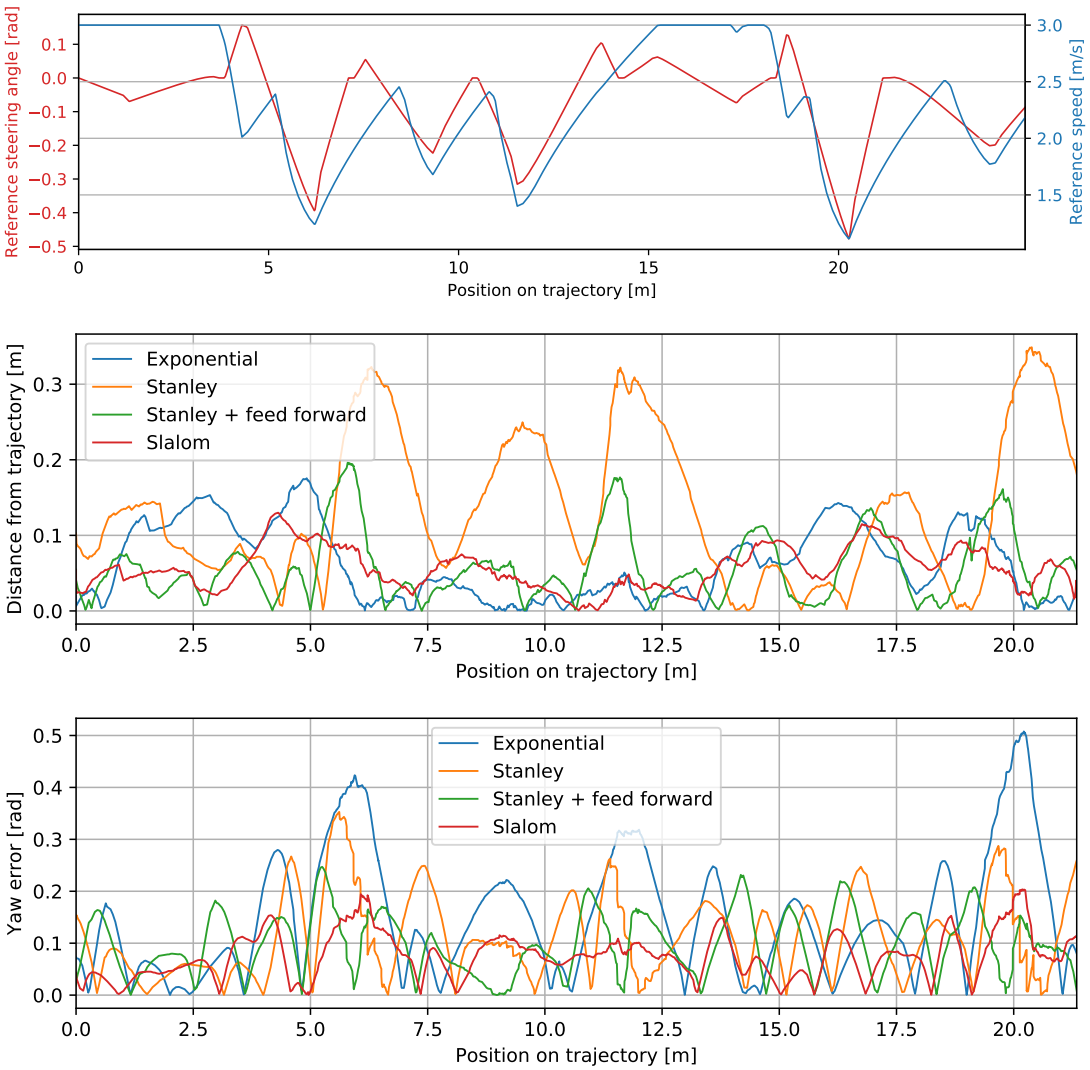


Obrázek 5.4. Okruh navržený tak, aby obsahoval časté a prudké zatáčky. První graf znázorňuje rychlostní profil trajektorie a graf referenčního natočení kol. Druhý graf znázorňuje vzdálenosti od trajektorie. Třetí graf znázorňuje odchylku úhlu.

