České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Plánování hladké trajektorie pro mobilní roboty

Praha, 1.6.2009 **Autor: Petr Beneš**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Děkuji vedoucímu práce Michalu Sojkovi a celému týmu CTU Dragons za možnost podílet se na zajímavém projektu.

Abstrakt

Cílem této práce je analyzovat možnosti plánování pohybu autonomního mobilního robota po spojitých křivkách, vybrat jednu z možností, která nejvíce odpovídá požadavkům a omezením spojeným s fyzickými parametry robota a časovou náročností použitých výpočetních algoritmů. Tyto algoritmy jsou následně modelovány v prostředí Matlab a nakonec implementovány jako knihovna v jazyce C++. Integrací této knihovny do celého projektu jsme dosáhnuli hezčího, plynulejšího pohybu, a také možnosti snížení doby průjezdu trajektorií, což přispěje k většímu počtu manévrů, které robot stihne během hrací doby.

Abstract

The aim of this work is to analyze possibilities of mobile autonomous robot trajectory planning using smooth curves, choose one of the options which fits the most to requests and restrictions related to the physical parameters of robot and time complexity of utilised algorithms. Consequently, these algorithms are simulated in Matlab and finally implemented as a C++ library. Thanks to the library integration we gained a more fluent motion and a possibility to lower the pass period. This contributes to increased number of movements during the game period.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Na Katedře řídicí techniky FEL je vyvýjen autonomní mobilní robot, na kterém pracují především studenti za účelem účasti na soutěžích. Tento robot se časem zdokonaluje a přibývají mu nové součásti. Jeden ze základních prvků je plánování trajektorie, které bylo potřeba vylepšit o jízdu po spojitých křivkách.

V kapitole 2 přiblížím zmíněnou soutěž a představím robota. V kapitole 3 popíši některé související součásti, které již v robotu fungují a budu s nimi spolupracovat nebo je měnit. Kapitola 4 ukáže, jaké spojité křivky je možné použít a proč. Od kapitoly 5 se jedná o mou vlastní práci, v této kapitole (5) se zabývám tím, jak vygenerovat námi požadovanou křivku a pak i celou trajektorii, další kapitola 6 je o způsobu průjezdu robota trajektorií. V poslední kapitole 7 je algoritmus vyhýbání se překážkám s použitím algoritmů předchozích. Práci končím závěrem v kapitole 8

Kapitola 2

Parametry robota a hřiště

Přestože je robot navržen tak, aby byl variabilní a použitelný pro různé druhy činností a obsahuje spoustu fukcionalit, které drobnými upravami mohou vytvořit nový funkční celek, jeho hlavní úlohou je v současné době účast na soutěžích Eurobot. Na úvod ukáži jak vypadá hřiště, jeho parametry, robota a jeho fyzikální model, aby bylo z čeho vycházet a na co se odkazovat v následujících kapitolách.

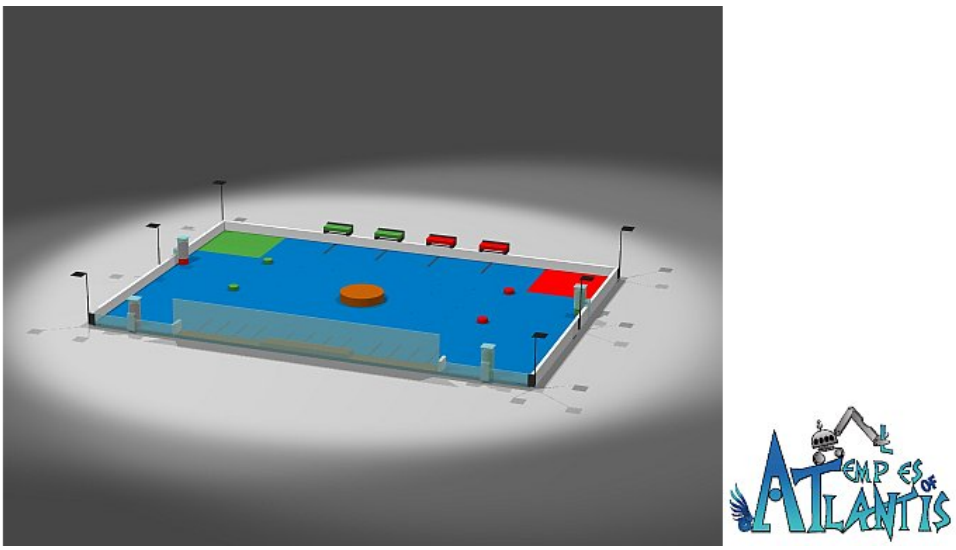
2.1 Hriště pro soutěž Eurobot

Soutěž Eurobot (*Webové stránky Českého kola Eurobot* [online], 2009) se pořádá jednou ročně a pokaždé má jiné zadání, tudíž je potřeba vytvořit nové manipulační schopnosti. Přestože hřiště pokaždé vypadá trochu jinak a také rozložení herních prvků je různé, co se týče rozměrů je vždy obdélníkového půdorysu o rozměrech $300 \times 210cm$.

Algoritmy robota mohou tedy počítat s relativně malým, omezeným hřištěm, na kterém je rozmístěno několik herních prvků a soupeř, který může bránit v pohybu nebo zapříčinit kolizi.

2.2 Model Robota

Samotný robot je obdélníkového půdorysu, má dvě hnaná kola a dvě malá vlečená kuličková kolečka. Pro některé kinematické úlohy, kdy je potřeba brát v úvahu rozměry, má tři stupně volnosti. Pro jiné úlohy, kdy je potřeba robota vidět pouze jako hmotný

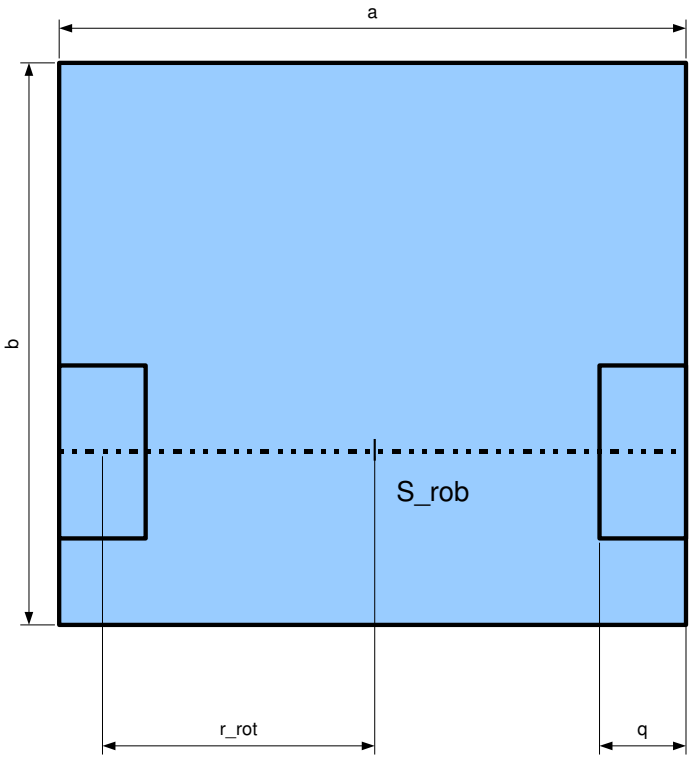


Obrázek 2.1: Hřiště pro Eurobot 2009 – Chrámy atlantidy

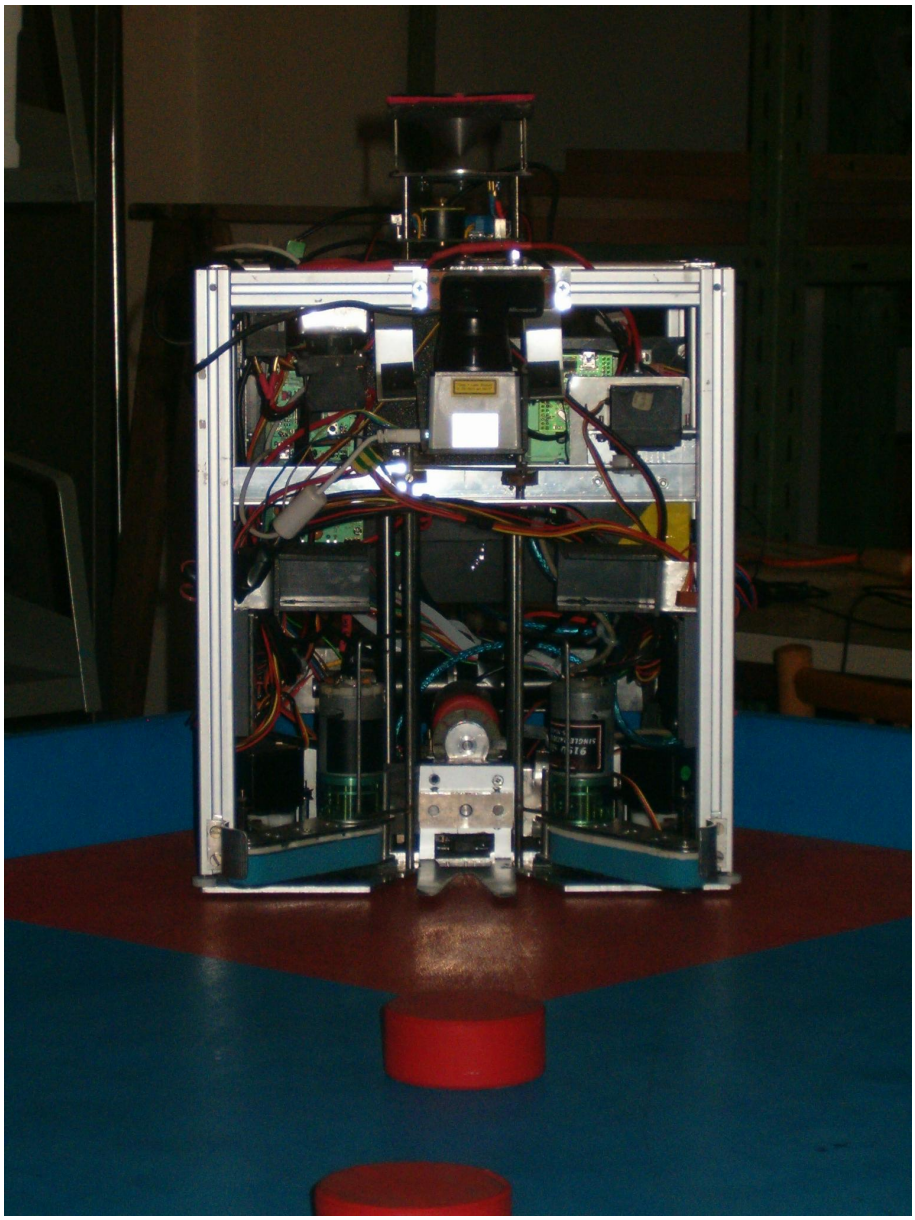
bod, uvažujeme jen dva stupně volnosti (souřadnice x a y). Tento hmotný bod je středem robota S_{rob}, který je umístěn ve středu osy hnaných kol.

