Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

# BAKALÁŘSKÁ PRÁCE

## Automatická tvorba mapy závodní trati pro autonomní model RC auta

Plzeň 2022 Petr Kuchař

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Petr KUCHAŘ** |
| Osobní číslo: | **A19B0360P** |
| Studijní program: | **B0714A150005 Kybernetika a řídicí technika** |
| Specializace: | **Automatické řízení a robotika** |
| Téma práce: | **Automatická tvorba mapy závodní trati pro autonomní model RC auta** |
| Zadávající katedra: | **Katedra kybernetiky** |

## Zásady pro vypracování

1. Realizace senzorové platformy pro model autonomního RC auta.
2. Seznámení s metodami automatické tvorby mapy prostředí ze senzorových dat.
3. Návrh a realizace systému řízení autonomního RC auta.
4. Návrh a realizace systému pro automatickou tvorbu mapy závodní trati.

# Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

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

V Plzni dne 20. května 2022 . . . . . .

vlastnoruční podpis

## Poděkování

Tímto bych chtěl poděkovat panu Ing. Miroslavu Flídrovi, Ph.D. za odborné vedení mé práce, poskytování cenných rad a připomínek. V neposlední řadě za veškerý věnovaný čas. Rád bych také poděkoval svým rodičům za podporu během celého studia.

# Abstract

This bachelor thesis deals with the design of a control system for an autonomous RC car. The goal is to design a control system that will be able to drive through the race track autonomously. In the first part, the problem of the design of the control system for the race track passage will be formulated. In the next part, the used robotic platform will be presented. The next section will deal with the automatic control of the autonomous racing vehicle. In the following section, the problem of mapping and localization in a race track will be introduced. The last theoretical part will introduce the used software framework and the following design of the control system. In the end, the experiments and the results obtained will be presented.



**Keywords**: robot, ROS, autonomous vehicle driving, mapping and localization



# Abstrakt

Tato bakalářská práce se zabývá návrhem řídicího systému autonomního RC autíčka. Cílem je navrhnout řídicí systém, který bude schopen autonomně projet závodní trať. V první části bude formulován problém návrhu řídicího systému pro průjezd závodní trati. V další části bude představena použitá robotická platforma. Další část se bude zabývat automatickým řízením autonomního závodního vozidla. V následující části bude představen problém mapování a lokalizace v závodní trati. Poslední teoretická část představí použitý softwarový rámec a následný návrh řídicího systému. Nakonec budou představeny experimenty a dosažené výsledky.



**Klíčová slova**: robot, ROS, řízení autonomního vozidla, mapování a lokalizace

# Obsah

# Kapitola 1

# Úvod

Motivací pro vznik této bakalářské práce byla především soutěž F1/10 [2]. Jedná se o soutěž, kde mezi sebou závodí standardizovaná 1:10 zmenšená autonomní závodní vozidla ve formě RC autíčka. Této soutěže se účastní především vysokoškolští studenti z celého světa. Víc než soutěž je však F1/10 komunitou výzkumníků, inženýrů a nadšenců autonomních systémů. Původně byla založena na Pensylvánské univerzitě [7] v roce 2016, ale od té doby se už rozšířila do mnoha institucí po celém světě. Jejich posláním je podporovat zájem, vzrušení a kritické myšlení o stále více všudypřítomné oblasti autonomních robotických systémů [2].

Pro realizaci autonomních robotických systému je důležitým krokem správný výběr senzorů. Senzory zde plní hlavní část pro získávání dat z prostředí. Robot obvykle obsahuje více než jeden senzor. Každý senzor totiž obsahuje nějakou chybu měření. Ty se částečně eliminují buď to právě použitím více senzorů a fúzí dat, které poskytují nebo použitím různých algoritmů. Další důležitou částí je zajištění správných akčních členů. Pro různé autonomní roboty se opět používá různých akčních členů. Každý autonomní systém pak obsahuje jednu či více řídicích jednotek, které plní funkci řízení akčních členů či zpracovaní informací ze senzorů. Na řídicí jednotce se pak nachází software, který informace zpracovává a následně pomocí nějakého algoritmu například ve formě regulátoru posílá signály na akční členy. Takový autonomní robot se často pohybuje v nějakém neznámém prostředí. Pokud však robot nemá nějakou informaci o jeho poloze v tomto prostředí například z GPS, musí si robot pro jeho následné autonomní řízení, nejdříve zmapovat prostředí a zároveň se ve vzniklé mapě lokalizovat.

Hlavním cílem práce je tedy navrhnout řízení robota, které bude schopné projet závodní trať. U zavodní trati se předpokládá, že bude jednoduše ležet v jedné rovině a bude uzavřená. Pro splnění cíle průjezdu závodní trati je potřeba nejdříve mít či postavit nějakého robota založeném na šasi RC autíčka. Zde se bude vycha-

zet z již postaveného robota. Dále pak vytvořit automatický systém tvorby mapy závodní trati. Po vytvoření systému tvorby mapy je nakonec potřeba navrhnout takový systém, který bude schopný tuto zavodní trať na základě informace o prostředí projet.

Tato práce se tedy bude ve druhé kapitole zabývat formulací problému. Především ve formě nadefinování problému. Následně bude představena použitá robotická platforma. Budou zde popsány jednotlivé mechanické struktury a použité senzory či akční členy. Ve čtvrté kapitole bude představeno řízení modelu autonomního vozidla. Budou zavedeny jednotlivé souřadné systémy a jejich transformace mezi sebou. Následně bude představen zjednodušený kinematický model auta. Nakonec se tato kapitola bude zabývat použitým řídicím algoritmem. Pátá kapitola se bude zaměřovat na lokalizaci a mapování prostředí. Budou představeny algoritmy pro lokalizaci a poté bude ukázána a popsána typická dvojrozměrná reprezentace mapy. Poslední teoretická část se bude zabývat použitým řídicím softwarem, jeho strukturou a použitými balíčky, které byly důležité pro ovládání autonomního robotického systému. Poslední část předvede dosažené výsledky z experimentů a jejich zhodnocení.

# Kapitola 2

# Formulace problému

Cílem této práce je navrhnout autonomní řízení robota pro projetí závodní trati. Pro splnění takového cíle je však zapotřebí splnit pár podcílů, které jako celek umožní projet zavodní trať. U trati se předpokládá, že leží v rovině, je uzavřená a má nějaké dostatečně vysoké ohraničení trati, které bude vyšší než auto samotné. Dále se předpokládá, že se robot bude pohybovat po trati sám, tudíž se tam nebude vyskytovat žádná dynamická překážka, která by se mohla v průběhu měnit.

V této práci se vycházelo z již postaveného robota. Jedním z podcílů je proto seznámit se s touto robotickou platformou a popřípadě si tohoto robota dle možností trochu upravit. Dále je potřeba robota nastavit tak, aby byl robot schopen přijímat správně data ze senzorů, správně generovat akční členy a v neposlední řadě se dal ovládat. Zde byl k tomuto účelu pro jednoduchost použit celosvětově známy softwarový rámec ROS (Robot Operating System) [35]. ROS je v podstatě soubor softwarových knihoven a nástrojů vhodný pro vývoj robotických aplikací.

Po správném nastavení robotu je pak zapotřebí pro následný průjezd trati rozšířit schopnosti robota. Vzhledem k tomu, že pro robota jsou informace o prostředí závodní trati neznámé, je potřeba aby byl robot schopen si tyto informace získat ve formě mapy. Proto je jedním z dalších cílů práce vytvořit systém automatické tvorby mapy. Vzhledem k předpokladům o trati, které byli stanoveny bude bohatě stačit vytvořit jednoduchou 2D mapu, která bude značit, kde se robot může pohybovat. Pokud robot zná prostředí je pro jeho následné řízení potřeba znát jeho polohu v mapě. Polohu však nelze dostatečně reprezentovat čistě ze senzorů, vzhledem k chybám, které můžou poskytovat. Proto bude ještě zaveden samotný systém určování polohy a orientace v mapě.

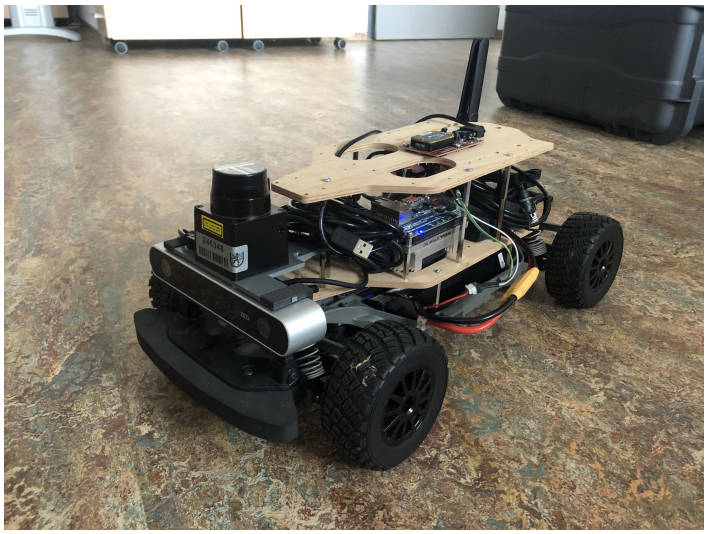
Když už je robot schopný se v mapě lokalizovat je následným krokem již samotné vytvoření trajektorie, kterou bude robot sledovat. Cílem proto bude dále vytvořit jednoduchou trajektorii, podle které bude robot schopný trať projet. Zde

bude využito předpokladu, že trať je uzavřená. Nakonec je potřeba navrhnout řídicí algoritmus, který bude schopný na základě jeho aktuální polohy generovat takové akční zásahy, aby byl robot schopný trajektorii sledovat. K návrhu takového řídicího algoritmu je však zapotřebí mít model řídicího systému, v tomto případě matematický model auta. Takový model je proto potřeba získat.

# Kapitola 3

# Robotická platforma

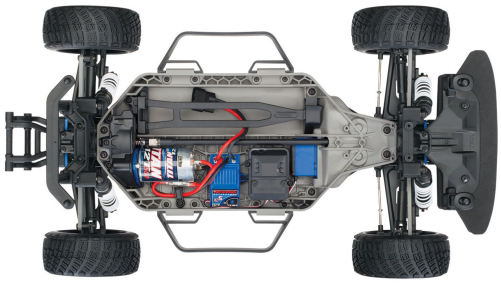
V této práci se vycházelo z postaveného robota [47], který byl částečně navržen podle již zmíněného projektu F1/10 [2]. Robot byl však ještě upraven. Byl z něj vyjmut celý GPS přijímač, jeden rozbočovač a na střed otáčení byla přesunuta Inerciální měřící jednotka (IMU) [47]. Na robotu tedy zůstali pouze součástky, které budou představeny v dalších podkapitolách. Postupně budou v této části popsány všechny části robotu. Nejdříve bude představena mechanická konstrukce. Dále se bude tato část zabývat použitými senzory a jejich možného potenciálního využití. Následně budou představeny použité akční členy zajišťující pohyb robotu. Na tuto část bude navazovat poslední část o řídicí elektronice robotu.



Obrázek 3.1: Autonomní RC autíčko

## 3.1 Mechanická konstrukce

Celá robotická platforma leží na podvozku Traxxas Ford Fiesta ST Rally [43]. Jedná se o podvozek 1:10 zmenšeného závodního auta s nízkým těžištěm a náhonem na všechny čtyři kola. Je zároveň postaven na principu takzvaného Ackermannova řízení. V diplomové práci [47] byl však tento podvozek ještě upraven. Z podvozku byl odstraněn původní stejnosměrný motor, regulátor a RC přijímač [47]. Aby bylo možné měřit otáčky motoru nachází se nově na podvozku bezkartáčový stejnosměrný motor Velineon 3500 [44], který se stará o přímočarý pohyb a původní servomotor sloužící k zatáčení předních kol.



Obrázek 3.2: Traxxas Ford Fiesta ST Rally

## 3.2 Senzory

Robot je osazen celkem třemi senzory. ZED kamerou [41], která se nachází na předku robotu. Nad ním se nachází laserový senzor vzdálenosti LiDAR [23] a na středu otáčení robota je osazena IMU [27]. Senzory zde plní funkci získávání dat z prostředí a informaci o poloze tak, aby se robot mohl buď to orientovat v prostředí a nebo mohl tvořit mapu z tohoto prostředí.

### 3.2.1 Inercální měřící jednotka

Na vrchní části robotu je připevněna průmyslová IMU firmy LORD, konkrétně typ 3DM-CV5-25. Tento senzor na rozdíl od ostatních použitých senzorů neposkytuje žádnou informaci o prostředí. Poskytuje pouze informace o zrychlení, rychlosti a orientaci v prostoru. K tomuto měření slouží trojice senzorů, teplotně kompenzovaný tříosý akcelerometr, gyroskop a magnetometr. Akcelerometr slouží k měření

lineárního zrychlení, gyroskop měří rychlost otáčení a magnetometr získává informace o magnetickém poli Země. Velkou nevýhodou může být pokud používáme tento senzor k určení polohy, protože se bude v průběhu pohybu robotu nahromaďovat chyba. Jelikož se totiž k určení polohy musí integrovat poskytující rychlost s ohledem na čas, bude se vlivem malých chyb, který senzor bude poskytovat, chyba postupně zvětšovat. Chyba se tedy bude hromadit a tím pádem vznikne takzvaný drift, což je stále rostoucí chyba mezi polohou, kterou poskytuje senzor a mezi jeho reálnou polohou. Měření však lze zpřesnit pomocí algoritmu rozšířeného Kalmanova filtru, který je v této jednotce obsažen.


Obrázek 3.3: Lord 3DM-CV5-25

### 3.2.2 ZED kamera

Jedná se o pasivní stereo kameru, která reprodukuje způsob, jakým funguje lidské vidění. Pomocí svých dvou kamer vytváří trojrozměrnou mapu scény porovnáním posunu pixelů mezi levým a pravým obrazem. O tento princip tvorby hloubkové mapy se stará ZED SDK dodávané výrobcem. Výsledkem je pak odhad hloubky nebo vzdálenosti kamery od objektů ve scéně.