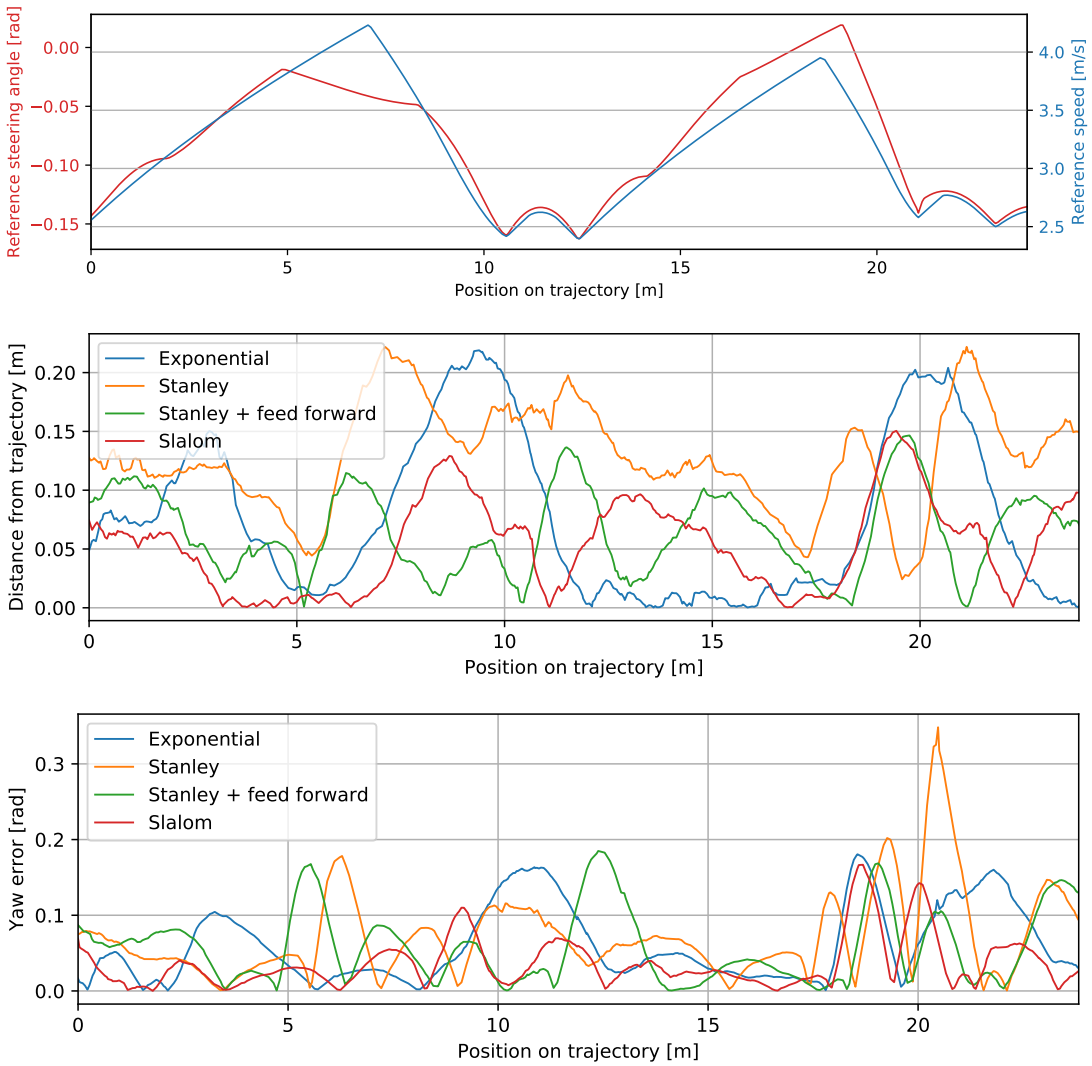
5.5.1 Příčiny chyby sledování

Velký dopad na výslednou chybu má použití jednoduchých sledovacích algoritmů zanedbávajících dynamiku pohybu. Využití složitějšího modelu by mohlo chybu zmenšit. Chyba sledování je způsobena také nepřesností lokalizace (přesnost na 10 cm – viz 2.2).

Dalším faktorem ovlivňujícím chybu je nepravidelné zpoždění řídící smyčky (viz 5.3). Odchylky regulátorů by se mohly snížit implementací všech klíčových systémů (komunikace s VESCem, komunikace se servem, zpracování dat z IMU) ve formě C++ knihoven.



Obrázek 5.5. Okruh s menšími zatáčkami. První graf znázorňuje rychlostní profil trajektorie a graf referenčního natočení kol. Druhý graf znázorňuje vzdálenosti od trajektorie. Třetí graf znázorňuje odchylku úhlu.



Obrázek 5.6. Okruh navržený pro vysokorychlostní jízdu, bez ostrých zatáček. První graf znázorňuje rychlostní profil trajektorie a graf referenčního natočení kol. Druhý graf znázorňuje vzdálenosti od trajektorie. Třetí graf znázorňuje odchylku úhlu.

Kapitola 6
Vyhodnocení

V této sekci jsem celé vytvořené řešení pro předjíždění otestoval. Testy jsem provedl nejprve v simulátoru Stage (6.1). Následně jsem algoritmy otestoval také v reálném prostředí na cílové platformě (6.2).

6.1 Testování v simulátoru

V simulátoru jsem předjíždění testoval nejprve na rovné dráze. Šířka dráhy byla 3 metry, rychlost předjíždějícího auta byla 2 m/s a rychlost soupeře byla 1 m/s. V experimentech na simulátoru nebyla detekce protivníka zatížena chybou, byla použita skutečná poloha a rychlost.

Příklad výsledného předjížděcího manévru je zachycen v grafech na obrázku 6.1. Okamžiky přeplánování trajektorie jsou v grafech znázorněny svislými přerušovanými čarami.

Manévr probíhá podle mé intuitivní představy, jak by předjížděcí manévr měl vypadat. Relativně rychlý přejezd je způsoben přednastavenou podélnou vzdáleností mezi průjezdními body (2.5 metru). Při vyšších rychlostech by bylo rozumné zvážit delší vzdálenost přejezdu.

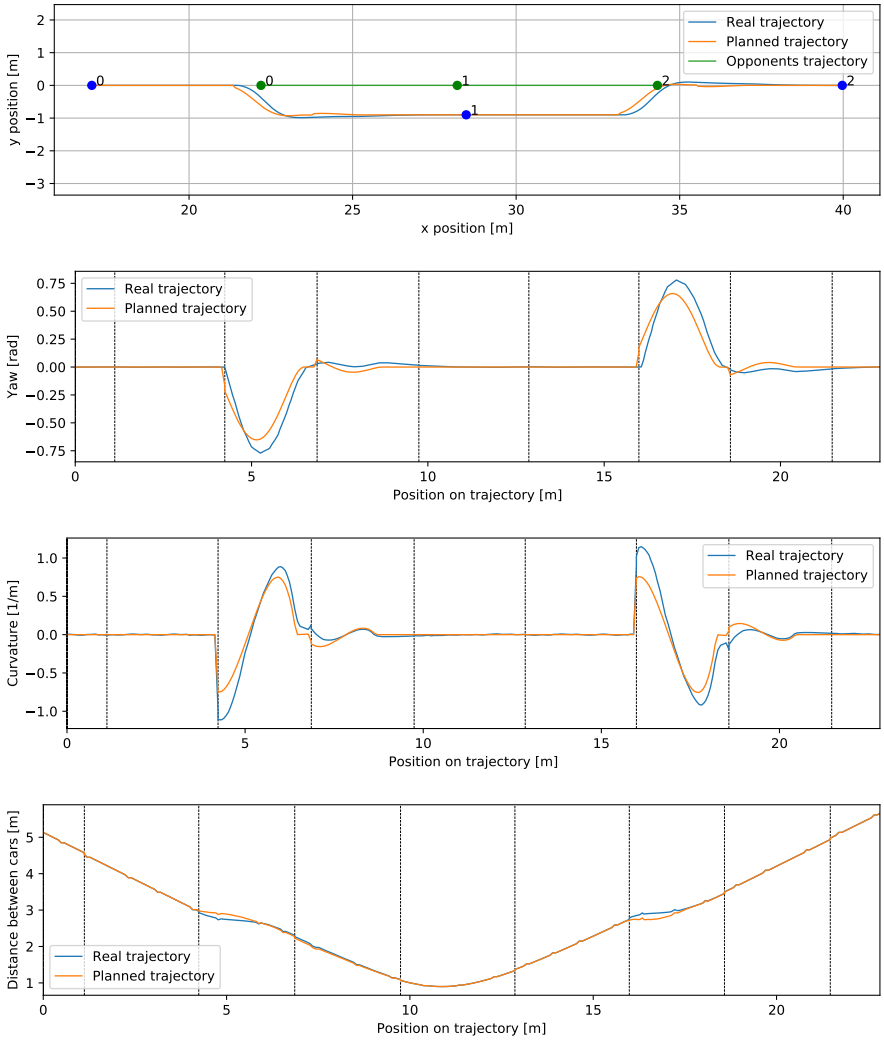
Ukazuje se, že díky zpoždění, které je způsobené vytvářením a přenosem trajektorie do sledovače, vznikají při přeplánování nespojitosti sledované trajektorie. Díky tomu není spojitá ani křivost trajektorie vytvořené gradientním plánovačem.

Vyzkoušel jsem také možnost, kdy je z plánování gradientní plánovač úplně vynechán a je použita trajektorie vytvořená s pomocí Bezierových křivek. Ukázalo se, že takto vytvořené manévry jsou o něco plynulejší a lépe sledovatelné (obrázek 6.2). To je způsobeno hlavně zkrácením doby mezi začátkem přeplánování a přijetím trajektorie sledovačem. Výpočet trajektorie s pomocí Bezierových křivek je rychlý (ve srovnání s ostatními zpožděními zanedbatelný). Ke zpoždění tak dochází už jen díky predikci kolize a přenosu trajektorie do sledovače (dohromady přibližně 40 ms).