Pokud označíme a jako šířku a b jako délku robota (viz obr. 2.2), dále q jako šířku zapuštěných kol, pak poloměr otáčení robota je

$$r_{rot} = \frac{a - q}{2} \tag{2.1}$$



Obrázek 2.2: Půdorys robota



Obrázek 2.3: DragonBot

Kapitola 3

Původní funkce plánování pohybu

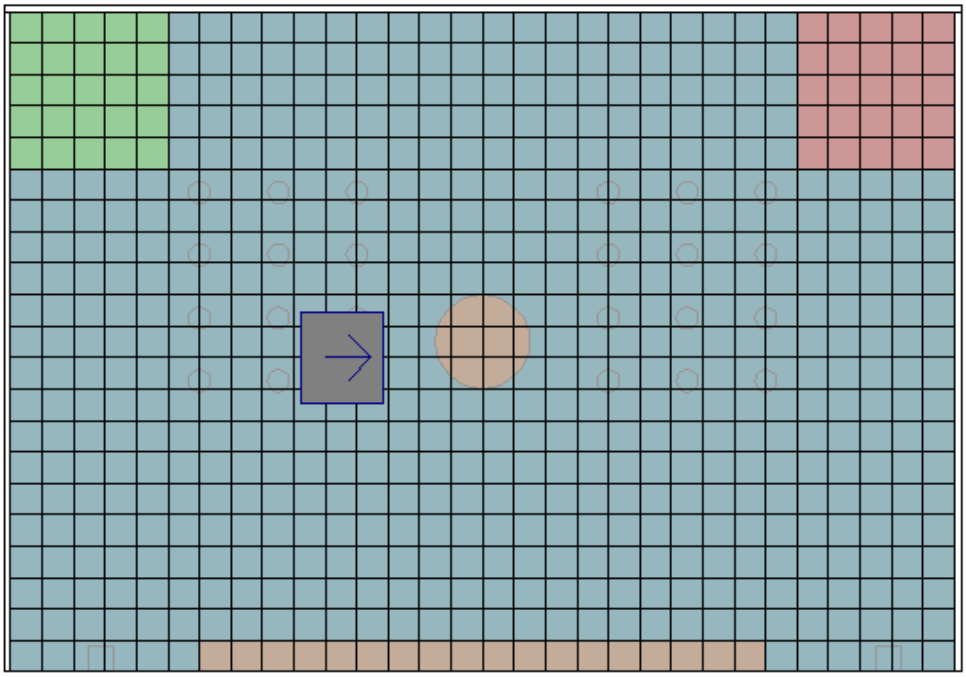
Nyní představím algoritmy a funkcionality, o které doposud opíralo plánování pohybu (*Webové stránky CTU Dragons* [online], 2009) a které jsem jen nepatrně poupravil, aby vyhovovaly požadavkům na integraci s dále zmíněnými částmi.

3.1 Mapa hřiště

Jak již jsem zmínil, na hřišti se nachází mnoho herních prvků, se kterými je třeba počítat, pokud chceme naplánovat trajektorii. Pokud polohy těchto prvků zjistíme, nebo je známe již před soutěží, zaznamenáme je do mapy. Z této mapy potom čerpáme informace o poloze jednotlivých objektů. Pomocí programu Robomon (obr. 3.1) s grafickým uživatelským rozhraním můžeme sledovat během testování polohu robota i objektů v mapě.

3.1.1 Struktura mapy

Mapa je reprezentována 2D mřížkou, která dělí hřiště na jednotlivé buňky. Současná velikost buněk je $10 \times 10cm$, což při rozměrech hřiště $300 \times 210cm$ tvoří pole o rozměru 30×21 buněk. Každá z buněk nese soubor informací o tom, jestli je do ní může robot vjet (myšleno středem S_{rob}). Některé z těchto informací můžeme nastavovat ručně, některé se dynamicky mění podle informací ze senzorů během hry.



Obrázek 3.1: Reprezentace mapy v prostředí Robomon

3.1.2 Informace v buňkách

Každá buňka je reprezentována strukturou `MapCell`

```
typedef struct map_cell {
        map_cell_flag_t flags;
        map_cell_detobst_t detected_obstacle;
                /**< 0 = no obstacle, 255 = new obstacle */
} MapCell;
```

a celé hřiště tedy

```
struct map {
        MapCell cells[MAP_HEIGHT][MAP_WIDTH];
};
```

Jednotlivé bity v proměnné `flags` ukazují status buňky. V prostředí Robomon vidíme jednotlivé statusy jako odlišné barvy buněk na hřišti. V současné době je k dispozici tato množina příznaků:

Jejich definice je pak:

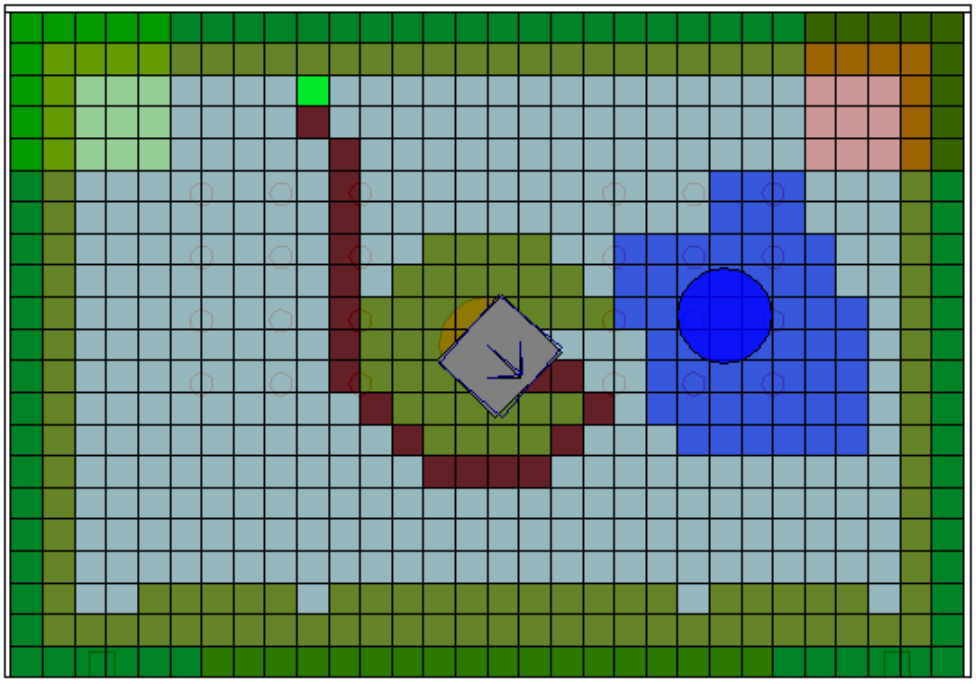
Tabulka 3.1: Tabulka příznaků pro buňky v mapě a jejich barevná reprezentace v prostředí Robomon

Název příznaku	Význam příznaku	Barva v prostředí Robomon
MAP_FLAG_WALL	Zeď	tmavě žlutá
MAP_FLAG_PATH	Trajektorie	hnědá
MAP_FLAG_START	Začátek trajektorie	červená
MAP_FLAG_GOAL	Konec trajektorie	zelená
MAP_FLAG_DET_OBST	Překážka ve hře	azurová
MAP_FLAG_SIMULATED_WALL	Simulovaná zeď v Robomonu	žlutá
MAP_FLAG_IGNORE_OBST	Ignorování detekované překážky	tmavě zelená
MAP_FLAG_PLAN_MARGIN	Bezpečnostní vzdálenost kolem překážky	–
MAP_FLAG_INVALIDATE_WALL	Zrušení existující zdi	průhledná

```
#define MAP_FLAG_WALL              1
#define MAP_FLAG_PATH              2
#define MAP_FLAG_START             4
#define MAP_FLAG_GOAL              8
#define MAP_FLAG_DET_OBST          16
#define MAP_FLAG_SIMULATED_WALL    32
#define MAP_FLAG_IGNORE_OBST       64
#define MAP_FLAG_PLAN_MARGIN       128
#define MAP_FLAG_INVALIDATE_WALL   256
```

Proměnná `detected_obstacle` obsahuje číslo od 0 do 255, které signalizuje, před jakou dobou byla na buňce detekována překážka. Je nutné si překážku pamatovat nějakou dobu, než ji prohlásíme za zmizelou, abychom odfiltrovali rychlé změny dat ze senzorů a zajistili jakousi setrvačnost, která souvisí s fyzikální podstatou překážky. V našem případě se jedná o robota soupeře. Po detekci je nastavena hodnota 255 a časem klesá až na 0, poté je z mapy překážka odstraněna.

Flag `MAP_FLAG_PLAN_MARGIN` jsem již přidal. Z praktických pokusů se ukázalo, že pokud se naplánuje cesta kolem robota v jeho nejtěsnější blízkosti, mohlo by se stát, že do mí v nejbližší době soupeř vjede, a to buď opravdu, nebo tuto situaci může způsobit zašumělá informace ze senzorů. Dochází tak k oscilacím a zbytečnému přeplánovávání



Obrázek 3.2: Příklad konkrétní situace v mapě

trajektorie. Tento příznak vytvoří jakousi ochrannou zónu v okolí robota, která se bere v úvahu pouze při plánování trajektorie, ale nikoli při detekci kolize se soupeřem.

Flag `MAP_FLAG_INVALIDATE_WALL` byl přidán tak, abychom se mohli přiblížit a dotknout zdi, pokud to během hry potřebujeme.

3.1.3 Vkládání objektů do mapy

Abychom samozřejmě nemuseli vždy plnit flagy na hřišti po jednotlivých buňkách, byly přidány funkce, které zjednoduší práci a vedou k intuitivnějšímu ovládání pozice detekovaných objektů na hřišti. Předpokládejme, že objekty, které chceme do mapy zaznačit, jsou obdélníkového nebo kruhového půdorysu, případně je lze z těchto útvarů poskládat.

Máme tedy k dispozici dvě funkce, kterými můžeme na určitou plochu zapnout či vypnout určitý príznak. Tato plocha se musí vhodně přepočítat na množinu buněk.