Obrázek 3.4: ZED kamera

### 3.2.3 LiDAR

Na robotu se nachází konkrétně model Hokuyo URG-04LX-UG01. Tento LiDAR senzor se stará o měření vzdáleností mezi objekty a senzorem. Senzor vysílá infračervený laserový paprsek, který rotuje. Poté co paprsek dopadne na objekt se odrazí zpátky. Fáze vysílajícího a odraženého paprsku jsou porovnány a vypočítá se vzdálenost od objektu. Z těchto vzdáleností se získají bodová data. Data poskytované tímto senzorem lze následně použít například k lokalizaci a mapování. Velkovou výhodou tohoto senzoru je jeho veliká přesnost.


Obrázek 3.5: Hokuyo URG-04LX-UG01

## 3.3 Akční členy

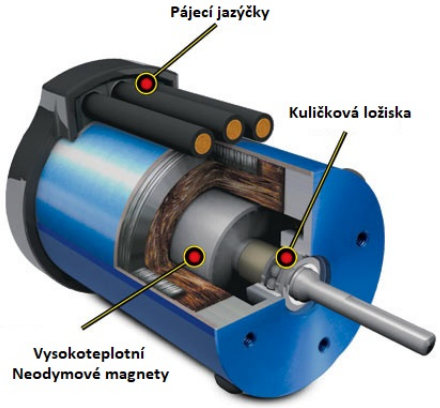
Akční členy se zde starají o kompletní pohyb robota. Robot dokáže jezdit buď to dopředu nebo dozadu. Natočením předních kol pod nějakým úhlem pak robot dokáže jet zhruba po kružnici dané právě natočením kol. Na robotu se nachází jak již bylo zmíněno celkem dva akční členy. Jeden servomotor, který se stará o natáčení kol. Druhý je třífázový bezkartáčový motor [44], který je použit pro přímočarý pohyb.

### 3.3.1 Bezkartáčový motor Velineon 3500

Bezkartáčové stejnosměrné motory jsou tvořeny ze dvou hlavních částí, ze statoru a rotoru, který otáčí hřídelí. Stator je navinut cívkami a rotor je tvořen permanentními, u tohoto motoru ultra vysokoteplotními neodymovými magnety. Mezi rotorem a hřídeli jsou umístěna kuličková ložiska. Výhoda tohoto motoru spočívá

v tom, že jelikož neobsahují žádné kartáče nebo komutátor nepotřebuje tento motor prakticky žádnou údržbu.

Základním úkolem elektromotoru je obecně přeměna elektrické energie na mechanickou, tedy uvést do pohybu hřídel poháněného zařízení. Princip motoru spočívá v tom, že do jednotlivých cívek navinutých na statoru je přiváděn vhodně orientovaný proud, který vytváří magnetické pole. Sledováním pohybu rotoru a postupným aktivováním jednotlivých cívek se začne rotor otáčet z důvodu působení sil mezi magnetem a cívkou [28]. Na hřídeli se pak vytváří točivý moment. Čím rychleji se cívky přepínají tím více se rotor otáčí a tím pádem jede robot rychleji.




Obrázek 3.6: Velineon 3500

## 3.4 Řídicí elektronika

Jednou z nejdůležitějších částí robotu je řídicí elektronika. Zajišťuje správný pohyb robota a získávání dat ze senzorů, které se pak dále zpracovávají k určení polohy, tvorby mapy nebo plánování pohybu robota. Robot je vybaven celkem dvěma řídicími jednotkami. První řídicí jednotkou je VESC 6 MKIV, která slouží především k řízení bezkartáčového motoru a servomotoru. Druhou a zároveň hlavní řídicí jednotkou je malý počítač NVIDIA Jetson TX2 [31]. Tento počítač slouží především k samotnému řízení a také získávaní dat ze senzorů.

### 3.4.1 VESC

VESC je elektronický regulátor rychlosti (ESC) s otevřeným zdrojovým kódem od Benjamina Veddera [45]. Jedná se v podstatě o vylepšenou verzi klasického

ESC, který v principu dává informaci motoru jak silně má přidávat plyn nebo jak moc brzdit. VESC oproti klasickému ESC poskytuje hodně možností úpravy. K regulaci rychlosti zde slouží PID regulátor. Nastavení určité rychlosti pak nastává tak, že zadáme požadovanou rychlost a VESC tuto instrukci převede do motoru. Je schopen měřit napětí a proud na všech fázích motorů a poskytnout informaci o rychlosti otáčení, dá se tedy dále použít k odhadu polohy robota. Všechna nastavení se pak upravují v softwaru VESC Tool [46]. Ten umožňuje například identifikovat parametry motoru, na základě něhož pak také automaticky stanovit parametry PID regulátoru nebo nastavit limity pro ochranu motoru a baterie.

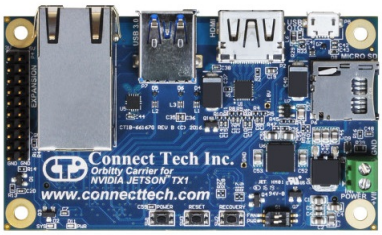
Obrázek 3.7: VESC 6 MKIV

### 3.4.2 NVIDIA Jetson TX2

Jetson TX2 je rychlé a energeticky úsporné vestavné výpočetní zařízení s umělou inteligencí [31]. Je postaven kolem GPU rodiny NVIDIA Pascal a osazen 8 GB paměti s propustností 59,7 GB/s. Tento počítač je dodáván ve vývojové sadě s nosnou deskou. Tato deska byla nahrazena menší deskou Orbitty Carrier [11]. Do této desky je pak připojen rozbočovač do kterého jsou zapojeny všechny senzory a regulátor VESC. Na zařízení byl nainstalován pomocí SDK manageru Linux Ubuntu 18.04 s NVIDIA Jetpack 4.2 který byl modifikován z důvodu použití jiné nosné desky. Celý proces instalace tak musel probíhat částečně i pomocí instalačních souborů poskytující firmou Connect Tech, která vyrábí pravě použitou nosnou deskou. Poté byl nainstalován dříve zmíněný ROS z důvodu jednoduché implementace programů a zároveň velkého obsahu balíčků pro navigaci či lokalizaci a mapování. Konkrétně byla nainstalována distribuce Melodic. Nakonec byl stažen balíček od F1/10 [2], který obsahuje všechny důležité ovladače zařízení a zároveň nastavení pro ovládání robotu.

| | |
|---|---|
| GPU | 256jádrová architektura GPU NVIDIA Pascal™ s 256 jádry NVIDIA CUDA |
| procesor | Dvoujádrový procesor NVIDIA Denver 2 64bitový Čtyřjádrový ARM® Cortex®-A57 MPCore |
| Paměť | 8GB 128bitová paměť LPDDR4 1866 MHz - 59,7 GB/s |
| Úložný prostor | 32GB eMMC 5.1 |
| Napájení | 7,5W / 15W |

Tabulka 3.1: NVIDIA Jetson TX2 specifikace




Obrázek 3.8: Orbitty Carrier

# Kapitola 4

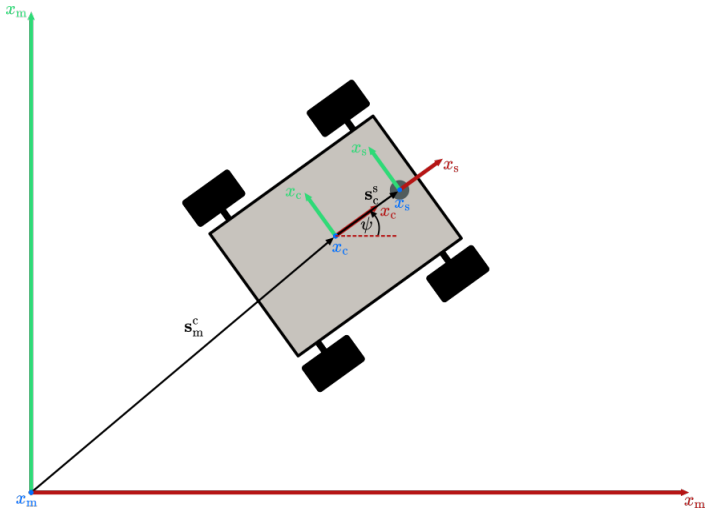
# Řízení modelu autonomního závodního vozidla

V této kapitole budou postupně zavedeny jednotlivé části pro potřeby řízení autonomního vozidla. Nejdříve budou zavedeny potřebné souřadné systémy. Abychom mohli následně vyjadřovat jednotlivé polohy či data ze senzorů v jiných souřadných systémech budou dále zavedeny jednotlivé transformace mezi souřadnými systémy. Po zavedení těchto transformací bude představen Ackermannův princip řízení na kterém je založeno řízení použitého robotu. Následně bude odvozen kinematický model bicyklu, který bude zjednodušením právě Ackermannova principu řízení. Pro potřeby řízení bude poté představena inverzní kinematika modelu, dále lineární aproximace modelu bicyklu a jeho diskrétní varianta. V poslední části této kapitoly pak bude představen použitý regulátor pro řízení autonomního vozidla.

## 4.1 Souřadné systémy

V této části budou zavedeny všechny potřebné souřadné systémy. Pro popis kinematiky bude potřeba zavést souřadný systém prostředí ve kterém se robot bude pohybovat a souřadný systém samotného robotu. Z důvodu, že hlavní použitý senzor LiDAR leží v přední části robotu a poskytuje data z měření ve svém souřadném systému bude zaveden i jeho souřadný systém.

Souřadný systém prostředí tvoří souřadný systém, který v této práci bude uvažován jako globální. Je tvořen třemi na sobě navzájem ortogonálními osami $x_{\mathrm{m}}$, $y_{\mathrm{m}}$ a $z_{\mathrm{m}}$ připojeními k počátku tohoto systému. Počátek ve většině případů leží na startovní pozici auta, ale může ležet kdekoli jinde v prostoru. Souřadný systém robotu je tvořen opět třemi osami $x_{\mathrm{c}}$, $y_{\mathrm{c}}$ a $z_{\mathrm{c}}$ natočenými tak, že osa $x_{\mathrm{c}}$ směřuje ve směru pohybu robota vpřed a jeho počátek leží ve středu otáčení robotu. Poslední




Obrázek 4.1: Zavedené souřadné systémy

použitý souřadný systém tvoří osy $x_s$, $y_s$ a $z_s$, kde počátek leží ve středu senzoru. Souřadné systémy robotu a senzoru jsou na sebe staticky vázány, kde souřadný systém senzoru je vůči souřadnému systému robotu pouze posunutý.

### 4.1.1 Transformace

Dále je potřeba pro vyjadřování poloh či bodů v různých zavedených souřadných systémech zavést transformace mezi těmito souřadnými systémy. Transformace mezi souřadnými systémy se skládá z translace a rotace. Translace je popsaná pomocí vektoru translace a rotace je pak popsána pomocí matice rotace. Celková transformace bude pak popsána homogenní transformační maticí.

Translace z počátku souřadného systému prostředí $\mathbf{O}_m$ do souřadného systému robotu $\mathbf{O}_c$ je popsaná pomocí vektoru $\mathbf{s}_m^c$ reprezentující polohu počátku souřadného systému robotu v prostředí

$$\mathbf{s}_m^c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}. \tag{4.1.1}$$

Jelikož se pohyb robotu uskutečňuje po rovině, tedy po souřadných osách $x_m$ a $y_m$, mění svoji orientaci kolem osy $z_m$ o úhel $\psi$. Toto otočení lze reprezentovat pomocí matice rotace

$$\mathbf{R}_m^c(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.1.2}$$

Celkovou transformaci mezi těmito souřadnými systémy pak získáme uspořádáním do homogenní transformační matice

$$\mathbf{T}_{\mathrm{m}}^{\mathrm{c}} = \begin{bmatrix} \mathbf{R}_{\mathrm{m}}^{\mathrm{c}} & \mathbf{s}_{\mathrm{m}}^{\mathrm{c}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & x_{\mathrm{c}} \\ \sin(\psi) & \cos(\psi) & 0 & y_{\mathrm{c}} \\ 0 & 0 & 1 & z_{\mathrm{c}} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.1.3}$$

Translace mezi počátky souřadného systému robotu $\mathbf{O}_{\mathrm{c}}$ a senzoru $\mathbf{O}_{\mathrm{s}}$ je dána opět vektorem popisující vzdálenost mezi těmito souřadnými systémy, která se dá jednoduše změřit

$$\mathbf{s}_{\mathrm{c}}^{\mathrm{s}} = \mathbf{O}_{\mathrm{s}} - \mathbf{O}_{\mathrm{c}} \begin{bmatrix} x_{\mathrm{c}} - x_{\mathrm{s}} \\ y_{\mathrm{c}} - y_{\mathrm{s}} \\ z_{\mathrm{c}} - z_{\mathrm{s}} \end{bmatrix}. \tag{4.1.4}$$

Vzhledem k tomu, že souřadné systémy robotu a senzoru nejsou vůči sobě natočeny, bude rotační matice $\mathbf{R}_{\mathrm{c}}^{\mathrm{s}}$ jednotková. Opětovným uspořádáním translace a rotace se získá homogenní transformační matice

$$\mathbf{T}_{\mathrm{c}}^{\mathrm{s}} = \begin{bmatrix} \mathbf{R}_{\mathrm{c}}^{\mathrm{s}} & \mathbf{s}_{\mathrm{c}}^{\mathrm{s}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_{\mathrm{c}} - x_{\mathrm{s}} \\ 0 & 1 & 0 & y_{\mathrm{c}} - y_{\mathrm{s}} \\ 0 & 0 & 1 & z_{\mathrm{c}} - z_{\mathrm{s}} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.1.5}$$

Celkovou transformační matici mezi souřadným systémem prostředí a senzorem získáme vynásobením matice 4.1.3 s maticí 4.1.5