Předjíždění jsem testoval také na závodní dráze. Zde předjíždění fungovalo také dobře. Nedocházelo ke kolizím způsobených předjíždějícím autem a soupeře se většinou dařilo téměř okamžitě předjet. Pouze v několika situacích musel algoritmus auto nouzově zastavit, aby zabránil kolizi. K tomu docházelo jen v úzkých (šířka dráhy $\approx$ 1.5 metru) částech dráhy.

6.2 Testování na reálném autě

Předjíždějící algoritmus jsem otestoval na dlouhé rovině (délka 18 m, šířka mezi 2 a 4 metry). Předjížděné auto jsem ovládal ručně. Jeho pohyb byl přibližně přímočarý s rychlostí 1 m/s. Ukázka průběhu předjížděcího manévru je na obrázku 6.4 (zde byla rychlost předjíždějícího auta 2 m/s) a na obrázku 6.6 (rychlost předjíždějícího auta 3 m/s).



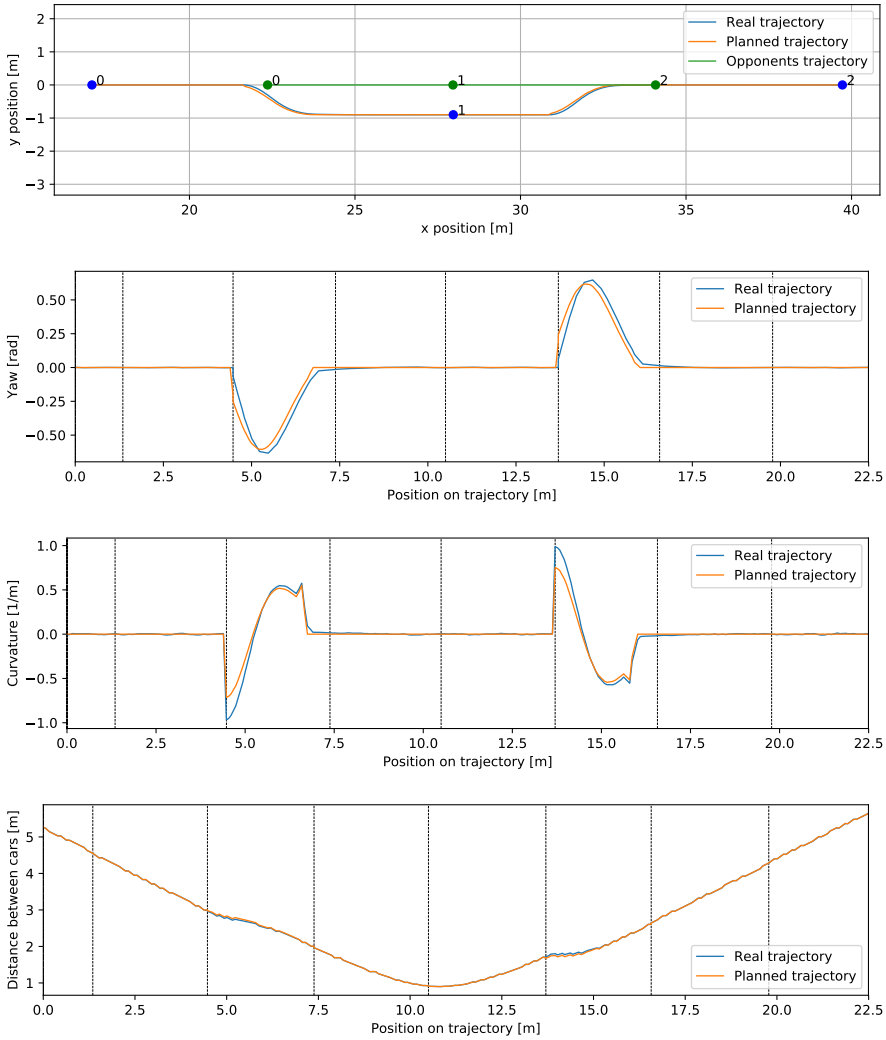
Obrázek 6.1. Průběh předjížděcího manévru v simulátoru – trajektorie vytvářená gradientním plánovačem, svislé přerušované čáry značí okamžiky přeplánování trajektorie. Na prvním obrázku jsou vyznačeny 3 dvojice časově korespondujících bodů.

Oproti výsledkům ze simulace se významně projevila chyba detekce soupeře. I přesto, že předjížděné auto jelo během experimentů rovně, je průběh jeho detekované polohy výrazně zvlněný (zelená křivka na prvním grafu obrázku 6.4 a 6.6). Výsledný manévr proto (a díky dráze proměnlivé šířky) také není přesně rovný a dochází k častějšímu přeplánovávání trajektorie než při testech v simulaci (aby bylo zabráněno kolizi).

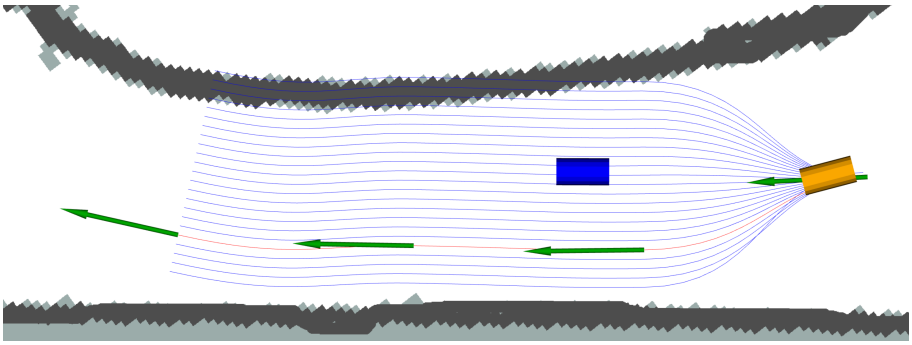
V reálném experimentu se výrazněji projevila chyba sledování trajektorie a to zejména při vyšších rychlostech, kdy auto kolem sledované trajektorie výrazně osciluje (6.6).

Stejně jako v simulaci dochází ke zpoždění při vytváření a přenosu trajektorie do sledovače a při přeplánování vznikají nespojitosti.