Nastavení příznaků na obdélníkovém objektu zařídí funkce

```
int ShmapSetRectangleFlag(double x1, double y1, double x2, double y2,
        map_cell_flag_t set_flags, map_cell_flag_t clear_flags);
```

a nastavení příznaků na kruhovém objektu

```
int ShmapSetCircleFlag(double xs, double ys, double r,
        map_cell_flag_t set_flags, map_cell_flag_t clear_flags);
```

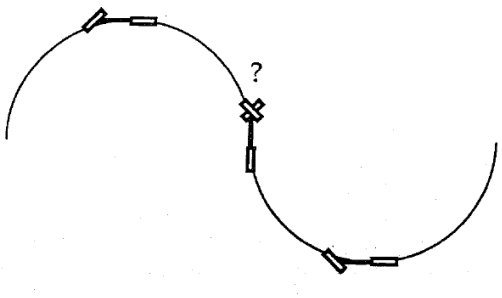
3.2 Hledání cesty

Máme-li všechny potřebné informace v mapě, můžeme přistoupit k hledání cesty. Algoritmus A-star, po zadání dvou bodů (start, cíl), nalézne nejkratší cestu. K tomu nám pomůže opět mřížkou rozdělená mapa (obr. 3.1). Nalezená cesta uspořádaný seznam buněk, které převedeme na body, středy těchto buněk. Nyní máme cestu poskládanou z mnoha bodů, které jsou od sebe navzájem vzdáleny maximálně délku úhlopříčky buňky. S tím bychom mohli být relativně spokojeni, ale pro zjednodušení ještě některé redundandní body odebereme. A to tak, že pokud zjistíme, že více bodů za sebou tvoří úsečku, nebo je lze proložit úsečkou, od níž jsou tyto body odchýleny jen do určité malé vzdálenosti, odebereme všechny vnitřní body úsečky a ponecháme pouze body krajní. Tím se sníží počet pamatovaných bodů, aniž by se změnila trajektorie, kterou bychom vytvořili z přímočarých segmentů. Pokud se ale budeme dále snažit o vytvoření hladké trajektorie, bude se hodit menší počet bodů průjezdu.

3.3 Plánování trajektorie

Dosud používaná metoda tvoření trajektorií byla skládání přímočarých segmentů a kružnicových oblouků, kterými byly prokládány ostré rohy, aby robot nemusel zastavovat v zatáčkách.

Co se týče stanovení průběhu rychlostí podél trajektorie, algoritmus popíši v kapitole 5, jelikož se příliš neměnil.



Obrázek 3.3: Použití kružnicových oblouků na proložení zatáček

3.4 Vyhýbání se překážkám

Pokud robot objeví v libovolném bodě trajektorie překážku, je potřeba se jí vyhnout a trajektorii přeplánovat. Řešení bylo dosud takové, že se v tomto případě robot zastavil, a poté začal počítat novou trajektorii. Tento přístup je dosti benevolentní vzhledem k časovým omezením a bude tudíž v dalších kapitolách také předmětem úprav.

Kapitola 4

Spliny

Tato kapitola se zabývá různými druhy křivek, které se dají použít jako trajektorie pro robota. Spline je spojitá křivka, má tedy složitější matematický popis než například přímka nebo kružnice. Na druhou stranu existuje velké množství tvarů, kterých můžeme dosáhnout. O spojité křivky mají zájem i jiné podobné robotické aplikace (Trdlička, J., 2005).

Nejprve bych rád zmínil, že existuje více přístupů k řízení pohybu robota. Jednou možností je kvalitní regulátor s velmi omezeným akčním zásahem, který bude pouze sledovat cílový bod jako referenci. Tuto možnost zavrhneme, protože při použití této metody je složité predikovat polohu robota v budoucnosti. Další možností je současné plánování trajektorie a průběhu rychlosti viz (Choset et al., 2005). Asi je zřejmé, že nejjednodušší bude postupovat směrem děleného plánování trajektorie a rozdělování rychlostí, jelikož tato filozofie je již v robotu implementována, bude tedy nejjednodušší ji integrovat s ostatními součástmi.

4.1 Symbolika

V této kapitole budeme používat množství geometrických symbolů, které bych nejprve rád ujasnil.

- PNT je množina bodů průjezdu, které jsou vstupem do algoritmu plánování trajektorie. Skládá se ze z x-ové a y-ové složky PNT_x a PNT_y.
- největší množinou je množina bodů trajektorie $TRAJ$

- trajektorie $TRAJ$ se skládá ze segmentů SEG, přičemž platí, že SEG_i je segment, který následuje po segmentu SEG_{i-1}. Skládá se ze z x-ové a y-ové složky SEG_x a SEG_y. Po své délce je parametrizován parametrem $t \in \langle 0; 1 \rangle$
- symbol SEG' znamená derivaci x-ové a y-ové složky SEG_x a SEG_y podle parametru
- notace $|PNT_i\ PNT_{i+1}|$ znamená vzdálenost dvou bodů

4.2 Motivace

Důvod, proč je vůbec potřeba nasadit spojité křivky místo kružnicových oblouků, je ten, že musíme splnit určitá kritéria, jak mohou segmenty na sebe navazovat. Musí tedy pro všechny segmenty SEG_i, kde $i \in 0, 1, 2, \cdots, n$ platit:

- $SEG_i(1) = SEG_{i+1}(0)$
- $SEG'_i(1) = SEG'_{i+1}(0)$
- $SEG''_i(1) = SEG''_{i+1}(0)$

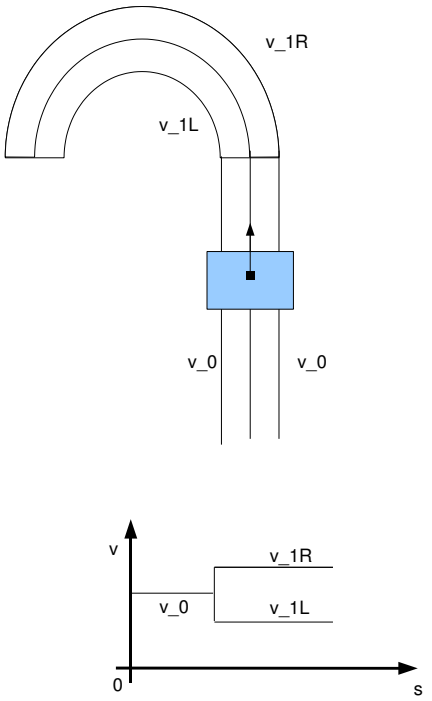
Nyní je třeba, abych zmínil pojem křivosti křivky (*Wikipedia* [online], 2009), jenž nás bude často provázet touto kapitolou. Křivost křivky v bodě je převrácená hodnota poloměru křivosti v určitém bodě. Pro parametricky zadané křivky lze vyjádřit vztahem

$$\kappa = \frac{1}{R} = \frac{|x'(t)y''(t) - y'(t)x''(t)|}{(x'^2(t) + y'^2(t))^{3/2}}, \tag{4.1}$$

kde R je poloměr křivosti křivky, $x(t)$ průběh x-ové a $y(t)$ y-ové souřadnice v závislosti na parametru t.

Při použití kružnic sice segmenty trajektorie splňují první dva body, navazují tedy v tečnách (tj. první derivace), ale ne už v křivosti (závisí na první i druhé derivaci). To znamená, že pokud se pohybujeme po takové trajektorii, přímočarý úsek má nulovou křivost a kružnicový oblouk má křivost konstantní nenulovou (obr. 3.3). Pokud se v tomto bodě robot octne, musí jedno z jeho koleček nekonečně zrychlit a druhé nekonečné zpomalit.

Pro představu uvedu příklad (obr. 4.1), kdy robot přejíždí z přímočarého segmentu SEG_0 na kružnicový oblouk SEG_1 o poloměru r. Střed robota S se pohybuje konstantní dopřednou rychlostí v. Během segmentu SEG_0 se obě kola pohybují dopřednou rychlostí

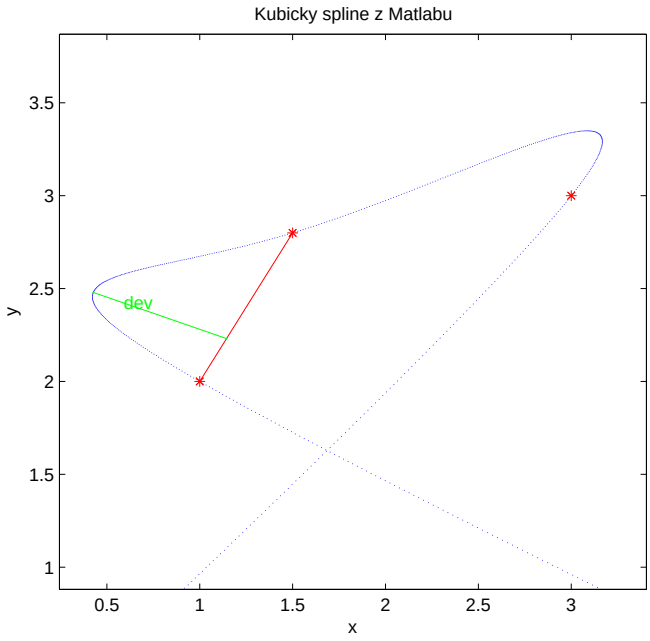


Obrázek 4.1: Nespojitá rychlost koleček na kružnicovém oblouku

$v_{0L} = v_{0P} = v$. Během segmentu SEG_1 se levé kolo pohybuje rychlostí $v_{1L} = v(r - r_{rot})$ a pravé kolo $v_{1R} = v(r + r_{rot})$. V bodě $SEG_0(1) = SEG_1(0)$ musí dojít tedy k nekonečnému zrychlení/zpomalení kolečka robota.

To může způsobit nepěkné cukání robota, a pokud se snažíme určovat svou polohu pomocí odometrie (měření otáček kol pomocí inkrementálních snímačů), kolečka mohou podkluzovat a lokalizace bude dosti nepřesná.

Jedno z dalších omezení, na která si musíme dát pozor je, že pokud chceme iterpolovat body trajektorie nějakou automaticky vygenerovanou křivkou, odchyluje se určitým způsobem od trajektorie přímočaré. Tuto vzdálenost musíme umět kontrolovat, jinak bychom nevěděli, jestli robot nekoliduje s nějakou překážkou. Na obrázku je ukázka možného průjezdu zadanými body a maximální odchylka od přímočaré trajektorie mezi dvěma body.



Obrázek 4.2: Odchylka od přímočaré trajektorie

4.3 Druhy spojitých křivek

Nyní představím možnosti, kterými jsem se zabýval během rešerše. V různých praktických oborech se používají různé popisy 2D křivek, které pro nás mají určité výhody a nevýhody.

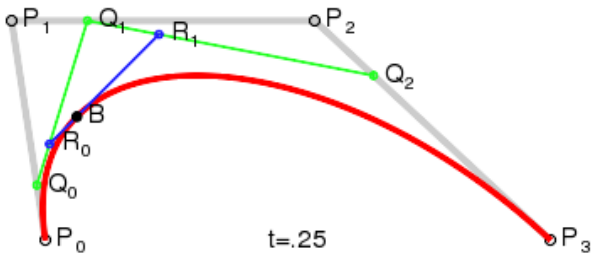
4.3.1 Bézierovy křivky třetího řádu

Bézierovy křivky jsou nejčastěji využívané hladné křivky v počítačové grafice. Jsou vyjádřeny parametricky pomocí polynomů třetího řádu.

$$P_x(t) = A_x t^3 + B_x t^2 + C_x t + D_x \tag{4.2}$$
$$P_y(t) = A_y t^3 + B_y t^2 + C_y t + D_y$$

kde $t \in \langle 0; 1 \rangle$.