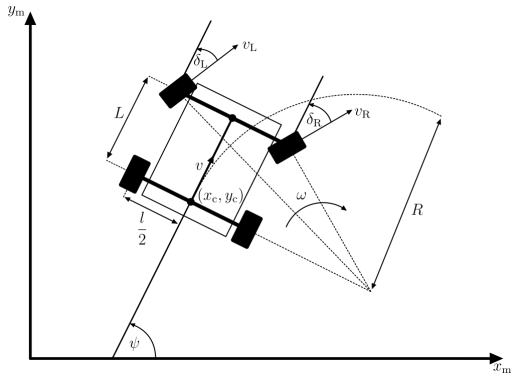
$$\mathbf{T}_{\mathrm{m}}^{\mathrm{s}} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & x_{\mathrm{c}} + \cos(\psi)(x_{\mathrm{s}} - x_{\mathrm{c}}) - \sin(\psi)(y_{\mathrm{s}} - y_{\mathrm{c}}) \\ \sin(\psi) & \cos(\psi) & 0 & y_{\mathrm{c}} + \cos(\psi)(y_{\mathrm{s}} - y_{\mathrm{c}}) - \sin(\psi)(x_{\mathrm{s}} - x_{\mathrm{c}}) \\ 0 & 0 & 1 & z_{\mathrm{s}} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.1.6}$$

Vyjádření polohy bodu v souřadném systému prostředí $\mathbf{P}^{\mathrm{m}}$ ze souřadného systému senzoru $\mathbf{P}^{\mathrm{s}}$ pak získáváme vztahem

$$\begin{bmatrix} \mathbf{P}^{\mathrm{m}} \\ 1 \end{bmatrix} = \mathbf{T}_{\mathrm{m}}^{\mathrm{s}} \begin{bmatrix} \mathbf{P}^{\mathrm{s}} \\ 1 \end{bmatrix}. \tag{4.1.7}$$

## 4.2 Ackermannův princip řízení

Ackermannův princip řízení je specifický tím, že při průjezdu zatáčkou je vnitřní kolo natočeno pod trochu větším úhlem něž kolo vnější. Zároveň se vnitřní kolo




Obrázek 4.2: Schéma Ackermannova principu řízení

otáčí pomaleji než kolo vnější. Větší natočení a pomalejší otáčení vnitřního kola je použito z důvodu toho, aby se zadní kola odvalovala bez smýkání.

Z obrázku 4.2 pomocí trigonometrie můžeme dostat vztahy definující úhly natočení jednotlivých předních kol [20]

$$\tan\left(\frac{\pi}{2} - \delta_{\mathrm{L}}\right) = \frac{R + \frac{l}{2}}{L}, \tag{4.2.1}$$

$$\tan\left(\frac{\pi}{2} - \delta_{\mathrm{R}}\right) = \frac{R - \frac{l}{2}}{L}. \tag{4.2.2}$$

Následnou úpravou těchto rovnic pak získáme finální vztah pro natočení předních kol

$$\delta_{\mathrm{L}} = \frac{\pi}{2} - \arctan\left(\frac{R + \frac{l}{2}}{L}\right), \tag{4.2.3}$$

$$\delta_{\mathrm{R}} = \frac{\pi}{2} - \arctan\left(\frac{R - \frac{l}{2}}{L}\right). \tag{4.2.4}$$

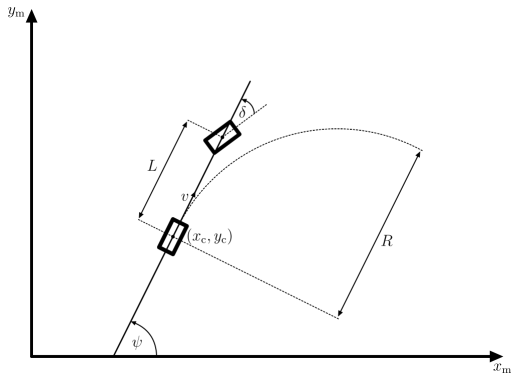
Při natočení předních kol budou zadní kola opisovat kružnici se stejnou úhlovou rychlostí $\omega$ a obvodová rychlost kol se vyjádří vztahy

$$v_{\mathrm{L}} = \omega\left(R + \frac{l}{2}\right), \tag{4.2.5}$$

$$v_{\mathrm{R}} = \omega\left(R - \frac{l}{2}\right). \tag{4.2.6}$$

## 4.3 Kinematický model robotu

Poslední částí před zavedením samotného použitého řídicího algoritmu je popis kinematiky modelu Ackermannova řízení. Pro zjednodušení bude uvažováno, že obě přední kola budou při průjezdu zatáčkou natočeny pod stejným úhlem. Díky tomuto předpokladu můžeme všechna čtyři kola nahradit pouze dvěma, předním a zadním kolem. Dalším zanedbáním bude případný úhel bočního skluzu, což je úhel mezi směrem kterým se robot pohybuje a směrem kam je robot reálně natočen. Tento jev může nastat například když robot jede ve smyku. Tímto zanedbáním můžeme pro jednodušší odvození přenést počátek robotu $\mathbf{O}_\mathrm{c}$ na zadní nápravu. Z těchto zjednodušení dostaneme takzvaný kinematický model bicyklu.




Obrázek 4.3: Kinematický model bicyklu

### 4.3.1 Kinematický model bicyklu

Konfigurační prostor bicyklu **x** je dán třemi stavy, polohou v ose $x_\mathrm{m}$ a $y_\mathrm{m}$ a směrovým úhlem $\psi$, který je definován jako úhel mezi osou $x_\mathrm{m}$ a natočením bicyklu

$$\mathbf{x} = \begin{bmatrix} x_\mathrm{m} \\ y_\mathrm{m} \\ \psi \end{bmatrix}. \tag{4.3.1}$$

řídicím vstupem pro ovládání bicyklu je pak vektor **u**, který je dán řídicím vstupem rychlosti $v$ a úhlu natočení předního kola $\delta$

$$\mathbf{u} = \begin{bmatrix} v \\ \delta \end{bmatrix}. \tag{4.3.2}$$

Délka mezi zadní a přední nápravou je pak reprezentována vzdáleností $L$.

Při velmi malé změně polohy limitně blížící se nule v souřadných osách $x_{\mathrm{m}}$ a $y_{\mathrm{m}}$ se bicykl bude pohybovat ve směru úhlu $\psi$ [26]. Tím pádem můžeme nadefinovat první dvě rovnice popisující kinematiku modelu

$$\dot{x}_{\mathrm{m}} = v\cos(\psi), \tag{4.3.3}$$

$$\dot{y}_{\mathrm{m}} = v\sin(\psi). \tag{4.3.4}$$

Pokud zafixujeme natočení kol $\delta$, bicykl se bude pohybovat po kružnici danou poloměrem $R$, viz obrázek 4.3. Pomocí trigonometrie si můžeme pak vyjádřit $R$ následujícím způsobem

$$R = \frac{L}{\tan(\delta)}. \tag{4.3.5}$$

Vzhledem k tomu že $\dot{\psi}$ je roven úhlové rychlosti $\omega$ můžeme využít vztahu

$$\omega = \frac{v}{R}. \tag{4.3.6}$$

Následným nahrazením rovnice 4.3.5 za $R$ získáme poslední rovnici definující kinematiku bicyklu

$$\dot{\psi} = \omega = \frac{v}{L}\tan(\delta). \tag{4.3.7}$$

### 4.3.2 Inverzní kinematika modelu bicyklu

Pro plánování pohybu modelu bicyklu je zapotřebí znát požadované hodnoty řídicího vstupu **u** a konfiguračního prostoru **x**. Proto je zapotřebí si tyto vztahy z kinematického modelu bicyklu vyjádřit. Díky tomu, že byl kinematický model robotu co nejvíce zjednodušen bude možné pro tento případ nalézt analytické řešení.

Umocněním a sečtením rovnic 4.3.3 a 4.3.4 spolu s použitím součtového vzorce $\sin^2(\psi) + \cos^2(\psi) = 1$ získáme požadovanou rychlost $v$ definovanou vztahem

$$v = \sqrt{\dot{x}^2 + \dot{y}^2}. \tag{4.3.8}$$

Pro vyjádření požadovaného natočení kol $\delta$ je zapotřebí si nejdříve vyjádřit směrový úhel $\psi$ a následně jeho časovou derivaci $\dot{\psi}$. Transformací a úpravou rovnice 4.3.3 a 4.3.4 získáme finální vztah pro směrový úhel [29]

$$\psi = \arctan\left(\frac{\dot{y}}{\dot{x}}\right). \tag{4.3.9}$$

Následnou časovou derivací vztahu 4.3.9 získáme vztah pro časovou derivaci směrového úhlu

$$\dot{\psi} = \frac{\ddot{y}\dot{x} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2}. \tag{4.3.10}$$

Následně si už můžeme ze vztahu 4.3.7 a jeho následnou úpravou vyjádřit vztah pro požadovanou hodnotu natočení kol

$$\delta = \arctan\left(\frac{L\dot{\psi}}{v}\right). \tag{4.3.11}$$

### 4.3.3 Linearizace modelu

Vzhledem k tomu, že pro řízení bude využita lineární varianta regulátoru je zapotřebí použít lineární odchylkovou aproximaci získaného nelineárního modelu. Jelikož se model bicyklu bude pohybovat podél nějaké referenční trajektorie nabízí se linearizovat model podél této trajektorie. Protože trajektorie je často funkcí času bude model převeden do formy lineárního časově variantního systému (LTV). Systém tedy bude nabývat standardního maticového předpisu v ochylkových proměnných

$$\Delta\dot{\mathbf{x}}(t) = \mathbf{A}(t)\Delta\mathbf{x}(t) + \mathbf{B}(t)\Delta\mathbf{u}(t), \tag{4.3.12}$$

kde matice $\mathbf{A}(t)$ a $\mathbf{B}(t)$ se budou v průběhu času měnit. Matici $\mathbf{A}(t)$ získáme parciálním derivováním jednotlivých rovnic 4.3.3, 4.3.4 a 4.3.7 podle vektoru 4.3.1, kde uspořádáním do matice dostaneme

$$\mathbf{A}(t) = \begin{bmatrix} 0 & 0 & -v\sin(\psi) \\ 0 & 0 & v\cos(\psi) \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.3.13}$$

Matici $\mathbf{B}(t)$ získáme podobným způsobem, tentokrát však parciálním derivováním podle vektoru řídicího vstupu **u**

$$\mathbf{B}(t) = \begin{bmatrix} \cos\psi & 0 \\ \sin\psi & 0 \\ \frac{\tan\delta}{L} & \frac{v}{L\cos^2\delta} \end{bmatrix}. \tag{4.3.14}$$

Jednotlivé proměnné vyskytující se v maticích $\mathbf{A}(t)$ a $\mathbf{B}(t)$ pak získáme z trajektorie pomocí vztahů odvozených v kapitole 4.3.2.

### 4.3.4 Diskretizace modelu

Kvůli tomu, že později použitý regulátor je diskrétní, potřebuje se pro přímý návrh převést použitý spojitý LTV model na diskrétní. Diskretizace modelu bude provedena pomocí Eulerovy metody.

Zavedeme tedy za aktuální časový okamžik $t$ proměnou $k$. Následující časový okamžik $k+1$ bude reprezentovat časový okamžik $t+T$, kde $T$ bude reprezentovat

délku vzorkovacího intervalu. Diskrétní model spojitého LTV modelu bude tedy nabývat nového tvaru

$$\Delta \mathbf{x}_{k+1} = \mathbf{A}_k \Delta \mathbf{x}_k + \mathbf{B}_k \Delta \mathbf{u}_k. \tag{4.3.15}$$

Souvislost mezi diskrétními a spojitými variantami matic můžeme reprezentovat vztahy [12]

$$\mathbf{A}_k = e^{\mathbf{A}(t)}, \tag{4.3.16}$$

$$\mathbf{B}_k = \mathbf{A}^{-1}(t)(e^{\mathbf{A}(t)} - \mathbf{I})\mathbf{B}(t). \tag{4.3.17}$$

Vztah 4.3.16 můžeme následně ještě rozvést do řady

$$\mathbf{A}_k = e^{\mathbf{A}(t)} = \mathbf{I} + \mathbf{A}(t)T + \frac{1}{2!}\mathbf{A}^2(t)T^2 \ldots, \tag{4.3.18}$$

kde použijeme pouze první řád a vyšší řády zanedbáme, tím dostaneme aproximační vztah definující souvislost mezi maticemi

$$\mathbf{A}_k \approx \mathbf{I} + \mathbf{A}(t)T. \tag{4.3.19}$$

Následným dosazením tohoto vztahu do rovnice 4.3.17 a následnou úpravou získáme další aproximační vztah pro druhou matici

$$\mathbf{B}_k \approx \mathbf{B}(t)T. \tag{4.3.20}$$

Aplikacemi těchto aproximací budou matice $\mathbf{A}_k$ a $\mathbf{B}_k$ nabývat tvaru

$$\mathbf{A}_k = \begin{bmatrix} 1 & 0 & -v\sin(\psi)T \\ 0 & 1 & v\cos(\psi)T \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.3.21}$$

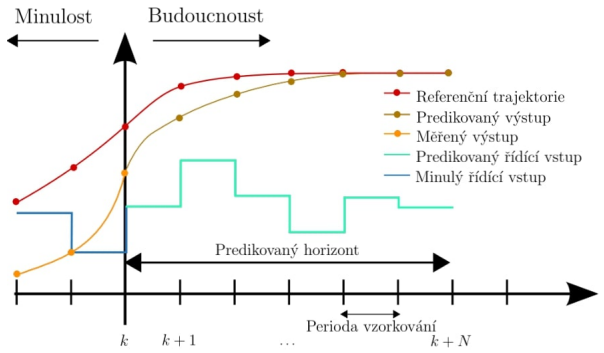
$$\mathbf{B}_k = \begin{bmatrix} \cos(\psi)T & 0 \\ \sin(\psi)T & 0 \\ \frac{\tan(\delta)}{L}T & \frac{v}{L\cos^2(\delta)}T \end{bmatrix}. \tag{4.3.22}$$

Pro zjednodušení budeme v následujících částech této práce zanedbávat použité označení $\Delta$ reprezentující odchylky od referenční trajektorie.

## 4.4 Model Predictive Control

Tato část práce se bude zabývat použitým algoritmem prediktivního řízení (MPC - Model Predictive Control). Cílem MPC je predikovat budoucí chování řízeného

diskrétního systému v konečném časovém horizontu $N$ a vypočítat optimální sekvenci řídicích vstupů, které při zajištění splnění daných omezení systému minimalizují predikovanou regulační odchylku přes časový horizont. Řídicí vstup se vypočítá tak, že se v každém časovém okamžiku vyřeší problém optimálního řízení s otevřenou smyčkou s konečným horizontem. Protože v aktuálním časovém okamžiku $k$ dokážeme měřit polohu robotu a v ostatních pouze predikujeme jeho polohu, dosáhneme zpětnovazebního řízení aplikováním pouze prvního vstupu. Celý proces se pak následně opakuje. Výhodou tohoto algoritmu řízení je, že dokáže pracovat se systémy, které mají více vstupů a výstupů (MIMO).




Obrázek 4.4: Princip MPC. Převzato z [4]

### 4.4.1 Formulace problému

MPC obecně řeší omezený optimalizační problém, který bude minimalizovat nějaké zvolené funkční kritérium. Snaží se tedy najít, jak již bylo zmíněno optimální sekvenci řídicích vstupů a predikovaných stavů systému.

Zavedením regulační odchylky mezi stavy $\mathbf{e}_{\mathbf{x},k} = \mathbf{x}_k - \mathbf{x}_{\mathrm{r},k}$ a odchylky mezi řídicími vstupy $\mathbf{e}_{\mathbf{u},k} = \mathbf{u}_k - \mathbf{u}_{\mathrm{r},k}$, kde $\mathbf{x}_{\mathrm{r},k}$ a $\mathbf{u}_{\mathrm{r},k}$ reprezentuje referenční hodnoty pro aktuální časový okamžik $k$, můžeme pak funkční kriterium formulovat jako kvadratickou funkci ve tvaru

$$J(\mathbf{x},\mathbf{u}) = \mathbf{e}_{\mathbf{x},N}^{\top}\mathbf{Q}_N\mathbf{e}_{\mathbf{x},N} + \sum_{k=0}^{N-1} \mathbf{e}_{\mathbf{x},k}^{\top}\mathbf{Q}\mathbf{e}_{\mathbf{x},k} + \mathbf{e}_{\mathbf{u},k}^{\top}\mathbf{R}\mathbf{e}_{\mathbf{u},k}, \tag{4.4.1}$$

kde $\mathbf{Q}$ je váhová matice stavů a $\mathbf{R}$ je váhová matice řídicích vstupů. Jelikož řídicí vstupy nemohou nabývat libovolných akčních zásahu je vhodné pro řešení optimalizační úlohy zavést tyto omezení. Dalším omezením je množina stavů, kterých v

daný moment může systém nabývat vzhledem k ohraničení závodní trati $\mathcal{X}$, kterou projíždí. Celý optimalizační problém za podmínky (s.t.) pak nadefinujeme následujícím způsobem

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & J(\mathbf{x},\mathbf{u}) \\ \text{s.t.} \quad & \mathbf{x}_0, \\ & \mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k, \\ & \mathbf{x} \in \mathcal{X} \implies \mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max}, \\ & \mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}. \end{aligned} \tag{4.4.2}$$

Tento optimalizační problém s omezením nelze řešit analyticky. Je proto nutné řešit tento problém pomocí numerické optimalizace. Vzhledem k tomu, že kritérium je kvadratické a máme lineární omezení můžeme tuto dynamickou optimalizační úlohu převést na statické kvadratické programování.

### 4.4.2 Kvadratické programovaní

Kvadratické programování je speciální typ konvexní optimalizace. Výhodou konvexní optimalizace je, že nalezené lokální minimum je zároveň globálním. Optimalizační úlohy kvadratického programování se definují následovně

$$\begin{aligned} \min_{\mathbf{z}} \quad & \frac{1}{2}\mathbf{z}^\top\mathbf{H}\mathbf{z} + \mathbf{g}^\top\mathbf{z} \\ \text{s.t.} \quad & \mathbf{b}_{\text{lb}} \leq \mathbf{A}_{\text{c}}\mathbf{z} \leq \mathbf{b}_{\text{ub}}, \end{aligned} \tag{4.4.3}$$

kde proměnná $\mathbf{z}$ je optimalizační proměnná, $\mathbf{H}$ je symetrická pozitivně semidefinitní Hessova matice a vektor $\mathbf{g}$ je jeho gradient. Lineární omezení je pak definováno maticí $\mathbf{A}_{\text{c}}$ a vektory $\mathbf{b}_{\text{lb}}$ a $\mathbf{b}_{\text{ub}}$.

Nadefinovaný MPC problém je následně potřeba převést do standardní formy kvadratického programování. Je tedy potřeba přepsat aditivní kritérium 4.4.1 do maticového zápisu 4.4.3. Optimalizační proměnná $\mathbf{z}$ v tomto případě nabývá tvaru

$$\mathbf{z} = \begin{bmatrix} \mathbf{x}_0 & \dots & \mathbf{x}_N & \mathbf{u}_0 & \dots & \mathbf{u}_N \end{bmatrix}^\top. \tag{4.4.4}$$

Kvadratické kritérium 4.4.1 se pak dá následně přepsat do maticového zápisu

$$J(\mathbf{x},\mathbf{u}) = \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}^\top \begin{bmatrix}\mathbf{Q} & \\ & \mathbf{R}\end{bmatrix} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix} + \begin{bmatrix}-\mathbf{Q}\mathbf{x}_{\text{r},k} \\ -\mathbf{R}\mathbf{u}_{\text{r},k}\end{bmatrix}^\top \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix} + \frac{1}{2}\mathbf{x}_N^\top\mathbf{Q}_N\mathbf{x}_N. \tag{4.4.5}$$

Poté se funkční kvadratické kritérium může přepsat do finální formy s optimalizační proměnou $\mathbf{z}$ v kompaktním maticovém zápisu

$$J(\mathbf{z}) = \frac{1}{2}\mathbf{z}^\top \underbrace{\begin{bmatrix} \bar{\mathbf{Q}} & \\ & \bar{\mathbf{R}} \end{bmatrix}}_{\mathbf{H}} \mathbf{z} + \underbrace{\begin{bmatrix} -\bar{\mathbf{Q}}\mathbf{x}_\mathrm{r} \\ -\bar{\mathbf{R}}\mathbf{u}_\mathrm{r} \end{bmatrix}}_{\mathbf{g}} \mathbf{z}, \tag{4.4.6}$$

kde matice $\bar{\mathbf{Q}}$ a $\bar{\mathbf{R}}$ nabývají tvaru

$$\bar{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & & \\ & \ddots & \\ & & \mathbf{Q}_N \end{bmatrix}, \tag{4.4.7}$$

$$\bar{\mathbf{R}} = \begin{bmatrix} \mathbf{R} & & \\ & \ddots & \\ & & \mathbf{R} \end{bmatrix}. \tag{4.4.8}$$

Pro kompletní nadefinování standardního zápisu kvadratického programování je zapotřebí ještě přepsat lineární omezení do maticového zápisu. Dolní a horní omezení přepíšeme do příslušných vektorů $\mathbf{b}_\mathrm{lb}$ a $\mathbf{b}_\mathrm{ub}$ následovně

$$\mathbf{b}_\mathrm{lb} = \begin{bmatrix} -\mathbf{x}_0 & \dots & \mathbf{0} & \mathbf{0} & \mathbf{x}_\mathrm{min} & \dots & \mathbf{x}_\mathrm{min} & \mathbf{u}_\mathrm{min} & \dots & \mathbf{u}_\mathrm{min} \end{bmatrix}^\top, \tag{4.4.9}$$

$$\mathbf{b}_\mathrm{ub} = \begin{bmatrix} -\mathbf{x}_0 & \dots & \mathbf{0} & \mathbf{0} & \mathbf{x}_\mathrm{max} & \dots & \mathbf{x}_\mathrm{max} & \mathbf{u}_\mathrm{max} & \dots & \mathbf{u}_\mathrm{max} \end{bmatrix}^\top. \tag{4.4.10}$$

Následně můžeme nadefinovat matici

$$\mathbf{A}_\mathrm{c} = \begin{bmatrix} \bar{\mathbf{C}} \\ \mathbf{I} \end{bmatrix}. \tag{4.4.11}$$

kde matice $\bar{\mathbf{C}}$ reprezentuje první a druhou podmínku. Jednotková matice $\mathbf{I}$ pak reprezentuje omezení mezi dosažitelnými stavy a řídicími vstupy. Matice $\bar{\mathbf{C}}$ bude nabývat tvaru

$$\bar{\mathbf{C}} = \begin{bmatrix} \bar{\mathbf{A}} & \bar{\mathbf{B}} \end{bmatrix}, \tag{4.4.12}$$

kde následně matice $\bar{\mathbf{A}}$ a $\bar{\mathbf{B}}$ budou nabývat tvaru

$$\bar{\mathbf{A}} = \begin{bmatrix} -\mathbf{I} & & & \\ \mathbf{A}_1 & -\mathbf{I} & & \\ & \ddots & \ddots & \\ & & \mathbf{A}_{N-1} & -\mathbf{I} \end{bmatrix}, \tag{4.4.13}$$

$$\bar{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_1 & & \\ & \ddots & \\ & & \mathbf{B}_{N-1} \end{bmatrix}. \tag{4.4.14}$$

Tímto jsme získali kompletní přepis z formulace MPC problému pomocí kvadratické funkce do standardního zápisu kvadratického programování. K nalezení optimálního řízení s konečným horizontem v každém časovém okamžiku pak musí být vyřešeno pomocí nějakého solveru.

# Kapitola 5

# Mapování a lokalizace v závodní trati

Aby bylo možné pro robota naplánovat správný pohyb nebo vyjadřovat svoji polohu v závodní trati je potřeba nejdříve získat mapu této trati ve které se bude následně pohybovat. Tento problém tvorby mapy z neznámého prostředí je známý jako simultánní lokalizace a mapování (SLAM). Proto bude cílem této kapitoly představit právě tento SLAM problém. Nejdříve bude proto popsán SLAM problém obecně. Poté bude představen problém lokalizace a následně problém reprezentace mapy

## 5.1 SLAM

SLAM řeší obecně problém postupné tvorby mapy neznámého prostředí a současné lokalizace v právě vytvářené mapě [13]. Získáváním dat z prostředí dochází k tvorbě mapy a současnému určování polohy v této mapě. SLAM problém lze tedy rozdělit na dvě části a to mapování prostředí a lokalizaci v prostředí. Mapování je obecně problém kdy známe polohu robota v každém časovém okamžiku a data ze senzoru se používají k tvorbě mapy. Naopak při lokalizaci má robot již vytvořenou mapu prostředí a používá data ze senzorů k určení polohy v mapě. SLAM řeší oba tyto problémy zároveň a proto je velmi náročný.

Existuje několik způsobů jak je možné SLAM implementovat. Pro různé typy SLAM algoritmů jsou potřeba různé senzory pro sběr dat z prostředí. Vzhledem k tomu, že se tato práce bude zaměřovat na tvorbu mapy závodní trati u které se předpokládá, že bude nějak ohraničena, bude k získávání dat z prostředí sloužit především LiDAR senzor. Vzhledem k velké přesnosti LiDAR senzoru, bude využit algoritmus tvorby mapy a lokalizace, který využívá jako hlavní senzor právě

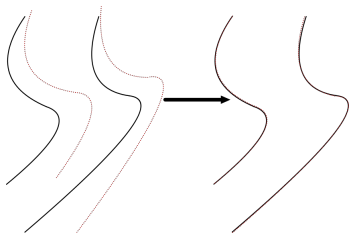
LiDAR a lze tak použít i bez senzoru odometrie, tedy bez použití dat z IMU jednotky nebo z otáček motoru a natočení serva. Protože se předpokládá, že trať je umístěna v jedné rovině, je možné ji reprezentovat pouze v ploše. Mapa bude tak reprezentována pomocí jemnozrnné mřížky obsazenosti.

## 5.2 Lokalizace v mapě

Jak již bylo zmíněno, vyjadřování polohy pouze z odometrie není vhodné. Senzor totiž není dokonalý a zároveň bude vlivem integrovaní nahromaďovat v průběhu času chybu. Ke zpřesnění může být využito například fůze dat z více různých senzorů nebo použití různých algoritmů. Mezi dva nejčastěji používané algoritmy využívající LiDAR data patří Monte Carlo lokalizace (MCL), známá také jako částicový filtr a Scan matching. Obě metody používají k lokalizaci především data z LiDAR senzoru a ze senzoru odometrie. Druhý zmíněný algoritmus však lze použit i bez senzoru odometrie.

### 5.2.1 Scan matching

Cílem scan matchingu je zaregistrovat dvě po sobě jdoucí mračna bodů poskytovaných z LiDAR senzoru a následně najít mezi nimi takovou transformaci, která maximalizuje překrytí mezi dvěma mračny bodů. Mračno bodů je v podstatě neuspořádaná množina bodů, které jsou dány svými souřadnicemi v souřadném systému. K nalezení optimálního překrytí se používají různé varianty Iterativního algoritmu nejbližšího bodu (ICP) [10]. Ten se snaží pomocí nějaké metriky minimalizovat rozdíl mezi dvěma mračny bodů. Následně se snaží najít optimální transformaci mezi mračny bodů. Nakonec aplikuje tuto transformaci na později příchozí mračno. Celý proces se pak opakuje dokud není buďto dosaženo optimální přesnosti a nebo nějakého stanoveného počtu kroků.



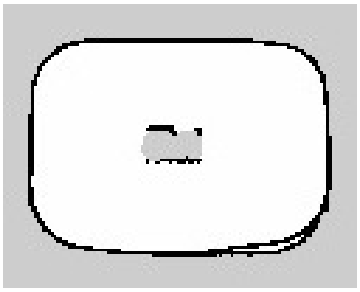
Obrázek 5.1: Funkce Scan matchingu

### 5.2.2 Monte Carlo lokalizace

Jedná se o pravděpodobnostní způsob lokalizace robota. Princip tohoto algoritmu spočívá v reprezentaci stavů prostředí pomocí konečné množiny generovaných vážených vzorků [19]. Pravděpodobnost polohy robota je tak dána hustotou a váhami vzorků v daném místě. Algoritmus má dvě základní fáze a to predikci a korekci. Predikce je dána posunem vzorků na základě informace o změně polohy například z odometrie. Korekce je pak úprava vah jednotlivých vzorků na základě shody či neshody naměřených dat s očekáváními, která by odpovídala pozici reprezentované příslušným vzorkem.

## 5.3 Reprezentace mapy mřížkou obsazenosti

Jednou z nejčastějších reprezentací 2D mapy je jemnozrnná mřížka obsazenosti mapy. Tento typ mapy reprezentuje prostředí jako pole binárních náhodných proměnných z nichž každá představuje přítomnost překážky v daném místě. Obsazenost každé buňky mřížky tedy můžeme brát jako binární náhodnou veličinu. Obsahuje buď to volné místo a nebo obsazené místo, kterým je přiřazeno reálné číslo. Každému statusu obsazenosti ve všech buňkách je pak přiřazena nějaká pravděpodobnost toho, že se jedná buď o volné či obsazené místo. V průběhu času, kdy se robot pohybuje v prostředí a získává o něm průběžně data se následně s použitím algoritmu založeném na Bayesově větě a přepočtu na mřížku, pravděpodobnost rekurzivně aktualizuje. Každá mapa je dána nějakým rozlišením a přepočtem velikosti mřížky na reálné prostředí.



Obrázek 5.2: Mřížka obsazenosti

## 5.4 SLAM algoritmy

Existují v podstatě dva základní SLAM algoritmy pro tvorbu mřížky obsazenosti, které jsou založeny na datech primárně z LiDAR senzoru. Liší se především tím jak se postupně v mapě lokalizují.

Prvním algoritmus je založen na pravděpodobnostním způsobu mapování a lokalizace. K tvorbě mapy používá částicový filtr, kde každá částice (vzorek) pak nese individuální mapu prostředí [21]. Ke správné funkci potřebuje data ze senzoru odometrie a zároveň data poskytující LiDAR senzorem. Výsledkem tohoto algoritmu je pak jemnozrnná obsazovací mřížka.

Druhým základním algoritmem je Hector SLAM [39]. Tento 2D SLAM algoritmus je založen na Scan matching technice a může být tedy použit bez senzoru odometrie. Nevýhodou tohoto algoritmu je, že neposkytuje možnost uzavírání smyček, tedy jednoduše řečeno k dosažení dobrého výsledku se trať musí projet jen jednou. Přesto je tento algoritmus velmi spolehlivý a lze i s jedním projetím trati dosáhnout dobrého výsledku. Výsledkem tohoto algoritmu je pak opět jemnozrnná obsazovací mřížka.

# Kapitola 6

# Řídicí software

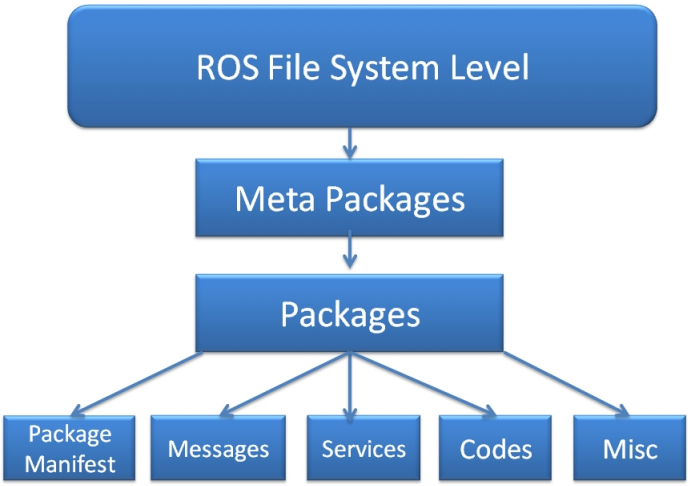
Hlavním použitým softwarem pro návrh řídicího softwaru je zde Robot Operating System (ROS), pomocí něhož se dají jednoduše získávat data ze senzorů nebo umožňuje jednoduché ovládání robotu. Kromě frameworku ROS byli využity již hotové moduly navržené pro projekt F1/10, který je postaven právě na frameworku ROS. Řídicí algoritmy pak byli napsány buď to v programovacím jazyku C/C++ nebo Pythonu. V této kapitole bude tedy nejdřív představen ROS samotný, bude popsán především jeho souborový systém a výpočetní graf, který představuje to jak ROS řídí jednotlivé procesy. Po představení frameworku ROS bude postupně popsán návrh a nastavení řídicího softwaru.

## 6.1 ROS

Jedná se o soubor softwarových knihoven a nástrojů pro psaní robotického softwaru. Poskytuje několik funkcí při řešení úkolů jako je předávání zpráv, opakované použití kódů a implementace nejmodernějších kódů pro robotické aplikace [9]. ROS má v dnešní době velkou komunitu uživatelů a vývojářů po celém světě, která se neustále rozrůstá. Velkou výhodou použití frameworku ROS je velké množství volně dostupných balíčků, knihoven, aplikací a spoustu ovládačů pro širokou škálu aktuátorů a senzorů.

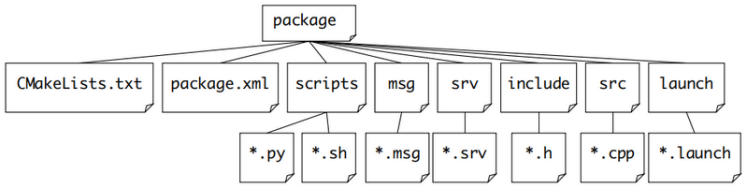
### 6.1.1 Souborový systém

ROS však nabízí víc než jen nástroje a knihovny, ale dokonce i funkce podobné operačnímu systému, jako je abstrakce hardwaru, správa balíčků a vývojářské nástroje. Podobně jako u operačního systému jsou i soubory ROS uspořádány na pevném disku určitým způsobem, který je znázorněn na obrázku 6.1. Meta Packages




Obrázek 6.1: Souborový systém frameworku ROS. Převzato z [9]

označuje jeden čí více souvisejících balíčků, které lze seskupit. Je v zásadě virtuálním balíčkem, který neobsahuje žádné zdrojové kódy ani typické soubory, které se obvykle v balíčcích nacházejí. Packages jsou nezákladnější jednotkou softwaru ROS. Obsahují například jeden či více ROS programů (uzlů), knihovny nebo konfigurační soubory které jsou uspořádány dohromady jako jeden celek. Příkladem co ve většině případů balíček obsahuje je znázorněno na obrázku 6.2.

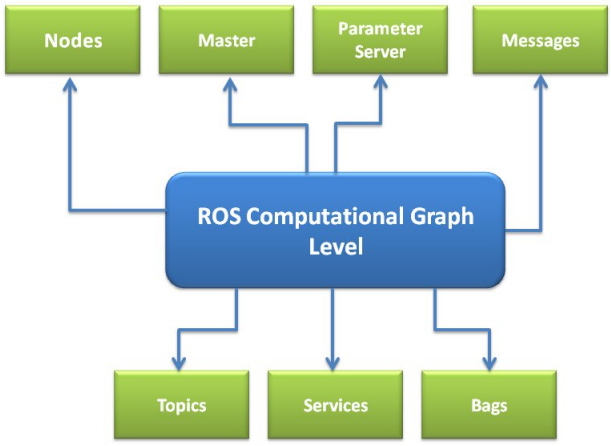



Obrázek 6.2: Struktura ROS balíčku. Převzato z [9]

Balíček (package) tak obsahuje spoustu dalších složek do kterých jsou uloženy další soubory či programy. Jednotlivé složky v balíčku obsahují zpravidla spustitelné soubory či definice, které mají příponu danou názvem složky ve které jsou.

### 6.1.2 Výpočetní graf

Všechny výpočty které se ve frameworku ROS provádí zachycuje výpočtový graf na obrázku 6.3. Nodes (uzly) jsou jednotlivé programy, které typicky zastávají ně-




Obrázek 6.3: Výpočtový graf frameworku ROS. Převzato z [9]

jakou funkci. Můžou například provádět lokalizaci nebo naplánovat trajektorii. Povětšinou tvoří jednoduché než velké procesy, které obsahují všechny funkce [9]. Více uzlů se najednou dá spustit pomocí nástroje roslaunch (`.launch`). Všechny uzly mohou mezi sebou komunikovat přes Messages (`.msg`), pomocí Topics, který se používají k identifikaci obsahu těchto zpráv. Zprávy jsou jednoduše datovou strukturou obsahující typové pole, které může obsahovat sadu dat [9]. Speciálním typem zpráv jsou pak Services (`.srv`), které na rozdíl od zpráv mají mezi sebou interakci požadavek a odpověď. ROS Master zajišťuje registraci a vyhledávání názvů pro zbytek výpočetního grafu. Bez něho by nemohli jednotlivé uzly mezi sebou komunikovat. Součastí ROS Masteru je nově Parametr Server, který umožňuje ukládat data podle klíče na centrálním místě [35]. Bags (`.bag`) je pak formát pro ukládání a zpětné přehrání ROS zpráv.

## 6.2 Návrh a nastavení řídicího softwaru

V této části bude postupně popsán návrh, nastavení a případné použité balíčky, které byli použity pro návrh řídicího softwaru. Nejdříve bude popsáno základní nastavení robota pro následné ovládání a řízení. Následně budou představeny balíčky pro mapování a lokalizaci a jejich základní nastavení. Poté bude představen systém tvorby globální trajektorie, kterou bude robot sledovat. Nakonec pak konečné nastavení řídicího algoritmu.

### 6.2.1 Základní nastavení robota

Prvním krokem pro zprovoznění robota bylo nainstalování sady vývojových nástrojů (SDK) JetPack [30] na Jetson TX2. Tato SDK obsahuje Jetson Linux balíček ovladačů, Linuxové jádro, desktopové prostředí Ubuntu a kompletní sadu knihoven pro akceleraci výpočtů na GPU, multimédia a počítačové vidění. Na mikropočítač byla nainstalována konkretně verze 4.5, kde byl nejdříve stažen SDK manager [32] a následně instalace probíhala podle [42]. Vzhledem k tomu, že probíhala instalace tímto způsobem, nebyly na Jetson TX2 nainstalovány knihovny CUDA, díky kterým můžeme spouštět programy na GPU. Ty musely být pak ještě pomocí SDK manageru doinstalovány.

Po základním nastavení mikropočítače byl nainstalován ROS, konkrétně distribuce Melodic pomocí návodu [38]. Po úspěšném nainstalování frameworku ROS byla pro následnou práci s balíčky potřeba vytvořit složka catkin workspace, kde se dají tyto balíčky upravovat, instalovat a překládat. Proto byla vytvořena tato složka s názvem `f110_ws` reprezentující workspace.

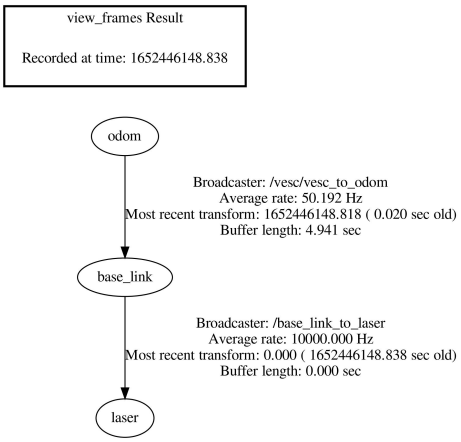
Vzhledem k tomu, že byl robot postaven podle [2], tudíž má robot téměř stejnou strukturu a je osazen až na pár výjimek stejnými součástky byl pro kompletní ovládání a nastavení robotu, po nastavení složky workspace stažen volně dostupný balíček [17] od komunity F1TENTH. Balíček obsahuje všechny potřebné ovladače a nastavení k tomu aby bylo možné jednoduše robota ovládat pomocí frameworku ROS. Příjem dat ze senzorů a zadávání akčních zásahů pak jednoduše probíhá pomocí příjmu čí posílání zpráv. Ovšem aby mohlo ovládání robotu a přijímání dat ze senzorů probíhat správně je potřeba aby byli v operačním systému nastaveny udev pravidla. Tyto nastavení umožňují přiřadit v Linuxu každému zařízení virtuální název. Každé zařízení lze pak jednoduše pomocí těchto pravidel najít. Tyto pravidla byly nastaveny pro LiDAR senzor a regulátor VESC.

Nakonec musí být nastaven VESC tak aby správně fungoval s použitým motorem. Pro jeho nastavení slouží nástroj VESC Tool [46]. Byly zde především identifikovány parametry motoru a následně nastaveny parametry PID regulátoru. Celé nastavení i s parametry použitého motoru je popsáno v [14].

### 6.2.2 ROS transformace

Další důležitou částí je potřeba zavést do systému souřadné systémy a transformace odvozené v kapitole 4.1. Ty jsou jednoduše provedeny pomocí ROS knihovny `tf` [37]. Ta umožňuje zavádět transformace mezi souřadnými systémy a převádět tak data jednoduše mezi jednotlivými souřadnými systémy. Nejdříve byla zavedena statická transformace mezi středem otáčení robota (`base_link`) a senzorem (`scan`). Nakonec dynamická transformace mezi středem otáčení robotu a souřad-

ným systémem mapy (`odom`), kde je tato transformace poskytována pomocí dat ze senzoru odometrie. Zde se jako senzor odometrie používá regulátor VESC, který dokáže měřit otáčky motoru, natočení serva a na základě kinematiky modelu odhadovat polohu robotu. Tato transformace je proto však jakýmsi odhadem mezi těmito dvěma souřadnými systémy. Přesnější transformaci pak bude mezi těmito systémy provádět algoritmus lokalizace.




Obrázek 6.4: Zavedené ROS transformace

### 6.2.3 Balíčky pro mapování a lokalizaci

Pokud jsou zavedeny všechny transformace uvedené v kapitole 6.2.2 můžou být použity balíčky pro mapování a lokalizaci. Pro tvorbu mapy prostředí byl využit ROS balíček `hector_mapping` [25]. Jedná se o balíček, který provádí mapování a lokalizaci druhého algoritmu popsaného v kapitole 5.4. Tento balíček vyžaduje aby byli ve frameworku ROS zavedeny transformace mezi robotem a LiDAR senzorem. Eventuálně může být zavedena právě dříve zmíněná transformace mezi souřadným systémem mapy a robotu pro počáteční odhad polohy.

Pokud máme vytvořenou mapu prostředí je následně potřeba pro získání daleko přesnější polohy se nějak v mapě lokalizovat. Pro lokalizaci v již vytvořené mapě prostředí byl proto použit balíček `particle_filter` [34] od MIT racecar [3]. Tento balíček funguje na principu popsaném v kapitole 5.2.2.

### 6.2.4 Generování trajektorie

Na vytvořené mapě je zapotřebí vygenerovat trajektorii, kterou bude robot sledovat. Vzhledem k tomu, že se jedná o uzavřenou trať bude generovaná trajektorie pro jednoduchost středová čára. Nejdříve je potřeba získat data z mapy pomocí ROS zprávy s topikem `/map`, kde nás bude hlavně zajímat jednorozměrné pole, které obsahuje informace o každé mřížce. Tato zpráva obsahuje ještě informace o poloze počátku mapy.

Pro vygenerování středové čáry je nejdříve potřeba mapu upravit. Ze získaných dat ROS zprávy (`/map`) se dá získat právě zmíněné jednorozměrné pole, které obsahuje informace o každé buňce mapy. Toto jednorozměrné pole bylo nejdříve převedeno na dvojrozměrné. Použitím tohoto převedení se dá využít některých užitečných knihoven v Pythonu. Jelikož vytvořená mapa není často dokonalá je vhodné pro vygenerování středové čáry na mapě dilatovat jednotlivé části mapy reprezentující překážku. Dilatace se provádí použitím Python knihovny `scipy.ndimage` [6], konkrétně pomocí funkce **`binary_dilatation(pole,`** $N$**`)`**, kde **pole** označuje vstupní dvojrozměrné pole a $N$ počet iterací.

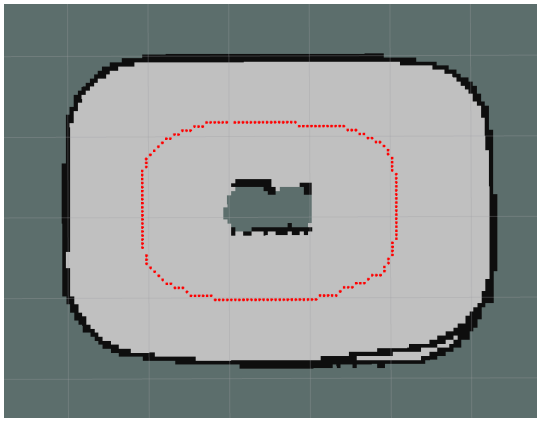
Pokud je mapa dostatečné upravená, tedy proběhl dostatečný počet iterací dilatace, může se následně vytvořit středová čára. Ta se vytvoří za použití další Python knihovny `scikit-image`[5], konkretně pomocí modulů `util` a `morphology`. Pomocí modulu `util` a funkce **`invert(pole)`**, kterou obsahuje se pro skeletonizaci nejdříve musí pole **vstup** invertovat. Následně za použití druhého zmíněného modulu `morphology` byla použita funkce **`skeleotonize(pole)`**. Tato funkce vytvoří z upravené mapy skeleton. Skeleton je pak složen z jednotlivých indexů pole reprezentující mapu.

Tímto dostaneme v jakých mřížkách se nachází jednotlivé body, které teď můžeme nazvat jako body středové čáry. Aby se ale středová čára dala reprezentovat v souřadném systému mapy je potřeba tyto body transformovat. Toho se dosáhne využitím informace o počátku a rozlišení mapy. Celý proces tvorby středové čáry pak reprezentuje pseudokód 1.

Funkce **`sortCenterLine(v)`** pak představuje postupné seřazení bodů od počátku za sebou tak, aby se část středové čáry dala jednoduše proložit nějakou parametrickou funkcí. Princip pak spočívá v tom, že se nejdříve vybere nejbližší bod k nule a ten se uloží do nového pole. K tomuto bodu se pak následně vybere další nejbližší a předchozí se z původního pole smaže. Následně se celý postup opakuje dokud původní pole není prázdné.

**Algorithm 1:** Generování středové čáry

**Input: mapa**
**Output: stredova_cara**
**for** $i = 0$ **to** $vertikalni\_rozliseni\_mapy$ **do**
  **for** $j = 0$ **to** $horizontalni\_rozliseni\_mapy$ **do**
    $\mathbf{M}[i][j] \leftarrow$ **mapa**$[i \cdot vertikalni\_rozliseni\_mapa + j]$;
  **end**
**end**
$\mathbf{M} \leftarrow binary\_dilatation($**pole**$, N)$;
$\mathbf{M} \leftarrow invert(\mathbf{M})$;
$\mathbf{M} \leftarrow skeletonize(\mathbf{M})$;
**for** $i = 0$ **to** $vertikalni\_rozliseni\_mapy$ **do**
  **for** $j = 0$ **to** $horizontalni\_rozliseni\_mapy$ **do**
    **if** $(\boldsymbol{M}[i][j] == \mathit{True})$ **then**
      $\mathbf{v}[i][j] \leftarrow [i, j]$;
    **end**
  **end**
**end**
$\mathbf{m} \leftarrow sortCenterLine(\mathbf{v})$;
**for** $i = 0$ **to** $vertikalni\_rozliseni\_mapy$ **do**
  **stredova_cara**$[i] \leftarrow$
   $[i \cdot velikost\_mrizky + pocatek\_x, i \cdot velikost\_mrizky + pocatek\_y]$
**end**



Obrázek 6.5: Vygenerovaná trajektorie

### 6.2.5 Návrh řídicího algoritmu

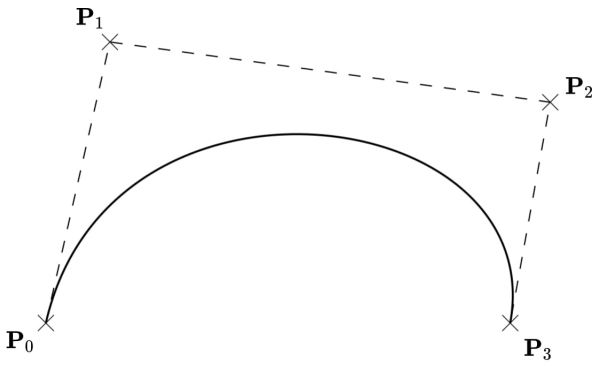
Pro návrh řídicího algoritmu je důležité získání referenčních vztahů, které byly odvozeny v kapitole 4.3.2. Protože středová čára vygenerovaná v kapitole 6.2.4 je složena pouze ze souřadnic bodů v souřadném systému mapy je vhodné tuto trajektorii alespoň v nějaké části proložit. K proložení trajektorie byla využita kubická Beziérova křivka. Z proložené křivky se pak následně dají v každém bodě křivky jednoduše určit referenční parametry. Po získání těchto referenčních parametrů pak může být využit použitý řídicí algoritmus.

#### Kubická Beziérova křivka

Jedná se o parametrickou křivku, která je popsána funkcí

$$\mathbf{C}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3, 0 \leq t \leq 1, \quad (6.2.1)$$

kde $\mathbf{P}_0$ až $\mathbf{P}_3$ jsou takzvané kontrolní body. Kubická Beziérova křivka začíná v bodě $\mathbf{P}_0$ a končí v bodě $\mathbf{P}_3$. Křivka obecně body $\mathbf{P}_1$ a $\mathbf{P}_2$ neprochází. Tyto dva body pouze určují jakého tvaru křivka bude nabývat. Derivací podle času pak získáme potřebné derivace funkce $\dot{\mathbf{C}}(t)$ a $\ddot{\mathbf{C}}(t)$. Z těchto funkcí pak můžeme jednoduše vytvořit referenční hodnoty. Celý algoritmus proložení bodů křivkou zachycuje pseudokód 2.




Obrázek 6.6: Kubická Bezierova křivka. Převzato z [1]

**Algorithm 2:** Získání referenčních hodnot

**Input: stredova_cara**
**Output: reference**
**x** $\leftarrow$ **poloha_robotu**;
$i \leftarrow nejblizsi\_bod($**x**, **stredova_cara**$)$;
$d\_sum \leftarrow 0$;
$v\_max \leftarrow maximalni\_rychlost$;
**while** $d\_sum \leq v\_max$ **do**
    $d\_sum \leftarrow d\_sum +$
     $spocti\_vzdalenost($**stredova_cara**$[i],$ **stredova_cara**$[i+1])$;
    **v**$.pridej($**stredova_cara**$[i])$;
**end**
$L \leftarrow velikost($**v**$)$;
$N \leftarrow pocet\_predikcnich\_kroku$;
$t \leftarrow 0$;
**for** $i = 0$ **to** $N$ **do**
    **bod** $\leftarrow bezier(t,$ **v**$[0],$ **v**$[integer : \frac{M}{3}],$ **v**$[integer : \frac{2M}{3}],$ **v**$[L-1])$;
    $t \leftarrow t + \frac{1}{N}$;
    **reference**$.pridej(vypocti\_reference($**bod**$))$
**end**

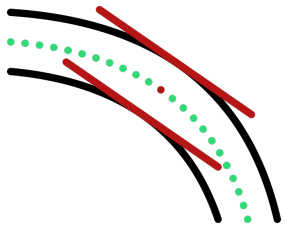
Funkce **`bezier(`**$t,$ **v**$[0],$ **v**$[integer : \frac{M}{3}],$ **v**$[integer : \frac{2M}{3}],$ **v**$[L-1]$**`)`** se zde stará o výpočet polohy souřadnic $x$ a $y$ a jejich derivací na Beziérově křivce na základě parametru $t$.

### 6.2.6 Nastavení řídicího algoritmu

Pro návrh řídicího algoritmu MPC je zapotřebí určit velikost predikčního horizontu. Ten určuje jak moc se bude předpovídat do budoucnosti. Dále je potřeba získat aktuální polohu robotu. Následně musí být získány referenční hodnoty. Tyto referenční hodnoty budou použity jednak k dosazení do matic 4.3.16 a 4.3.17, zároveň ke stanovení referencí, které určují v každém časovém okamžiku v jaké poloze by se ideálně měl robot nacházet. Dále je potřeba zavést lineární omezení na polohu, kde se robot může nacházet, maximální rychlost a natočení kol. Poté už jen stačí všechno až na jednu změnu poskládat do matic a vektorů, jak je popsáno v kapitole 4.4.2.

Jelikož se ale ukázalo jako vhodné reprezentovat omezení trati v každém časovém okamžiku pomocí přímek, nemůže se použít reprezentace omezení stavů ukázané v kapitole 4.4.2. Musí se tak upravit část jednotkové matice **I** obsažené v matici 4.4.11. Horní část této jednotkové matice definuje omezení právě stavů.

Zároveň musí být změněná část vektoru 4.4.9 a 4.4.10, definující maximální a minimální hranici možných stavů. Jak lze vidět na obrázku 6.7 dvě červené čáry repre-



Obrázek 6.7: Přímky definující omezení trati

zentují v daném okamžiku ohraničení trati pomocí dvou přímek, respektive každá z přímek může rozdělovat prostor na dvě části. Toto omezení se dá zapsat maticově

$$\mathbf{Mx} \leq \mathbf{b}, \tag{6.2.2}$$

kde řádek této matice reprezentuje právě jednu přímku. Jelikož každá z těchto přímek bude reprezentovat rozdělení části prostoru na dvě části, můžeme použitím dvou přímek definovat právě prostor mezi těmito přímkami. Část vektoru 4.4.9 definující pouze horní omezení bude nabývat tvaru

$$\mathbf{b}_{\text{ub}} = \begin{bmatrix} \dots & b_1 & -\infty & \psi_{\max} & \dots & b_1 & -\infty & \psi_{\max} & \dots \end{bmatrix}^\top. \tag{6.2.3}$$

A část vektoru 4.4.9 pak následujícího tvaru

$$\mathbf{b}_{\text{lb}} = \begin{bmatrix} \dots & \infty & b_2 & \psi_{\min} & \dots & \infty & b_2 & \psi_{\min} & \dots \end{bmatrix}^\top. \tag{6.2.4}$$

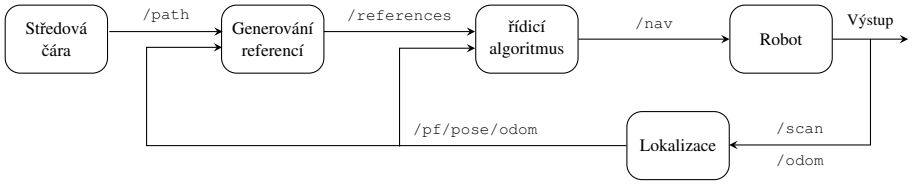
Oba vektory jsou tak změněny pouze v části, kde se definuje omezení na stavy. Nakonec část jednotkové matice určující stavy bude nabývat tvaru

$$\mathbf{I} = \begin{bmatrix} \mathbf{M} & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & \mathbf{M} & & & \\ & & & & 1 & & \\ & & & & & \ddots & \end{bmatrix}. \tag{6.2.5}$$

Zbytek matice definující omezení na řídicí vstupy má opět na diagonále jedničky.

K nalezení optimálního řízení s konečným horizontem v každém časovém okamžiku byl použit solver `OSQP` [40]. Vzhledem k tomu, že byl pro implementaci

využit programovací jazyk C++ a k tomu byla použita knihovna `Eigen` [22], byla následně pro převedení nadefinovaných matic do solveru `OSQP`, použita wrapper knihovna `osqp-eigen` [36]. Celý proces řídicího algoritmu je zachycen na obrázku 6.8, který především zachycuje jak se hodnoty z jednotlivých částí přenáší. Každý z těchto bloků pak poskytuje jako výstup jednu čí více ROS zpráv, ozna-




Obrázek 6.8: Struktura řídicího algoritmu

čených pomocí topiku. Zpráva pak obsahuje informace které daný blok pomocí algoritmů popsaných výše vygeneroval.

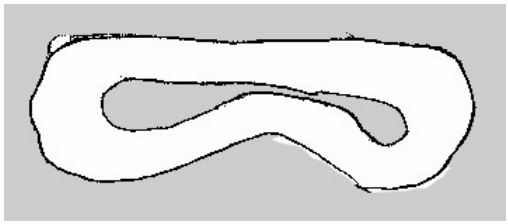
# Kapitola 7

# Experimenty

V této části budou ukázány a popsány dosažené výsledky experimentů. První experimenty pro ověření funkčnosti algoritmů byli provedeny nejdříve v simulátoru [16] od F1TENTH. Po úspěšném otestování byly provedeny následně experimenty i na použité robotické platformě, kde bylo provedeno i kontrolní měření pomocí systému VICON [8]. Algoritmy lokalizace a mapování budou testovány pouze u experimentu na reálné robotické platformě.

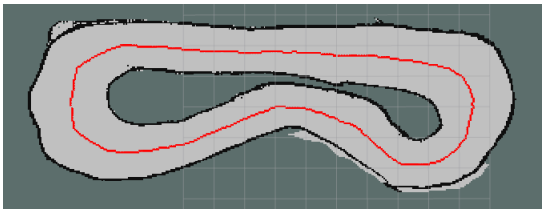
## 7.1 Experimenty v simulátoru

V simulátoru byl testován pouze algoritmus generování trajektorie a řídicí algoritmus. Simulátor nabízí velký počet map závodních tratí, které se dají použít. Výhodou simulátoru je to, že nepotřebuje žádný externí systém lokalizace, neboť poskytuje pomocí ROS zprávy přesnou informaci o poloze robotu. Pro experimenty v simulátoru byla vybrána mapa zobrazená na obrázku 7.1. Na této mapě byl ná-



Obrázek 7.1: Použitá mapa pro experiment v simulátoru

sledně otestován algoritmus generování středové čáry získaný v kapitole 6.2.4. Vygenerovaná středová čára je pak zobrazena na obrázku 7.2. Vygenerovaná čára, tak jak lze vidět z obrázku nabývá právě tvaru středové čáry, která byla požadována.



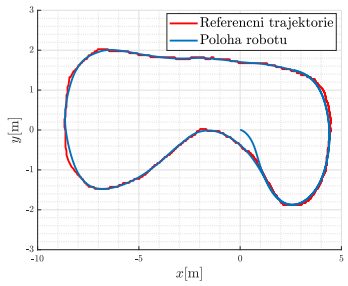
Obrázek 7.2: Vygenerovaná středová čára

Obrázek 7.2 zde ukazuje, že vygenerovaná referenční trajektorie není příliš hladká. To je způsobeno tím jak se tato trajektorie generuje. Jelikož funkce tvořící středovou čáru využívá ohraničení trati, které také není hladké, nevznikne tak ani hladká středová čára. Jelikož se pak tato středová čára v každém časovém okamžiku na základě aktuální polohy prokládá Beziérovou křivkou není tato kostrbatost vyložený problém. Na druhou stranu, pokud je zatáčka příliš ostrá, proložení křivky nebude příliš přesné z důvodu volby kontrolních bodů.

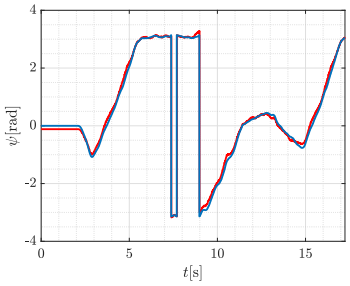
Po vygenerovaní středové čáry byl otestován právě řídicí algoritmus pro sledování vygenerované středové čáry představený v kapitole 6.2.6. V simulaci se vycházelo ze simulačního robotu ve formě 1:10 zmenšeného autíčka. Délka rozvoru $L$ tohoto simulačního autíčka měří $0.3302$ m. Algoritmus byl spuštěn se vzorkovací periodou $T = 50$ ms. Predikční horizont pak nabýval hodnoty $N = 10$. Matice **Q** a **R** byli pro simulaci nastaveny s hodnotami

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.4 \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}. \tag{7.1.1}$$

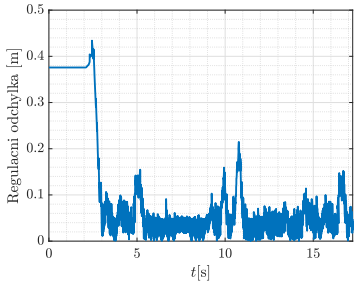
Maximální rychlost autíčka $v_{\max}$ byla nastavena na $3\frac{\mathrm{m}}{\mathrm{s}}$ a maximální natočení předních kol $\delta_{max}$ bylo nastaveno na hodnotu $-\frac{\pi}{6} \leq \delta \leq \frac{\pi}{6}$. Trajektorie průjezdu autíčka, jak lze vidět na obrázku 7.3(a) je oproti vygenerované středové čáře hladká. Zároveň lze vidět, že trajektorii sleduje celkem pěkně, dokonce má lepší průběh než středová čára. Na obrázku 7.3(b) pak lze vidět, že referenční natočení autíčka a reálné natočení se téměř shodují. Následně byla vykreslena regulační odchylka polohy auta od referenční trajektorie v průběhu času, která je zobrazena na obrázku 7.3(c). Z tohoto obrázku lze vidět že po najetí autíčka na trajektorii nepřesáhla regulační odchylka $0.25$ m. Téměř polovina hodnot regulační odchylky však nepřesáhla hodnotu $0.1$ m.




(a) Sledování trajektorie




(b) Sledování referenčního natočení autíčka




(c) Velikost regulační odchylky

Obrázek 7.3: Výsledky experimentu v simulátoru

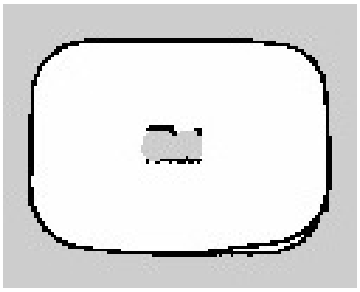
## 7.2 Experimenty na reálném robotu

Následně byly provedeny experimenty právě na reálném robotu 3. Experimenty byli prováděny na vyrobené trati zobrazené na obrázku 7.4. Vzhledem k dostup-

Obrázek 7.4: Fotky vyrobené dráhy pro experimenty

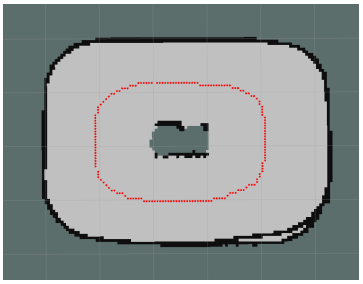
ným prostorům není trať příliš velká. Nabývá jednoduše tvaru oválu, aby robot byl vůbec schopen vzhledem k maximálnímu otáčení robotu trať projet.

Tuto trať bylo tak nejdříve potřeba zmapovat. Výsledkem mapování této trati je mapa zobrazená na obrázku 7.5. Vytvořená mapa odpovídá až na pár nedokonalostí právě vyrobené dráze, kterou mapoval. Lze vidět, že překážka v prostředku dráhy není na kratší stranách vidět. To je nejspíše z důvodu toho, že robot, který se zde otáčí neměl možnost tuto překážku zaznamenat, protože rozsah LiDAR senzoru je $180$ deg. Následně jelikož algoritmus neumožňuje pro mapování uzavírání smyček je na obrázku v levé dolní polovině vidět mírná nesrovnalost na začátku mapování a na konci, kde robot dojel do místa, kde začínal. Na této mapě byla pak opět



Obrázek 7.5: Mapa vyrobené dráhy

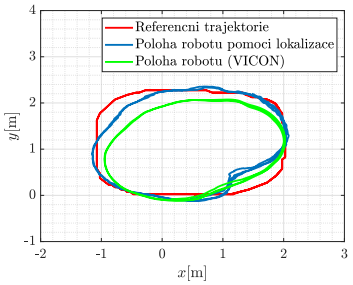
vygenerována středová čára zobrazená na obrázku 7.6. Opět jako u experimentu v simulaci trajektorie nabývá středové čáry a není opět úplně hladká.



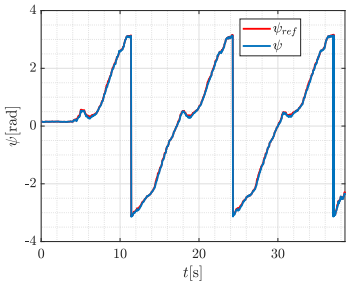
Obrázek 7.6: Vygenerovaná středová čára

Následně byl otestován řídicí algoritmus. U reálného robotu se rozvor $L$ oproti simulačnímu autíčku nepatrně liší. Měří zhruba $0.325$ m. Algoritmus byl opět spuštěn se stejnými parametry jaké byli uvedeny u experimentu v simulátoru až na maximální rychlost, která byla $v_{\text{max}} = 1\ \frac{\text{m}}{\text{s}}$. Při tomto experimentu bylo však zapotřebí již použít systém lokalizace. Zároveň byl pro testování systému lokalizace

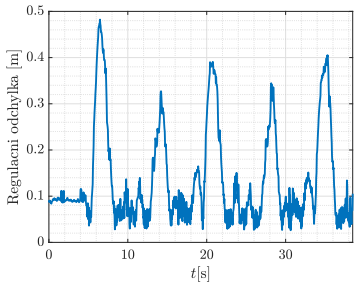
průjezdu trati použit dříve zmíněný VICON. Trajektorie průjezdu trati je pak zobrazena na obrázku 7.7(a). Na obrázku lze vidět především rozdíl v poloze robota




(a) Sledování trajektorie



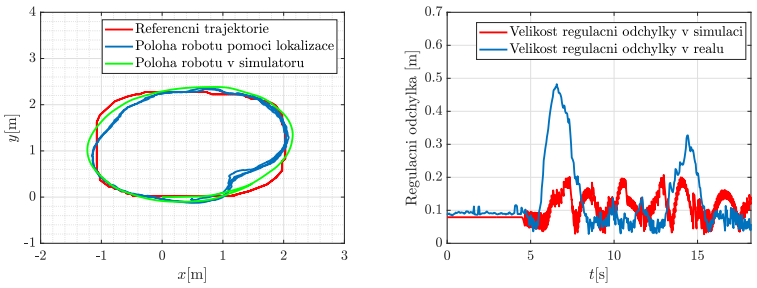

(b) Sledování referenčního natočení autíčka




(c) Velikost regulační odchylky

Obrázek 7.7: Výsledky experimentu na vytvořené trati

poskytovaného pomocí systému lokalizace a určené poloze pomocí systému VICON. Data ze systému VICON by měli ukazovat na přesnější polohu. Obě trajektorie robota však trajektorii nesledují zcela přesně, především pak na levém horním a pravém dolním kraji trati v případě určení polohy pomocí systému lokalizaci. Důvodů proč trajektorii nesledují přesněji může být tak více. Na obrázku 7.7(b) pak lze vidět, že požadované natočení robotu bylo sledováno docela přesně. Nakonec regulační odchylka zobrazená na obrázku 7.7(c) dosahovala maximální hodnoty $0.5$ m. Většinu času však nepřesáhla hodnoty $0.4$ m. To je oproti výsledku v simulátoru daleko horší. Jelikož byl test v simulátoru proveden na jiné mapě bude pro úplnost srovnán tento průjezd s výsledkem průjezdu se stejnými parametry a stejnou mapou v simulátoru. Bude se tedy porovnávat pouze předchozí experiment bez systému VICON a experiment v simulátoru. Srovnání těchto dvou experimentů lze vidět na obrázcích 7.8. Konkrétně na obrázku 7.8(a) lze vidět, že řídicí algoritmus opět přesně nesleduje středovou čáru. Nicméně jak potvrzuje obrázek 7.8(b), který




(a) Sledování trajektorie (b) Velikost regulačních odchylek

Obrázek 7.8: Srovnání výsledků v simulaci a na reálném robotu

srovnává regulační odchylku zhruba ve stejném čase, nestávají zde tak velké odchylky. V některých částech však má menší regulační odchylku právě experiment na reálném robotu.

Jak bylo zmíněno předtím, důvodů může být více. Nejpravděpodobnějším důvodem je však to, že středová čára je prokládána právě Beziérovou křivkou. Tento způsob má však jednu nevýhodu, která zapříčiňuje to, že při prudších zatáčkách neprokládá Beziérova křivka tolik bodů. Svým způsobem si pomocí Beziérovy křivky může zkrátit trasu, což tak trochu lze vidět právě na obrázku 7.8(a). Protože algoritmus MPC vždy predikuje do budoucnosti a sleduje v predikčním horizontu Beziérovu křivku, která právě při větších zatáčkách není tak zakroucená jako středová čára může být právě tohle problém. Toto tvrzení podporuje výsledek z experimentu na obrázku 7.3(a), kde na rovince, trajektorii sleduje nejlépe.

Důvodem proč experiment na reálném robotu má v některých místech větší regulační odchylku může být špatnou lokalizací v daném místě. Například mohl být robot trochu nakloněn a tím pádem nemusel LiDAR senzor zaznamenat překážku.

# Kapitola 8

# Závěr

Hlavní cílem této práce bylo navrhnutí řídicího systému robotu pro projetí závodní trati. K dosažení takového cíle bylo zapotřebí splnit několik podcílů. Nejdříve proto bylo zapotřebí se seznámit s použitou robotickou platformou. Následně navrhnout automatický systém tvorby mapy. Nakonec samotný řídicí algoritmus, který umožní na základě vytvořené mapy trať projet.

V první části byla proto popsána použitá robotická platforma ve formě RC autíčka. Pro řízení tohoto autíčka byli ve druhé kapitole zavedeny potřebné souřadné systémy, transformace a nakonec zjednodušený kinematický model autíčka. Následně byl představen řídicí algoritmus prediktivního řízení. Při testování bylo zjištěno, že pro dosažení lepších výsledků je vhodné reprezentovat ohraničení trati pomocí přímek rozdělující prostor. Musela být proto upravena obecná definice podmínek pro stavy optimalizačního problému.

Ve třetí části pak byl představen problém simultánní lokalizace a mapování. Během testování se ukázalo jako vhodné použít pro mapování závodní trati ROS balíček HECTOR SLAM. Ten sice neumožňuje uzavírání smyček, ale na druhou stranu dokázal při jednom průjezdu vytvořit velice slušnou mapu. Mapa sloužila nejen k následné lokalizaci v prostředí, ale také především k návrhu trajektorie. Výsledky ukazují, že vygenerovaná trať není příliš hladká. Během experimentů se následně ukázalo, že pro co nejpřesnější sledovaní trajektorie není použití Beziérovy křivky příliš vhodné.

Jedním z vylepšení, které by tak v budoucnu mohli být řešeny a aplikovány je lepší generace referenční trajektorie. Nejlépe pak generování časově optimální trajektorie. Dále by mohl být pro predikci použit dynamický model. Zároveň by se pak mapa mohla generovat zcela automaticky bez ovládání člověkem.

# Příloha A

# Nastavení pro otestování v simulátoru

Všechny algoritmy i samotný simulátor, který byl použit pro otestování právě řídicích algoritmů je k nalezení v repositáři na adrese `https://github.com/petrkuchar/sim_ws`. Ke zprovoznění je zapotřebí mít na počítači nainstalovaný operační systém Linux. Simulátor byl odzkoušen konkrétně na Ubuntu 20.04 a Ubuntu 18.04, proto by bylo nejvhodnější použít jednu z těchto distribucí. Dále je zapotřebí mít na počítači nainstalovaný softwarový rámec ROS, nejlépe distribuci Melodic. Dále je zapotřebí nainstalovat balíčky, které jsou nutné pro spuštění simulátoru. Nejdříve tak aktualizujeme dostupné balíčky a jejich verze pomocí příkazu:

```
$ sudo apt-get update
```

Poté se balíčky jednoduše nainstalují konkrétně pro ROS Melodic pomocí příkazů v terminálu:

```
$ sudo apt-get install ros-melodic-tf2-geometry-msgs
$ sudo apt-get install ros-melodic-ackermann-msgs
$ sudo apt-get install ros-melodic-joy
$ sudo apt-get install ros-melodic-map-server
```

Dále je potřeba stáhnout Python knihovny pro spuštění algoritmu vygenerování trajektorie. Všechny potřebné knihovny se dají nainstalovat pomocí dvou příkazů:

```
$ sudo apt-get install python-skimage
$ sudo apt-get install python-scipy
```

Poslední částí, která je důležitá pro otestování řídicího algoritmu je nainstalování dříve zmíněných knihoven `OSQP`, `Eigen` a `osqp-eigen`. Celý proces instalace

knihovny `OSQP` je popsán přímo v návodu [33]. Stejně tak nainstalování knihovny `osqp-eigen`, kde návod na instalaci je popsán rovnou v repositáři [36]. Nakonec knihovna `Eigen` se nainstaluje pomocí příkazu:

```
$ sudo apt-get install libeigen3-dev
```

Po nainstalovaní všech těchto knihoven se bude následně dát celý workspace přeložit pomocí následujících příkazů:

```
$ cd ~/simulation_ws
$ catkin_make
$ source devel/setup.bash
```

Po úspěšném přeložení je už možné vyzkoušet simulátor, generování trajektorie a řídicí algoritmus pomocí sady příkazů, kde každý příkaz bude zadán v jiném terminálu:

```
$ roslaunch f1tenth_simulator simulator.launch
$ rosrun navigation line.py
$ roslaunch navigation controller.launch
```

První příkaz spouští simulátor, druhý generování trajektorie a třetí spouští řídicí algoritmus. Jelikož řídicí algoritmus vysílá řídicí vstupy na topik `/nav` je potřeba v terminálu ve kterém je spuštěn simulátor napsat písmeno **n**, které zajistí že autíčko bude přijímat zprávy z topiku `/nav`, kde by následně autíčko by mělo sledovat trajektorii.

# Příloha B

# Nastavení robota pro otestování

Použité balíčky a implementované programy, které byli použity pro testovaní na použité robotické platformě 3 jsou k nalezení v repositáři na odkazu `https://github.com/petrkuchar/f110_ws`. Pro zprovoznění je zapotřebí mít minimálně takovou robotickou platformu, která bude postavena podle návodu [15] na sestavení autonomního závodního robota od F1TENTH.

Hlavní nastavení robota je popsáno v kapitole 6.2.1. Repositář už obsahuje workspace, tudíž stačí pouze nastavit samotný mikropočítač, nainstalovat ROS a popřípadě nastavit regulátor VESC. Po stažení bude nejspíše potřeba nainstalovat navíc nějaké ROS balíčky. Ty se nainstalují pro ROS Melodic pomocí příkazů v terminálu:

```
$ sudo apt-get update
$ sudo apt-get install ros-melodic-driver-base
```

Následně potřebují být všechny Python skripty spustitelné. To se provede pomocí příkazů:

```
$ cd f110_ws
$ find . -name "*.py" -exec chmod +x {} \;
```

Poté je zapotřebí nastavit udev pravidla. Především pak pro regulátor VESC a LiDAR senzor. Tyto pravidla se jednoduše nastaví pomocí druhé části návodu [18]. Po nastavení těchto pravidel je následně potřeba pro mapování nainstalovat balíček HECTOR SLAM. Nainstalování tohoto balíčku se provede pomocí příkazu:

```
$ sudo apt-get install ros-melodic-hector-slam
```

Následně je speciálně pro algoritmus generování trajektorie potřeba nainstalovat určité Python knihovny. Ty se nainstalují pomocí příkazů v terminálu:

```
$ sudo apt-get install python-skimage
```

```
$ sudo apt-get install python-scipy
```

Následně je zapotřebí pro zprovoznění řídicího algoritmu nainstalovat C++ knihovny. Jedná se o knihovny `OSQP`, `Eigen` a `osqp-eigen`. Celý proces instalace knihovny `OSQP` je popsán přímo v návodu [33]. Knihovnu `Eigen` nainstalujeme pomocí příkazu v terminálu:

```
$ sudo apt-get install libeigen3-dev
```

Nakonec je pro nainstalování knihovny `osqp-eigen` potřeba postupovat pomocí návodu [36]. Pro zprovoznění systému lokalizace je ještě zapotřebí následně nainstalovat knihovnu `RangeLibc` [24] podle návodu, který je u ní uveden. Po úspěšném nastavení a nainstalovaní příslušných knihoven stačí workspace už jen přeložit pomocí sady následujících příkazů:

```
$ cd ~/f110_ws
$ catkin_make
$ source devel/setup.bash
```

Pro mapování je potřeba spustit následující sekvenci příkazů, každý v jiném terminálu:

```
$ roslaunch racecar teleop.launch
$ roslaunch racecar mapping.launch
```

Vytvořenou mapu pak lze následně uložit pod názvem `track` následující sadou příkazů:

```
$ cd ~/f110_ws/src/particle_filter/maps
$ rosrun map_server map_saver -f track
```

Po zmapování lze již spustit řídicí algoritmus. To se provede opět sadou příkazů, kde každý příkaz bude v jiném terminálu:

```
$ roslaunch racecar teleop.launch
$ roslaunch partcile_filter localize.launch
$ rosrun navigation line.py
$ rosrun navigation controller.launch
```

Ovládání robotu bylo testováno na XBOX One ovladači. Pro ovládání robotu ručně je potřeba mít stisknuté tlačítko LB a pro autonomní řízení pak tlačítko RB.

# Literatura


[1] Bézier curve. https://en.wikipedia.org/wiki/Bézier_curve. Navštíveno: 2022-04-22.

[2] F1TENTH. https://f1tenth.org/. Navštíveno: 2022-04-20.

[3] MIT Racecar. https://racecar.mit.edu/. Navštíveno: 2022-04-26.

[4] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control. Navštíveno: 2022-04-22.

[5] scikit-image. https://scikit-image.org/. Navštíveno: 2022-04-26.

[6] scipy.ndimage. https://docs.scipy.org/doc/scipy/reference/ndimage.html. Navštíveno: 2022-04-26.

[7] University of Pennsylvania. https://www.upenn.edu/. Navštíveno: 2022-02-24.

[8] VICON. https://www.vicon.com/. Navštíveno: 2022-5-15.

[9] Lentin Joseph a Jonathan Cacace. *Mastering ROS for Robotics Programming - Second Edition*. Bantam, London, 1988.

[10] Andrea Censi. ICP algoritmus. https://censi.science/pub/research/2008-icra-plicp.pdf. Navštíveno: 2022-04-26.

[11] Connect Tech. Orbitty Carrier. https://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/. Navštíveno: 2022-03-20.

[12] Ph.D. Doc. Ing. Jiří Melichar, CSc. Ing. Martin Goubej. *Lineární systémy 1*. Plzeň, 2017.

[13] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[14] F1TENTH. Configuring the VESC. `https://f1tenth.readthedocs.io/en/stable/getting_started/firmware/firmware_vesc.html`. Navštíveno: 2022-04-26.

[15] F1TENTH. F1TENTH - Build. `https://f1tenth.org/build.html`. Navštíveno: 2022-04-26.

[16] F1TENTH. `f1tenth_simulator`. `https://github.com/f1tenth/f1tenth_simulator`. Navštíveno: 2022-04-26.

[17] F1TENTH. `f1tenth_system`. `https://github.com/f1tenth/f1tenth_system/tree/braking`. Navštíveno: 2022-04-26.

[18] F1TENTH. ROS Workspace Setup. `https://github.com/f1tenth/f1tenth_doc/blob/stable/getting_started/firmware/drive_workspace.rst`. Navštíveno: 2022-05-01.

[19] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. pages 343–349, 01 1999.

[20] G. Klančar, A. Zdešar, S. Blažič, I. Škrjanc. *Mobile Robotics*. Joe Hayton, 2016.

[21] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[22] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[23] HOKUYO. URG-04LX-UG01. `https://hokuyo-usa.com/products/lidar-obstacle-detection/urg-04lx-ug01`. Navštíveno: 2022-03-20.

[24] kctess5. `range_libc`. `https://github.com/kctess5/range_libc`. Navštíveno: 2022-05-01.

[25] Stefan Kohlbrecher. `hector_mapping`. `http://wiki.ros.org/hector_mapping`. Navštíveno: 2022-04-26.

[26] Steven M. LaValle. A simple car. http://planning.cs.uiuc.edu/node658.html. Navštíveno: 2022-03-20.

[27] LORD. 3DM-CV5-25. https://www.microstrain.com/inertial-sensors/3dm-cv5-25. Navštíveno: 2022-03-20.

[28] How To Mechatronics. How Brushless Motor and ESC Work. https://howtomechatronics.com/how-it-works/how-brushless-motor-and-esc-work/. Navštíveno: 2022-03-20.

[29] Vu Minh. *Trajectory Generation for Autonomous Mobile Robots*, volume 540, pages 195–214. 02 2014.

[30] NVIDIA. JETPACK SDK. https://developer.nvidia.com/embedded/jetpack. Navštíveno: 2022-03-20.

[31] NVIDIA. Jetson TX2 Module. https://developer.nvidia.com/embedded/jetson-tx2. Navštíveno: 2022-03-20.

[32] NVIDIA. NVIDIA SDK Managers. https://developer.nvidia.com/nvidia-sdk-manager. Navštíveno: 2022-03-20.

[33] OSQP. Build from sources. https://osqp.org/docs/get_started/sources.html. Navštíveno: 2022-04-26.

[34] MIT Racecar. particle_filter. https://github.com/mit-racecar/particle_filter. Navštíveno: 2022-04-26.

[35] Willow Garage Stanford Artificial Intelligence Laboratory Open Robotics. ROS. https://www.ros.org/. Navštíveno: 2022-04-26.

[36] Robotology. osqp-eigen. https://github.com/robotology/osqp-eigen. Navštíveno: 2022-05-01.

[37] ROS. tf. http://wiki.ros.org/tf. Navštíveno: 2022-05-01.

[38] ROS. Ubuntu install of ROS Melodic. http://wiki.ros.org/melodic/Installation/Ubuntu. Navštíveno: 2022-04-26.

[39] Shahrizal Saat, WN Rashid, MZM Tumari, and MS Saealal. Hectorslam 2d mapping for simultaneous localization and mapping (slam). *Journal of Physics: Conference Series*, 1529:042032, 04 2020.

[40] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[41] STEREOLABS. ZED Stereo Camera. https://www.stereolabs.com/zed/. Navštíveno: 2022-03-20.

[42] Connect Tech. CTI-L4T Board Support Package Installation for NVIDIA JetPack with Connect Tech Jetson™ Carriers. https://connecttech.com/resource-center/kdb373/. Navštíveno: 2022-03-20.

[43] Traxxas. Ford Fiesta ST Rally. https://traxxas.com/products/models/electric/ford-fiesta-st-rally. Navštíveno: 2022-03-26.

[44] Traxxas. Velineon 3500. https://traxxas.com/products/parts/motors/velineon3500motor. Navštíveno: 2022-03-20.

[45] Benjamin Vedder. VESC – Open Source ESC. http://vedder.se/2015/01/vesc-open-source-esc/. Navštíveno: 2022-03-20.

[46] VESC. VESC Tool. https://vesc-project.com/vesc_tool. Navštíveno: 2022-03-05.

[47] Švec Josef. Návrh řídicího systému kolového robota s využitím softwarového rámce ROS. Diplomová práce, Fakulta aplikovaných věd Západočeské univerzity v Plzni, Plzeň, Česká republika, 2019.