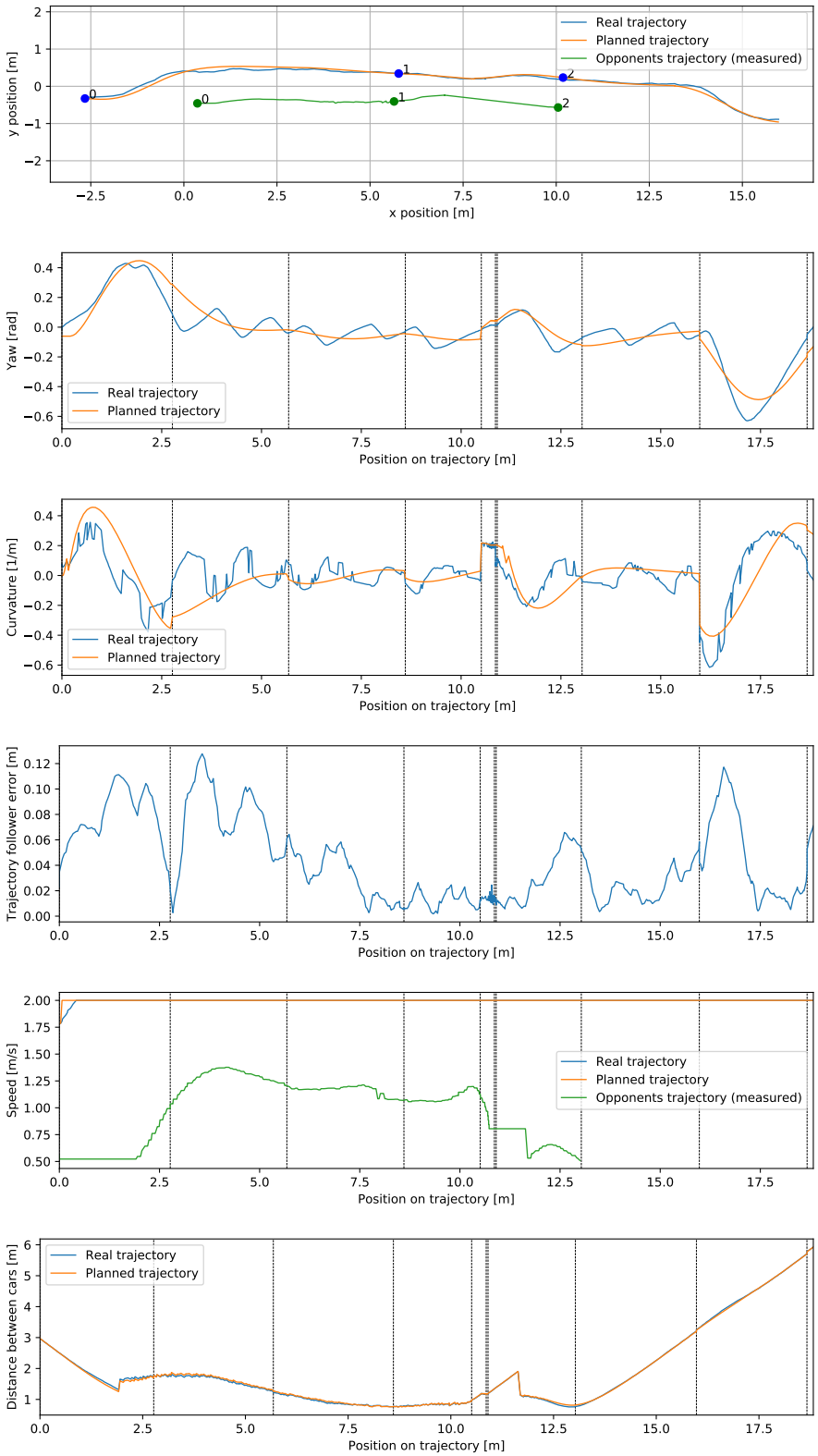
Otestoval jsem opět i vytváření trajektorie přímo s pomocí Bezierových křivek. Ukázka průběhu manévru je na obrázku 6.5. Manévr je srovnatelný s manévrem vytvořeným s pomocí gradientního plánovače.



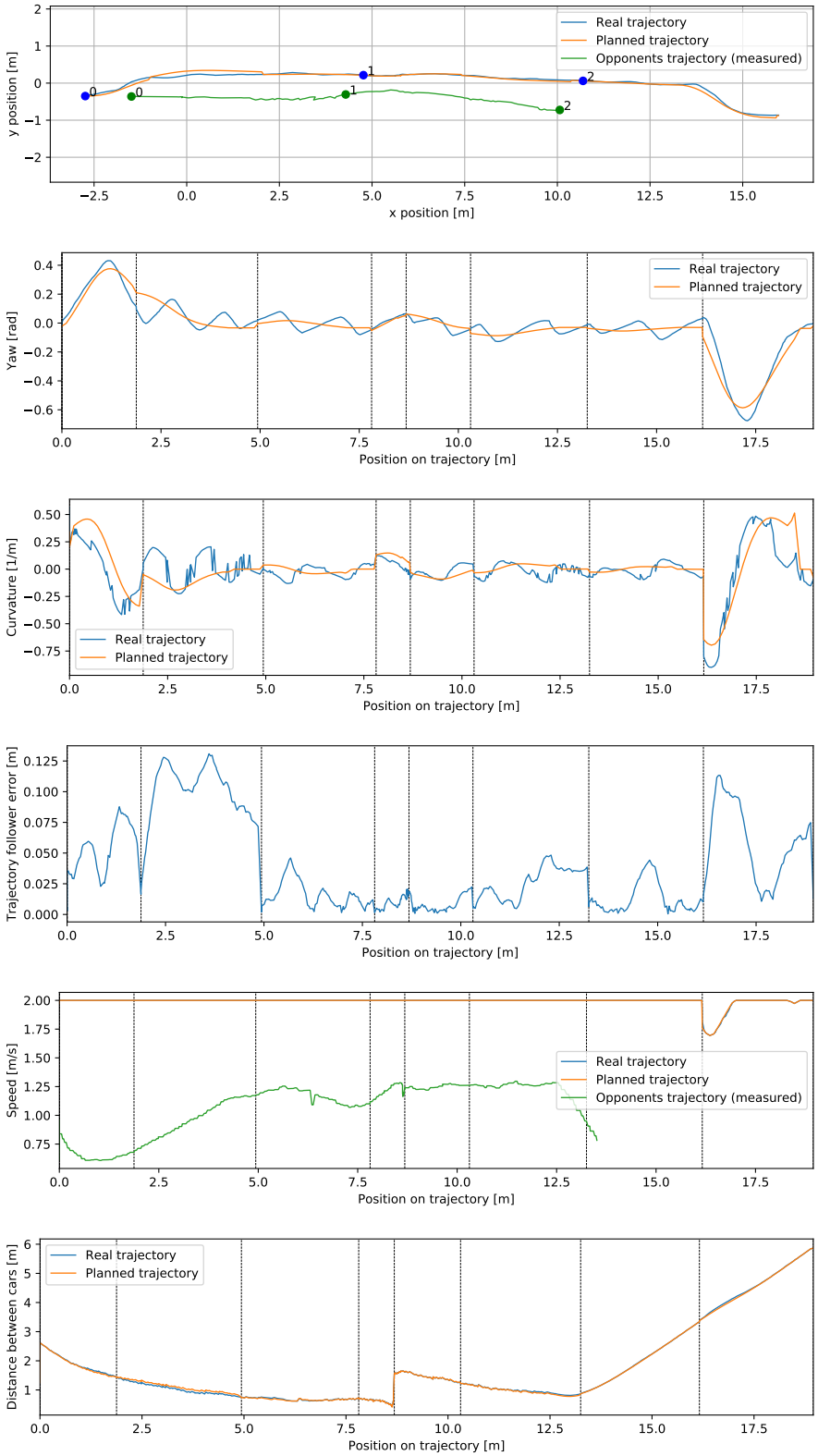
Obrázek 6.2. Průběh předjížděcího manévru v simulátoru – trajektorie vytvářená Bezierovými křivkami, svislé přerušované čáry značí okamžiky přeplánování trajektorie. Na prvním obrázku jsou vyznačeny 3 dvojice časově korespondujících bodů.