Pokud si uvědomíme praktický význam některých parametrů, můžeme jednoduše křivku přesouvat a tvarovat. Tím, že je polynom pouze třetího řádu, nelze s křivkou příliš divokce tvarovat. Dosáhnout můžeme pouze tvaru bez inflexe nebo s právě jednou



Obrázek 4.3: Bezierova křivka třetího řádu

inflexí, což by se nám celkem hodilo. Na obrázku obr. 4.4 můžeme vidět několik možností, jak lze tuto křivku natvarovat.

Existují jednoduché, veřejně známé algoritmy viz např. (Karolík, A., 2006), kde na vstup zadáme dva body a tečné vektory v těchto bodech a dostaneme křivku. Pokud potřebujeme vygenerovat křivku delší, zadanou více body, lze využít algoritmu, kterým se zabýval již kolega přede mnou (Karolík, A., 2006). Tento algoritmus spojuje jednotlivé polynomiální křivky tak, že zachovává spojitost, spojitost směru tečny i spojitost křivosti.

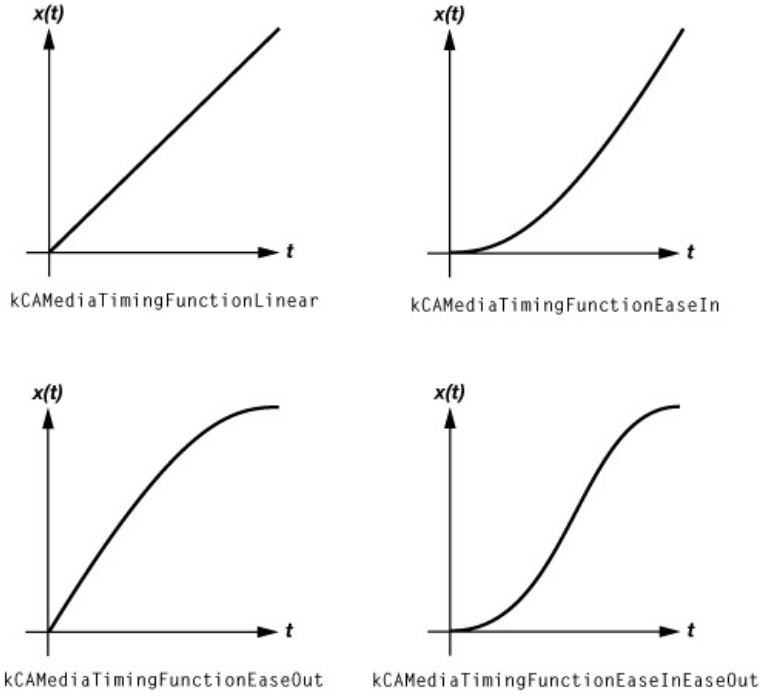
Problém je v tom, že při použití tohoto algoritmu nevíme dopředu, jak daleko se bude trajektorie odchylovat od přímočaré. Samozřejmě, že z parametrů křivky průběh odchylky vypočítáme, ale nelze už analyticky vyjádřit závislost těchto parametrů na maximální odchylce

$$dev_{max} = \max_{t=0}^{1}(|SEG(t)\ SEG_{LINE}|) \tag{4.3}$$

Vlastním ideologickým problémem je snaha body trajektorie interpolovat (obr. 4.5), to vede vždy k nějaké odchylce, kterou nebudeme schopni zadat jako vstup do algoritmu. Jiný přístup je pak tyto body aproximovat (obr. 4.6) podobným způsobem jako v případě kružnicových oblouků. Na první pohled by se zdálo, že se nijak zmíněného problému nezbavíme, ale realita je jiná. Výhoda je taková, že sice máme nějaké odchylky od přímočaré trajektorie, ale tyto odchylky jsou vždy pouze uvnitř konvexních úhlů. Podle následující věty se tedy nemusíme zabývat kolizemi s obvodovými zdmi.

Pokud jsou všechny body průjezdu PNT uvnitř hrací plochy, která je vymezena konvexními úhly (v našem případě pravými), pak všechny body vnitřní aproximační křivky leží uvnitř tohoto hřiště. Pro vnitřní aproximační křivku platí, že všechny její body leží uvnitř úhlu γ, který je tvořen třemi následujícími body průjezdu.

Z toho plyne, že je potřeba již pouze ošetřit situaci, kdy máme překážku (herní prvek) uvnitř tohoto úhlu γ. V kapitole 3.2 jsem uvedl způsob generování bodů průjezdu. Pokud zaručíme, že tyto body vždy dodrží určitou minimální vzdálenost od překážky, můžeme



Obrázek 4.4: Možné vzhledy bezierovy křivky

počítat s touto vzdáleností jako konstantním požadavkem pro každé generování aproximační křivky, čímž si výrazně zjednodušíme práci.

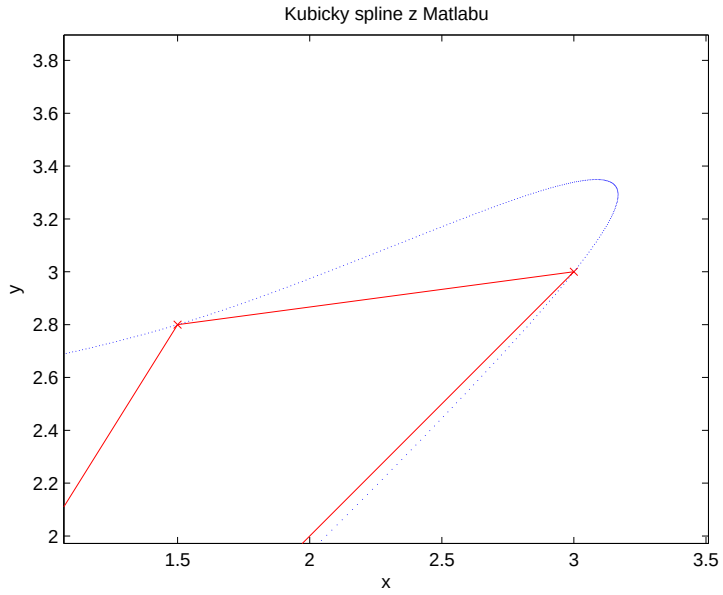
4.3.2 Aproximační polynomiální křivka

Další bádání vede k otázce, jak natvarovat polynoiální křivku, aby aproximovala body průjezdu trajektorie. Řekněme, že vzhledem ke snaze rychlého průjezdu nebude vůbec vadit, ponecháme-li si možnost tvořit přímočaré úseky a polynom použijeme jen pro změny směru.

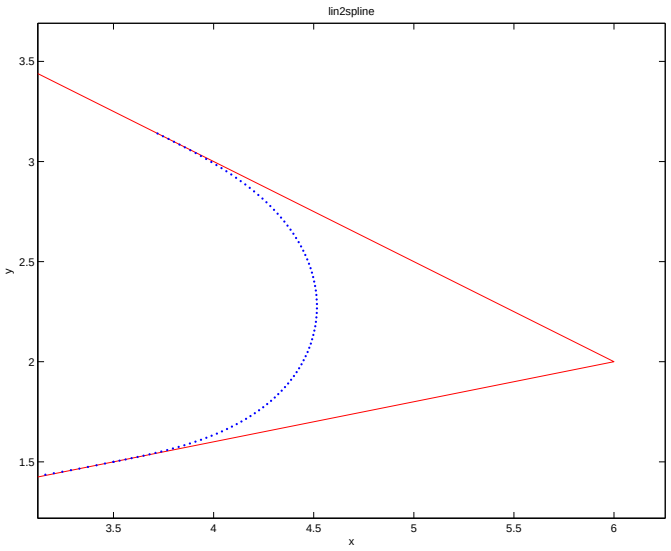
Pokud budeme chtít navazovat přímočarý segment na spline, z požadavku na spojitou rychlost koleček je jasné, že na začátku i na konci křivky je potřeba zajistit nulovou křivost. Důsledkem bude také to, že není třeba rozlišovat případ, kdy za splinem navazuje přímočarý segment nebo další spline.

Nyní si shrneme matematické podmínky pro vytvoření takové polynomiální křivky, jakožto segmentu trajektorie. Je třeba zajistit možnost definování parametrů uvedených v tabulce (tabulka 4.1).

To znamená celkem 10 stupňů volnosti, jimiž oplývá 2D polynomiální křivka minimálně



Obrázek 4.5: Interpolace



Obrázek 4.6: Aproximace

Tabulka 4.1: Obecné podmínky pro navazování segmentů trajektorie

Podmínka	Počet ubraných stupňů volnosti
poloha počátečního bodu $SEG(0)$	2 stupně volnosti
poloha koncového bodu $SEG(1)$	2 stupně volnosti
směr tečny v bodě $SEG(0)$	1 stupeň volnosti
směr tečny v bodě $SEG(1)$	1 stupeň volnosti
křivost $\kappa = 0$ v bodě $SEG(0)$	2 stupně volnosti
křivost $\kappa = 0$ v bodě $SEG(1)$	2 stupně volnosti

řádu 4. Jelikož se nám bude hodit ještě křivku dále tvarovat pro lepší kinematické vlastnosti, použijme polynom řádu 5.

4.3.3 Klotoida

Křivka, která je hojně využívána v dopravním stavebnictví, se nazývá klotoida (Seng, R. a Severdia, M., n.d.) (obr. 4.9). Využívá se například při navrhování dálničních křižovatek, ale i v jiných praktických fyzikálních aplikacích (obr. 4.7 a obr. 4.8).

Základní vlastností klotoidy, pro kterou je využívána, je, že s rostoucí vzdáleností se lineárně mění její křivost (Seng, R. a Severdia, M., n.d.).

$$\kappa(s) = ks + \kappa_i; \qquad k \in R\backslash\{0\} \tag{4.4}$$

Problémem ovšem je, že tuto křivku nelze analyticky popsat jako závislost polohy na parametru. Její vyjádření pomocí tzv. Fresnelových integrálů je:

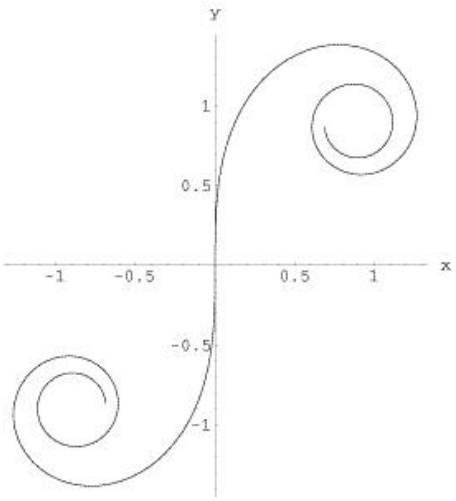
$$f(t) = \int_0^t \sin(\frac{x^2}{2})\mathrm{d}x \tag{4.5}$$

$$g(t) = \int_0^t \cos(\frac{x^2}{2})\mathrm{d}x \tag{4.6}$$

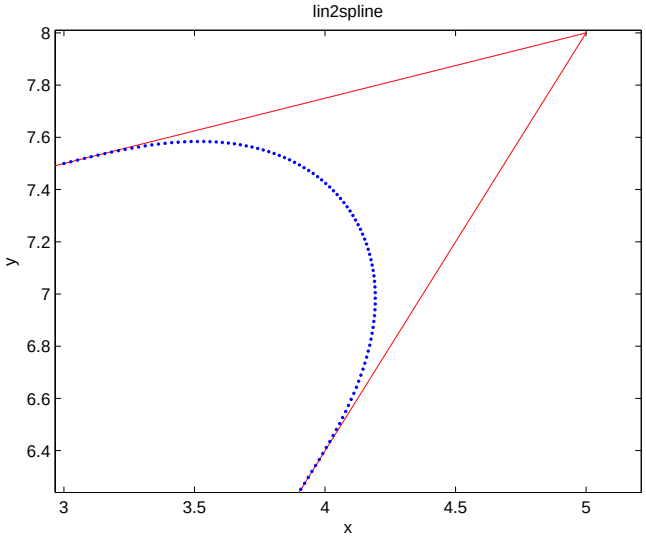
Z výše zmíněné závislosti (4.4) plyne, že můžeme jednoduše klotoidu napojovat na přímočaré úseky. Pokud bychom složili zatáčku ze dvou klotoid, dosáhneme krásného tvaru zatáčky (viz obr. 4.10), který splňuje všechna omezení uvedená v tabulce (tabulka 4.1).