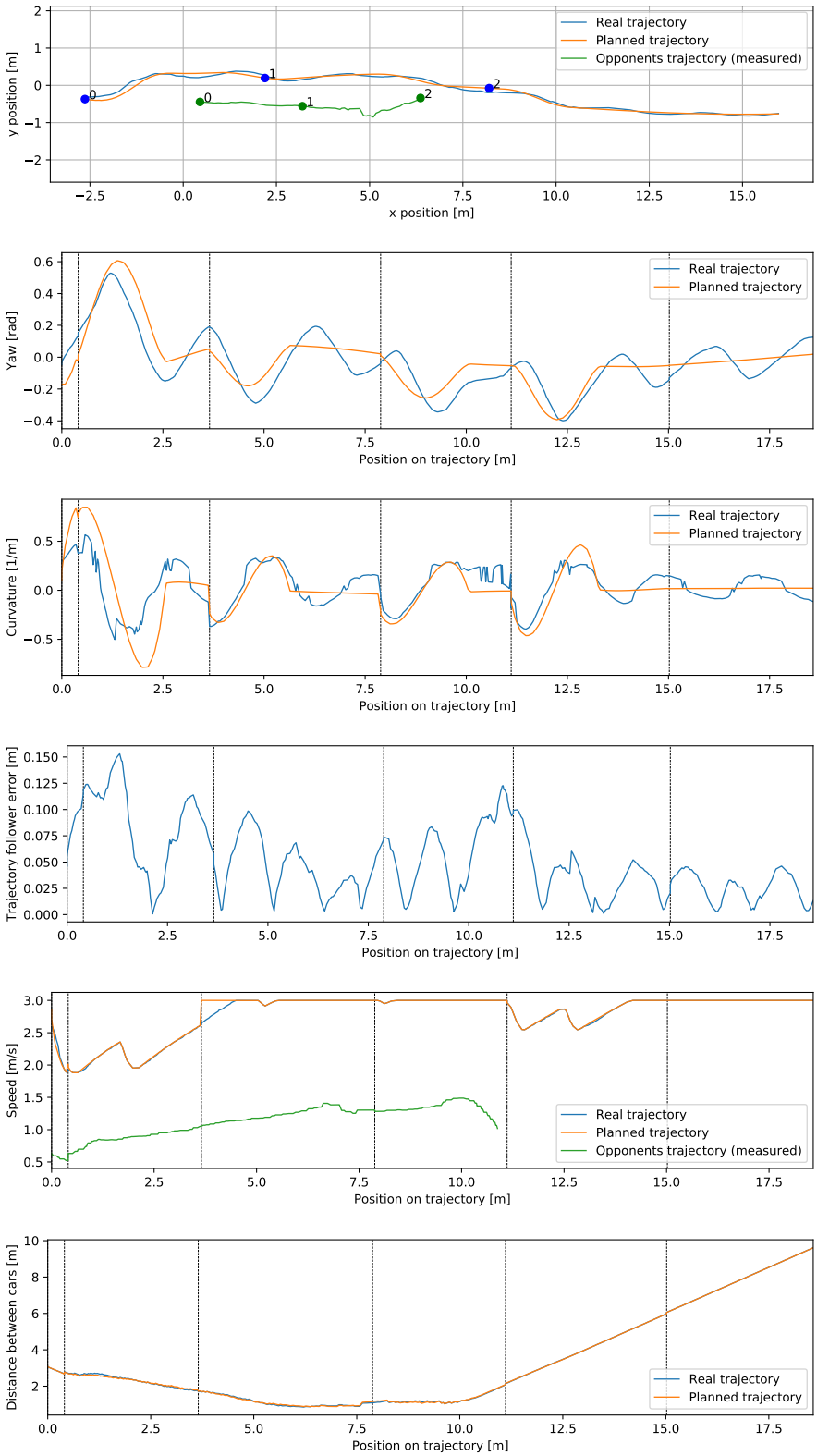
Obrázek 6.3. Vizualizace výběru trajektorie, modře označené trajektorie jsou všechny trajektorie, ze kterých lokální plánovač vybírá. Červeně je označena vybraná trajektorie.



Obrázek 6.4. Průběh předjížděcího manévru – trajektorie vytvářená s gradientním plánovačem, svislé přerušované čáry značí okamžiky přeplánování trajektorie. Na prvním obrázku jsou vyznačeny 3 dvojice časově korespondujících bodů.



Obrázek 6.5. Průběh předjížděcího manévru – trajektorie vytvářená Bezierovými křivkami, svislé přerušované čáry značí okamžiky přeplánování trajektorie. Na prvním obrázku jsou vyznačeny 3 dvojice časově korespondujících bodů.



Obrázek 6.6. Průběh předjíždějícího manévru – rychlost 3 m/s (trajektorie vytvořená s gradientním plánovačem), svislé přerušované čáry značí okamžiky přeplánování trajektorie. Na prvním obrázku jsou vyznačeny 3 dvojice časově korespondujících bodů.

6.2.1 Testování na závodním okruhu

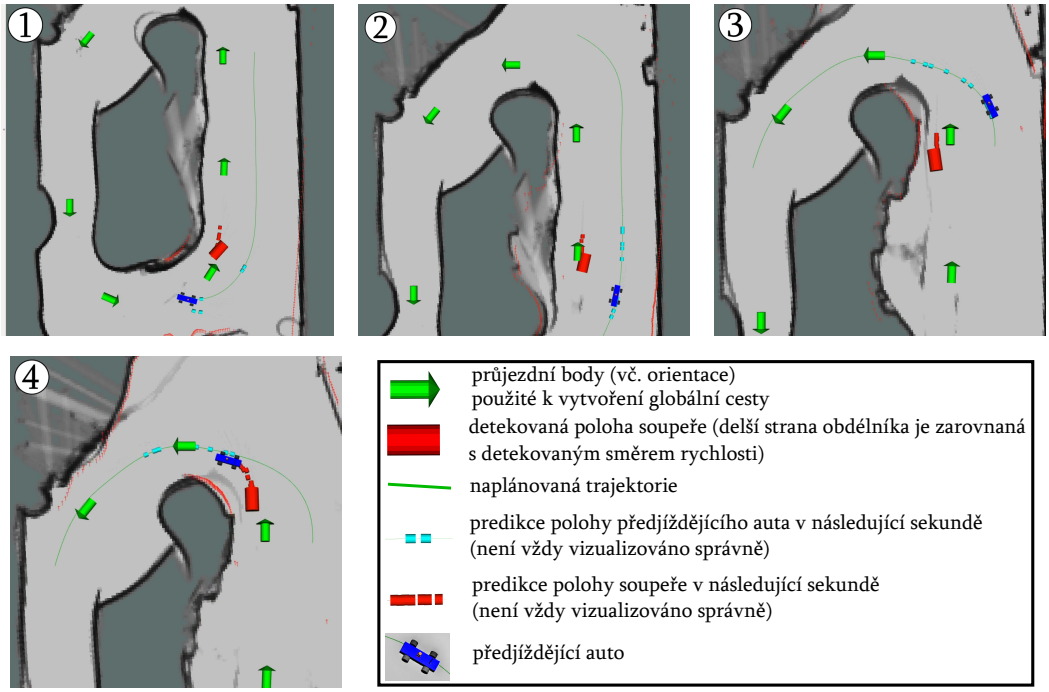
Jízdu jsem otestoval také na závodním okruhu. Rychlost předjíždějícího auta během experimentů byla omezena na 2 m/s. Druhé auto jsem ovládal manuálně v rychlostech přibližně 1 m/s.