Obrázek 4.7: Dálniční křižovatka složená z úseků klotoid

Obrázek 4.8: Roller Coaster



Obrázek 4.9: Klotoida



Obrázek 4.10: Zatáčka složená ze dvou částí klotoidy

4.4 Výběr konkrétní křivky

Po shlédnutí všech možností jsem se rozhodl, že nejlepší varianta bude následující. Polynom pátého řádu nám zajistí analytické vyjádření křivky, která má dostatečný počet stupňů volnosti k tomu, abychom ji dokázali natvarovat tak, abychom splnili všechny podmínky. Ale protože jsme nevyužili všech stupňů volnosti, je třeba ještě doplnit další požadavky. Vzhledem k tomu, že se budeme dále snažit optimalizovat dobu průjezdu trajektorií, nebylo by špatné, kdyby se tvar našeho polynomu alespoň trochu blížil tvaru klotoidy. Popis 2D polynomu pátého řádu:

$$\begin{aligned} P_x(t) &= A_x t^5 + B_x t^4 + C_x t^3 + D_x t^2 + E_x t + F_x \\ P_y(t) &= A_y t^5 + B_y t^4 + C_y t^3 + D_y t^2 + E_y t + F_y \end{aligned} \tag{4.7}$$

kde $t \in \langle 0; 1 \rangle$.

Kapitola 5

Generování trajektorie

5.1 Popis křivky

Vycházejme prozatím ze situace, kdy máme zadán bod průjezdu Q, který se budeme snažit aproximovat, dále směr příjezdu a směr odjezdu z tohoto bodu. Těmito dvěma směry vymezíme úhel $\gamma \in \langle 0°; 180° \rangle$, který se budeme snažit proložit zatáčkou. Vycházejme ze situace, kdy známe vzdálenost d od bodu Q, kde bude zatáčka začínat. Jak tuto vzdálenost vybrat, se dozvíme později. V zatáčce dále platí vztahy

$$SEG(t) = P(t) \tag{5.1}$$

$$d = |X0\ Q| = |Q\ X1|, \qquad P(0) = X0, \qquad P(1) = X1 \tag{5.2}$$

kde $P(t)$ je polynomiální křivka dle (4.7) s parametrem $t \in \langle 0; 1 \rangle$. Z výrazu (5.2) plyne, že začátek polynomu bude stejně daleko od vrcholu zatáčky jako jeho konec.

Nyní sestavme rovnice (5.3), které chceme, aby platily pro náš spline (viz obr. 5.1).

$$
\begin{aligned}
P_x(0) &= X0_x \\
P_y(0) &= X0_y \\
P_x(1) &= X1_x \\
P_y(1) &= X1_y \\
P'_x(0) &= X0'_x = m(Q_x - X0_x) \\
P'_y(0) &= X0'_y = m(Q_y - X0_y) \\
P'_x(1) &= X1'_x = m(X1_x - Q_x) \\
P'_y(1) &= X0'_y = m(X1_y - Q_y) \\
\kappa(0) &= \frac{|P'_x(0)P''_y(0) - P'_y(0)P''_x(0)|}{(P'^2_x(0) + P'^2_y(0))^{3/2}} = 0 \\
\kappa(1) &= \frac{|P'_x(1)P''_y(1) - P'_y(1)P''_x(1)|}{(P'^2_x(1) + P'^2_y(1))^{3/2}} = 0
\end{aligned}
\tag{5.3}
$$

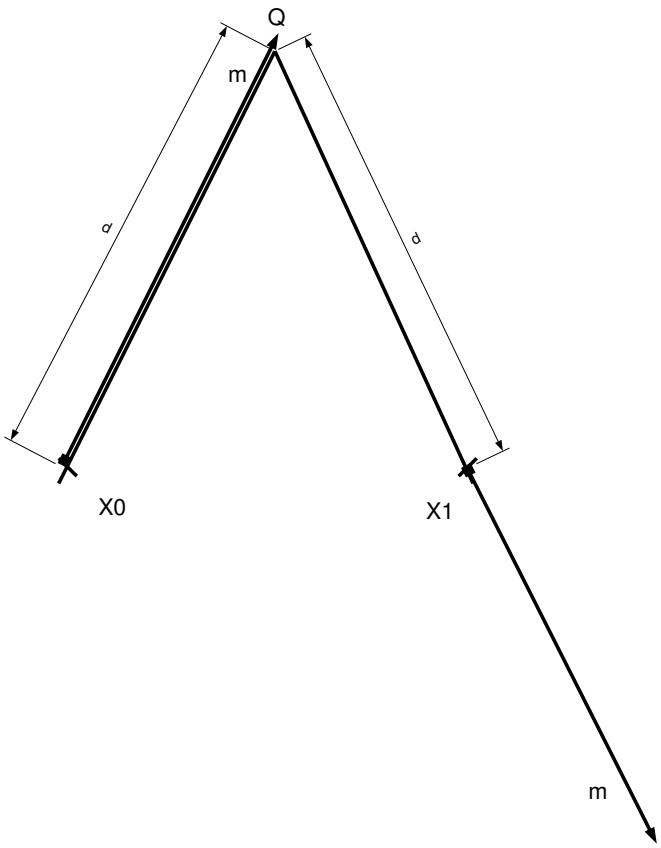
přičemž parametr m určuje délku obou tečných vektorů, ale jeho hodnotu zatím neznáme. Pro oba směry, směr příjezdu i směr odjezdu, je shodný kvůli zachování symetrie křivky. Totéž platí i pro hodnotu d.

Pro úplnost je třeba uvést hodnoty derivací polynomu (4.7)

$$
\begin{aligned}
P'_x(t) &= 5A_xt^4 + 4B_xt^3 + 3C_xt^2 + 2D_xt + E_x \\
P'_y(t) &= 5A_yt^4 + 4B_yt^3 + 3C_yt^2 + 2D_yt + E_y \\
P''_x(t) &= 20A_xt^3 + 12B_xt^2 + 6C_xt + 2D_x \\
P''_y(t) &= 20A_yt^3 + 12B_yt^2 + 6C_yt + 2D_y
\end{aligned}
\tag{5.4}
$$

Po úpravě soustavy (5.3) dostaneme vyjádření jednotlivých parametrů splinu:

$$
\begin{aligned}
A_x &= -3X1'_x - 3X0'_x - 6X0_x + 6X1_x \\
B_x &= 7X1'_x + 8X0'_x + 15X0_x - 15X1_x \\
C_x &= -4X1'_x - 6X0'_x - 10X0_x + 10X1_x \\
D_x &= 0 \\
E_x &= X0'_x \\
F_x &= X0_x
\end{aligned}
\tag{5.5}
$$



Obrázek 5.1: Význam symbolů v zatáčce

$$
\begin{aligned}
A_y &= -3X1'_y - 3X0'_y - 6X0_y + 6X1_y \\
B_y &= 7X1'_y + 8X0'_y + 15X0_y - 15X1_y \\
C_y &= -4X1'_y - 6X0'_y - 10X0_y + 10X1_y \\
D_y &= 0 \\
E_y &= X0'_y \\
F_y &= X0_y
\end{aligned} \tag{5.6}
$$

Víme tedy nyní, jak vypočítat jednotlivé parametry splinu, za předpokladu, že známe polohu bodů $X0$ a $X1$ a parametr m.

5.2 Tvarování křivky jako klotoidy

Parametr m se pokusíme nastavit tak, aby se křivka co nejvíce podobala klotoidě. To se nám podaří ovšem pouze numerickými optimalizačními metodami. Vzhledem k tomu, že tento výpočet je časově náročný, nemůžeme jej provádět za jízdy, ale je třeba jej předpočítat off-line. Využijme vlastnosti klotoidy, že má lineárně měnící se křivost.

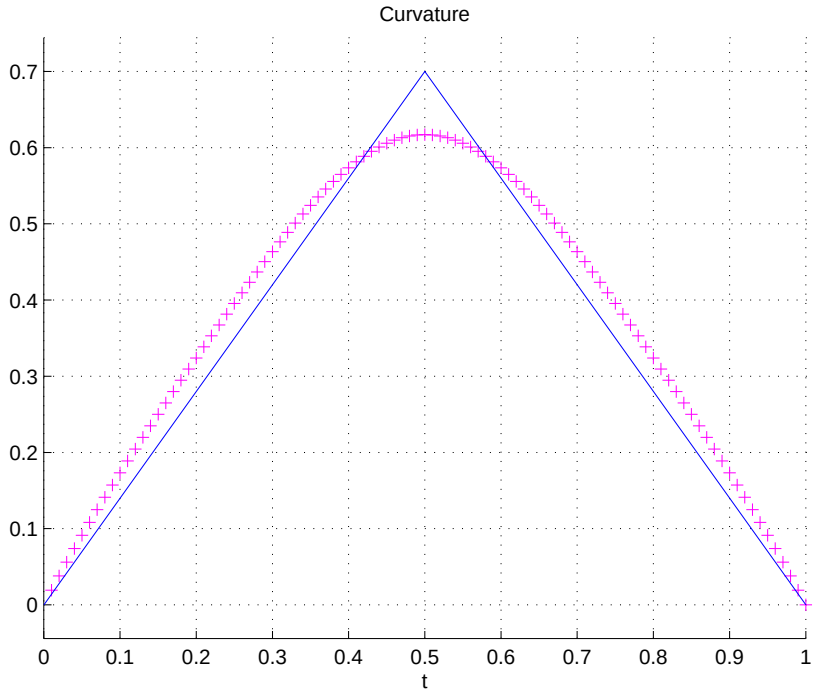
Pokud budeme tvarovat polynomiální křivku, vypočteme si průběh její křivosti podél parametru. Tento průběh srovnáme s průběhem klotoidy a jejich rozdíl se budeme snažit minimalizovat. Ideální křivka tedy splňuje podmínku:

$$
m = \min(m)_m \left[\int_0^1 |\kappa(t,m) - \kappa_{clothoid}(t,m)| \partial t \right] \tag{5.7}
$$