Zde předjíždění fungovalo přibližně v polovině případů. Snížení spolehlivosti bylo způsobeno zejména chybnou detekcí a predikcí polohy soupeře.

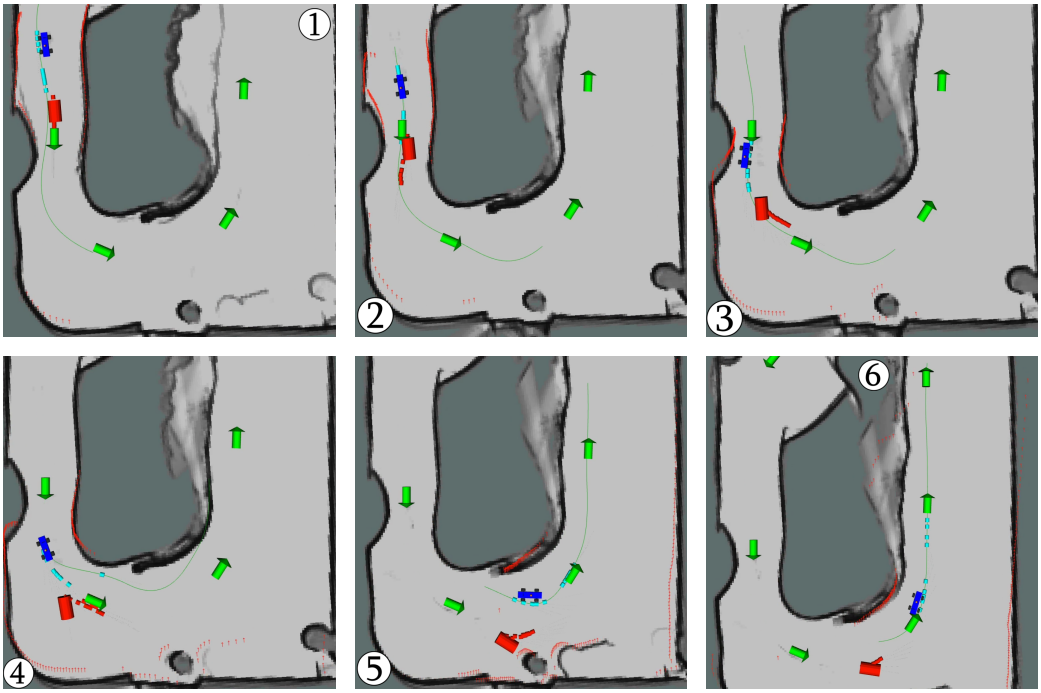
Během experimentů na závodním okruhu došlo k několika kolizím s předjížděným autem. Ve většině případů k nim došlo, jelikož soupeřící auto vůbec nebylo detekováno. Předjíždějící algoritmus potom neuvažoval jeho pohyb a došlo ke kolizi. Příklad, kdy auto vůbec nebylo detekováno je znázorněn na obrázku 6.10.

V několika případech došlo ke kolizi způsobené chybnou predikcí pohybu soupeře. V predikčním algoritmu předpokládám, že je pohyb auta podélný se směrem globální cesty (viz sekce 4.5.4). Tento předpoklad je příliš zjednodušující. Během závodu se auta často pohybují i jinými směry. Predikce pohybu by se mohla zlepšit, pokud bych do predikce směru jízdy zahrnul i detekovaný směr jízdy. Detekovaný směr jízdy je sice nepřesný (viz sekce 4.5.3), ale v případě kdy se auto pohybuje výrazně jiným směrem než je směr globální cesty by predikci zlepšil.

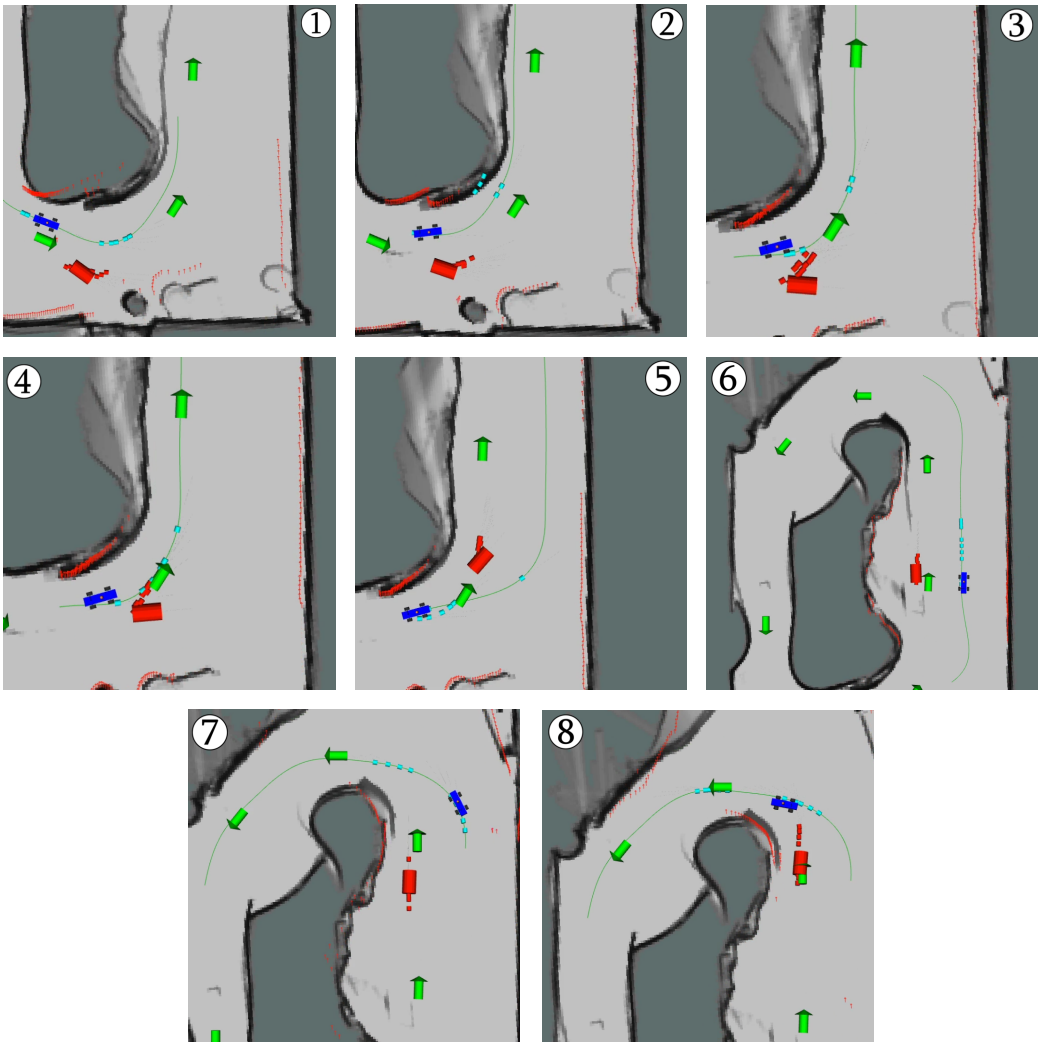
V případech, kdy bylo druhé auto detekováno a jeho směr pohybu se příliš nelišil od směru globální cesty (rozdíl do 30 stupňů), předjíždějící algoritmus fungoval správně. Sekvence obrázků demonstrující průběhy několika uskutečněných předjíždějících manévrů jsou na obrázcích 6.7, 6.8 a 6.9. Tyto sekvence byly pořízeny při jízdě podle trajektorie vytvořené pouze s Bezierovými křivkami. Jízda s gradientním plánovačem byla srovnatelná, ale bohužel jsem při experimentech zapomněl zapnout záznam dat.



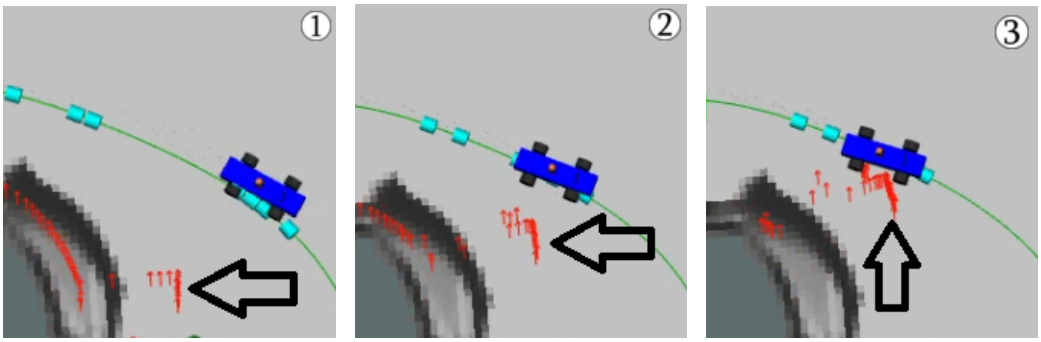
Obrázek 6.7. Příklad předjetí I. Sekvence obrázků demonstruje průběh předjížděcího manévru na závodním okruhu.



Obrázek 6.8. Příklad předjetí II. Význam značek v obrázku je stejný jako na obrázku 6.7.



Obrázek 6.9. Příklad předjetí III. Význam značek v obrázku je stejný jako na obrázku 6.7.



Obrázek 6.10. Na modrém autě je spuštěn předjížděcí algoritmus. Černé šipky ukazují na lidarové body (červené body) odpovídající druhému autu. Druhé auto v tomto případě nebylo detekováno a došlo ke kolizi.

Kapitola 7
Závěr

V této práci jsem se zabýval problémem vytvoření předjížděcího algoritmu pro model auta do soutěže F1/10. Bylo potřeba vyřešit několik dílčích problémů – detekce soupeře, predikce kolize, plánování trajektorie a sledování trajektorie.

Pro detekci soupeře jsem použil hotovou implementaci detektoru objektů [10]. Ukázalo se však, že toto řešení není pro autonomní předjíždění vhodné. Soupeřící auto často vůbec není algoritmem detekováno nebo je detekce zatížena velkou chybou (střední kvadratická odchylka detekce polohy je 33 cm).

Pro vytváření trajektorie jsem použil gradientní plánovač, jehož vstupem jsou průjezdní body. Ty jsem volil metodou lokálního plánovače. Abych snížil velkou výpočetní náročnost gradientního plánovače, navrhl jsem způsob, jak s pomocí Bezierových křivek trajektorii nejprve odhadnout.

Pro sledování vytvořené trajektorie jsem implementoval a porovnal tři jednoduché sledovací algoritmy – Stanley, Look forward sledovač a sledovač přenesený z autonomního slalomu s Porsche Panamera. Parametry sledovacích algoritmů jsem naladil pomocí gradientní metody. Nejmenších odchylek od trajektorie dosahoval algoritmus z autonomního slalomu. Laterální vzdálenost od trajektorie se během sledování pohybovala do 15 cm.

Předjíždění jsem otestoval nejprve v simulaci. Zde algoritmy fungovaly dobře a soupeře se dařilo úspěšně a bezpečně předjíždět. Při testech na reálném autě se předjíždecí manévr dařil přibližně v polovině případů. Snížená spolehlivost byla způsobena zejména chybnou detekcí a predikcí polohy soupeře. Ve vyšších rychlostech (3 m/s) se navíc významněji projevovala odchylka sledování trajektorie.

7.1 Navrhovaná zlepšení

V této části na závěr shrnuji možná zlepšení.

7.1.1 Detekce soupeřícího auta

Pro vytvoření spolehlivého předjížděcího algoritmu je potřeba spolehlivý způsob detekce druhého auta. Použitý algoritmus často druhé auto vůbec nezdetekuje.

Jelikož je mapa okruhu předem známá je možné z lidarových dat nejprve odstranit body odpovídající okolnímu statickému prostředí. Detekční algoritmus by potom ve zbylých datech mohl lépe rozpoznat druhé auto.

Použitý detektor je určený pro detekci překážek, jejichž pohyb je omezen pouze pohybovými rovnicemi hmotného bodu v rovině. Pohyb soupeřícího auta je však omezen rovnicemi kinematického modelu auta[1]. Detekci auta je možné zlepšit přidáním této informace do detektoru. Tím by se částečně potlačil šum v detekci polohy a zpřesnila detekce směru jízdy.

[1] při aproximaci kinematickým modelem

7.1.2 Sledování trajektorie

Pro jízdu ve vyšších rychlostech je potřeba upravit sledovač trajektorie.

Velký vliv na chybu sledování má zpoždění řídící smyčky. Ta je tvořena zbytečně velkým počtem nodů. Zpoždění je možné zkrátit přenesením základních systémů auta do C++. Komunikace s aktuátory by navíc mohla být implementována ve formě knihoven.

Druhý způsob, jak se lépe vypořádat se zpožděním je použití sledovacího algoritmu, který sleduje bod dále na trajektorii, například Pure Pursuit [23].