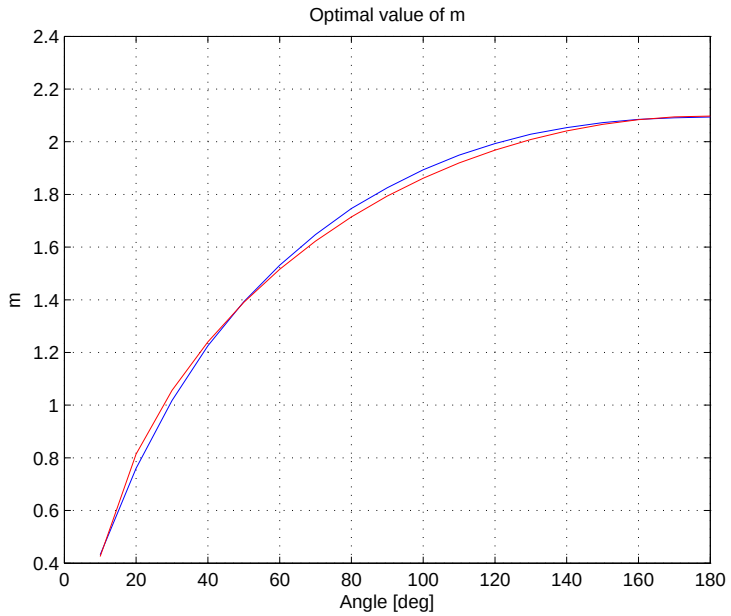
V našem případě pro numerický výpočet a numerickou integraci bude vhodné použít diskrétní ekvivalent předešlého vztahu.

$$
m = \min(m)_m \left[\sum_{t=1}^{1} \left((s(t) - s(t-1)) \left| \kappa(t,m) - \kappa(t,m)_{clothoid} \right| \right) \right] \tag{5.8}
$$

přičemž $s(t)$ je vzdálenost od začátku $X0$ splinu v závislosti na parametru t. Konkrétní výrazy nebudu uvádět z důvodu jejich přílišného rozsahu, především co se týče vzorců s křivostí (výraz (4.1) je třeba derivovat). K optimalizaci nám poslouží v Matlabu funkce `fminsearch`. Výsledkem bude křivka, ktrá se nejvíce podobá klotoidě, ovšem je třeba si uvědomit, že průběh křivosti polynomiální křivky je stále spojitá funkce, čili nemůžeme dosáhnout průběhu ostrého jako u klotoidy. Nicméně, z praktického hlediska nám to



Obrázek 5.2: Průběh křivosti klotoidy a námi generované křivky



Obrázek 5.3: Optimalizovaný parametr m v závislosti na úhlu

vůbec v ničem nepřekáží. Na obrázku (obr. 5.2) je znázorněn jeden z finálních stavů optimalizace.

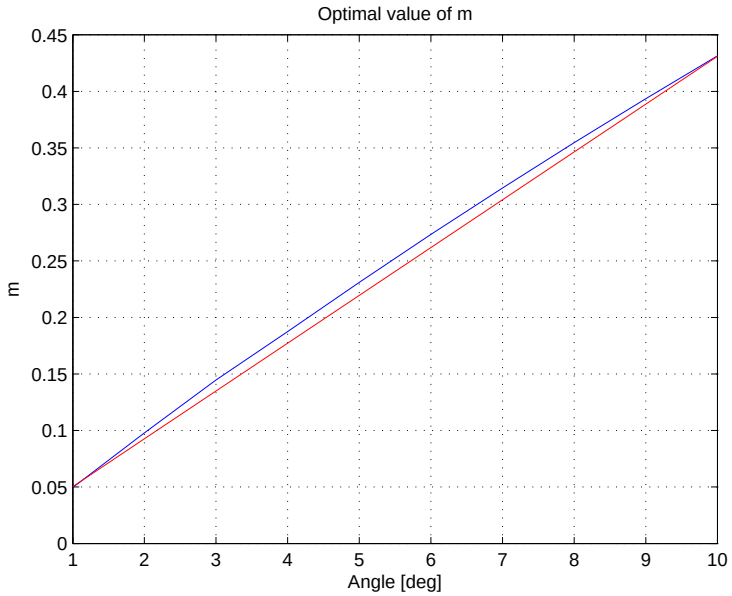
Výsledkem optimalizace tedy je uspořádaná dvojice $[\gamma; m]$, kde γ je vnitřní úhel zatáčky a m je hledaný parametr polynomiální křivky. Těchto uspořádaných dvojic bychom měli najít v ideálním případě nekonečně mnoho pro $\gamma \in (0°; 180°)$.

Zjednodušme si situaci tím, že budeme hledat parametr m pouze pro velikosti úhlů, které jsou násobky 10°.

Tyto získané body proložme vhodně zvolenou křivkou. Po testování různých polynomů se nejvhodnější zdála elipsa, pro kterou je třeba natvarovat dvěma parametry A a B. Tyto se podařilo ručně nastavit na hodnoty $A = 6860$ a $B = 4,4$. Proložení bodů elipsou (5.9) je znázorněno na obrázku (obr. 5.3).

$$m(\gamma) = \sqrt{B - \frac{(\gamma - 180)^2}{A}} \tag{5.9}$$

Z pozdějších poznatků vyplynulo, že se tato křivka nedá použít pro malé úhly zhruba menší než 10°, protože v této oblasti nedostačuje přesnost proložení. Proto je třeba se na tuto oblast malých úhlů zaměřit zvlášť a podrobněji. Úplně stejnou optimalizační metodou dostaneme body v intervalu $\gamma \in (0°; 10°\rangle$ a proložíme křivkou (5.10),



Obrázek 5.4: Optimalizovaný parametr m v závislosti na malých úhlech

znázorněnou na obrázku (obr. 5.4).

$$m(\gamma) = C \cdot \gamma + D \qquad C = 0,0423 \qquad D = 0,008 \tag{5.10}$$

Výsledek celé optimalizace už může používat robot během plánování pohybu za jízdy jen se znalostí těchto parametrů A, B, C, D.

Celá předchozí část vycházela z předpokladu, že vliv vzdálenosti d neuvažujeme, respektive d je známým vstupem, z čehož vyplývá i poloha bodů $X0$ a $X1$. Jak se tedy vypořádat se situací, kdy potřebujeme zatáčku jinak velkou (myšleno $d \neq 1$, ale úhel γ je stejný)? Vypočítaný polynom lze jednoduše transformovat přenásobením všech jeho parametrů konstantou, která udává zvětšení křivky ve směru osy x i y. Přitom se zachovají požadované vlastnosti. Chceme-li tedy posunout body $X0$ a $X1$ do jiné vzdálenosti, definujeme poměrné zvětšení $\epsilon = \frac{d_{new}}{d}$. Pro jednotlivé parametry polynomu bude tedy platit:

$$A_{xnew} = \epsilon \cdot A_x \tag{5.11}$$

$$\ldots$$

$$F_{ynew} = \epsilon \cdot F_y$$

Nyní máme k dispozici postup, pomocí něhož umíme vytvořit libovolně velkou zatáčku

v libovolném úhlu $\gamma \in (0°; 180°)$, která navazuje na přímočaré úseky.

Pokud bychom měli splnit omezení, že se trajektorie nesmí odchylovat od vrcholu zatáčky Q více než určitou zadanou hodnotu e_{max}, je potřeba spline transformovat pomocí konstanty ϵ (5.12).

$$\epsilon = \frac{e}{e_{max}} \tag{5.12}$$

5.3 Napojení jednotlivých segmentů

S již představenými nástroji se pokusíme kompletně sestavit trajektorii ze zadaných bodů. Nejprve je třeba ošetřit některé záležitosti:

- Dva následující body půjezdu nesmí být stejné
- Mezi třemi následujícími body průjezdu nesmí být úhel $\gamma = 0$

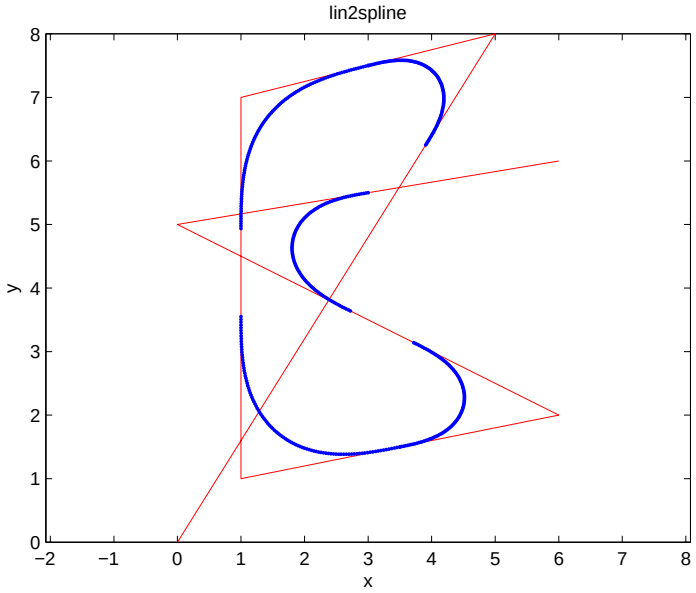
V těchto případech bychom nebyli schopni trajektorii vygenerovat, protože v prvním případě nelze vypočítat úhel mezi úsečkami, z nichž má jedna nulovou velikost. S tím si poradíme jednoduše tak, že všechny duplicitní body smažeme. Ve druhém případě je více možných zásahů, já jsem vybral možnost přidat bod, který je nepatrně vzdálen od vrcholu zatáčky Q.

Algoritmus, kterým vybudujeme trajektorii, skládající se z přímočarých a splinových segmentů, funguje tak, že nejdříve pospojujeme sekvenci bodů průjezdu úsečkami, a následně vyhledáme body, kde budou začínat a končit spliny. Tyto body najdeme tak, že vezmeme dvě přilehlé úsečky bodu Q_i, a v polovině kratší z nich bude ležet jeden z bodů $X0_i$, $X1_i$. Na druhé úsečce, ve stejné vzdálenosti d, pak druhý z těchto bodů. Ukázka takové trajektorie je na obrázku (obr. 5.5).

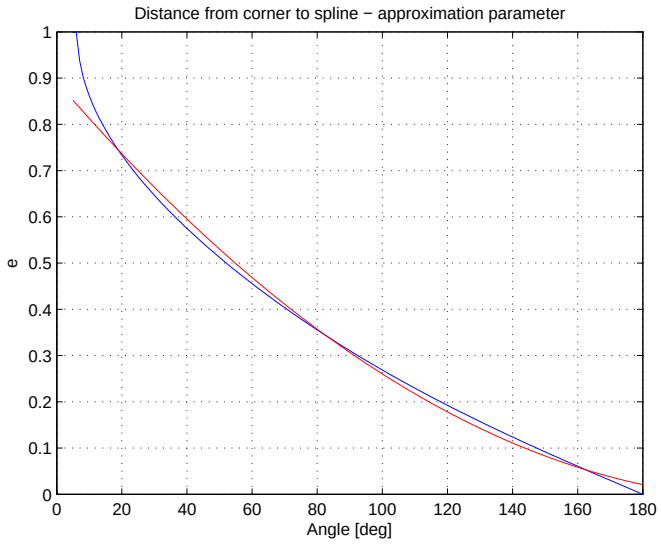
Nyní je potřeba vzít v úvahu omezení e_{max}. Bod Q_i je od středu splinu $P(0, 5)$ ve vzdálenosti e_i. Pokud platí, že $e_i > e_{max}$, je třeba spline zmenšit a zajistit rovnost $e_i = e_{max}$. Vzhledem k tomu, že vzdálenost e je přímo úměrná vzdálenosti d, posuneme body $X0$ a $X1$ tak, že platí:

$$\epsilon_i = \frac{e_{max}}{e_i} \tag{5.13}$$

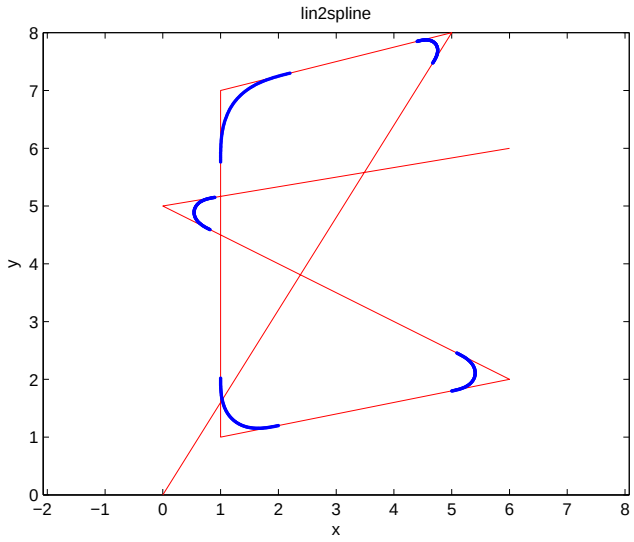
Jakým způsobem ale vypočítáme e? Nejjednudušší možnost je spline vygenerovat a potom se podívat na polohu bodu $P(0, 5)$. To je ale zbytečně časově náročné. Lepší by bylo znát tuto hodnotu předem. Vytvoříme si tedy off-line tabulku závislosti e na vnitřním úhlu γ. Tuto tabulku opět proložíme nějakou analyticky vyjádřitelnou křivkou:



Obrázek 5.5: Trajektorie se spliny, nebere v úvahu omezení e_{max}



Obrázek 5.6: Závislost vzdálenosti e na úhlu γ při konstantním d



Obrázek 5.7: Výsledná trajektorie

$$e(\gamma)|_{d=1} = 1,849 \cdot 10^{-5}\gamma^2 - 8,169 \cdot 10^{-3}\gamma + 0.892 \qquad [m, m, °] \tag{5.14}$$

Nyní, když už víme, jak budou vypadat všechny křivky, je potřeba zkrátit přímočaré úseky. Čili každá úsečka bude vymezena krajními body $X1_{i-1}$ a $X0_i$, pokud nastane případ, že $X1_{i-1} = X0_i$, bude tento přímočarý úsek odstraněn. Nyní jsme u konce s veškerou geometrií trajektorie a máme připravenou sekvenci navazujících přímočarých a splinových úseků.

Kapitola 6

Kinematika robota

Máme-li hotovou trajektorii, můžeme se zabývat tím, jak rozdělíme rychlosti, kterými tuto trajektorii budeme projíždět. Aby se nám to povedlo, je potřeba vzít v úvahu omezení způsobená fyzikální povahou robota. S těmito omezeními budeme muset dále počítat. Musí platit, že v každém okamžiku pohybu má robot:

- aktuální rychlost nižší než v_{max}
- aktuální tečné zrychlení nižší než a_{max}
- aktuální dostředivé zrychlení nižší než $cenacc_{max}$
- aktuální úhlovou rychlost nižší než ω_{max}
- aktuální úhlové zrychlení nižší než α_{max}

Tyto hodnoty jsou konstantní a jsou nastavovány experimentálně. V programu jsou reprezentovány strukturou `TrajectoryConstraints`:

```
struct TrajectoryConstraints {
    double maxv;
    double maxomega;
    double maxangacc;
    double maxacc;
    double maxcenacc;
    double maxe;
};
```

Tabulka 6.1: Kinematické parametry přímočarého úseku

Parametr	Význam	Vztah
v_1	rychlost na začátku úseku	
v_2	rychlost na konci úseku	
acc	zrychlení během úseku	$acc = \frac{(v_2 - v_1)(v_2 + v_1)}{l}$

6.1 Průběh rychlosti na přímočarém úseku

Tato část již byla implementována již dříve, průběh rychlosti na úsečce je jednoduše vyjádřen třemi parametry v tabulce (tabulka 6.1).

Po úsečce se tedy můžeme pohybovat třemi možnými způsoby:

- pohyb rovnoměrný
- pohyb rovnoměrně zrychlený
- pohyb rovnoměrně zpomalený

6.2 Průběh rychlosti na splinu

Kdybychom chtěli dokonale optimalizovat průběh rychlosti během splinu, museli bychom popsat funkci (6.1), která je minimem všech průběhů rychlostí závislých na různých omezení.

$$v(s) = \min(v_{max}, v(a_{max}, s), v(cenacc_{max}, s), v(\omega_{max}, s), v(\alpha_{max}, s)), \tag{6.1}$$

což by bylo dosti náročné. Pro jednoduchost vycházejme z toho, že křivost splinu během dráhy lineárně roste a potom zase lineárně klesá. Řekněme, že je potřeba do poloviny zpomalovat a ve druhé polovině zrychlovat. Přijměme tedy parametrizaci v tabulce (tabulka 6.2).

Čili závislost rychlosti na čase se skládá ze dvou úseček, které obvykle mají po řadě zápornou a kladnou derivaci. Tento průběh vypadá jako písmeno „V“. Zmíněný „obvyklý“ případ vychází z předpokladu, že nejsme ovlivněni okolními úseky natolik, aby musela být rychlost v_1 či v_2 nižší než v_c, což se ovšem prakticky stát může.

Tabulka 6.2: Kinematické parametry splinu

Parametr	Význam	Vlastnost
v_1	rychlost v bodě $P(0)$	
v_c	rychlost v bodě $P(0,5)$	
v_2	rychlost v bodě $P(1)$	
$acc1$	zrychlení mezi body $P(0)$ a $P(0,5)$	obvykle $acc1 < 0$
$acc2$	zrychlení mezi body $P(0,5)$ a $P(1)$	obvykle $acc2 > 0$

6.3 Maximální rychlosti segmentů

Nejdříve nastavíme rychlosti v segmentech tak, jako kdyby nebyly ovlivňovány podmínkami segmentů sousedních. Všechny úsečky tedy lze projet nejvyšší rychlostí maxv: $v_1 = v_2 = v_{max}$.

U splinů už není situace tak jednoduchá, protože musíme brát v úvahu i ostatní omezení (35). Nejprve nastavíme maximální hodnotu rychlosti uprostřed segmentu. K tomu potřebujeme poloměr křivosti v bodě $SEG(0.5)$

$$r_{min} = \frac{1}{\kappa(t = 0.5)} \tag{6.2}$$

Pokud vezmeme v úvahu všechna kinematické omezení (Kotlík, B. et al., 2003), rychlost v_c bude minimální hodnota ze všech maximálních rychlostí pro dané omezení.

$$v_c = \min\left(v_{max}, \omega_{max} r_{min}, \sqrt{cenacc_{max} r_{min}}, \sqrt{\alpha_{max} \frac{l r_{min}}{2}}\right) \tag{6.3}$$

kde l je délka celého splinu. Poslední člen je založen na následujícím odvození:

$$\begin{aligned} \alpha = \mathrm{d}\omega \mathrm{d}t &= \frac{v^2}{\mathrm{d}r\mathrm{d}s} \\ v^2 &= \alpha \mathrm{d}r\mathrm{d}s \\ v &= \sqrt{\alpha \mathrm{d}r\mathrm{d}s} \\ \mathrm{d}r\mathrm{d}s &= \frac{1}{\frac{\mathrm{d}\kappa}{\mathrm{d}s}} = konst. \end{aligned} \tag{6.4}$$

protože $\frac{\mathrm{d}\kappa}{\mathrm{d}s} = konst.$ je vlastnost klotoidy. Tato konstanta lze jednoduše vypočítat z průběhu křivosti klotoidy viz obr. 5.2.

$$\frac{\mathrm{d}\kappa}{\mathrm{d}s} = \frac{\kappa(t = 0.5) - 0}{l/2} \tag{6.5}$$

Máme tedy nastavenou maximální rychlost uprostřed segmentu. Dále je potřeba vyvážit rychlosti v_1, v_2 a v_C tak, aby nikde během segmentu nedošlo k překročení maximálních omezení. Za tímto účelem byl implementován iterativní optimalizační algoritmus začínající na hodnotách $v_1 = v_2 = v_{max}/2$, který nebudu blíže komentovat, je uveden v přiloženém zdrojovém kódu (A). Tento algoritmus je sice prakticky docela funkční, ale příliš intuitivní. Především je problémem to, že se provádí zbytečná iterace on-line. Tato část by se hodila v budoucnu ještě upravit.

6.4 Zajištění spojitosti rychlostí

Nyní, když máme nastaveny maximální rychlosti u jednotlivých segmentů, jsme v situaci, kdy potřebujeme v některých segmentech rychlost snížit tak, aby navazovala se segmentem sousedním.

Algoritmus byl jen nepatrně změněn oproti původní verzi s kružnicovými oblouky a pracuje následovně:

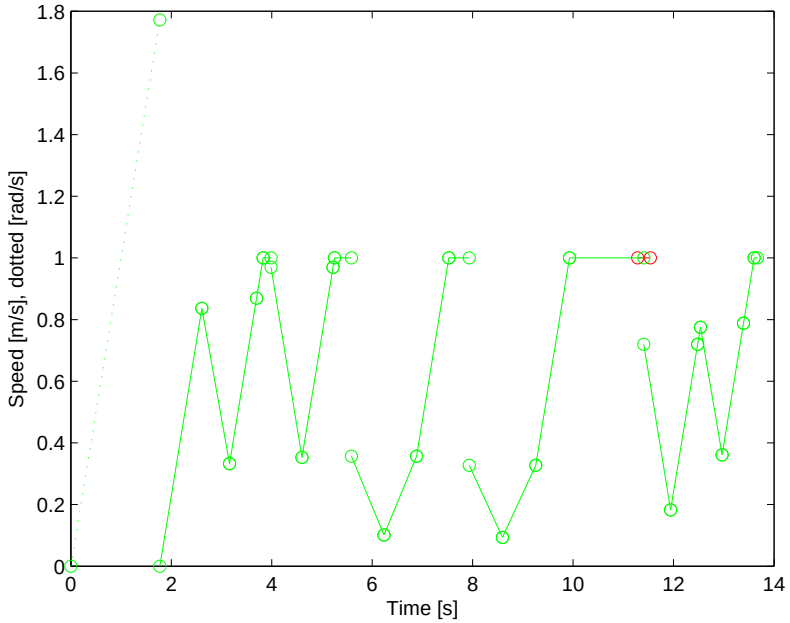
1. V první části procházíme segmenty od začátku do konce a rychlost v_1 nastavíme na hodnotu v_2 segmentu předchozího následujícím způsobem:

   - pokud se jedná o přímočarý segment, rozdělíme jej na dva tak, že první část bude mít $acc = acc_{max}$ a druhá část bude mít rychlosti v_1 i v_2 původní. Může se samozřejmě stát, že druhá část vůbec nebude existovat.
   - pokud se jedná o spline, pouze nastavíme hodnotu v_1 na hodnotu v_2 segmentu předchozího. Následně přepočteme hodnotu acc1.

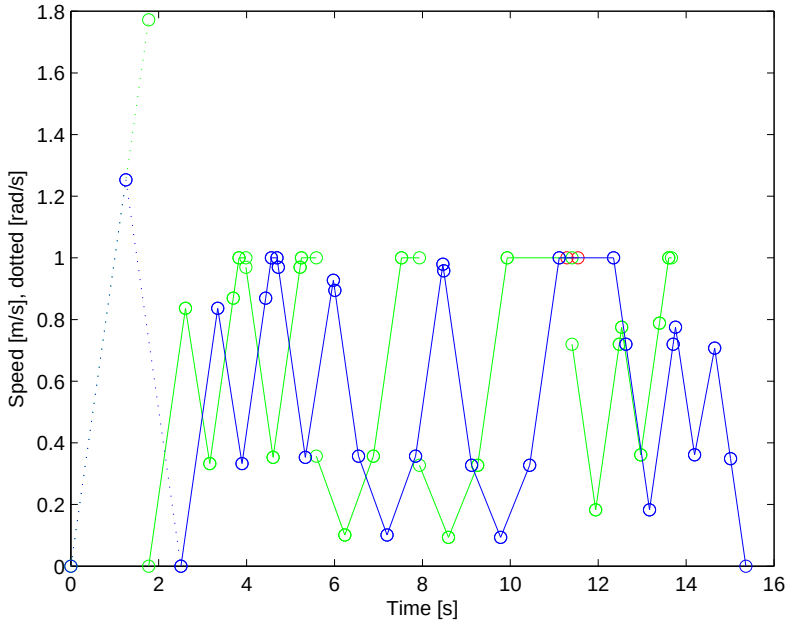
2. Ve druhé části procházíme sekvenci segmentů zpět od konce na začátek a nastavujeme rychlost v_2 na hodnotu v_1 segmentu předchozího (myšleno ve směru od konce):

   - jedná-li se o přímočarý úsek, rozdělíme jej na dva tak, že jeden bude mít hodnotu $acc = -acc_{max}$ a druhý bude mít průběh rychlosti nezměněn.
   - jedná-li se o spline, nastavíme hodnotu v_2 na hodnotu v_1 segmentu předchozího (myšleno ve směru od konce). Následně přepočteme hodnotu acc2.

Tím jsme dostali spojitý průběh rychlosti podél celé trajektorie. Na obrázku (obr. 6.1) je vidět, že části ve tvaru „V“, které reprezentují průběh rychlosti na splinu, navazují na



Obrázek 6.1: Průběh rychlosti po přidání zrychlení mezi segmenty



Obrázek 6.2: Průběh rychlosti po přidání zpomalení mezi segmenty

okolní části jen z jedné strany, kdežto na obrázku (obr. 6.2) už z obou stran a celý průběh je spojitý (modrá barva).

Kapitola 7

Algoritmus vyhýbání se překážkám

V kapitole 3 jsem uvedl původní metodu vyhýbání se překážkám, nyní, když už jsme věnovali úsilí zrychlování průjezdu pomocí splinů, bude se hodit vylepšit i tuto metodu.

Během pohybu robota je pravidelně kontrolována trajektorie, zda nekoliduje s nějakou překážkou. Typicky jde o soupeře, protože herní prvky, které by nám vadily, jsou většinou statické.

Pokud tedy zjistíme možnou kolizi, je třeba přeplánovat na trajektorii novou. Naším cílem je, aby toto robot prováděl za jízdy. Ovšem celý algoritmus plánování nové trajektorie trvá delší dobu než jeden cyklus vzorkování trajektorie, což znamená, že budeme ještě nějakou chvíli potřebovat jet po trajektorii staré. Plánovat trajektorii novou nebudeme z bodu, kde se právě robot nachází, ale z určitého bodu v budoucnosti

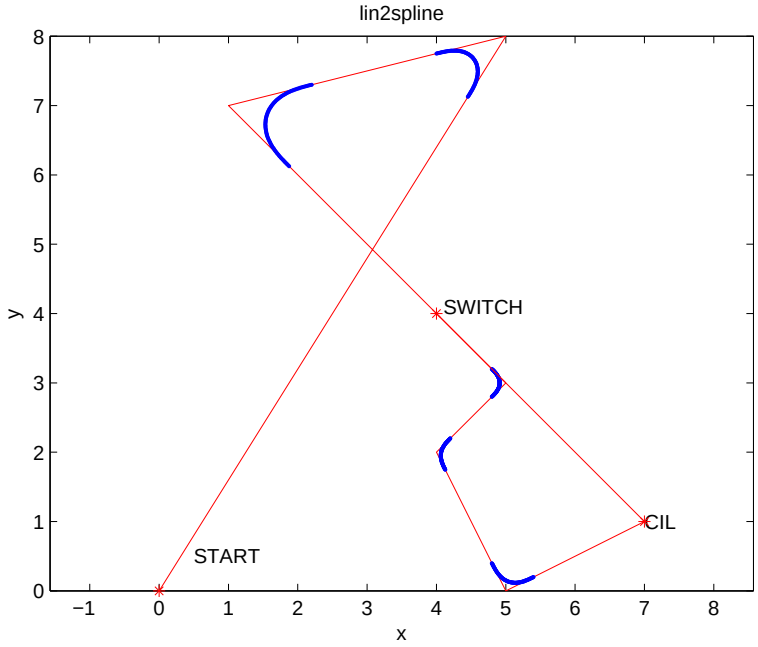
$$TRAJ(t_{switch}); \qquad t_{switch} \geqq t_{now} + t_{trgen} \tag{7.1}$$

Je třeba si uvědomit, že existuje rozdíl mezi výše zmíněným plánováním a přeplánováním za jízdy. Pokud plánujeme trajektorii za jízdy, v bodě $TRAJ(0)$ máme nenulovou rychlost. To znamená, že nejenže potřebujeme zajistit spojitost trajektorie a její tečny, ale musíme dát pozor na to, že nemůžeme libovolně generovat první bod průjezdu, abychom splnili kinematická omezení (souvisí se setrvačností robota). Tento požadavek jednoduše ošetříme tím, že první bod vygenerujeme násilně ve směru rychlosti ve vzdálenosti l_{brake} takové, během níž je schopen úplně zastavit.

$$l_{brake} = 1/2 \frac{v^2_{PNT'(0)}}{a_{max}} \tag{7.2}$$

$$PNT(1)_x = PNT(0)_x + l_{brake} \frac{v_0 x}{|v_0|} \tag{7.3}$$

$$PNT(1)_y = PNT(0)_y + l_{brake} \frac{v_0 y}{|v_0|} \tag{7.4}$$



Obrázek 7.1: Přeplánování trajektorie za jízdy

Bod $PNT(1)$ zajistí, že po vygenerování trajektorie je algoritmus přiřazování rychlostí schopen zajistit spojitý průběh rychlosti, jelikož robot je schopen do tohoto bodu zpomalit na nulovou rychlost.

Jiný rozdíl oproti generování trajektorie z klidu není. Když máme novou trajektorii hotovou, je třeba ji navázat na starou v bodě $S = TRAJ_i(t_{switch}) = TRAJ_{i+1}(0)$ a udělat z nich trajektorii jedinou. Rozdělíme tedy starou trajektorii na dvě poloviny v bodě C a druhou polovinu smažeme. Nepatrný problém nastává, pokud k rozdělení dojde uvnitř spline segmentu, protože trajektorie končí nenulovou křivostí a my nejsme schopni zajistit jinou než nulovou křivost v bodě $TRAJ(0)$ pro trajektorii následující. Tento problém bohužel nelze s použitím stávajících algoritmů žádnými modifikacemi odstranit a budeme se s ním tudíž muset smířit.

Kapitola 8

Závěr

Podařilo se vyjádřit popis spojitých křivek a implementovat funkční knihovnu v C++. Knihovna má určité nedokonalosti, především není dořešena situace, kdy je třeba přeplánovat trajektorii za jízdy, a také práce s kinematickými omezeními pro průjezd splinem není úplně dokonalá. Dalším vylepšením robota pro lepší lokalizaci pomocí odometrie by mohla být výměna příliš širokých koleček za užší.

Literatura

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E. a Thrun, S. (2005), *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, Cambridge, MA.

Karolík, A. (2006), 'Kubické spliny pro vícerozměrné prostory'. Semestrální práce, FEL ČVUT.

Kotlík, B., Lank, V., Růžičková, K., Vondra, M. a Vošický, Z. (2003), *Matematické, fyzikální a chemické tabulky*, Fragment. ISBN 80-7200-521-9.

Seng, R. a Severdia, M. (n.d.), 'The Clothoid', [online] `http://online.redwoods.cc.ca.us/instruct/darnold/CalcProj/Fall07/MollyRyan/Paper.pdf`.

Trdlička, J. (2005), 'Řízení pro robotický fotbal'. Diplomová práce, FEL ČVUT.

Webové stránky CTU Dragons [online] (2009). [cit. 2009], ⟨http://rtime.felk.cvut.cz/robot/⟩.

Webové stránky Českého kola Eurobot [online] (2009). [cit. 2009], ⟨http://www.eurobot.cz/⟩.

Wikipedia [online] (2009). [cit. 2009], ⟨http://en.wikipedia.org/⟩.

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD, na kterém jsou uloženy zdrojové kódy a video.

- Adresář `pdf`: Elektronická podoba bakálářské práce
- Adresář `src`: Zdrojové kódy knihovny v C++
- Adresář `app`: Přiložená nepublikovaná literatura (`.pdf`)
- Adresář `matlab`: Skripty v Matlabu k simulacím
- Adresář `video`: Záznam jízdy robota na cvičném hřišti (`.avi`)