7.1.3 Predikce pohybu

Předpoklad pohybu protivníka ve směru globální cesty je pro prostředí autonomního závodu příliš zjednodušující. Predikce by se mohla zlepšit kombinací s detekovaným směrem pohybu (viz sekce 6.2). Zjednodušující je také předpoklad konstantní rychlosti druhého auta. Pro lepší predikci rychlosti by mohla být využita také informace o rychlostním profilu globální cesty.

7.1.4 Gradientní plánovač

Ukázalo se, že při použití metody lokálního plánování je předjíždění bez gradientního plánovače (trajektorie vytvářená pouze s pomocí méně výpočetně náročných Bezierových křivek) podobné jako pokud je pro plánování použit.

Vlastnosti gradientního plánovače je možné zlepšit lepší volbou souřadnicového systému. V sekci 3.5 jsem navrhl způsob použití souřadnicového systému cesty. Jeho použitím by se odstranili problémy plánování v zatáčkách.

Pak by bylo možné použít plánování přes více průjezdních bodů najednou. Výsledkem by byla trajektorie s menšími křivostmi a derivacemi křivostí. Taková trajektorie by byla lépe sledovatelná.

Literatura

[1] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio a Marko Bertogna. "F1/10: An Open-Source Autonomous Cyber-Physical Platform". *arXiv e-prints*. 2019, arXiv:1901.08567.

[2] *F1TENTH competition*. (Online) Naposledy navštíveno 17. 5. 2021. `https://f1tenth.org/`.

[3] Jiří Záhora, Michal Sojka a Zdeněk Hanzálek. Cone slalom with automated sports car. (Zatím nepublikováno),

[4] Moritz Diehl a Sebastien Gros. Numerical Optimal Control. 2017, 60-62.

[5] *Robotic operating system*. (Online) Naposledy navštíveno 17. 5. 2021. `https://www.ros.org/`.

[6] *(Online) Nodes in ROS*. (Online) Naposledy navštíveno 17. 5. 2021. `http://wiki.ros.org/Nodes`.

[7] *ROS Stage simulator*. (Online) Naposledy navštíveno 17. 5. 2021. `http://wiki.ros.org/stage`.

[8] *Google Cartographer*. (Online) Naposledy navštíveno 17. 5. 2021. `https://google-cartographer-ros.readthedocs.io/en/latest/`.

[9] David Zahrádka. *Optimization-based control of the F1/10 autonomous racing car*. Diplomová práce, ČVUT. 2020. `http://hdl.handle.net/10467/90254`.

[10] Mateusz Przybyła. *Detection and tracking of 2D geometric obstacles from LRF data*. In: *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*. 2017. 135-141.

[11] Jaroslav Klapálek. *Dynamic obstacle avoidance for autonomous F1/10 car*. Diplomová práce, ČVUT. 2019. `http://hdl.handle.net/10467/83424`.

[12] R Rajamani. *Vehicle Dynamics and Control*. 2006. ISBN 0-387-26396-9.

[13] Wenda Xu, Wen ZhaYao, Huijing Zhao a Hongbin Zha. *A vehicle model for micro-traffic simulation in dynamic urban scenarios*. In: *2011 IEEE International Conference on Robotics and Automation*. 2011. 2267-2274.

[14] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen a Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*. 2015, 60 416-442. DOI https://doi.org/10.1016/j.trc.2015.09.011.

[15] Shilp Dixit, Saber Fallah, Umberto Montanaro, Mehrdad Dianati, Alan Stevens, Francis Mccullough a Alexandros Mouzakitis. Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects. *Annual Reviews in Control*. 2018, 45 76-86. DOI https://doi.org/10.1016/j.arcontrol.2018.02.001.

[16] Liang Ma, Jing Yang a Meng Zhang. *A Two-level Path Planning Method for On-road Autonomous Driving.* In: *2012 Second International Conference on Intelligent System Design and Engineering Application*. 2012. 661-664.

[17] Mihail Pivtoraiko a Alonzo Kelly. *Differentially constrained motion replanning using state lattices with graduated fidelity.* In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008. 2611-2616.

[18] Matthew McNaughton, Chris Urmson, John M. Dolan a Jin-Woo Lee. *Motion planning for autonomous driving with a conformal spatiotemporal lattice.* In: *2011 IEEE International Conference on Robotics and Automation*. 2011. 4889-4895.

[19] John Subosits a J. Christian Gerdes. *Autonomous vehicle control for emergency maneuvers: The effect of topography.* In: *2015 American Control Conference (ACC)*. 2015. 1405-1410.

[20] *Scanning Laser Range Finder Smart-URG mini UST-10LN Specification*. (Online) Naposledy navštíveno 17. 5. 2021.
`https://hokuyo-usa.com/`.

[21] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo a Sebastian Thrun. *Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing.* In: *2007 American Control Conference*. 2007. 2296-2301.

[22] Astrid Rupp a Michael Stolz. *Survey on Control Schemes for Automated Driving on Highways.* In: Daniel Watzenig a Martin Horn, eds. *Automated Driving: Safer and More Efficient Future Driving*. Cham: Springer International Publishing, 2017. 43–69. ISBN 978-3-319-31895-0.
`https://doi.org/10.1007/978-3-319-31895-0_4`.

[23] Jarrod M. Snider. Automatic Steering Methods for Autonomous Automobile Path Tracking. 2009, (CMU-RI-TR-09-08, Technical report, Carnegie Mellon University),

[24] Fernando Auat Cheein a Gustavo Scaglia. Trajectory Tracking Controller Design for Unmanned Vehicles: A New Methodology. *Journal of Field Robotics*. 2014, 31 (6), 861-887. DOI https://doi.org/10.1002/rob.21492.

[25] Gorazd Karer, Mitja Kolbe, Janez Leskovec, Vito Logar a Prof Matko. *Robot ballet.* In: 2021.
`https://www.researchgate.net/publication/228418464_Robot_ballet`.

[26] *Gradient descent cpp (C++ library)*. (Online) Naposledy navštíveno 17. 5. 2021.
`https://github.com/Rookfighter/gradient-descent-cpp`.

Příloha A
Terminologie

A.1 Pojmy

stav	množina parametrů jednoznačně popisující polohu a orientaci auta v prostoru a křivost jízdy
cesta	naplánovaná posloupnost n stavů
trajektorie	cesta s rychlostním profilem
nejbližší referenční stav	stav na cestě/trajektorii, jehož Euklidovská vzdálenost od zvoleného referenčního bodu na autě je nejkratší
globální cesta	předem známá cesta celým závodním okruhem
křivost cesty/trajektorie	křivost v nejbližším referenčním stavu cesty/trajektorie
směr cesty/trajektorie	orientace v nejbližším referenčním stavu cesty/trajektorie

A.2 Zkratky

MPC	Model predictive control
RMSE	Root Mean Square Error
LIDAR	Light Detection and Ranging
IMU	Inertial Measurement Unit
VESC	Vedder Electronic Speed Controller
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping