Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Aplikace metod strojového učení v robotickém fotbalu

Plzeň 2013 Jakub Vališ

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 12. srpna 2013

Jakub Vališ

Abstract

The diploma thesis concentrates on an extension of robotic soccer module providing an elementary intelligence, it is the capability on the basis of accessible information from the other modules to determine an optimal direction of move a robot on playground, an avoiding the obstacles etc. including methods of machine learning to replace current rule based control

Poděkování

Panu Ing. Kamilovi Ekšteinovi, Ph.D. za za odborné vedení, za pomoc a rady při zpracování této práce.

Obsah

1 Úvod

Robotický fotbal je disciplína, ve které se každoročně na šampionátech utkávají přední světové univerzity, aby porovnaly vzájemnou technologickou vyspělost. Pro úspěch je totiž nezbytné zkombinovat různé algoritmy především z oblastí umělé inteligence, rozpoznávání obrazu, softwarového inženýrství a teorie řízení. Ačkoli myšlenka robotického fotbalu vznikla již v roce 1953, Česká republika stále nemá důstojného zástupce na světových šampionátech. Tým, který vznikl v roce 2010 na Západočeské univerzitě v Plzni, má za cíl vytvořit koncurenceschopný řídící software a utkat se s velmocemi robotického fotbalu na mistrovství světa.

Řídící software je modulární a každý modul má jiný úkol. Pro jednodušší vývoj celého řídícího software jsou moduly rozděleny tak, aby se vývojem každého mohl zabývat pouze jeden člověk. Tato diplomová práce má za cíl rozšířit původní modul elementární inteligence, který má na starost výpočet optimální trajektorie pohybu robota tak, aby co nejlépe plnil příkazy generované modulem herní strategie.

Vývoj modulu elementární inteligence se ubírá směrem nahrazování pravidlového řízení strojovým učením. Autor před samotným vývojem získával teoretické znalosti o těchto moderních metodách, které by mohly, při správném využití mít pozitivní vliv na kvalitu celého modulu. Na začátku práce je velmi stručné shrnutí teorie strojového učení.

Dále se práce zabývá vlastním vývojem modulu. Problemetika elementární inteligence je však velmi rozsáhlá. Obsahuje algoritmy pro plánování trasy, regulaci pohybu robota, vyhýbání se překážkám, apod. Ačkoli jsou některé algoritmy modulu poměrně složité, ve skutečnosti se jedná o sofitistifikované výpočty z oblasti analytické geometrie. Činnost modulu je tedy založena na výpočtech kružnic, jejich tečen, přímek, apod. Přesto, že jim v této diplomové práci není věnováno příliš prostoru, jsou pro činnost elementární inteligence nezbytné.

Vývoj modulu probíhá zatím pouze v simulaci, neboť reální roboti stále nejsou připraveni. Nahrazení pravidlového řízení robotů strojovým učením, přechod z simulace do reálného prostředí usnadnit. Je to z toho důvodu, že pouhým přetrénováním algoritmů, bez nutnosti explicitního přeprogramování, by se měl modul elementární inteligence přizpůsobit rozdílům reálného prostředí oproti podmínkám v simulaci.

2 Strojové učení

Strojové učení je podoblast umělé inteligence, která má stále větší a větší uplatnění. Jedná se o algoritmy a techniky, které umožňují programu měnit svoji vnitřní strukturu nebo hodnoty parametrů tak, aby se adaptoval na řešení zadaného problému. Tato adaptace, bez nutnosti explicitního přeprogramování programu, je označována jako schopnost se učit. Poměrně přesnou definici učení stanovil Tom Mitchel (1988):

> *O počítačovém programu řekneme, že se učí vykonávat úlohu T (Task) ze zkušenosti E (Experience) s úspěšností P (Performance) tehdy, když úspěšnost P na úloze T roste se vzrustající zkušeností E.*

Existuje několik důvodů, proč techniky strojového učení používat, ačkoliv se zdá, že naprogramovat aplikaci pomocí pravidel tak, aby rovnou plnila správně zadanou úlohu, je mnohem jednodušší.

- Problém je obtížně definovatelný jinak, než na příkladech. To znamená, že je možné specifikovat velké množství vstupních a výstupních párů, ale ne vztah mezi nimi. Cílem takových programů je, aby si stroj sám dokázal vztahy mezi vstupními a požadovanými výstupními daty co nejpřesněji nadefinovat.
- Data mining: Vyhledávat informace ve velkém množství dat je téměř nemožné. Mezi jednotlivými informacemi však často existují závislosti a programy strojového učení bývají používány právě k extrakci těchto vztahů.
- Programy mohou být často vyvíjeny v jiném prostředí, než pro které jsou určeny, a dynamika cílového prostředí nemusí být zcela známá. Z toho důvodu je potřeba, aby se program po nasazení v novém prostředí dokázal jeho dynamice přizpůsobit. Prostředí se navíc může v průběhu času měnit a program tomu musí své chování neustále přizpůsobovat.
- Znalosti potřebné pro správné provedení požadovaného úkolu mohou být natolik rozsáhlé, že není možné, aby je programátor mohl explicitně

naprogramovat. Stroje tak mohou tyto znalosti postupně získávat a vylepšovat svoje chování natolik, že zvládnou více věcí, než kolik by byl programátor schopen či ochoten naprogramovat.

- O zadané úloze jsou neustále objevovány nové vědomosti, které nebyly známy v průběhu implementace programu. Metody strojového učení lze využít jako nástroje, který nově nabité vědomosti postupně integruje do systému, a tím se lze vyhnout neustálému explicitnímu přeprogramovávání aplikace.

Ačkoli to nemusí být často znát, metody strojového učení jsou dnes již integrovány do aplikací, které často používá i široká veřejnost. Přestože se tato diplomová práce zabývá především implementací technik strojového učení pro zkvalitnění řídícího systému robotického fotbalu, považuje autor za vhodné na tyto aplikace poukázat.

- **Strojový překlad:** Strojový překlad (Machine Translation) je jednou z nejstarších oblastí umělé inteligence. Jedná se o automatickou transformaci textu z jednoho přirozeného jazyka do druhého pomocí počítače. První systémy strojového překladu aplikovaly rozsáhlé množství pravidel pro překlad z jednoho jazyka do druhého. Překlad samotný však skrývá tolik úskalí, že pomocí pravidel uspokojivý překladač naprogramovat nelze. Moderní automatické překladače tak využívají metody strojového učení, kdy se pravidla překladu sama extrahují z rozsáhlého množství dvojjazyčných textů. Tyto překladače stále nejsou dokonalé, ale jejich výkonnost je daleko vyšší, než u těch pravidlově řízených.

- **Nákupní košík:** Jedná se o metody strojového učení, které jsou použity v e-shopech. Aplikace registruje prohlížené zboží, případně zboží v košíku, a podle toho určuje, které další zboží zákazníkovi přímo nabídne či na které mu zobrazí reklamu. Cílem těchto technik strojového učení je co nejvíce zvýšit prodej.

- **Data mining:** Množství informací, které se v dnešní době nachází na internetu, je obrovské. Prohledat v omezeném čase takové množství dat a extrahovat potřebné informace je velice náročná úloha. Z toho důvodu využívají strojového učení pro data mining veškeré moderní vyhledávače jako například Google.

Při aplikaci algoritmů strojového učení jsou objekty v daném prostředí popsány vektorem příznaků $\vec{x}$. Cílem algoritmů strojového učení je nalézt funkci $h_w(\vec{x})$, která se nazývá **hypotéza** (angl. hypothesis) takovou, aby co nejlépe mapovala vektor příznaků $\vec{x}$ na očekávaný výstup y. Hledání hypotézy $h_w(\vec{x})$ je proces učení, ke kterému je potřeba tzv. **trénovací množina**. Jedná se o seznam objektů, na něž se aplikují metody strojového učení. Hypotéza $h_w(\vec{x})$ je pak schopná predikovat dostatečně přesný výstup i pro objekty, které se v trénovací množině nenachází. Metody strojového učení se dělí na **učení s učitelem** a **učení bez učitele** podle toho, zda pro objekty trénovací množiny známe přesný výsledek y nebo ne. Dalším, dnes velice oblíbeným typem alogoritmů, je **reinforcement learning** (posilované učení), který se od předchozích dvou odlišuje tím, že systémy jsou trénovány většinou za běhu, nikoli předem na trénovací množině, viz dále.

Na téma strojové učení lze napsat mnoho set stránek textu a také se jím obrovské množství publikací zabývá. V těchto publikacích jsou popsány jak teoretické principy fungování strojového učení, tak praktické příklady použití. Autor pro diplomovou práci čerpal z těchto publikací a online kurzů [Ng(2011)], [Hinton(2012)], [Alpaydin(2010)], [Smola(2008)], [Rojas(1996)], [Sutton, Barto(2005)] a další, viz použitou literaturu.

Jelikož lze především na internetu najít dostatečné množství materiálů, rozhodl se autor pouze krátce shrnout jednotlivé typy strojového učení a poté se přímo zaměřit na modul elementární inteligence robotického fotbalu.

2.1 Strojové učení s učitelem

Strojové učení s učitelem (v angličtině Supervised Machine Learning) hledá parametry hypotézy $h_w(\vec{x})$ na základě rozdílu mezi skutečným výsledkem y a výstupem hypotézy $h_w(\vec{x})$.

Výstup y může mít buď podobu reálného čísla nebo může být diskrétní. Podle toho rozlišujeme dva druhy úloh:

- **Regrese** (angl. **regression**): Regresní úlohy mají reálný výstup, může se jednat například o předpovídání ceny bytu v určité oblasti na základě jeho rozlohy, stáří, apod. Nejznámějšími algoritmy pro trénování regresních úloh jsou **lineární regrese** a **support vector regression** (SVR).

- **Klasifikace** (angl. **Classification**): Klasifikační úlohy třídí objekty do několika předem definovaných tříd. Výsledkem hypotézy $h(x)$ je pořadí třídy, do které má být objekt zařazen, a její výstup je tedy diskrétní. Mezi klasifikační úlohy patří například dělení e-mailů na spamy a hamy nebo třídění kovů na hliník, ocel, titan, apod., na základě hmotnosti, tvrdosti a pevnosti. Klasifikační algoritmy jsou používané daleko častěji než ty regresní a jedná se o **logistickou regresi**, **support vector machines** nebo **neuronové sítě**. Neuronové sítě lze použít i pro regresi, ale jejich použití v klasifikačních úlohách je daleko obvyklejší.

Metody učení s učitelem jsou velice často používané a autor jich částečně využil i pro vývoj modulu elementární inteligence.

2.2 Strojové učení bez učitele

Algoritmy strojového učení bez učitele se používají k nalezení skryté struktury dat. Tato data jsou uložena bez jakékoli dodatečné informace (bez správného výsledku y), na rozdíl od strojového učení s učitelem.

Typickým příkladem algoritmů strojového učení bez učitele je hledání shluků, tedy objektů v trénovací množině s podobnými vlastnostmi. Shluk se v angličtině nazývá **cluster**, proto se o shlukování častěji mluví jako o **clusterování**. Clusterování je mnoho druhů, z nichž velice často používané je například **K-means clustering**, **Hierarchical clustering**, apod.

Dále se strojové učení bez učitele používá ke snížení dimenze příznakového prostoru. Jednoduše řečeno se jedná o snížení počtu příznaků na základě jejich vzájemné závislosti, která nemusí být zřejmá. Redukce dimenzí se používá především kvůli kompresi trénovacích dat. Nejčastěji používanými algoritmy jsou **PCA** (Principal Component Analysis) nebo **FA** (Factor Analysis).

Strojové učení bez učitele lze použít pro inteligentní dolování dat (**data mining**), které neustále nabývá na důležitosti. Dále se jedná o některé problémy v biologii, například hledání shluků v genetickém kódu, nebo v astronomii k analýze shluku galaxií apod. Přestože metody strojového učení bez učitele jsou ve světě počítačů používány stále častěji, pro modul elementární inteligence pro ně zatím autor žádné využití nenašel.

2.3 Reinforcement Learning

Algoritmy **posilovaného učení** (reinforcement learning) se učí jaké akce použít v daných situacích, tak aby číselná hodnota zpětné odezvy systému byla z dlouhodobého hlediska co nejvyšší. Tato odezva systému se nazývá odměna (**reward**), je-li kladná, nebo trest (**punishment**), je-li záporná. Je-li systém správně natrénován, preferuje takové chování, které přináší největší užitek z dlouhodobého hlediska, přestože okamžitá hodnota odezvy systému nemusí být nejvyšší.

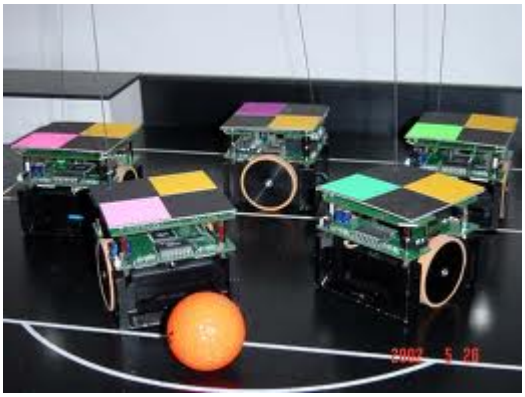
Jedním z problému posilovaného učení je nalezení rovnováhy mezi využíváním již naučených znalostí (**exploitation**) a zkoumáním nových možností, které by mohli přinést v budoucnu vyšší užitek (**exploration**). Nalezení rovnováhy mezi průzkumem a využíváním znalostí je velice intenzivně studované téma, neboť obě části jsou nezbytné pro správnou činnost posilovaného učení. Nejjednodušším a velice často používaným způsobem je takzvaný ϵ-hladový (ϵ-greedy) algoritmus. Systém v takovém případě s pravděpodobností ϵ použije akci, která je v daném okamžiku optimální a s pravděpodobností $(1 - \epsilon)$ vybere akci náhodně. Dalšími algoritmy, ktere lze použít jsou **Softmax** nebo posilované porovnávání (**Reinforcement Comparison**).

Hlavní třídy algoritmů strojového učení jsou dynamické programování (**dynamic programing (DP)**), **Monte Carlo** a **temporal difference (TD)**. Každá z těchto metod má své silné a slabé stránky. Metody dynamického programování jsou výborně prozkoumány z matematického hlediska, ale vyžadují úplnou znalost dynamiky prostředí. Monte Carlo metody jsou jednoduché a nepotřebují model prostředí, ale odezva prostředí není okamžitá. TD metody nepotřebují model, odezva je okamžitá, ale vyžadují před použitím hlubší analýzu. Většina systému výše popsané metody kombinuje tak, aby využil všech možných výhod každé z nich.

3 Robotický fotbal

V předchozí kapitole byly shrnuty základy strojového učení a v následujících kapitolách bude popsána implementace těchto metod pro účely robotického fotbalu. Robotický fotbal je dnes již tradiční disciplína mezi univerzitami po celém světě. Každá z univerzit postaví mužstvo robotických hráčů, kteří se spolu utkají ve fotbalovém zápase. V zápasech robotického fotbalu jde především o srovnání technologické úrovně jednotlivých univerzit

Vzhled robotů, jejich počet a pravidla zápasů se liší dle kategorií. Tým Západočeské univerzity vyvíjí software a roboty pro střední ligu kategorie MiroSot, kde se spolu utká vždy pět hráčů z každého týmu. V této kategorii jsou roboti limitováni velikostí $75 \times 75 \times 75$ mm a hrají s oranžovým golfovým míčkem, viz obrázek 3.1.

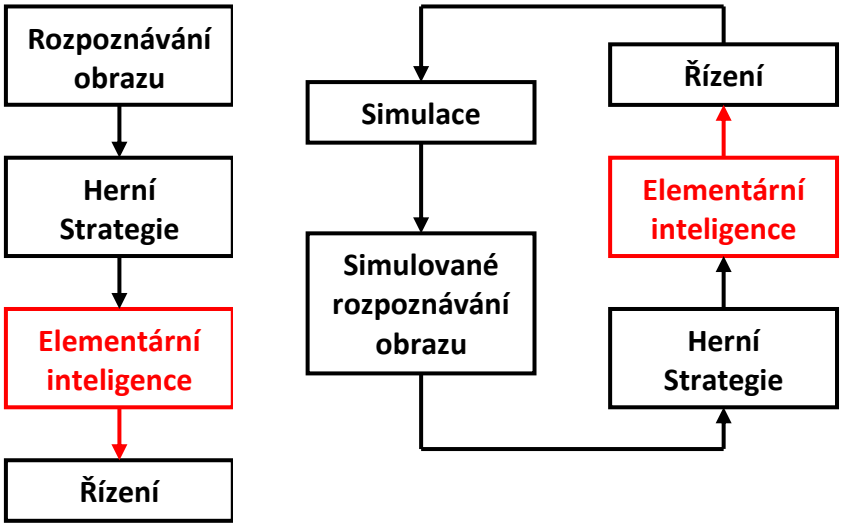


Obrázek 3.1: Tým robotického fotbalu

Hráči nejsou autonomní, jsou řízeni z centrálního počítače. Aktuální stav hry je snímán kamerou, která se nachází v předepsané výšce nad středem hřiště. Data z kamery jsou přenášena do centrálního počítače a jsou jediným zdrojem informací pro řídící software o aktuálním stavu hry na hřišti. Data jsou v řídícím software vyhodnocena a každému z robotů jsou poslány povely. Změny polohy hráčů a míče jsou opět kamerou snímány a posílány do centrálního počítače. Tato smyčka je natolik rychlá, že pohyb hráčů po hřišti je víceméně plynulý. Technologická úroveň řídícího software má nejzásadnější vliv na výsledek zápasu.

3.1 Řídící software

Řídící software má modulární strukturu. Každý modul má odlišný úkol a jednotlivé moduly mezi sebou komunikují pomocí zpráv. Na obrázku 3.2a je zobrazen graf a propojení všech modulů ve finální verzi řídícího software. Bohužel, prototypy robotů stále nejsou k dispozici, a proto byla tato práce implementována a testována pouze v simulaci. Aktuální struktura modulů řídícího software tak odpovídá struktuře na obrázku 3.2b.



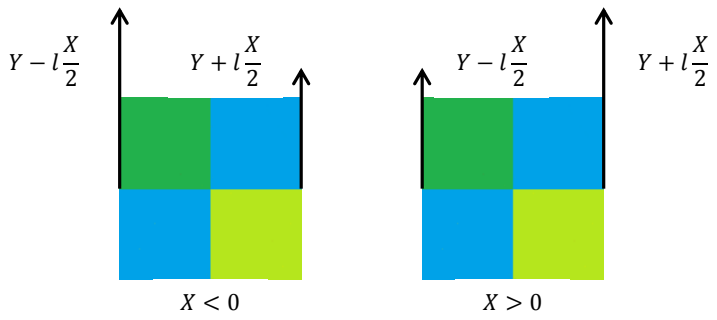
(a) Finální verze (b) Aktuální verze

Obrázek 3.2: model řídícího softwaru

Simulační modul obsahuje fyzikální engine využívající knihovny NVIDIA PhysX, který nikdy nebyl zcela odlazen, a je příčinou mnohých problémů při vývoji řídícího software. Modul elementární inteligence přímo s fyzikálním enginem pracuje, tudíž nestabilita prostředí výrazně zpomalila vývoj, ale implementovaný software by měl být dostatečně robustní, aby obstál i v přirozených podmínkách.

3.2 Modul Elementární Inteligence

Modul elementární inteligence slouží jako prostředník mezi herní strategií a samotnými roboty. Výstup herní strategie je příliš abstraktní a právě v modulu elementární inteligence je transformován na výstup, který lze použít pro ovládání robotů. Výstupem z modulu elementární inteligence je 2 dimenzionální pohybový vektor, který určuje směr a rychlost pohybu robota. Význam složek X a Y pohybového vektoru je zobrazen na obrázku 3.3, kde l je velikost rozvoru kol robota.



Obrázek 3.3: Převod pohybového vektoru

Cílem diplomové práce je rozšíření původního modulu elementární inteligence, na kterém autor pracoval v rámci bakalářské práce. Jelikož se ale od doby vývoje změnil celý řídící systém, původní modul elementární inteligence byl prakticky nefunkční a musel být zcela přepracován. Hlavní změna řídícího software se týká převodu složky X a Y pohybového vektoru na rychlost jednotlivých motorů. Rovnice 3.1 a 3.2 definují převod v původním systému. Převod pohybového vektoru na pohyb robota v původním systému je podle autora logičtější, ale pro vývoj nového modulu elementární inteligence se současnému stavu musel přizpůsobit.

$$W_l = Y - X \tag{3.1}$$
$$W_r = Y + X \tag{3.2}$$

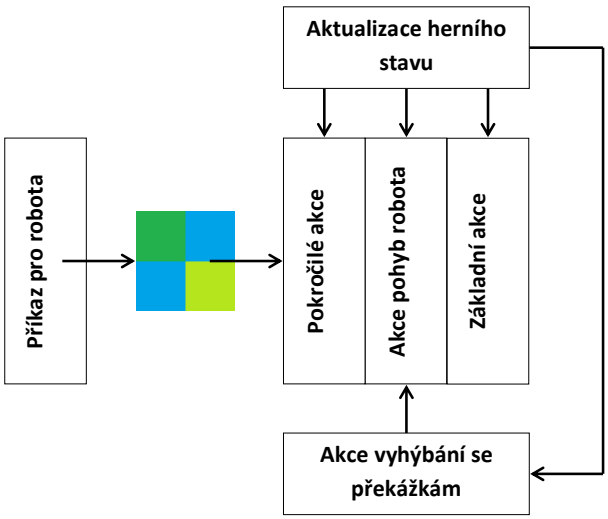
Formát příkazů herní strategie ještě není definován, neboť oba moduly (elementární inteligence i herní strategie) jsou stále ve vývoji, a tudíž může

mít různou formu abstrakce. V nejjednodušším případě posílá herní strategie pouze souřadnice, na které se má robot přesunout a případně, jak má být v tomto cílovém bodě natočen. Elementární inteligence je pak daleko jednodušší a nemá žádnou informaci o účelu robotova pohybu.

V nejsložitějším případě jsou výstupem herní strategie velmi abstraktní příkazy typu: *„nahraj robotovi číslo 2“*. Elementární inteligence musí příkaz zpracovat, vypočítat optimální dráhu robota a příkaz provést. Modul elementární inteligence v rámci diplomové práce byl navržen tak, aby zpracovával příkazy s nejvyšší formou abstrakce.

3.2.1 Akce

Každá akce má na starosti některou činnost pro správný pohyb robota po hrací ploše a je základní jednotkou modulu elementární inteligence. **Základní akce** pracují přímo s pohybovými vektory a fyzikálním enginem a jedná se o samotné jádro modulu, na jehož odladění závisí kvalita elementární inteligence především. **Pokročilé akce** tyto základní akce využívají pro zpracování příkazů z modulu herní strategie. Tyto akce stojí na pomezí elementární inteligence a herní strategie a jejich implementace umožňuje zpracování velmi abstraktních příkazů, viz výše. Kromě těchto akcí jsou implementovány i akce pro vyhýbání se překážkám, které stojí mimo obě předchozí skupiny.



Obrázek 3.4: Systém akcí robota

Příkazy jsou pro každého robota zpracovány separátně a nezávisle na ostatních. Při inicializaci modulu elementární inteligence je pro každého robota spuštěno vlastní vlákno. Každá akce pro každého robota je také zpracovávána separátně ve vlastním vlákně. Tento systém je náročnější na synchronizaci jednotlivých vláken, ale díky tomu je možné zpracovávat v jednom okamžiku více akcí najednou. Většinou má každý robot aktivní 3 akce – základní, pokročilou a pro vyhýbání se překážkám. Ostatní akce jsou uspány, aby nesnižovaly svojí činností výkon řídícího systému.

Pokročilé akce většinou aktivují základní akci a poté čekají na dokončení její činnosti. Základní akce naopak přímo počítají pohybové vektory a je nezbytné je synchronizovat se smyčkami celého systému. K tomu slouží semafor, který po aktualizaci informace o stavu hry na hřišti propustí základní akci k další činnosti a po vypočtení pohybového vektoru pro robota opět zablokuje. Se stejnou frekvencí pracuje i akce pro vyhýbání se překážkám, která vždy po aktualizaci pozic robotů kontroluje, zda není třeba dráhu robota upravit, aby nedošlo ke kolizi. Celý systém akcí pro jednoho robota si lze prohlédnout na obrázku 3.4

4 Základní akce

Základní akce tvoří nejspodnější vrstvu elementární inteligence. Tvoří abstrakci od fyzikálního enginu pro všechny ostatní akce a na jejich kvalitě závisí i kvalita celého modulu. Jelikož ostatní akce jsou od fyzikálního enginu zcela izolovány, lze je natrénovat pro všechny roboty najednou. Základní akce se trénují pro každého robota zvlášť, neboť každý může mít jinak seřízená kola, apod., což ovlivňuje jeho chování při pohybu. Při použití reálných robotů bude navíc potřeba veškeré základní akce znovu natrénovat.

Problémy s fyzikálním enginem se projevily zejména při implementaci základních akcí. Odezva systému není pro každý běh stejná, takže akce natrénovaná v předchozím běhu programu byla často při znovuspuštění zcela nepoužitelná. Použité trénovací algoritmy by tak měly být dostatečně robustní i pro trénování modulu elementární inteligence se skutečnými roboty.

Fyzikální engine navíc ke všem již zmíněným problémům s nepravidelnou frekvencí generuje chybové hlášky, které celou aplikaci pozastaví a znesnadňují natrénování jakýchkoliv akcí. V podkapitolách jednotlivých základních akcí jsou podrobněji nastíněna řešení těch problémů, které se přímo týkaly jejich fungování.

4.1 Pohyb po přímce

Jak je patrné z obrázku 3.3, aby se robot pohyboval po přímce, musí být X složka pohybového vektoru nulová. Naopak Y je rovno ve většině případů 1, aby robot co nejrychleji překonal požadovanou vzdálenost, viz rovnice 4.1 a 4.2. Robot je schopen pohybu dopředu i dozadu a tato akce dokáže podle robotova natočení a pozice cílového bodu sama určit, kterým směrem se má robot pohybovat. Pohybuje-li se robot dozadu, hodnota Y není 1, ale -1.

$$X = 0 \tag{4.1}$$

$$Y = 1 \tag{4.2}$$

Směrnice plánované přímky, po které se má robot pohybovat, většinou neodpovídá natočení robota na začátku pohybu zcela přesně. Jedná se o ne-

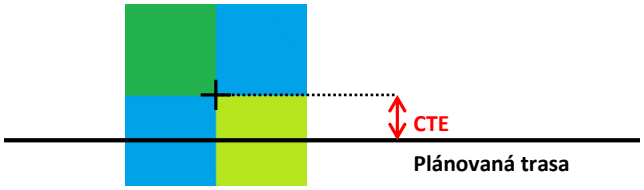
přesnosti vzniklé předchozím pohybem či natáčením robota nebo může jít o nepřesnosti v rozpoznávání obrazu z kamery. V každém případě, pokud by rovnice pro X a Y zůstaly v podobě rovnic 4.1 a 4.2, robot se během okamžiku vzdálí od plánované trasy.

Pro udržení robota na plánované trase je třeba jeho pohyb regulovat. V rámci diplomové práce se autor inspiroval informacemi z on-line kurzu [Thrun(2012)] a použil PID-regulátor pro regulaci pohybu, který se velice často používá, a to nejen v robotice.

4.1.1 PID-regulátor

PID-regulátor reguluje na základě odchylky od očekávané či požadované hodnoty. Tato odchylka se nazývá **cross track error (CTE)** a pro regulaci pohybu po přímce představuje vzdálenost robota od plánované trasy, viz obrázek 4.1.

Regulátor mění podle rovnice 4.7 hodnotu X složky tak, aby se robot při odchýlení od plánované trasy vrátil zase zpátky. Je-li robot odchýlen od plánované trasy napravo, hodnota X musí být kladná, aby se robot pohyboval proti směru hodinových ručiček. Je-li robot nalevo X je naopak záporná, proto se v takovém případě CTE násobí -1 a rovnice 4.7 funguje beze změny pro všechny případy. Parametry c_p, c_i a c_d jsou základní koeficienty ovlivňující činnost PID-regulátoru a pro správnou činnost je třeba je nastavit. Nastavení lze udělat manuálně, ale vyžaduje to obrovské množství práce a zkušeností, které autor postrádá. PID-regulátor je tedy nastaven automaticky pomocí algoritmu **Twiddle**, který je popsán níže.



Obrázek 4.1: Cross track error

PID-regulátor, jak lze snadno poznat z názvu i z rovnice 4.7, se skládá ze tří částí – proporcionální, integrační a derivační.

- **Proporcionální část** regulátoru reguluje pohyb robota podle hodnoty CTE, viz rovnici 4.3. Při odchýlení robota od plánované trasy nastaví P-regulátor hodnotu X tak, aby se vrátil zpět. Dokud se nevrátí, je hodnota X nenulová, což robota neustále natáčí směrem k plánované trase, i když už je poměrně blízko. Robot tak plánovanou trasu překročí a je nutné jeho pohyb regulovat na opačnou stranu. Tomuto se anglicky říká **overshoot** a robot se tak místo po přímce pohybuje ve vlnovkách, viz obrázek 4.2a.

$$X_{temp} = c_p \cdot cte \tag{4.3}$$

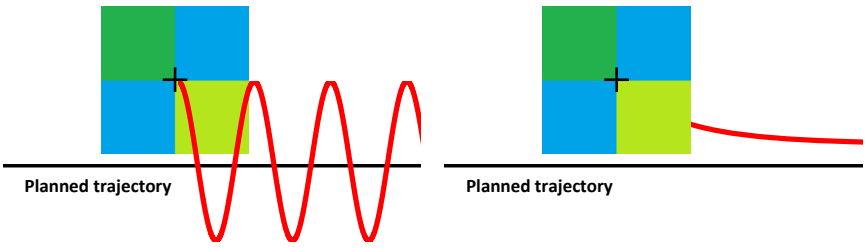
- **Derivační složka** regulátoru vlnovitý pohyb eliminuje. Regulace je ovlivněna nejen aktuální odchylkou, ale i změnou odchylky za určitý časový úsek, viz rovnici 4.4. Pokud se odchylka robota od plánované trasy zmenšuje, působí derivační složka proti proporcionální. Je-li robot dostatečně blízko, hodnota X může být i záporná a robot se tak opět natáčí ve směru pohybu a plánovanou trasu nepřekročí. Naopak, pokud se robot od plánované trasy vzdaluje, derivační složka proporcionální v regulaci podporuje.

$$X_{temp} = c_d \cdot \frac{cte_t - cte_{t-1}}{\Delta t} \tag{4.4}$$

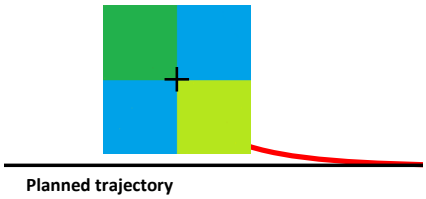
- Při idélním nastavení systému k regulaci stačí PD-regulátor. Může se ale stát a stává se, že systém není nastaven dokonale. V případě robotického fotbalu nemusí být kola robota seřízena dokonale ve směru pohybu. PD-regulátor není tuto chybu systému schopen regulovat a robot se tak k plánované trase pouze přiblíží, ale až k ní se nikdy nedostane, viz 4.2b. K regulaci systémové chyby slouží **integrační část** regulátoru, která reguluje na základě součtu odchylek od počátku pohybu robota do aktuálního času T, viz 4.5.

$$X_{temp} = c_i \cdot \sum_{t=0}^{T} cte \tag{4.5}$$

  Ve spojitém prostředí by místo součtu odchylek byl v rovnici integrál křivky průběhu CTE. Pokud systémová chyba způsobí, že se robot nedokáže dostat zpět na plánovanou trasu, součet odchylek se neustále navyšuje a X je větší, než by bylo pouze s použitím PD-regulátoru. Tento rozdíl hodnot X způsobí, že se robot na plánovanou trasu zcela vrátí, viz obrázek 4.2c.



(a) P-regulátor (b) PD-regulátor



(c) PID-regulátor

Obrázek 4.2: Vlastnosti regulátorů

Výsledné hodnoty X a Y pohybového vektoru se spočtou z rovnic 4.7 a 4.6. Převod pohybového vektoru na rychlosti jednotlivých kol je popsán na začátku kapitoly. Žádná z hodnot $Y + l\frac{X}{2}$ a $Y - l\frac{X}{2}$ nesmí být větší než 1 nebo menší než -1, jinak jsou na tyto hranice zarovnány. Toto zarovnání ale změní poměr mezi rychlostí levého a pravého kola a tím i regulaci pohybu. Z toho důvodu je nutné parametry X a Y upravit podle vzorců 4.9 a 4.8

$$Y_{temp} = 1 \tag{4.6}$$

$$X_{temp} = c_p \cdot cte + c_d \cdot \frac{cte_t - cte_{t-1}}{\Delta t} + c_i \cdot \sum_{t=0}^{T} cte \tag{4.7}$$

$$Y = \frac{Y_{temp}}{(Y_{temp} + l\frac{X_{temp}}{2})} \tag{4.8}$$

$$X = \frac{X_{temp}}{(X_{temp} + l\frac{X_{temp}}{2})} \tag{4.9}$$

4.1.2 Algoritmus Twiddle

Algoritmus Twiddle slouží k natrénování parametrů PID-regulátoru c_p, c_d a c_i definovaných v rovnici 4.7. Teoreticky se jedná o algoritmus reinforcement learningu, který metodou pokus omyl nalezne optimální parametry regulátoru.

Algoritmus funguje tak, že pro velkou množinu různých trojic parametrů PID-regulátoru se spustí určitý **trénovací cyklus** a vybrány jsou parametry, pro které je celková chyba nejmenší. Způsob výběru trojic parametrů je popsán níže. Trénovací cyklus je v případě trénování regulátoru pro pohyb na přímce seznam bodů, které má robot postupně projet. Tento seznam bodů i jejich pořadí je shodné pro všechny vybrané trojice parametrů. Chybou testované trojice je pak průměrná odchylka CTE v průběhu trénovacího cyklu.

Inteligentní výběr trojic parametrů je důvod, proč je v aplikaci použit algoritmus Twiddle místo **brute force**, tedy zkoumání všech možných kombinací parametrů. Před začátkem algoritmu inicializujeme dvě pole proměnných $c = [0, 0, 0]$ a $dc = [1, 1, 1]$. Pro počáteční trojici parametrů necháme robota projet body trénovacího cyklu a výslednou chybu uložíme do proměnné *bestError*, která bude po celou dobu trénování udržovat informace o nejlepším dosaženém výsledku. Samotný algoritmus je popsán níže v pseudokódu a probíhá, dokud je některý z parametrů *dc* větší, než zvolená konstanta ϵ (typicky 0.01, podle požadavků na přesnost).

```
pro každé c[i] z c:
        c[i] = c[i] + dc[i]
        error = TrainingCycle(c)
        if(error < bestError)
        {
                bestError = error
                dc[i] *= 1.1
        }
        else
        {
                c[i] = c[i] - 2 * dc[i]
                error = TrainingCycle(c)
                if(error < bestError)
```

```
                {
                        bestError = error
                        dc[i] *= 1.1
                }
                else
                {
                        c[i] = c[i] + dc[i]
                        dc[i] *= 0.9
                }
        }
```

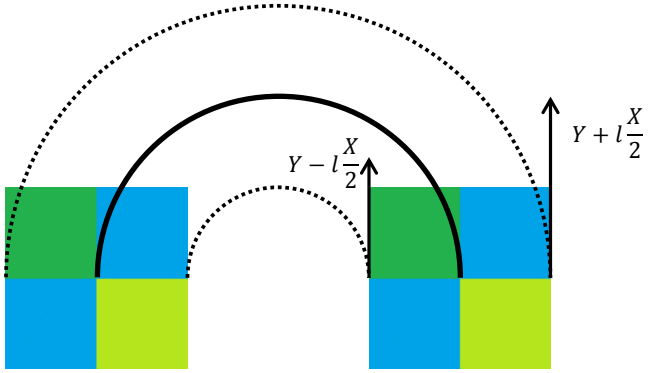
Algoritmus tedy neustále vybírá hodnoty parametrů v okolí těch, pro které byla chyba v průběhu trénovacího cyklu nejmenší. Pokud nejsou na hranici daného okolí nalezeny lepší parametry velikost hodnoty $dc[i]$ se neustále zmenšuje a tím se zkoumá stále blíže aktuálně optimálním parametrům. Naopak, pokud algoritmus naopak nalezne lepší parametry, pak se nadále zkoumá okolí této nejlepší trojice.

V této základní podobě bohužel Twiddle se současným fyzikálním enginem nefunguje. Parametry PID-regulátoru nalezené při jednom běhu aplikace byly v dalším zcela nepoužitelné. Pro nalezení odpovídajících parametrů bylo potřeba algoritmus upravit.

Hlavní změna se týká uchovávání parametrů, které sice nebyly nejlepší, ale chyba v průběhu trénovacího cyklu byla menší, než určitá přijatelná mez. Po skončení Twiddlu se pro všechny uložené parametry v paměti provede trénovací cyklus znovu a pokud chyba přijatelnou mez překročí, jsou tyto parametry z paměti vymazány. Tento třídící cyklus se provede ještě několikrát a na konci zbydou pouze ty parametry, které regulují pohyb robota po přímce stabilně. Z nich se vyberou ty, jejichž průměrná chyba ze všech běhů trénovacích cyklů byla nejmenší.

4.2 Pohyb po kružnici

Robot se pohybuje po kružnici, je-li X složka pohybového vektoru nenulová. Kromě této odchylky jsou si však obě akce (pohyb po přímce a pohyb po kružnici) principiálně velice podobné. Na obrázku 4.3 a v rovnicích 4.10 až 4.12 je znázorněno, jak velký vliv má hodnota X na zakřivení dráhy robota.



Obrázek 4.3: Pohyb po kružnici

$$Y_{temp} = 1 \tag{4.10}$$

$$Y_{temp} + l \cdot \frac{X_{temp}}{2} = \phi(r + \frac{l}{2}) \tag{4.11}$$

$$Y_{temp} - l_{temp} \cdot \frac{X_{temp}}{2} = \phi(r - \frac{l}{2}) \tag{4.12}$$

Poměr rychlostí obou kol je roven poměru délek oblouků o poloměrech $r + \frac{l}{2}$ a $r - \frac{l}{2}$, viz rovnici 4.13. Dalšími úpravami této soustavy rovnic lze dojít k výsledku rovnice 4.14, což znamená, robot se pohybuje po oblouku o velikosti $r = \frac{1}{X}$.

$$\frac{Y_{temp} + l\frac{X_{temp}}{2}}{Y_{temp} - l\frac{X_{temp}}{2}} = \frac{r + \frac{l}{2}}{r - \frac{l}{2}} \tag{4.13}$$

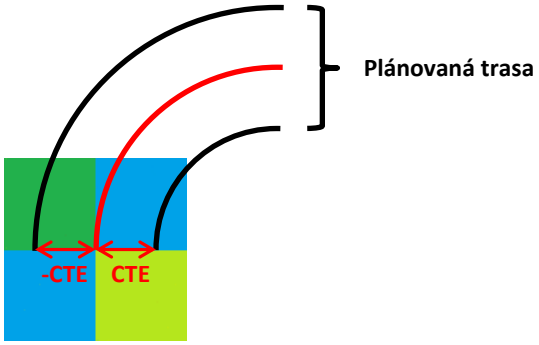
$$X_{temp} = \frac{1}{r} \tag{4.14}$$

Pokud by fyzikální engine fungoval podle teoretických předpokladů, bylo by možné pro pohyb robota po oblouku použít stejný stejnou akci jako pro pohyb po přímce. Hodnota X by se nastavila na hodnotu $\frac{1}{r}$, což, pokud uvažujeme o přímce jako o kružnici s poloměrem $r = \infty$, platí ve všech případech. Regulovat pohyb robota po kružnici v současné simulaci je však

mnohem náročnější, než regulovat pohyb po přímce. Je-li hodnota X nulová, robot se pohybuje zcela rovně a PID-regulátor vyrovnává pouze nepřesnosti vzniklé předchozím pohybem. Naopak pohyb robota po kružnici teorii vůbec neodpovídá, viz následující tabulku. V prvním sloupci jsou hodnoty poloměrů kružnic, po kterých by se měl robot podle vzorců popsaných výše pohybovat. V dalších sloupcích jsou však naměřené hodnoty pro několik běhů aplikace a jejich rozptyl. Jak je vidět, tyto hodnoty se od teoretických velice liší.

Naměřené hodnoty poloměru a jejich rozptyl						
předp.	**1. běh**		**2. běh**		**3. běh**	
radius	poloměr	rozptyl	poloměr	rozptyl	poloměr	rozptyl
0,08	0,098	0,0002	0,105	0,0002	0,115	0,0019
0,10	0,131	0,0003	0,130	0,0006	0,133	0,0005
0,12	0,150	0,0005	0,154	0,0005	0,165	0,0028
0,15	0,196	0,0004	0,192	0,0003	0,195	0,0013

Tabulka 4.1: Pohyb robota po kružnici bez regulace



Obrázek 4.4: CTE při pohybu po kružnici

Popis PID-regulátoru i algoritmus jeho fungování je výše. Výpočet CTE pro pohyb po kružnici je však odlišný. Hodnota CTE je rozdíl vzdálenosti robota od středu kružnice, po které se robot pohybuje, a poloměru této kružnice, viz obrázek 4.4. Kompletní vzorec pro výpočet hodnoty X je pak definován rovnicí 4.15. Pokud se má robot pohybovat po oblouku ve směru

hodinových ručiček, je třeba hodnotu X násobit -1. Nakonec se celý pohybový vektor normalizuje podle rovnic 4.9 a 4.8

$$X_{temp} = \frac{1}{r} + c_p \cdot cte + c_d \cdot \frac{cte_t - cte_{t-1}}{\Delta t} + c_i \cdot \sum_{t=0}^{T} cte \tag{4.15}$$

PID-regulátor funguje pro regulaci pohybu po kružnici velice dobře. Autor provedl stejné měření pohybu robota po kružnici jako v předchozím případě, ale s použitím regulace. Výsledky jsou v zapsány v následující tabulce a již se dají s těmi teoretickými srovnávat.

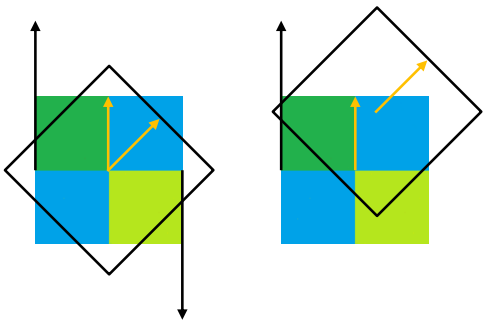
Naměřené hodnoty poloměru a jejich rozptyl						
předp.	**1. běh**		**2. běh**		**3. běh**	
radius	poloměr	rozptyl	poloměr	rozptyl	poloměr	rozptyl
0,08	0,087	0,0000	0,087	0,0000	0,088	0,0000
0,10	0,106	0,0000	0,107	0,0000	0,107	0,0000
0,12	0,127	0,0000	0,127	0,0000	0,129	0,0000
0,15	0,155	0,0000	0,155	0,0000	0,156	0,0000

Tabulka 4.2: Pohyb robota po kružnici s regulací

4.3 Rotace s reverzací

Rotace s reverzací je akce, při které se každé kolo robota pohybuje jiným směrem. Na obrázku 4.5 jsou znázorněny dva okrajové případy rotace s reverzací.

První obrázek ukazuje situaci, kdy se robot otáčí kolem vlastního středu. Y složka pohybového vektoru musí být 0 a obě kola se sice pohybují opačným směrem, ale stejnou rychlostí. Maximální rychlost každého kola, počítaná podle vzorečku na obrázku 3.3, je $|1|$. Maximální velikost X pro rozvor kol 75 mm (0,075 m) je 27, viz rovnice 4.16 a 4.17, ale pochopitelně může nabývat i nižších hodnot. Rotace s reverzací kolem středu robota se používá pro okamžité natočení robota do směru následného pohybu, více viz kapitolu 5.



Obrázek 4.5: Rotace s reverzací

$$l\frac{X}{2} = 1 \tag{4.16}$$

$$X = \frac{2}{l} = \frac{2}{0,075} = 26,666 \tag{4.17}$$

Na obrázku vpravo je druhý okrajový případ, kdy jedno kolo se nepohybuje a druhé ano. Tento případ již ve skutečnosti není rotace s reverzací, ale cokoli mezi tímto a otáčením robota kolem středu rotace s reverzací je. Rotaci robota tímto způsobem lze použít pro odehrání míče, poté co se k němu robot dostatečně přiblíží, viz kapitolu 7.1.1. Aby robot mohl rotovat s reverzací mimo vlastní střed, byla celá akce rozšířená o proměnnou *center*. Tato proměnná nabývá hodnot od v intervalu <-1,1> a určuje pozici středu rotace, kdy *center* = -1 znamená rotaci kolem levého kola, *center* = 0 rotaci kolem středu robota a *center* = 1 kolem pravého kola. Výpočet složek pohybového vektoru je znázorněn v rovnicích 4.18 až 4.20. Takto spočtený vektor je ještě třeba normalizovat podle rovnic 4.9 a 4.8

$$Y_{temp} = \frac{|center|}{1 + |center|} \tag{4.18}$$

$$X_{temp} = (1 - Y_{temp}) \cdot X_{max} \tag{4.19}$$

$$if \ \ (center < 0) \ \ Y_{temp} = -Y_{temp} \tag{4.20}$$

Rotace s reverzací je poměrně nepřesná záležitost i za ideálních podmínek. V reálném prostředí lze regulovat pohyb robota do správného natočení

velice obtížně a simulované fyzikální prostředí řídícího software je od ideálních podmínek ještě vzdálenější. Z toho důvodu se snažíme rotaci s reverzací v modulu elementární inteligence vyhýbat. Nicméně existují případy, kdy to nelze, a pro tyto případy je nezbytné akci rotace s reverzací odladit.

4.3.1 Lineární regrese

Lineární regrese je algoritmus strojového učení s učitelem. Cílem je nalézt funkci ve tvaru $w_0 + w_1x_1 + \ldots + w_nx_n$, která co nejlépe mapuje vektor příznaků $\vec{x}^T = \begin{bmatrix} x_0^{(i)} & x_1^{(i)} & \ldots & x_n^{(i)} \end{bmatrix}$ na výstup y. O použití lineární regrese autor uvažoval v úplných začátcích vývoje modulu elementární inteligence, kdy měla být rotace s reverzací použitá pouze pro rotaci kolem středu robota.

V případě reverzní rotace je nejdůležitějším příznakem rozdíl Φ mezi aktuálním rot_{act} a požadovaným natočením rot_{des} robota. Tento příznak nabývá hodnot v intervalu $<0, \frac{\pi}{2}>$. Je to z toho důvodu, že robot se může otáčet v obou směrech a má-li se otáčet doprava, je hodnota X záporná. Navíc, jak bude popsáno dále, se robot může pohybovat dopředu i dozadu, a proto stačí robota natočit tak, aby aktuální natočení robota odpovídalo jedné z rovnic 4.21 nebo 4.22.

$$rot_{act} = rot_{des} \tag{4.21}$$

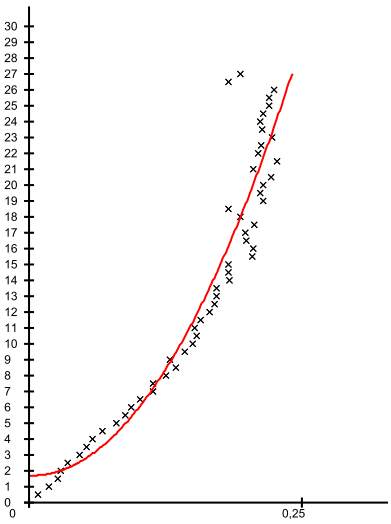
$$rot_{act} = rot_{des} + \frac{\pi}{2} \tag{4.22}$$

Výsledek lineární regrese y pak určuje hodnotu X složky pohybového vektoru, kterou se má robot pohybovat pro danou odchylku ϕ. Jak již bylo zmíněno výše, maximální hodnota X je 27. Pokud je výsledek y vyšší, je na tuto hranici zarovnán. Označení složek pohybového vektoru a vektorů lineární regrese může být matoucí, proto autor upozorňuje, že složky pohybového vektoru jsou psané kapitálkami.

Trénovací data a učení

Pro použití lineární regrese je nutné nejprve získat trénovací data. Hodnoty jsou získávány tak, že robot je rotován kolem vlastního středu s různými

hodnotami X. Ty se konkrétně pohybují od 0 do 27 s krokem 0,5 a při každé aktualizaci stavu hry na hřišti je spočteno, jakou úhlovou rychlostí se robot pohybuje. Průměrná hodnota úhlové rychlosti pro 50 – 150 iterací je výsledek y pro každou hodnotu X. Výsledky měření jsou zobrazeny na obrázku 4.6.



Obrázek 4.6: Trénovací data 1

Jak z obrázku vyplývá, hodnota X závisí spíše na druhé mocnině odchylky ϕ. Příznakový vektor $\vec{x}$ se rovná $\begin{bmatrix}1, & \phi, & \phi^2\end{bmatrix}$. Po získání příznakových vektorů je potřeba použít některý z trénovacích algoritmů pro získání hodnot $w = \begin{bmatrix}w_0, & w_1, & w_2,\end{bmatrix}$ tak, aby byly rozdíly mezi výstupem hypotézy $h(x)$ a naměřenými hodnotami y co nejmenší.

K natrénování lineární regrese se používají dva algoritmy – **gradientní sestup** a **normální rovnice**. Gradientní sestup je iterační metoda, která se používá především pro případy, kdy je velikost vektoru příznaků v řadu stovek hodnot a vyšších. Normální rovnice je naopak metoda analytická. Existuje několik způsobů, jak analyticky spočítat nejvýhodnější hodnoty vektoru w a autor použil **přeurčenou soustavu**. Výpočet koeficientů w probíhá podle rovnice 4.23.

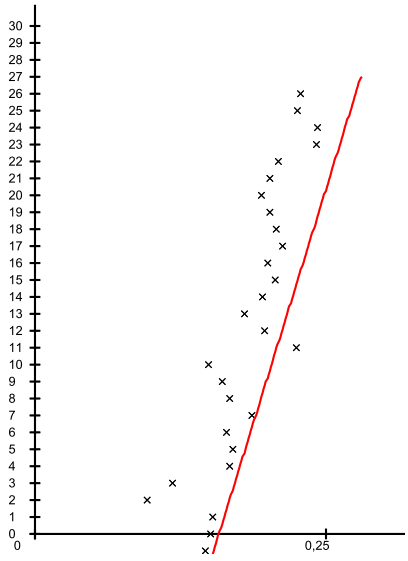
$$w = (X^T X)^{-1} X^T y \tag{4.23}$$

Výsledky

Tento koncept bohužel nefunguval podle představ autora, robot se ve většině případů přetočil a musel se otáčet zpět. Tento problém nastal nejspíše ze dvou důvodů. Prvním, méně důležitým, je určitá nestabilita fyzikálního enginu, kdy se úhlová rychlost robota v závislosti na pohybovém vektoru liší běh od běhu aplikace. Hlavním důvodem je významná odlišnost navrženého trénovacího modelu vůči reálnému nasazení akce. Robot se totiž většinou pohybuje maximální rychlostí a teprve, je-li odchylka dostatečně malá, je X složka pohybového vektoru menší, než maximální hodnota. Robot se tedy ve

skutečnosti pohybuje rychleji než při měření dat a úhel, o který se otočí při stejné hodnotě X vůči trénovacím datům, je větší.

Pro odstranění nedostatků předchozího modelu se trénovala nová data, ale postup měření byl odlišný. Robot se roztočil na maximální rychlost a teprve poté se měřila úhlová rychlost robota pro stejné hodnoty X jako v předchozím případě. Jak si lze prohlédnout na obrázku 4.7, naměřená data jsou zcela nevyhovující. Nejenom že odchylky jednotlivých trénovacích vzorků od proložené přímky by byly vysoké, ale opakovaná měření dávala pokaždé zcela jiný výsledek. Tentokrát je na vině především celková nestabilita řídícího softwaru, jehož fyzikální engine neodpovídá realitě. Navíc modul pro řízení robotů přibližně každou desátou zprávu s hodnotou pohybového vektoru pro následující smyčku přeslechne, takže robot se při měření nepohybuje podle nové hodnoty X, ale stále maximální rychlostí. Z těchto důvodů se autor rozhodl od použití lineární regrese v akci rotace s reverzací upustit.



Obrázek 4.7: Trénovací data 2

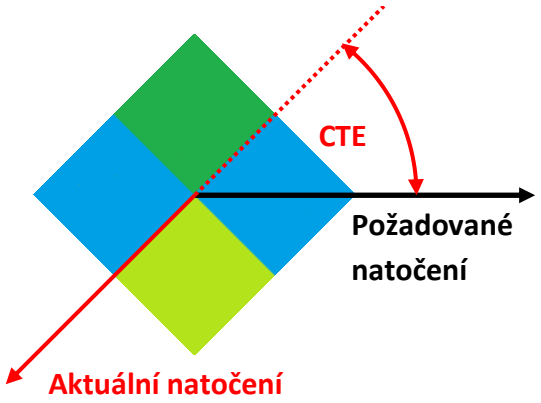
4.3.2 PID regulátor

Pro regulace reverzní rotace je nakonec použit PD-regulátor. Integrační složka je vynechána kvůli snaze o co největší zjednodušení natrénování takto nestabilní akce, jakou reverzní rotace bezesporu je. Vliv, jaký by mohla systémová chyba mít na cílové natočení robota pomocí reverzní rotace, by měl být nižší, než nastavená tolerance, která je v současnosti 5 stupňů. Pokud se v reálném prostředí podaří akci stabilně natrénovat, může být integrační složka regulátoru použita i pro potřeby rotace s reverzací.

Definice CTE pro regulaci reverzní rotace je odchylka aktuálního natočení robota od požadovaného, viz obrázek 4.8. Jak již bylo psáno výše, robot se může po natočení pohybovat směrem dopředu i dozadu. Natáčí se tedy

takovým směrem, aby počáteční odchylka natočení robota od požadovaného stavu byla co nejmenší. Na obrázku 4.8 je pro demonstraci znázorněn stav, kdy se po ukončení reverzní rotace začne robot pohybovat směrem vzad. Výhodou je, že robot se otáčí maximálně o úhel $\frac{\pi}{2}$.

Výstup PID-regulátoru je tedy hodnota X pohybového vektoru, pokud je vyšší než 27, je na tuto hranici zarovnán.



Obrázek 4.8: CTE pro rotaci s reverzací

Během tréninku akce reverzní rotace se objevil další problém s fyzikálním enginem simulace. Pokud se robot otáčí, úhlová rychlost, o kterou se otočí během jedné smyčky je diskretizovaná a z neznámého důvodu se vždy jedná o násobek čísla 0,0235 radiánu. V praxi to znamená, že pro nízké hodnoty X se někdy robot neotočí vůbec, někdy naopak o dvojnásobek výše zmíněného čísla. Pro vysoké hodnoty X se úhlová rychlost pohybuje vždy v násobcích, ale pouze průměr z několika posledních hodnot úhlových rychlostí dává smysluplný výsledek. Přehledně je to zobrazeno v tabulce 4.3

Pro derivační složku PD-regulátoru použitého pro regulaci reverzní rotace je jako změna CTE považován průměr z posledních pěti aktualizací stavu na hřišti. S přibývajícím počtem průměrovaných hodnot klesá určitá odezva této akce na změnu stavu a pět hodnot už odezvu snižuje nezanedbatelně. V kontrastu s tím použití posledních pěti hodnot rozdílů v natočení robota je naprosté minimum pro získání smysluplných hodnot úhlové rychlosti.

Použití reverzní rotace v reálném prostředí se sice bude potýkat s jinými problémy, ale alespoň se bude celý systém chovat realisticky. Současný simulovaný fyzikální engine je na hranici použitelnosti, ale všechny další akce již s ním přímo nepracují. Úspěšné natrénování základních akcí tak umožnilo

Naměřené hodnoty úhlové rychlosti						
číslo	hodnota X pohybového vektoru					
měření	1	5	7	15	18	25
1	0,024	0,094	0,071	0,259	0,259	0,306
2	0,024	0,094	0,141	0,189	0,212	0,189
3	0.000	0,071	0,094	0,212	0,189	0,329
4	0,024	0,094	0,094	0,189	0,212	0,141
5	0,024	0,071	0,117	0,212	0,235	0,282
průměr	0,019	0,0848	0,103	0,212	0,221	0,250

Tabulka 4.3: Naměřená úhlová rychlost robota

další pokračování ve vývoji modulu elementární inteligence.

5 Akce pohyb robota

Akce pro plánování pohybu robota tvoří rozhraní mezi pokročilými a základními akcemi, jak si lze povšimnout na obrázku 3.4. Plánování pohybu počítá optimální pohyb robota tak, aby se co nejrychleji dostal na určenou pozici.

Tato akce je implementována tak, aby tvořila jediný přístupový bod k základním akcím, a to jak pro modul herní strategie, tak pro pokročilé akce popsané dále. Je to z toho důvodu, že pokud by více akcí najednou zavolalo kteroukoli ze základních akcí, každá z nich by posílala pohybové vektory podle svých potřeb a výsledkem by byl zcela zmatený pohyb robota.

Vyšší akce mohou pomocí vstupních metod nastavit cílovou pozici, na kterou se má robot přesunout. Pro splnění příkazu může být nezbytné, aby robot přijel do cílové pozice z určitého směru, a proto akce pohybu robota nabízí možnost zadat kromě cílové pozice i natočení robota v cílové pozici.

Pro pohodlné použití je navíc implementována i možnost zadat více bodů, které má robot postupně projet. Pro každý takový bod je možné zadat i natočení robota v tomto bodě. Akce pohybu robota tak nabízí poměrně širokou funkcionalitu.

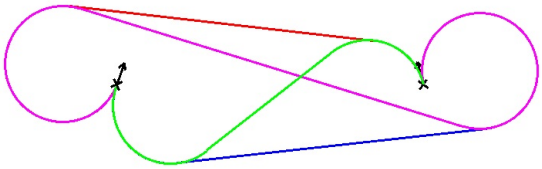
5.1 Plánování trasy

Trasa robota je plánována tak, aby se robot do cílové pozice dostal co nejrychleji. Cílem je najít ideální kompromis mezi nejkratší možnou cestou a plynulostí pohybu robota. Plánování trasy robota částečně vychází z autorovy bakalářské práce, kterou nadale rozšiřuje. Implementačně je však i tato akce zcela nová, neboť původní plánování pohybu bylo se současným systémem akcí zcela nekompatibilní a navíc úroveň původní implementace byla pro potřeby diplomové práce nedostatečná.

5.1.1 Trasa mezi dvěma body

Tvar trasy se velice liší podle toho, jestli má robot do cílové pozice přijet z určitého směru nebo jestli je důležité, aby se pouze na cílovou pozici přemístil.

První možnost se týká především případů, kdy je cílem pohybu robota udeřit do míče tak, aby se míč pohyboval zamýšleným směrem. Pouhé přesunutí na pozici může být využito k obraným účelům, kdy je cílem robota blokovat nahrávku soupeře, apod.



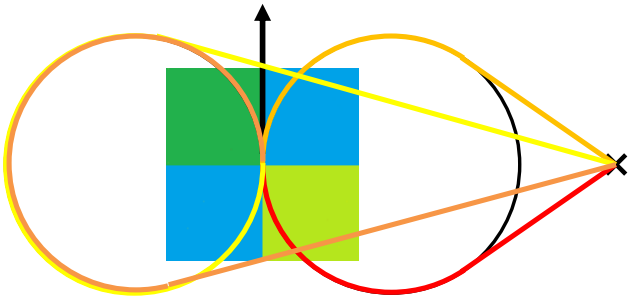
Obrázek 5.1: Čtyři S-křivky mezi počátečním a cílovým stavem

Robot se pohybuje mezi dvěma body po tzv. S-křivkách, viz obrázek 5.1. Největší výhodou pohybu po S-křivkách je, že robot může být k cílovému bodu jakkoli natočen a požadované natočení robota v cílovém bodě může být také jakékoliv, a přesto se robot pohybuje, aniž by musel použít rotaci s reverzací. Není-li zadán požadavek na natočení robota v cílové pozici, S-křivka může postrádat závěrečný oblouk a její výpočet je jednodušší. Detailní výpočet S-křivky je popsán v autorově bakalářské práci, zde budou popsána především rozšíření diplomové práce oproti bakalářské.

Výpočet dráhy bez cílového natočení

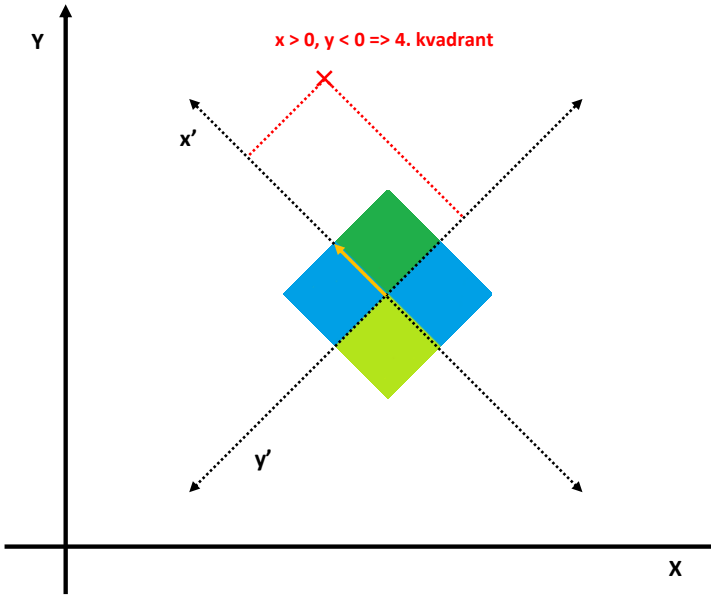
Nejprve se zaměřme na popis plánování trasy pro jednodušší případ, kdy není požadováno speciální natočení robota v cílovém bodě. Dráha robota se skládá z jednoho oblouku a poté tečny ke kružnici, na které oblouk leží, procházející cílovým bodem. Jelikož přímá část pohybu robota je tečnou k projížděnému oblouku, je tento pohyb plynulý. Počítáme-li s tím, že se robot může pohybovat dopředu i dozadu, existují mezi počáteční a cílovou pozicí čtyři různé trasy, viz obrázek 5.2.

Výběr správné trasy lze provést algoritmicky jednodušším způsobem nebo výpočetně jednodušším způsobem. První je výpočet všech čtyř drah a výběr té nejkratší z nich. Druhý způsob, který je při plánování trasy robota nakonec použit, je vypočtení vzájemné pozice počátečního a cílového bodu. V aplikaci se nejdříve vytvoří nový souřadný systém, který má počátek na aktuální pozici robota a osu X rovnoběžnou s jeho natočením. V tomto novém souřadnicovém systému určíme, ve kterém kvadrantu se nachází cílový bod, viz



Obrázek 5.2: Čtyři trajektorie mezi počátečním a cílovým stavem

obrázek 5.3. Výpočet probíhá tak, že pro cílový bod nalezneme kolmý průmět do osy x i do osy y v novém souřadnicovém systému. Pro tyto průměty je důležité pouze zda se nacházejí v kladné nebo záporné části osy. Další určení kvadrantu probíhá podle standardních pravidel, čili je-li hodnota jak x, tak y kladná, je cílový bod v prvním kvadrantu, apod.



Obrázek 5.3: Výpočet kvadrantu cílového bodu

Nachází-li se robot v prvním nebo ve čtvrtém kvadrantu, pak se pohybuje

dopředu, v opačném případě dozadu. Tím zbývají pouze dvě možné trajektorie pohybu robota a z nich lze podobně podle hodnoty kvadrantu vybrat tu správnou. Akce plánování pohybu vybranou cestu vypočte. V aplikaci jsou přímo implementovány geometrické třídy typu *Arc*, *Circle* nebo *Line*, které definují příslušné geometrické křivky. Navíc tyto třídy obsahují velké množství metod, které veškeré analytické výpočty s těmito křivkami zjednodušují.

Poloměr obloukové části trajektorie se může pohybovat pouze v poměrně úzkém rozmezí. Je-li menší než rozvor kol robota, robot už se nepohybuje po kružnici, ale pomocí rotace s reverzací, které se snažíme vyhnout. Je-li naopak poloměr oblouku příliš velký, délka trasy se úměrně prodlužuje, aniž by to mělo vliv na plynulost pohybu robota. Optimální poloměr obloukových částí je konfigurovatelný (viz dále) a výchozí hodnota je na základě úsudku autora nastavena na 0,15 m. Po přechodu do reálného prostření se provedou měření, která optimální poloměr určí přesněji.

Je-li potřeba naplánovat trasu pro příliš malou vzdálenost mezi počátečním a cílovým bodem, reverzní rotaci se nevyhneme. Teoreticky je ošetření takovýchto případů popsáno již v bakalářské práci, implementačně se však tímto problémem autor zabýval až při vývoji modulu elementární inteligence v diplomové práci.

Výpočet dráhy s cílovým natočením

Výpočet trajektorie tak, aby se robot dostal do cílové pozice v určitém směru, je mnohem náročnější. Za prvé je nutné vypočítat úvodní i cílovou obloukovou část a ještě najít jejich společnou tečnu. Není třeba dodávat, že výpočet společné tečny dvou kružnic je náročnější, než výpočet tečny ke kružnici z určitého bodu.

Mezi počátečním a cílovým bodem existuje celkem osm různých trajektorií. Na obrázku 5.1 jsou sice zobrazeny jenom čtyři, pokud se robot může pohybovat i dozadu, je třeba celkový počet možných trajektorií zdvojnásobit. Z těchto drah lze poměrně jednoduše určit čtyři, které pro pohyb robota nebudou použity. Stejně jako v předchozím případě vypočteme vzájemnou polohu počátečního a cílového pohybu robota.

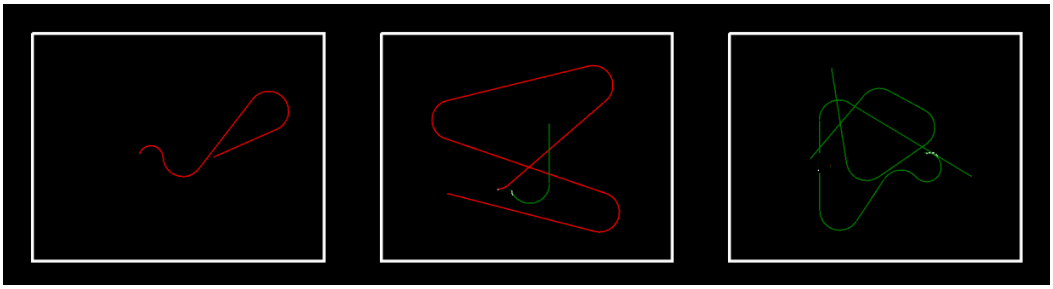
Je-li cílový bod v prvním nebo čtvrtém kvadrantu, robot se pohybuje dopředu, jinak dozadu. Dále už zúžit výběr možných trajektorií použitím kvadratů nelze, neboť existují singulární případy, kdy tento způsob selekce

selhává. Jedná se o případy, kdy se cílový bod nachází blízko hranice prvního a druhého nebo třetího a čtvrtého kvadrantu a vybraná dráha může být mnohem delší, než ta optimální. Pro výběr správné trajektorie je tedy nezbytné vypočítat všechny čtyři možné zbývající trasy a z nich vybrat tu nejkratší.

Stejně jako v předchozím případě je potřeba ošetřit výpočet dráhy pro málo vzdálený cílový bod. Robot se opět musí pohybovat s reverzní rotací. Teoreticky je ošetření těchto případů opět podrobně popsáno v autorově bakalářské práci. Přestože se autor zabýval velice intenzivně vyladěním aplikace pro pohyb mezi takto blízkými body, stále se ještě najdou případy, kdy se robot začne pohybovat zmateně. Techto případů již není příliš mnoho.

5.1.2 Plánování trasy s mezibody

Modul elementární inteligence je rozšířen o možnost zadat více než jeden bod hřiště, kterými má robot projet. Trasa je poté plánována tak, aby robot projel všemi určenými body podle pořadí, v kterém byli zadány. Robot při průjezdu trasy nezastavuje, pohybuje se plynule, dokud se nedostane na poslední požadovanou pozici. Toto rozšíření však obnáší některé změny v plánovacím algoritmu a hlavně změny v průjezdu trasy. Plánovaná trasa může nabývat mnoha různých podob, některé příklady jsou zobrazeny na obrázku 5.4. Pochopitelně pro každý ze zadaných bodů může být zadáno i natočení, které má v tomto bodě robot mít, a akce pohybu robota je schopná takovou trasu vypočítat.



Obrázek 5.4: Příklady plánovaných trajektorií

Robot tedy při průjezdu na mezipozicích nezastavuje a tomu je přizpůsoben i plánovací algoritmus. Robot se pohybuje stále stejným směrem (do-

předu nebo dozadu) od počátečního do cílového bodu. Tento směr je určen stejným postupem popsaným výše, ale pouze pro první ze zadaných bodů. Pro plánování trasy je tedy nezbytné uchovávat předpokládaný směr robota v každém ze zadaných bodů.

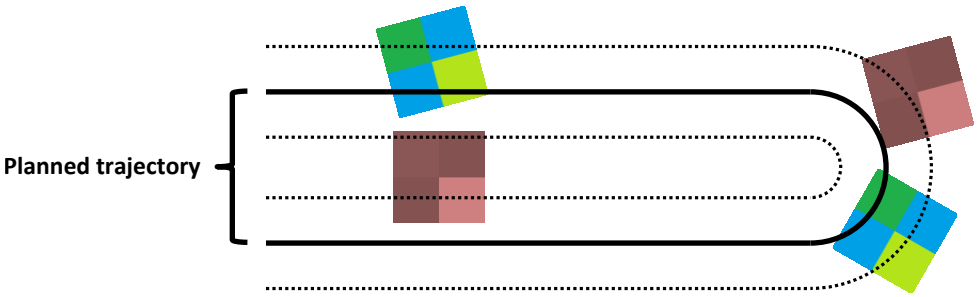
Možnost zadat více bodů využívají především pokročilé akce pro práci s míčem. První bod je zadán tak, aby se robot přiblížil k míči ze správného směru, a druhý pak již koliduje s pozicí míče. Tím je zajištěno, že robot narazí do míče rovně a ve vysoké rychlosti. Jak bude popsáno dále, vyhýbání se překážkám není zakomponováno do plánování trasy, ale až do jejího průjezdu. Důvody, které k tomuto rozdělení autora vedly, jsou popsány v příslušné kapitole, ale pokud má tvůrce herní strategie na stejnou problematiku jiný názor, může využít zadání mezibodů k zakomponování vyhýbaní se překážkám už do fáze plánování.

5.2 Průjezd plánované trasy

Jednotlivé části naplánované trasy se uloží do seznamu struktur `PathParts`. V této struktuře jsou uloženy informace vždy o jednom segmentu plánované trasy, buď'to úsečky nebo oblouku, včetně jejich okrajových bodů, apod. Po spočtení trasy robota z počátečního bodu do cílového je pro průjezd každé části předána kontrola příslušné základní akci. Pro průjezd přímých částí je to akce **pohyb po přímce**, pro obloukové části je použita akce **pohyb po kružnici**, je-li poloměr oblouku větší než rozvor kol, nebo **reverzní rotace** v opačném případě.

Předání kontroly akci probíhá tak, že příslušná základní akce se aktivuje a začne způsobem popsaným v příslušných kapitolách počítat pohybový vektor robota. Akce plánování pohybu je během činnosti základní akce pozastavena. Tento postup se opakuje pro všechny části plánované trasy. Po skončení akce pohybu robota je tak robot na požadované pozici.

Robot se po plánované trase pochopitelně nepohybuje zcela přesně, i když u přímých částí funguje regulace pomocí PID-regulátoru velice dobře. Je však třeba stanovit hranice, kdy už je vzdálenost robota od plánované trasy příliš vysoká. Tato vzdálenost je konfigurovatelná a její velikost může být na začátku a na konci každé části plánované trasy jiná. Pokud se robot od plánované trasy vzdálí příliš, je třeba robotovu trasu do cílového bodu přepočítat, viz obrázek 5.5. Nejkritičtější částí jsou přechody z obloukových částí, robot



Obrázek 5.5: Tolerance pohybu robota po trase

se mimo tolerované hranice dostane nejčastěji v těchto místech. Při dalším vývoji elementární inteligence by se vyplatilo výpočet průjezdu přechodů optimalizovat.

5.2.1 Přepočtení trasy

K přepočítání trasy může dojít se dvou důvodů. Prvním je, že se cílová pozice, do které se má robot přesunout, změní a druhým je, že se robot při svém pohybu od plánované trasy příliš vzdálí. V obou případech probíhá přepočet stejně. Akce plánování pohybu převezme zpět kontrolu od základní akce, která se stará o průjezd aktuální části plánované trasy. Poté vypočítá trasu pro požadovaný cílový bod z aktuální pozice robota na hřišti a opět pohyb robota zahájí. Celý výpočet je natolik rychlý, že pohyb robota je plynulý. Jediným rozdílem mezi zmíněnými důvody přepočtu dráhy je, že pokud cílový robot překročí hranice tolerance, počítá novou trasu do stále stejného bodu.

Algoritmus průjezdu trasy pro více než jeden cílový bod je prakticky stejný, pro více bodů existuje pouze více segmentů plánované trasy. Rozdíl nastává v okamžiku, kdy se robot od plánované trasy příliš vzdálí a je nutno trajektorii přepočítat. Nová trasa robota se počítá pouze pro body, kterými robot zatím ještě neprojel. Tento rozdíl je řešen tak, že zadané body jsou uložené v poli podle pořadí a jak robot projíždí plánovanou trasu, tyto body se postupně smazávají. Je-li nutno dráhu přepočítat, v paměti jsou pouze neprojeté pozice a pro ty je dráha vypočtena.

5.3 Synchronizace

Modul elementární inteligence pracuje v mnoha vláknech a jelikož akce plánování pohybu má prakticky exklusivní přístup k základním akcím, může se stát, že se v této akci sejde více požadavků na pohyb robota. Typickým případem je, že se přepočítává trasa, protože se robot příliš vzdálil od původní trasy, a zároveň je poslán požadavek na vyhnutí se jinému robotovi, aby nedošlo ke kolizi. Z toho důvodu je nezbytné vytvořit synchronizační mechanismy a určit prioritu jednotlivých akcí.

Priority jednotlivých požadavků nejsou zatím v této akci příliš řešeny, robot se pohybuje podle posledního požadavku. Dokud nebude modul elementární inteligence vyzkoušen v zápasovém režimu a ideálně v reálném prostředí, nelze určit jak velké problémy může absence priorit způsobit a zda je nutné akci pohybu robota o tuto funkcionalitu rozšířit. V případě nutnosti se tím autor bude zabývat při dalším vývoji modulu.

Požadavek na vstupu musí přepsat body dráhy robota a podle aktuálního stavu akce plánování buď převzít kontrolu na robotem od základní akce nebo akci plánování aktivovat. Aby bylo zajištěno, že body trasy robota odpovídají poslednímu požadavku, jsou všechny příchozí požadavky kromě prvního zastaveny semaforem.

Zatímco je počítána dráha, může přijít jiný požadavek na pohyb robota. Takto vypočítaná dráha už není aktuální, ale robot se po ní přesto začne pohybovat. Pro tyto singulární případy není výše popsaný semafor dostatečným synchronizačním prostředkem. Jako řešení je po vypočtení dráhy zkontrolován vlajkový příznak, zda nepřišel nový požadavek. Pokud ano, je dráha propočtena znova, pokud ne, začne se robot pohybovat.

6 Akce vyhýbání se překážkám

Jedná se o samostatnou kategorii akcí, které stojí stranou celého procesu postupného zjednodušování příkazu od herní strategie přes pokročilé akce, základní akce až po poslání hodnot konkrétních rychlostí motorům robota. Toto odloučení od ostatních akcí je patrné už z obrázku 3.4.

Tyto akce slouží k minimalizaci kontaktu robota s překážkami, kterými mohou být jak stěny hřiště, tak všichni ostatní hráči na hřišti. Každá kolize nejenže přináší riziko poškození, ale také pohyb robota velmi zpomalí. Kolize robotů tedy nejsou žádoucí, i když v rámci souboji o míč, apod. se jim stejně nelze vyhnout.

Robot se tedy vyhýbá jak stěnám hřiště, tak ostatním robotům. Algoritmicky se vyhýbání těmto dvěma druhům překážek liší natolik, že je nelze zkombinovat v jedné akci. Principiální rozdíly jsou podrobně popsány v jednotlivých podkapitolách.

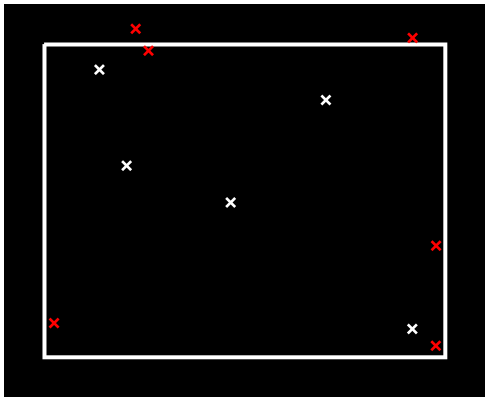
6.1 Vyhýbání se stěnám

Vyhýbání se stěnám, na rozdíl od všech dříve i později zmíněných akcí, neběží ve vlastním vlákně. Jedná se vlastně jen o skupinu metod, využívanou akcí plánování pohybu, kterou bylo z hlediska přehlednosti výhodné umístit do samostatné třídy.

Cílem této akce je omezit jakýkoliv kontakt se stěnami hřiště. Ty na rozdíl od robotů nemění svoji pozici, a proto jsou tyto algoritmy použity už ve fázi plánování. Odstranění kolizí probíhá ve dvou krocích. Prvním z nich je kontrola, zda body, kterými je průjezd robota požadován, leží všechny mezi stěnami hřiště, viz obrázek 6.1. Na tomto obrázku jsou všechny body, které leží mimo hranice označeny červeně. Poněkud matoucí se může zdát, že jsou červeně označeny i body, které se mezi stěnami nacházejí, ale to je z toho důvodu, že křížky označují střed robota a ten se musí nacházet v určité vzdálenosti od stěn, aby nedošlo ke kolizi.

Rozlišujeme tedy dvoje stěny hřiště, jedny fyzické, které představují skutečné stěny hřiště a na obrázku 6.1 jsou vyznačené silnými bílými čarami. Logické stěny jsou od fyzických odsazeny směrem ke středu hřiště o vzdále-

nost minimálně $\frac{l}{2}$, kde l je rozvor kol robota. Tyto logické stěny hřiště jsou používány pro veškeré dále popsané výpočty detekce kolizí, apod. V celé kapitole, pokud není výslovně uvedeno jinak, se za stěny hřiště považují právě logické stěny hřiště.



Obrázek 6.1: Body mimo a uvnitř hřiště

Body ležící mimo hřiště je potřeba nahradit. Pochopitelně by bylo možné je z plánování trasy zcela vynechat, ale to autor nepovažuje za dobré řešení. Tyto body lze nahradit dvěma způsoby.

- První způsob je algoritmicky složitější. Trasa je do bodu ležícího mimo hřiště plánována stejně jako kdyby mimo neležel. V okamžiku, kdy se robot přiblíží ke stěně hřiště, odchýlí se od plánované trasy a pohybuje se těsně vedle stěny, dokud se na plánovanou trasu nevrátí, viz obrázek 6.2a.

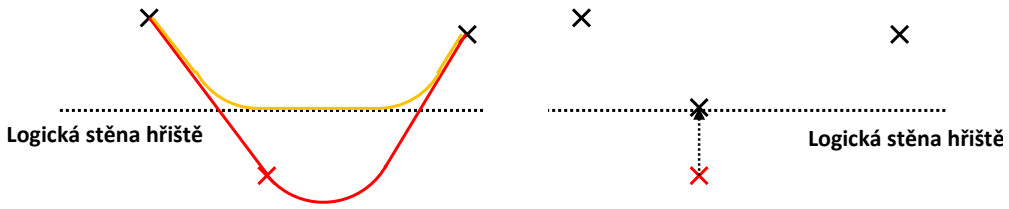
  Pro použití tohoto způsobu nahrazování bodů je potřeba ošetřit velké množství singulárních případů. Například pokud se více než jeden bod (případně všechny) za sebou nacházejí mimo hranice hřiště. Algoritmická náročnost takového nahrazování mimo ležících bodů je příliš vysoká, a proto jsou body nahrazovány jinak.

- Druhý způsob je algoritmicky velice jednoduchý. Bod je nahrazen kolmým průmětem do nejbližší stěny hřiště, viz obrázek 6.2b. Ve skutečnosti to pouze znamená, že jsou nastaveny maximální hodnoty X a Y, na kterých se může robot nacházet, viz rovnice 6.1 a 6.2. Je-li absolutní hodnota některého z koordinátů zadaného bodu vyšší, je na tuto hranici zarovnána. Střed hřiště má pozici $S = [0, 0]$, proto je ve vzorcích vždy

pouze polovina šířky nebo délky hřiště. Poslední člen *tol* znamená, že hranice je odsazená od stěn hřiště o víc, než o půlku rozvoru kol robota, aby se kompenzovaly nepřesnosti pohybu robota a zvýšila se úspěšnost vyhýbání se stěnám. Tato tolerance *tol* je konfigurovatelná a výchozí hodnota je nastavená zhruba na 1 cm.

$$X_{max} = \frac{delkaHriste}{2} - \frac{l}{2} - tol \tag{6.1}$$

$$Y_{max} = \frac{sirkaHriste}{2} - \frac{l}{2} - tol \tag{6.2}$$



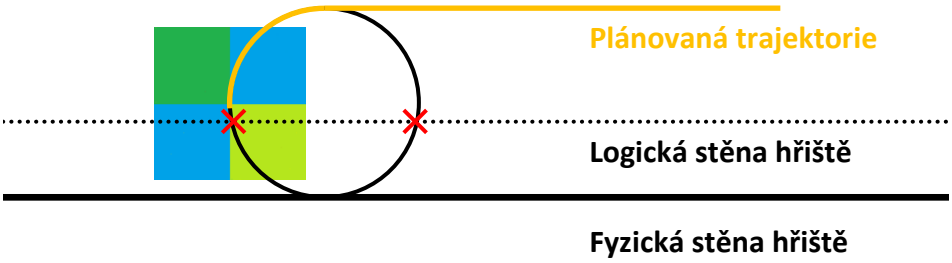
(a) Nahrazování bodů 1 (b) Nahrazování bodů 2

Obrázek 6.2: Nahrazování bodů mimo hřiště

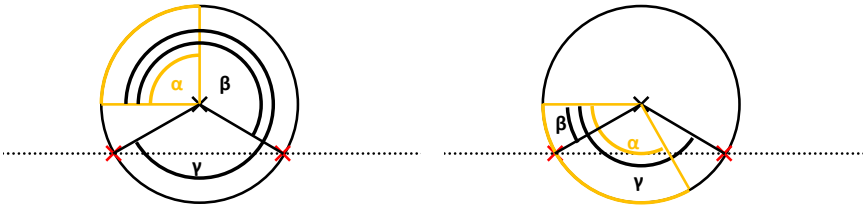
Krokem číslo dva je kontrola všech obloukových částí, zda se se stěnou hřiště nekříží. Přímé části není třeba kontrolovat, neboť tím, že se všechny body, kterými má robot projet, nacházejí uvnitř hřiště, je zajištěno, že kříží-li se přímá část pohybu robota se stěnou, kříží se i některá z obloukových.

Výpočet probíhá tak, že se hledají průsečíky kružnice, která tvoří základ obloukové části se stěnou hřiště. Pokud žádný takový bod neexistuje, pak ke kolizi se stěnou nemůže dojít. V opačném případě je pouze možnost, nikoli jistota, že by při průjezdu trasy došlo ke kolizi robota se stěnou. Na obrázku 6.3 je znázorněn jeden z případů, kdy se kružnice se stěnou kříží, ale v místech, která je mimo obloukovou část plánované trasy.

Abychom zjistili, zda je nutné řešit vyhýbání se stěně hřiště, musíme spočítat úhly α, β, γ. Přehledně je význam všech tří úhlů zobrazen na obrázcích 6.4a a 6.4b. Ke kolizi se stěnou dochází pouze v případě, je-li některý z úhlů β nebo γ menší než úhel α. Tyto úhly je nezbytné počítat ve směru pohybu po oblouku, takže všechny mohou nabývat hodnot $<0, 2\pi>$.



Obrázek 6.3: Výhýbání se stěnám není potřeba



(a) Bez kolize se stěnou (b) Kolize se stěnou

Obrázek 6.4: Význam úhlů α, β, γ

V případě, kdy ke kolizím dochází, je potřeba dráhu robota upravit. Prvním krokem je výpočet maximálního poloměru oblouku, po kterém se robot může pohybovat ve stejném směru, aniž by do stěny hřiště narazil. Je-li tento poloměr větší, než rozvor kol robota, pak se směr pohybu robota nemění, ale změní se velikost poloměru oblouku a přepočte se i zbytek plánované trasy. Je-li naopak radius oblouku menší, než rozvor kol robota, další výpočet závisí na tom, jestli se jedná o úvodní nebo o cílový oblouk. V prvním případě se robot zastaví a začne se pohybovat po původní kružnici, ale opačným směrem. Názorně je podobná situace znázorněna na obrázcích 6.4a a 6.4b, kdy při pohybu po stejné kružnici opačným směrem ke kolizi se stěnou nedochází. Dochází-li ke kolizi se stěnou při průjezdu cílového oblouku, změnit směr pohybu robota nelze a robot se musí pohybovat s pomocí reverzní rotace po oblouku o poloměru menším, než je rozvor kol.

Na závěr je nutno podotknout, že v současné implementaci vyhýbání se stěnám není počítáno s prostorem branek, kde by se robot mohl teoreticky

také pohybovat, například při pokusu dotlačit míč do branky přes soupeřova brankáře. Toto rozšíření podstatně zvyšuje náročnost detekce kolizí se stěnami hřiště, a proto bude implementováno až při dalším vývoji elementární inteligence.

6.2 Vyhýbání se robotům

Vyhýbání se ostatním robotům je daleko obtížnější úloha, než vyhýbaní se stěnám. Stěny nemění svoji pozici, proto stačí vhodný plánovací algoritmus a pokud nedojde k nějakému nečekanému vnějšímu vlivu, robot by do stěny narazit neměl. U vyhýbání se robotům je situace zcela opačná. Jejich pozice se může kdykoli měnit a predikovat jejich pohyb je extrémně náročná úloha, která není součástí modulu elementární inteligence. Z toho důvodu se autor rozhodl zakomponovat vyhýbání se překážkám jako samotnou akci, která má plnit funkci jakéhosi reaktivního chování robota. To znamená, že tato akce nezasahuje do pohybu robota, dokud nehrozí nebezpečí kolize. Pro plánování trasy tak pozice ostatních robotů není důležitá.

6.2.1 Detekce kolizí

Detekce kolizí je poměrně důležitá součást vyhýbání se překážkám, ale její důležitost je spjata především s tréninkem vyhýbání se překážkám. Na ostrý provoz modulu už prakticky žádný vliv nemá.

Účel detekce kolizí je dvojí. Za prvé slouží autorovi pro ladění úhybných manévrů, které jsou popsány dále. Pokud je úhybný manévr špatně navržen, robot se kolizi i při jeho provedení nevyhne. Odezva systému skrze detekci kolizí je tak nezbytná pro případné úpravy.

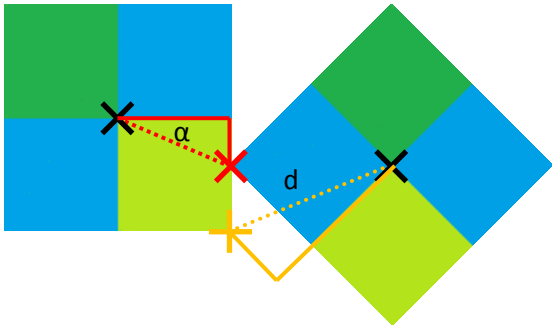
Druhým důvodem detekce kolizí je odezva systému kvůli trénování neuronové sítě, sloužící jako klasifikátor pro výběr úhybného manévru. Bližší podrobnosti tréninku jsou popsány dále, v této kapitole je popis pouze samotné detekce kolizí.

Kontrola, zda nedošlo ke kolizi, probíhá zvlášť pro každou překážku. Pro výpočet stačí znát polohu a natočení obou robotů. Jsou-li pozice obou robotů vzdálenější než je maximální možná vzdálenost d_{max} podle vzorce 6.3, pak ke

kolizi dojít nemůže. Kolize robotů, v případě, že jsou od sebe ve vzdálenosti d_{max}, znamená, že se dotýkají svými rohy.

$$d_{max} = \sqrt{2} \cdot l \tag{6.3}$$

Na obrázku 6.5 je zobrazena kolize dvou robotů. Pokud do sebe dva roboti narazí, vždycky se alespoň jeden robot musí toho druhého dotýkat rohem. Tento roh a zároveň kolizní bod je na obrázku znázorněn červeným křížkem. Algoritmus určení detekce kolizí probíhá tak, že pokud je vzdálenost obou robotů menší než d_{max}, vypočteme pro každého robota pozici rohu, který je k tomu druhému nejblíže. Na obrázku je jeden označen jako kolizní bod a druhý žlutým křížkem.



Obrázek 6.5: Kolize robotů

Dále je pro detekci kolizí nezbytné určit úhel α mezi spojnicí středu a nejbližšího rohu druhého robota (na obrázku značena přerušovanou čárou) a tzv. projekční přímkou (označeny plnými čarami). Projekční přímka je rovnoběžná nebo kolmá k aktuálnímu natočení robota. Názorně to ukazuje rovnice 6.4, kde rot je aktuální natočení robota a rot_{proj} je směrnice projekční přímky tak, aby úhel mezi projekční přímkou a spojnicí středů obou robotů byl co nejmenší. Je-li splněna rovnice 6.5 pro kterýkoliv z označených rohů, pak došlo ke kolizi. Značka d označuje délku spojnice středu s příslušným rohem druhého robota a l rozvor kol.

$$rot_{proj} = rot + k \cdot \frac{\pi}{2}; k = 0, 1, 2, 3 \tag{6.4}$$

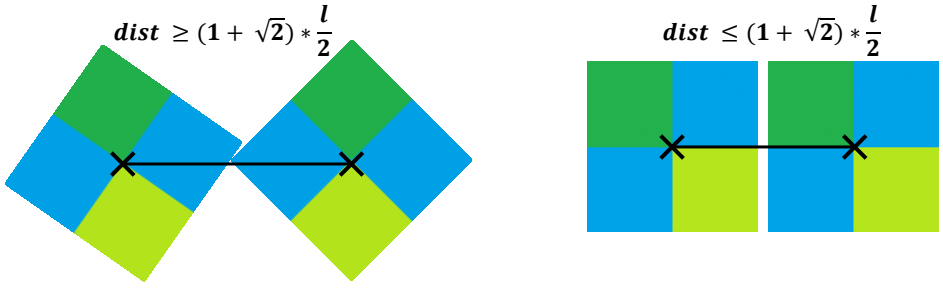
$$l \geq d \cdot cos(\alpha) \tag{6.5}$$

Tento algoritmus je poměrně výpočetně složitý vzhledem k tomu, že probíhá při každém aktualizování stavu hry pro všechny dvojice robotů. Navíc pokud je náhodou kolize detekována špatně, kolaps systému nehrozí. Z toho důvodu se autor rozhodl celý algoritmus detekce kolizí zjednodušit a za kolizi označit stav, kdy je splněna rovnice 6.6, kde *dist* je vzdálenost středů obou robotů.

$$dist \leq (1 + \sqrt{2}) \cdot \frac{l}{2} \tag{6.6}$$

Zjednodušení algoritmu s sebou přináší riziko chybné detekce kolizí. Na obrázku 6.6a je zobrazena situace, kdy roboti nabourali, ale kolize detekována není. V takovýchto případech se většinou roboti vlivem srážky protočí a dostanou se do pozice, kdy už je kolize detekována správně.

V druhém případě naopak algoritmus označí situaci jako srážku dvou robotů, i když ke kolizi zatím nedošlo. Pokud se však roboti nacházejí v takové blízkosti, je velice pravděpodobné, že se srazí o pár okamžiků později.



(a) Falešně negativní (b) Falešně pozitivní

Obrázek 6.6: Singulární případy

6.2.2 Úhybné manévry

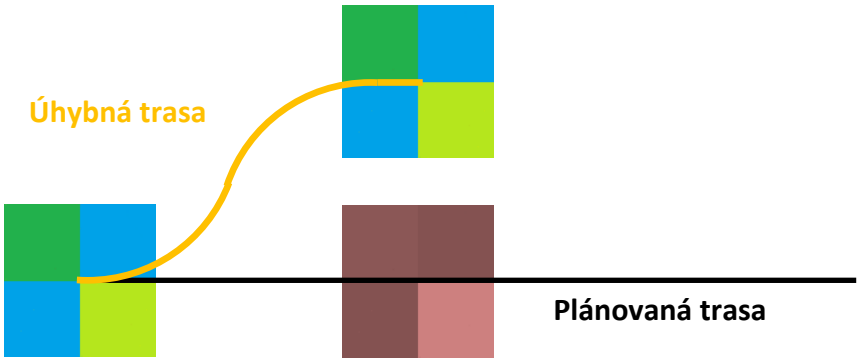
Úhybný manévr je prostředek k zabránění kolize, pokud je detekováno hrozící nebezpečí. Výběru úhybného manévru se věnuje následující podkapitola, zde jsou popsány samotné úhybné manévry.

Tyto manévry jsou velice jednoduché a jsou pouze čtyři. Jedná se o zastavení, pokračování v pohybu, úhyb vlevo a úhyb vpravo.

- **Zastavení** je manévr, kdy robot zůstane stát na místě. Zdržení v pohybu, které způsobí zpomalování, stání a rozjíždění je obrovské, a proto není tento manévr příliš žádaný. Používá se především jako poslední záchrana před kolizí nebo pokud se robot nachází blízko cílové pozice. Pokud se k sobě přiblíží dva roboti a oba použijí tento manévr, zůstanou stát, dokud do nich nějaký jiný robot nevrazí. Z toho důvodu je po zastavení spuštěn časovač, a pokud robot po vypršení času stále stojí, je donucen se rozjet.

- **Pokračování v pohybu** ruší všechny ostatní manévry. Jedná se o manévr, který je robotům posílán ve většině situací, a plánovaná trasa ani pohyb robota se nemění.

- **Úhyb vlevo** používá robot, aby se vyhnul překážce a přitom stále pokračoval v pohybu. Dráha manévru se skládá ze dvou obloukových částí a jedné rovné tak, jak je to vyobrazeno na obrázku 6.7. Velikost obou oblouků by měla být dostatečná, aby se robot mohl vyhnout stojící překážce.



Obrázek 6.7: Úhyb vlevo

  Vyhýbání se stojící překážce považuje autor za nejhorší případ, protože pokud se druhý robot pohybuje, lze vybrat úhybný manévr doleva nebo doprava tak, aby směřoval proti jeho pohybu.

- **Úhyb vpravo** je shodný s předchozím manévrem, pouze robot uhýbá na opačnou stranu.

Úhybné manévry nejsou příliš sofistikované, ale v rámci simulace fungují relativně dobře. Tyto manévry nesmí příliš narušit pohyb robota. Problémem,

který autor řešil, bylo propojení akce pro uhýbání robotům s akcí pohybu robota. Prvním řešením bylo přidat na začátek pole požadovaných pozic nové mezibody, kterými má robot projet. Tento návrh nebyl příliš úspěšný. Pokud se robot příliš vzdálil od plánované trasy a vypočítala se nová, mohly nové mezibody nacházející se v přílišné blízkosti počáteční pozice robota způsobit jeho zmatený pohyb.

Tento problém je vyřešen tak, že akci pohybu robota jsou poslány přesné segmenty dráhy, které má projet. Pro tyto segmenty je však nezbytné zakázat přepočítávání dráhy, pokud se robot od trasy příliš oddálí, neboť trasa by se přepočítala pro původní cílové pozice a úhybný manévr by byl vynechán.

6.2.3 Výběr úhybného manévru

Výběr úhybného manévru je nejdůležitější částí celé akce. Pro výběr manévru byla použita neuronová síť typu **vícevrstvý perceptron** se třemi vrstvami, trénovaná algoritmem **Q-Learning** posilovaného učení (reinforcement learning). Neuronová síť v tomto systému slouží k reprezentaci stavu, kdy pokud jsou podobné situace na hřišti, by měl být podobný i výstup perceptronu. Algoritmus Q-learning slouží ke generování „správných“ výsledků y, které jsou použity pro natrénování vah perceptronu. Výběr algoritmu Q-Learning byl čistě dočasný, cílem autora bylo porovnat rychlost trénování pro více metod reinforcement learningu. Samotné natrénování nebylo příliš úspěšné (viz dále), proto se zde zabýváme pouze použitím této techniky strojového učení.

Jak bylo psáno v sekci 2.3, cílem algoritmů posilovaného učení je získání co největší budoucí odměny (**reward**). V názvosloví strojového učení se vybírá za stavu s hry na hřišti akce a robota. Autor se v této kapitole bude tímto názvoslovím řídit, ale akce a v tomto případě znamená jeden z manévrů popsaných výše a nikoli akci modulu elementární inteligence.

Pro každou akci pomocí vícevrstvého perceptronu spočteme předpokládanou budoucí odměnu $Q(s_t, a_t)$ a vybereme tu akci, pro níž je hodnota nejvyšší. Velikost hodnoty $Q(s_t, a_t)$ by měla odpovídat vzoru 6.7. Pokud se od této chyby liší, velikost odchylky je označována za chybu e, která se použije pro úpravu vah perceptronu.

$$Q(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \tag{6.7}$$

Hodnota $Q(s_{t+1}, a_{t+1})$ je nejvyšší hodnota předpokládaných odměn v následujícím stavu hry na hřišti. Pokud je však pro robota detekována kolize, je tato hodnota nastavena konstantně na -10. Postupným upravováním vah perceptronu by hodnoty $Q(s_{t+1}, a_{t+1})$ nekolizních stavů měly začít odpovídat realitě, tedy pokud se robot nachází ve stavu těsně před kolizí, měla by hodnota $Q(s_{t+1}, a_{t+1})$ být blízká -10. Hodnota r_{t+1} označuje odměnu nebo trest za použití určité akce. Tímto způsobem lze dát učícímu algoritmu najevo, že některé akce jsou preferovanější než jiné. V případě vyhýbání se překážkám je preferovaná akce pro pokračování pohybu robota, naopak nejméně preferovaná je akce zastavení.

Na vstupní vrstvu perceptronu je přiveden vektor příznaků x, který představuje reprezentaci stavu hry na hřišti a jeho výpočet je popsán dále. Jako aktivační funkce perceptronu je použita sigmoida pro skrytou vrstvu a lineární funkce pro výstupní vrstvu. Lineární aktivační funkce výstupní vrstvy je použita z toho důvodu, že potřebujeme spojitý výstup, který budeme moci porovnat s příchozí hodnotou $Q(s_{t+1}, a_{t+1})$.

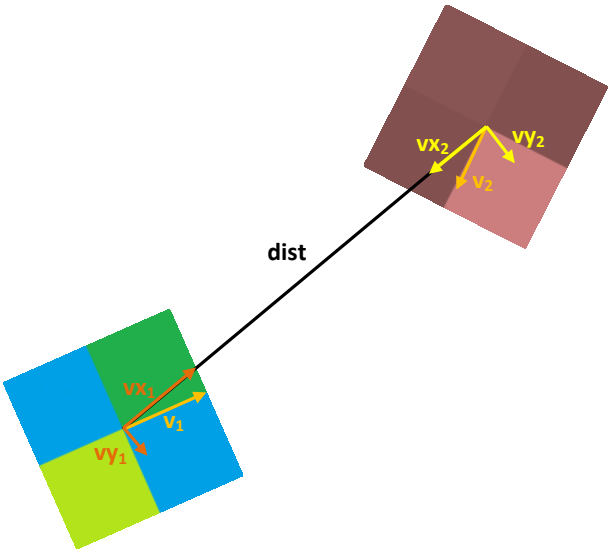
Pro práci s neuronovými sítěmi je v aplikaci vytvořen namespace `NeuralNetworks`, který obsahuje i třídu `Perceptron`. Hlavním cílem implementace této třídy, příp. jmeného prostoru byla jejich znovupoužitelnost. Uživatel si tedy může definovat libovolný počet vrstev perceptronu, v každé vrstvě libovolný počet neuronů a libovolnou aktivační funkci pro každý neuron. Jako trénovací algoritmus pro učení perceptronu je použit **backpropagation**.

Reprezentace stavu

Při použití neuronové sítě je naprosto nezbytné použít správnou reprezentaci stavu, v našem případě stavu hry na hřišti. To ovšem vyžaduje obrovské zkušenosti s používáním neuronových sítí a extrakce příznaků, které autor nemá, a proto se inspiroval článkem [Beitelsaicher].

Reprezentace stavu musí být Markovská, tedy musí obsahovat veškeré informace nutné k předpovídání budoucnosti. Tato podmínka bývá většinou nesplnitelná, ale je třeba se snažit najít co nejlepší aproximaci Markovského stavu. Pro potřeby robotického fotbalu stačí znát pozice všech objektů na hřišti a vektor jejich rychlosti. Vektor rychlosti je počítán z jejich předchozích pozic. Z aktuálního stavu hry na hřišti je nezbytné získat vektor příznaků, který je přiveden na vstupní vrstvu neuronové sítě. Pro každou překážku na hřišti je spočteno následujících pět hodnot, vzdálenost $dist$, vx_1, vy_1, vx_2 a

vy_2. Tyto hodnoty jsou přehledně zobrazeny na obrázku 6.8. Všechny tyto příznaky jsou pro rychlejší trénování neuronové sítě naškálovány.



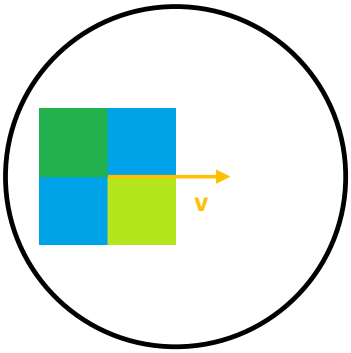
Obrázek 6.8: Vektor příznaků pro každou překážku

Vektory příznaků pro všechny překážky jsou seřazeny podle hodnoty vzdáleností, tedy na prvních pět neuronů neuronové sítě je přiveden vektor příznaků pro nejbližší překážku, na dalších pět neuronů vektor příznaků pro druhou nejbližší překážku apod. Podle výše zmíněného článku tento postup, kdy jsou překážky seřazeny podle vzdáleností, významně pomáhá natrénování neuronové sítě. Pro zjednodušení nejsou použity příznaky pro všechny překážky, ale pouze pro určitý počet těch nejbližších. V době, kdy se tímto způsobem pokoušel autor neuronovou síť natrénovat, to byly čtyři překážky.

Trénování neuronové sítě nebylo pro výše definovaný model úspěšné. Autor předpokládal, že po natrénování neuronové sítě nebude robot většinu času používat žádný úhybný manévr. Teprve v okamžiku, kdy se přiblíží k některému robotovi, který mu brání v průjezdu plánované trasy, se podle situace pokusí uhnout doleva nebo doprava. Úplné zastavení by mělo být používáno co nejméně, proto tento úhybný manévr je poměrně vysoce penalizován. Realita však byla taková, že robot používal stále stejný úhybný manévr, dokud nedošlo ke kolizi, pak většinou akci změnil, ale zase jí používal stále dokola. Autor se pokoušel celý systém rozšiřovat o odměnění robotů, kteří dorazí do cíle nebo se vyhnou překážce, či se pokoušel měnit počet neuronů ve skryté

vrstvě, apod. Tímto se místo zlepšení výkonosti celý systém zkomplikoval a hodnota odměny (reward) se v průběhu zkoumání výrazně nezlepšovala.

V předchozí podobě tedy nebyl řídící software schopen neuronovou síť natrénovat a žádné drobné úpravy solidní řešení situace nepřinesly. Dalším pokusem tedy bylo celý systém výrazně zjednodušit a zajistit, že po většinu času nebude robot provádět žádný úhybný manévr, aby se nepohyboval zmateně jako doposud. Robot se tedy pohybuje po plánované trase a na své cestě zkoumá pouze nejbližší okolí, zda se nenachází v blízkosti překážka. Zkoumané okolí má tvar kružnice, která ale není soustředná s aktuální pozicí robota. Její střed je posunut ve směru pohybu robota, viz obrázek 6.9.



Obrázek 6.9: Zkoumané okolí robota

Dokud se ve zkoumaném prostoru nenachází žádná překážka, robot se pohybuje po plánované trase. V okamžiku, kdy se některý z robotů nachází v blízkosti, vypočte se vektor příznaků stejně jako v původním případě, tedy hodnoty $dist$, vx_1, vy_1, vx_2 a vy_2. Pokud se ve zkoumané oblasti nachází více překážek, jsou spočteny vektory příznaků pouze pro nejbližší dva. Situace, kdy se v okolí robota nachází více než jeden robot, jsou poměrně vzácné a na výběr úhybného manévru má největší vliv ten nejbližší.

Vektor příznaků je tedy přiveden na vstupní vrstvu neuronové sítě. Pokud se nachází v blízkosti robota pouze jedna překážka, je vektor příznaků, rezervovaný pro případnou druhou překážku, nastaven na výchozí hodnotu, kdy vzdálenost je maximální hodnota datového typu **double** a ostatní hodnoty jsou 0. Výběr úhybného manévru je pak shodný s předchozím případem.

Takto zjednodušený systém už vykazoval jisté známky natrénování. Pokud robot měl za úkol objíždět neustále stojící překážku, naučil se zadané

úloze po 3 – 4 kolizích. Ani při vyhýbání se pohybující se překážce nebyly výsledky zcela špatné, ale autor není spokojen s tím, že robot neustále uhýbá pouze na jednu stranu. Z toho důvodu je pro demonstraci vyhýbání se překážkám ve finální verzi diplomové práce výběr úhybných manévrů řízen pravidly.

Pravidlové řízení

Výběr úhybných manévrů pomocí pravidel je velice jednoduchý a slouží pouze k demonstraci elementární inteligence. Přes svou jednoduchost funguje poměrně spolehlivě a jednou z možností dalšího vývoje vyhýbání se překážkám může být právě rozšíření pravidlového řízení.

Výběr úhybného manévru nastává až v okamžiku, kdy se v blízkosti robota nachází překážka. Detekce překážek probíhá pouze v okolí robota, znázorněném na obrázku 6.9, tedy stejném jako při výběru úhybného manévru pomocí neuronové sítě. Úhybný manévr je vybírán pouze podle vzájemné polohy a vzájemné rychlosti s nejbližší překážkou.

Po detekování překážky je nutné nejprve detekovat tzv. **kolizní bod**. Princip jeho detekce je jednoduchý, predikujeme budoucí pozice jak robota, tak překážky, podle jejich aktuální pozice a vektoru rychlosti. V takto predikovaných pozicích sestrojíme kružnice o poloměru $\sqrt{2} \cdot \frac{l}{2}$, kde l je rozteč kol robota. Pokud existuje průsečík obou kružnic, je označen za kolizní bod. Pokud existují 2 průsečíky, je za kolizní bod označen střed jejich spojnice. Pokud žádný průsečík neexistuje, výše popsaný postup se opakuje, a to do doby, než je kolizní bod nalezen nebo než je překročen limit počtu iterací. Pokud bude vývoj vyhýbání se překážkám pokračovat směrem pravidlového řízení, pak autor navrhuje především vylepšení predikce pohybu obou robotů.

Pokud nebyl kolizní bod nalezen, robot se pohybuje rovně. V opačném případě se určí, zda se kolizní robot nachází vlevo nebo vpravo od přímky definované robotovou aktuální pozicí a směrnicí jeho rychlosti. Nachází-li se vpravo, pak robot uhýbá doleva a naopak.

Výše zmíněné pravidlo o uhýbání platí pouze v případě, že se robot nachází v určité vzdálenosti od cílového bodu. Pokud se nachází v jeho blízkosti, jsou tyto úhybné manévry nepoužitelné, neboť by bylo obtížné robota zpátky na průjezd požadovaného bodu nasměrovat, zvlášť, má-li jím projíždět s cílovým natočením. V takovýchto případech je použit úhybný manévr zastavení. Robot počká, až nebezpečí kolize pomine, a poté se znova rozjede

k cíli. V tomto okamžiku by mohl nastat problém, neboť robot by mohl být blokován nekonečně dlouho, pokud by se překážka, která ho brzdí nikdy nerozjela. To se může stát v případě, že se takto vyhýbají dva roboti, kteří se náchází v blízkosti cílového bodu a vzájemně se zablokují. Autor tedy pro tuto akci nastavil časový limit a pokud se do té doby robot nerozjede sám je k pohybu donucen.

7 Pokročilé akce

Vsechny akce popsané v předchozích kapitolách jsou bezesporu součástí modulu elementární inteligence, neboť se staraly o efektivní a plynulý pohyb robota. Pro akce popsané v této kapitole to však neplatí, neboť se jedná o jisté rozhraní, které někdo může považovat za součást herní strategie a někdo za součást elementární inteligence. Modul elementární inteligence je navržen pro použití těchto akcí, ale nic nebrání programátorovi herní strategie, aby používal pouze akci pohyb robota (viz kapitolu 5) a funkcionalitu pokročilých akcí implementoval v modulu herní strategie.

Strojové učení modul elementární inteligence využívá pro natrénování PID-regulátorů základních akcí a pro výběr úhybných manevrů pro vyhnutí se překážkám. Hlavní nasazení strojového učení je plánováno pro potřeby pokročilých akcí, protože ty už jsou příliš složité pro kvalitní explicitní naprogramování.

Pokročilé akce nejsou v této diplomové práci implementovány. Tato kapitola je tedy z větší části teoretická a shrnuje plány pro budoucí vývoj elementární inteligence. Na vině je z větší části současný fyzikální engine, který velice zpomalil vývoj předchozích akcí a pro práci s míčem je zcela nepoužitelný, viz dále. Dalším důvodem je, že autor je se strojovým učením obeznámen především teoreticky, ale použití metod strojového učení v praxi je daleko složitější. Pro potřeby robotického fotbalu existuje jen velice málo materiálů, ze kterých se lze inspirovat praktickými příklady, a proto se autor velice zdržel implementací strojového učení pro potřeby akce vyhýbání se překážkám popsané výše.

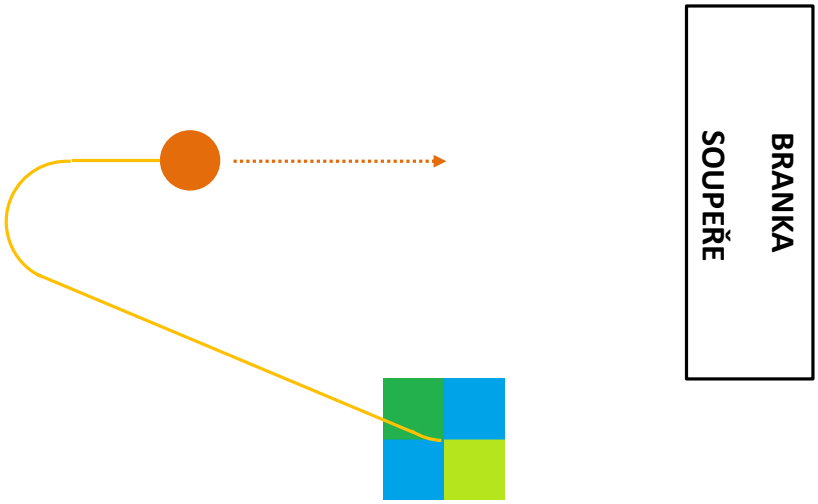
7.1 Práce s míčem

Fyzika řídícího systému pro pohyb míče je zcela nerealistická. Po kontaktu s robotem se míč začne pohybovat, ale v okamžiku, kdy se od něj oddělí, okamžitě zpomalí na cca 10% původní rychlosti a poté téměř nezpomaluje. Jediný způsob, jakým lze v současnosti s míčem pracovat, je zůstat s ním neustále v kontaktu. V takovém případě ale pokročilé akce pro práci s míčem, navržené pro sofistikovanější činnost, ztrácejí svůj smysl a jsou nahraditelné obyčejným pohybem robota.

Modul elementární inteligence je připraven pro implementaci pokročilých metod, ale jejich vývoj začne až s dokončením výroby reálných robotů.

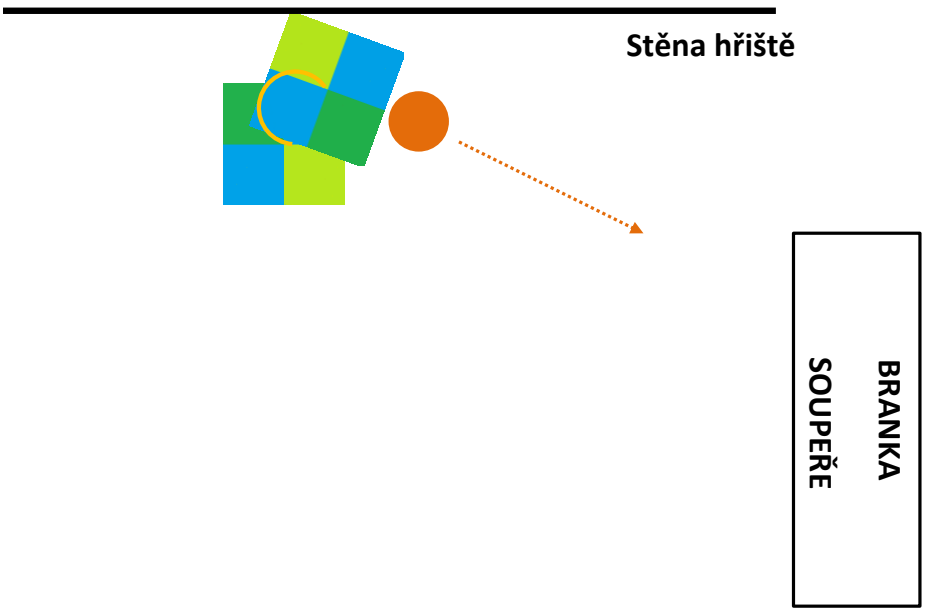
7.1.1 Střelba

Střelba je akce, kdy robot narazí do míče v takovém směru, aby se začal pohybovat směrem k soupeřově bráně. Robot se pomocí akce pohyb robota přiblíží k pozici míče tak, aby se míč nacházel mezi robotem a soupeřovou bránou. Robot na tuto pozici musí přijet ve správném směru tak, aby pouhou akcelerací a pohybem po přímce míč poslal požadovaným směrem. Přehledně je střelba z pohybu robota zobrazena na obrázku 7.1.



Obrázek 7.1: Střelba z pohybu

Pokud se míč nachází na těžko dostupných místech, především u stěn a v rozích hřiště, nelze střelbu z pohybu pokaždé použít. V takových případech se používá střelba pomocí rotace. Robot se přiblíží k míči co nejblíže a poté se protočí. Tato střelba je mnohem méně přesná a jejím účelem je především dostat míč na přístupnější pozici. Plocha robota bývá poměrně malá a samotným protočením robota kolem jeho středu by se mohl robot pouze protočit, aniž by míč zasáhl. Robot se tedy neotáčí kolem své osy, ale využívá možnosti popsané v podkapitole 4.3 protočení kolem středu, který se nachází na ose mezi koly robota. Obrázek 7.2 zobrazuje výše popsanou situaci.



Obrázek 7.2: Střelba z rotace

Použití technik strojového učení je pro akci střelba plánováno. Jejich účelem však není samotná střelba, ale rozhodování, zda se robot nachází ve vhodné střelecké pozici či nikoliv. V ideálním případě vhodná střelecká pozice znamená, že existuje místo soupeřovy brány, kam může robot míč vystřelit, a pokud se žádný ze soupeřových robotů nepohne, padne gól. Ve skutečnosti není střelba příliš přesná, ze středu hřiště je robot schopen se trefit do brány, ale pokaždé na jinou pozici. Skutečnou přesnost střelby půjde ověřit až v reálném prostředí.

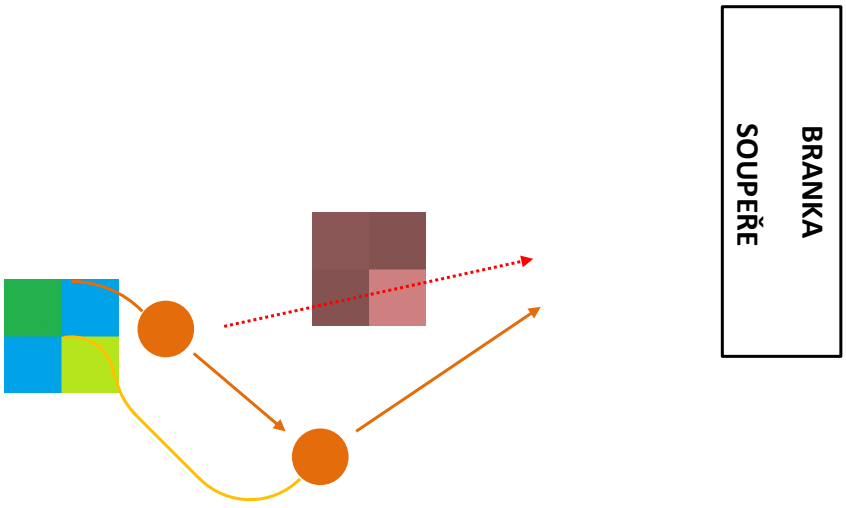
Pro účely akce střelba je plánováno použít shodné nebo velmi podobné techniky strojového učení jako pro akci vyhýbání se překážkám. Plánováno je tedy nasazení vícevrstvého perceptronu napojeného na některou z technik reinforcement learningu. Učení proběhne tak, že robot bude střílet na bránu z různých pozic na hřišti. Padne-li z jeho střely gól, bude robot odměněn (reward +1), zarazí-li se míč o brankáře, nebude ani odměněn ani potrestán (reward 0), a pokud míč zblokuje soupeřův obránce, pak bude robot potrestán (reward -1).

Manévry, které může robot použít jsou teoreticky velice jednoduché, ale jejich implementace bude podle názoru autora poměrně obtížná. Jedná se

o manévry:

- **Shot:** Robot z aktuální pozice vystřelí na bránu. Podle výše popsaného systému odměn získá robot odezvu pro úpravu vah perceptronu. Tento manévr je ve skutečnosti v současnosti již částečně implementován. Robot je schopen vystřelit stojící míč na bránu s přesností popsanou výše. Vzhledem k tristní úrovni fyziky pohybu míče se autor nezabýval pokusy trefit míč v pohybu.
- **Left (Right):** Robot popostrčí míč do strany, aby se dostal do lepší střelecké pozice. Robot není odměněn ani potrestán, dokud nevystřelí, to znamená, že robot může tuto akci teoreticky použít kolikrát chce dokud se nenachází ve vhodné střelecké pozici. Teoreticky by měl být robot potrestán, pokud použitím této akce ztratí nad míčem kontrolu, ať už vlivem příliš prudkého posunu míče nebo zásahem soupeřova robota. Cílem technik strojového učení pro potřeby akce střelba je však schopnost rozeznat vhodnou střeleckou pozici, a proto autor o trestech za ztrátu míče prozatím neuvažuje.

Samotná střelba je zobrazena již na obrázcích 7.1 a 7.2. Manévr **Right** je znázorněn na obrázku 7.3, manévr **Left** by se prováděl na opačnou stranu, jinak je shodný s předchozím.



Obrázek 7.3: Manévry robota před střelou

7.1.2 Přihrávka

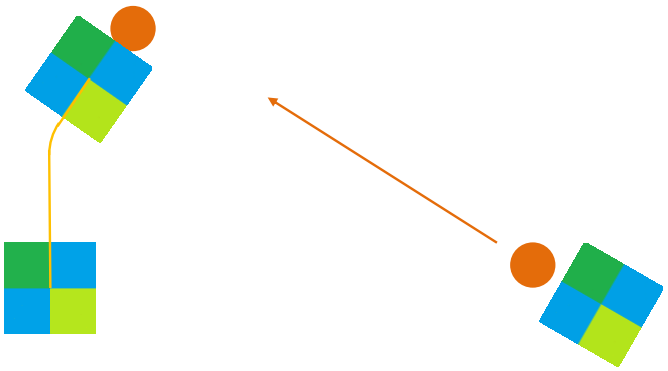
Přihrávka je akce principiálně shodná s akcí střelba. Rozdílem je, že míč není posílán na bránu soupeře, ale na místo, kde jej některý ze spoluhráčů může zpracovat a dále s ním pracovat. Akce přihrávka se používá většinou pro přesun míče k hráči, který se nachází v lepší pozici pro skórování. Míč se často přesouvá od brankáře k obráncům, od obránců k útočníkům nebo mezi útočníky pro vyšachování soupeřovy obrany.

Tato akce vyžaduje kooperaci dvou robotů, z nichž ten, který kontroluje míč, použije akci přihrávka a druhý akci zpracování míče, viz dále. Trénování těchto akcí tedy musí probíhat paralelně a autor předpokládá, že odladění bude velice náročné.

Techniky strojového učení a manévry prozatím plánuje autor shodné s akcí střelba. Pokud druhý robot míč zpracuje, je robot odměněn, pokud přihrávku zachytí hráč soupeře, pak je robot potrestán. Pokud nenastane ani jeden z obou případů, což se s největší pravděpodobností alespoň ze začátku bude stávat velice často, nebude nejspíš robot odměněn ani potrestán, maximálně potrestán řádově 10× menším trestem, než při nahrátí soupeři.

7.1.3 Zpracování míče

Zpracování míče je akce o jejíž podobě má autor zatím jen mlhavou představu. Slouží k získání kontroly nad míčem, většinou po nahrávce spoluhráče, ale možná ji bude možné použít i k blokování soupeřovy nahrávky.



Obrázek 7.4: Zpracování míče

Průběh akce probíhá tak, že se predikuje pohyb míče a robot se ho pokusí zastavit a získat nad ním kontrolu v nejvhodnějším místě, viz obrázek 7.4. Jestli bude akce využívat technik strojového učení není zatím jisté a pokud ano, akce robota nemá autor zatím promyšleny.

7.2 Obrana

Obrana je první akce, která není přímo vázaná na fyziku míče, její natrénování ale musí proběhnout společně s trénováním útočných akcí (střela, přihrávka, zpracování). Účelem této akce je znemožnění jakékoli činnosti soupeřova robota, ze které by mohla být ohrožena branka, v rámci pravidel. Pokud tedy soupeřův robot nekontroluje míč, obránce by ho měl nechat hýbat. Pokud naopak soupeř míč získá, obránce do něj nebude narážet (to je proti pravidlům), ale bude mu blokovat jakýkoliv pohyb k bráně. Konečným cílem, za který může být robot v rámci trénování technik strojového učení odměněn, je získání míče pod vlastní kontrolu.

Výše popsaná činnost je pouze základ úspěšné obrany. Bránící robot ve finální podobě této akce by měl zabránit soupeři i převzetí kontroly nad míčem. To znamená, že by měl blokovat i přihrávky na soupeřova robota nebo blokovat jeho pohyb, pokud se míč nachází v blízkosti apod. Finální podobu této akce autor ještě nemá zcela promyšlenou.

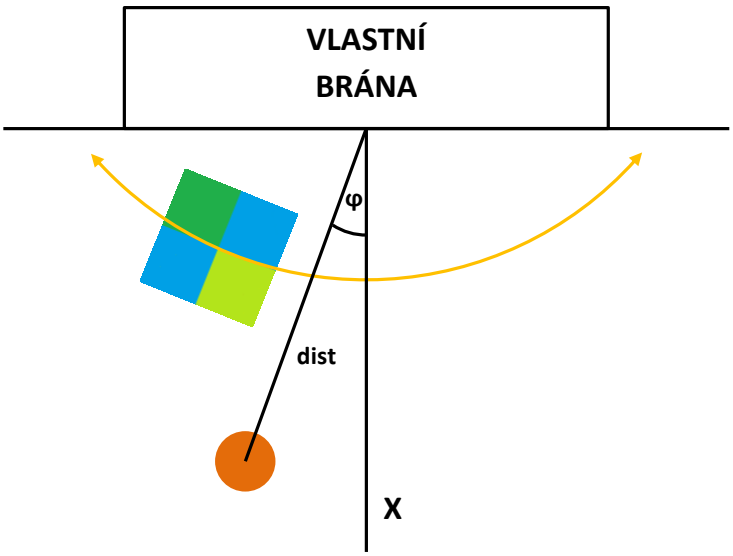
Trénování obraných a útočných akcí bude muset probíhat najednou nebo po fázích. Po fázích znamená, že například nejdříve natrénujeme akci střelba se statickými obránci. Poté jsou trénováni obránci pomocí již natrénované akce střelba. Na závěr se natrénuje akce střelba znovu na již pohybujících se obráncích. Takto, po fázích, by musel trénink proběhnout i pro všechny ostatní pokročilé akce. Trénování všech akcí najednou bude velice zmatené, autor tedy musí najít nějaký „ideální“ kompromis, v každém případě trénink pokročilých akcí bude náročná záležitost, která dalece přesahuje rámec zadání této diplomové práce.

7.3 Brankář

Činnost brankáře slouží k omezení počtu vstřelených gólů. Konečné rozhodnutí je na autorovi herní strategie, ale s největší pravděpodobností bude jeden

ze členů týmu zastávat funkci brankáře a měla by pro něj být vytvořena samostatná akce, kterou bude většinu herního času vykonávat.

Akce brankáře lze natrénovat tak, že jeden z hráčů střílí na branku z různých pozic na hřišti. Aktuální stav na hřišti pro akci brankáře představuje pozice míče na hřišti, především úhel ϕ, který svírá s osou X, jeho vzdálenost od brány a jeho rychlost, viz obrázek 7.5. Cílem akce brankáře je výběr optimální pozice kolem brány, kam se má brankář přesunout, aby měl největší šanci na chycení vystřeleného míče. Na obrázku 7.5 by se tedy měl začít pohybovat doleva, blíže ke středu brány.



Obrázek 7.5: Pohyb brankáře

Brankář se pohybuje na půlkruhu kolem brány, viz obrázek 7.5. Použité techniky strojového učení jsou podobné těm použitým v akci vyhýbání se překážkám, tedy vícevrstvý perceptron napojený na některou z technik reinforcement learningu. Výstupem perceptronu je rozhodnutí, jestli se má brankář pohnout vlevo, vpravo nebo zůstat na místě. Pokud útočník vystřelí a dá gól, je brankář potrestán (reward -1). Naopak, pokud míč míří do brány a brankář gólu zabrání, je odměněn (reward +1). Hlavní činnost této akce probíhá při pohybu míče na bránu, neboť až v okamžiku, kdy se vzdálí od soupeřova robota lze přesně určit pozici, kam se má brankář přesunout.

8 Integrace do systému

Modul elementární inteligence je součástí řídícího softwaru. Jako takový od systému dostává informace o aktuálním stavu hry na hřišti a zpět do systému posílá okamžitý směr a rychlost pohybu robota.

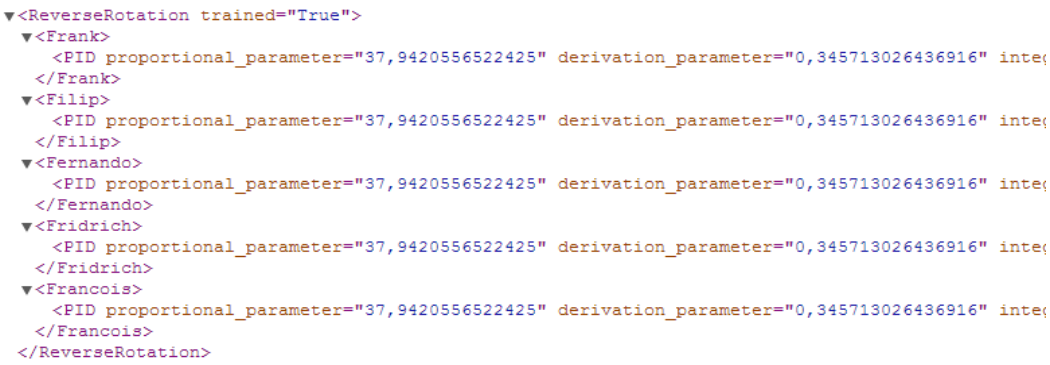
Hlavní třída elementární inteligence je oddděná od abstraktní třídy `Motion`, která je součástí jádra řídícího softwaru. Implementace abstraktních metod třídy `Motion` definuje chování modulu při inicializaci a ukončení běhu řídícího software a definuje chování při každé aktualizaci stavu hry na hřišti. Tyto abstraktní metody tedy definují rozhraní mezi modulem elementární inteligence a zbytkem řídícího softwaru.

Kromě aktualizace stavu hry na hřišti potřebuje modul elementární inteligence pro svůj běh další informaci – příkazy jednotlivým robotům. K tomu slouží třída `OrderProcessing`, která by měla v budoucnu sloužit jako rozhraní mezi herní strategií a elementární inteligencí. Modul herní strategie je poslední z modulů řídícího software, který ještě není vyvíjen, a proto není definované ani rozhraní mezi oběma moduly. Pro demonstraci diplomové práce jsou tak prozatím roboti řízeni přímo v modulu elementární inteligence.

8.1 Inicializace a natrénování

Veškerá data relevantní pro běh modulu elementární inteligence jsou uložena v XML souboru `ElementaryIntelligence.xml`. Během inicializace je tento soubor načten do paměti. Konfigurovatelné či natrénované proměnné jsou pro každou akci uloženy ve vlastním XML elementu nejvyšší úrovně. Pro základní akce jsou navíc všechny tyto proměnné uloženy pro každého robota zvlášť, viz obrázek 8.1. Každý robot má své jméno, které autor používá jak pro přiřazení proměnných základních akcí, tak pro logování informací jednotlivých akcí na disk.

Samotný trénink by měl teoreticky probíhat pouze v prvním běhu programu, pak už by měly být všechny akce natrénované a fáze tréninku přeskočena. Jak lze spatřit na obrázku 8.1, každá akce má vlajkový příznak `trained`, který určuje, zda je akce natrénovaná nebo ne. Pokud má tento vlajkový příznak pro některou akci hodnotu `false`, pak je před samotným začátkem



Obrázek 8.1: ElementaryIntelligence.xml

hlavního běhu modulu elementární inteligence akce natrénována. Pokud uživatel není s natrénováním některých akcí spokojen, může před spuštěním aplikace v konfiguračním souboru změnit hodnotu vlajkového příznaku pro akce, které chce přetrénovat.

Při hlavním běhu modulu elementární inteligence se každý robot stará o zpracování příkazů herní strategie samostatně. Pro každého robota tedy probíhá přiřazení úkolů jednotlivým akcím ve vlastním vlákně a činnost robotů je koordinována pouze v modulu herní strategie, nikoli v modulu elementární inteligence. Pro trénink akcí (zvláště pokročilých akcí), je však nutná určitá kooperace robotů. Například pro trénování vyhýbání se překážkám se všichni roboti pohybují náhodně po hrací ploše a trénovaný robot se jim snaží vyhýbat.

9 Realizace

Modul elementární inteligence je programován jako DLL knihovna v projektu robotického fotbalu. Je tedy nezávislý na konkrétní implementaci ostatních částí projektu a s ostatními moduly komunikuje pomocí zpráv. Rozhraní mezi jednotlivými moduly je podrobně popsáno v bakalářské práci [Altman(2011)]. Rozhraní mezi moduly elementární inteligence a herní strategie ještě není pevně stanoveno, neboť oba moduly jsou stále ve vývoji. Prozatím je definováno pomocí třídy `OrderProcessing`, kterou obsahuje knihovana modulu elementární inteligence. Později by rozhraní i mezi těmito dvěma moduly mělo mít formu zpráv.

Modul elementární inteligence stejně jako zbytek řídícího software byl vyvíjen v objektovém programovacím jazyce C# s využitím platformy Microsoft .NET Framework verze 2.0. Po zkušenostech s výkonem i implementací aplikací v různých programovacích jazycích by autor raději zvolil, kdyby měl na výběr, programovací jazyk C++ společně s využitím knihoven QT.

Hlavní třídou elementární inteligence je třída `ElementaryIntelligence`, která je oddděděná od třídy `MotionModule`. Přes rodičovskou metodu může hlavní třída posílat zprávy o pohybových vektorech do řídícího systému. Informace o stavu hry na hřišti modul získává, pomocí metody `OnGameState (GameState state)`, která je v rodičovské třídě abstraktní a musí být ve třídě `ElementaryIntelligence` implementována. Struktura `GameState` obsahuje aktuální pozici a natočení každého robota, uložené ve struktuře `RobotState` a pozici míče na hřišti, více viz [Altman(2011)].

Po spuštění řídícího systému dochází i k inicializaci modulu elementární inteligence. Modul se může nacházet v několika různých fázích:

- **Inicializace**: Řídící třídou této fáze je třída `InitializeEI`, jejímž hlavním úkolem je načtení XML dokumentu `ElementaryIntelligence.xml` a nastavení ošetření událostí, pokud dojde ke změně XML elementů v dokumentu, především během fáze tréninku akcí.
- **Trénink**: Třída `TrainingEI` kontroluje veškeré akce, zda již byly natrénovány, či nikoliv. Každá akce obsahuje metodu `Train()`, ve které je definován postup jejího tréninku. Trénink akce se poté opakuje pouze v případě, že uživatel změní hodnotu vlajkového příznaku v konfiguračním souboru.

- **Hlavní fáze**: Hlavní fáze začíná pro ukončení tréninku všech akcí. Pro každého robota v týmu je spuštěna instance třídy `RobotEI`, jejíž činnost probíhá ve vlastním vlákně. Tato třída iniciuje a kordinuje činnost všech akcí dle příkazů z herní strategie. Činnost robotů během hlavní fáze je nezávislá na ostatních.

9.1 Akce

Činnost jednotlivých akcí je popsána v předchozích kapitolách. Modul elementární inteligence obsahuje abstraktní třídu `Action`, která je rodičovskou třídou všech dalších tříd akcí. Tato třída definuje metody a proměnné společné všem akcím. Dále obsahuje abstraktní metody, které musí oddděné třídy implementovat. Jedná se například o metodu `Train()`, definující činnost robota během tréninku akce, nebo `Launch()`, která řídí činnost robota po aktivaci funkce v průběhu hlavní fáze.

Každá akce je spuštěna ve vlastním vlákně, které je ihned po spuštnění uspáno pomocí synchronizačního primitiva `AutoResetEvent`. Probuzení tohoto vlákna autor nazývá aktivací funkce. Většina akcí je uspána a nezvyšuje tak náročnost řídícího softwaru na výpočetní výkon. Synchronizační primitivum `AutoResetEvent` se používá i pro synchronizaci výpočtů s příchozími snímky z kamery. Činnost akcí probíhá v cyklu `while`, ale po skončení výpočtů je vlákno uspáno a probuzeno až po aktualizaci stavu hry na hřišti. K tomuto účelu třída `Action` obsahuje metodu `NextLoop(RobotState[] states)`. V této metodě se nejprve aktualizuje stav robota, spočte rychlost, úhlová rychlost, apod. a poté je vlákno probuzeno. Neaktivní akce jsou uspány jiným semaforem, a proto nejsou po aktualizaci snímku z kamery probuzeny.

Od abstraktní třídy `Action` jsou odděděny jiné abstraktní třídy – `BasicAction` a `AdvancedAction`. Obě abstraktní třídy jsou si velice podobné, implementují především metody pro výstup informací do souboru nebo na konzoli. Třída `BasicAction` je rodičovská třída všech základních akcí. Pracuje s pohybovými vektory, které po spočtení posílá dále do řídícího modulu. Přístup k metodám systému mimo modul elementární inteligence má pouze hlavní třída `ElementaryIntelligence`. Autor tento problém vyřešil pomocí struktury jazyka C# zvané delegáty (**delegate**). Delegát je bezpečný ukazatel na funkci. Jedním z parametrů konstruktoru všech základních akcí je delegát na metodu, která má přístup k rozhraní mezi modulem elementární inteligence a řídícím modulem. Pohybový vektor je tedy možné do systému

posílat přímo z třídy `BasicActions` nebo jejích potomků.

Pokročilé akce jsou odděděny od třídy `AdvancedAction`. Tato třída v počátcích vývoje měla obsahovat především metody pro práci se základními akcemi. Během vývoje však přístup k základním akcím získala výhradně akce pohybu robota implementovaná v třídě `Movement`. Z toho důvodu byl obsah třídy `AdvancedAction` redukován pouze na metody pro výstup do souboru, na konzoli nebo na uživatelské rozhraní, apod. Každá z akcí je implementovaná ve vlastní třídě, oddděné podle typu buď od třídy `AdvancedAction` nebo `BasicActions`.

9.2 Geometrie

Výpočet optimální trasy robota, její průjezd i vyhýbání se překážkám obsahuje veliké množství výpočtů z oblasti analytické geometrie. Implementace těchto výpočtů byla součástí vývoje modulu elementární inteligence. Třídy a metody implementující tuto funkcionalitu byly programovány s důrazem na znovupoužitelnost a v budoucnu budou obsaženy v samostatné knihovně, kterou může modul elementární inteligence využívat. V současnosti jsou však pevnou součásti knihovny modulu.

Každá třída implementuje metody pro práci s jiným typem křivky. Jmenný prostor `Geometrie` obsahuje třídy typu `Line` pro práci s přímkami a úsečkami, `Circle` a `Arc` pro práci s kružnicemi a oblouky nebo `Point` pracující s jednotlivými body. V bakalářské práci byla veškerá funkcionalita implementována ve statických metodách třídy `GeometryOperations`, která přetrvala i v diplomové práci, ale její význam je daleko menší. Autor většinu metod implementoval znovu a rozdělil do výše popsaných tříd. Kromě přesunu metod z třídy `GeometryOperations` jsou nové třídy rozšířeny a jmenný prostor `Geometrie` se stal velice silným nástrojem.

10 Uživatelská příručka

Modul elementární inteligence poskytuje nástroje pro vývoj modulu herní strategie. Jedná se tedy o vývojový kit a ne o finální aplikační produkt, proto není uživateli umožněno tento modul jakkoli ovládat. Pro demonstraci správné funkcianality modulu se po hrací ploše zcela náhodně pohybují dva roboti. Pro každého z nich jsou v nekonečné smyčce vygenerovány čtyři body, pro které je naplánována optimální trasa a které robot postupně projíždí. Pro polovinu těchto bodů se počítá dráha s cílovým natočením a pro polovinu bez. V okamžiku, kdy robot takto náhodně generovanou trasu projede, jsou náhodně vygenerovány nové body.

10.1 Spuštění aplikace

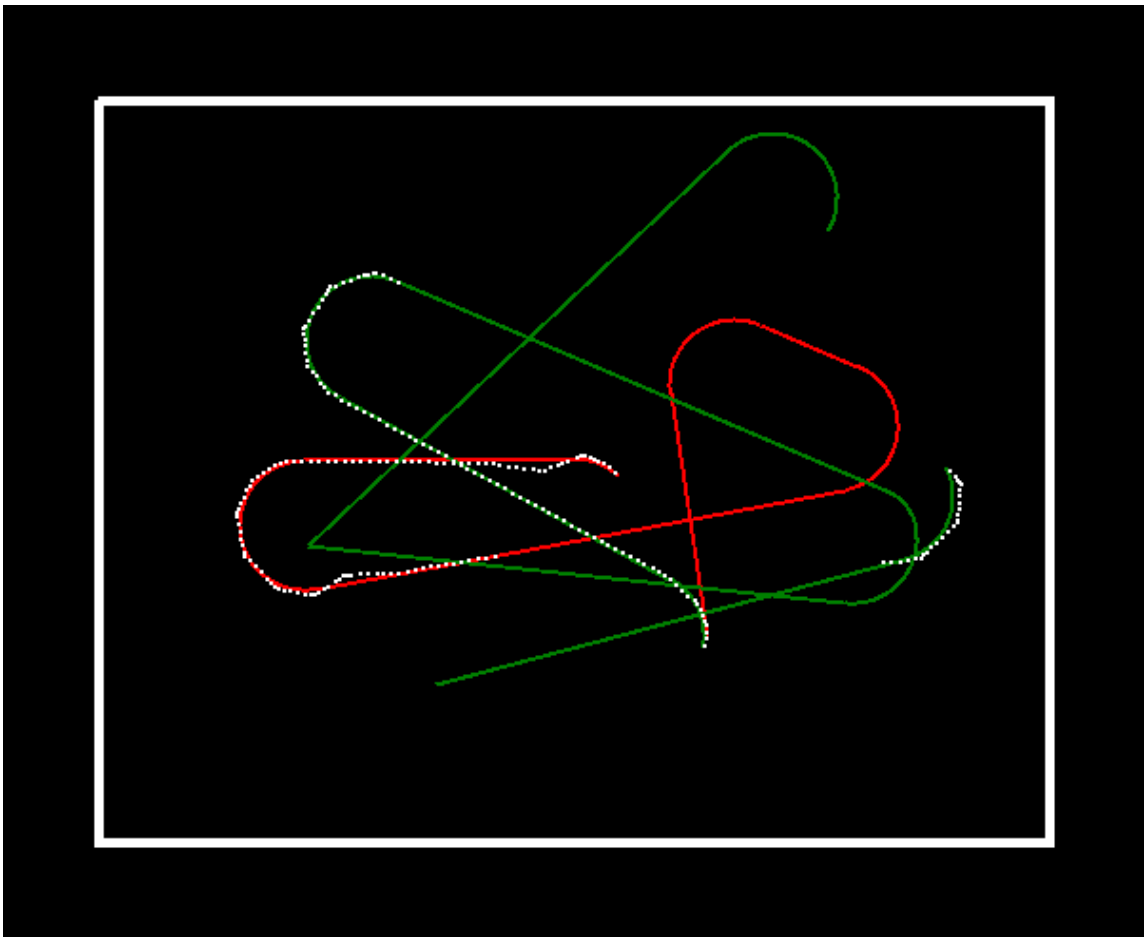
Pro zjednodušení spouštění aplikaci vytvořil autor v kořenovém adresáři skript `run.bat`. Samotný spustitelný soubor je vinou návrhu řídícího software skryt poměrně hluboko v souborovém systému. Pro spuštění aplikace je nutné mít nainstalován physX kvůli současnému fyzikálnímu enginu. Ten by měl být součástí ovladačů grafických karet NVIDIA. Pro uživatele konkurečních grafických karet je physX přiložen spolu s aplikací na doprovodném DVD.

Po spuštění aplikace se objeví GUI celého řídícího systému, které autor nastavil tak, aby obsahovalo na větší části obrazovky 3D-vizualizaci a grafické rozhraní přímo modulu elementární inteligence.

10.2 Uživatelské rozhraní

Uživatelské rozhraní modulu elementární inteligence je grafické. Na plátně je neustále vykreslována aktuální plánovaná cesta pro každého robota. Plánovaná trasa prvního robota, který byl během tréninku akcí vždycky hlavním robotem, je vykreslována červeně, trasy ostatních robotů zeleně, jak si lze prohlédnout na obrázku 10.1.

Kromě plánovaných akcí je bílou tečkou po každé aktualizaci stavu hry na



Obrázek 10.1: Uživatelské rozhraní

hřišti znázorněna aktuální pozice každého robota. Uživatel tak může pozorovat rozdíly mezi plánovanou trasou robota a trasou, kterou robot skutečně projede. Na obrazovce zůstávají pro každého robota viditelné všechny předchozí pozice, ve kterých se nacházel od naplánování trajektorie jeho pohybu. V okamžiku, kdy je trasa robota přeplánována z jakéhokoli důvodu, veškeré předchozí pozice robota přestanou být vykreslovány.

Vykreslování pozic, ve kterých se robot nacházel od naplánování současné trajektorie jeho pohybu, je zcela nezávislé na ostatních robotech. To znamená, že pokud je plánovaná trasa pro některého robota přeplánována,

předchozí pozice všech zbylých robotů jsou nadále vykreslovány.

10.3 Konfigurace elementární inteligence

Některé konstanty ovlivňující pohyb robotů jsou konfigurovatelné. V kapitole 8.1 je popis XML souboru `ElementaryIntelligence.xml`, ve kterém se nacházejí proměnné, jejichž hodnoty systém získá během tréninku jednotlivých akcí a které ukládá pro další použití. Kromě těchto proměnných se zde nachází i proměnné, které autor umožňuje uživateli konfigurovat tak, aby mohl částečně optimalizovat pohyb robotů. Jedná se především o hodnoty tolerancí, či optimální poloměr oblouku, po kterých se robot pohybuje, či parametry pro trénovaní neuronové sítě pro vyhýbání se překážkám, apod.

Uživatel může pochopitelně měnit i hodnoty parametrů, které systém získal během tréninku, ale autor podobný postup doporučuje jen s maximální opatrností, neboť vliv těchto parametrů může být větší než uživatel očekává.

11 Závěr

Autor vytvořil modul elementární inteligence od základů. Pokusil se implementovat metody strojového učení, ale to bylo úspěšné pouze u základních akcí pro určení koeficientů PID-regulátoru. Další nasazení strojového učení je plánováno až při dalším vývoji modulu elementární inteligence, který by měl pokračovat po výrobě reálných robotů.

Původní modul elementární inteligence, se kterým měl autor činnost aktuálního modulu porovnat, je vlivem úprav v řídícím softwaru nepoužitelný. Aktuální modul elementární inteligence je však velice spolehlivým nástrojem, který může modul herní strategie využívat. Především regulace pohybu robota po plánované trase pomocí PID-regulátoru je silnou stránkou modulu. V diplomové práci jsou ošetřeny i případy, se kterými měl původní modul veliké problémy, a to především plánovaní trasy pro případy, kdy se cílová pozice robota nachází v blízkosti počáteční.

Modul elementární inteligence je rozšířen o vyhýbání se překážkám. To ještě není zcela dokonalé, ale ke kolizím dochází méně často. Pro tyto účely výhýbání se překážkám již autor implementoval řízení pomocí neuronové sítě, ale to zatím nepřináší lepší výsledky než pravidlové řízení. Zvýšení účinnosti řízení pomocí neuronové sítě považuje autor za úkol nejvyšší priority pro další vývoj, tak aby bylo možné současné řízení pomocí pravidel nahradit.

Největším problémem při vývoji modulu elementární inteligence byl nedostatek jakýchkoliv písemných materiálů, které by autorovi umožnily se inspirovat na praktických příkladech. To se týká především použití metod strojového učení, s jejichž praktickým nasazením má autor nedostatek zkušeností a jejichž integrace do systému vývoj modulu výrazně zpomalovala.

Návrhy dalšího rozšiřování modulu elementární inteligence jsou popsány již v předchozích kapitolách. Jedná se především o nástroje, které jsou schopné zpracovat abstraktní případy herní strategie a které autor nazývá pokročilé akce. Autor však považuje za vhodné s jejich vývojem začít až v reálném prostředí s reálnými roboty. Je to z toho důvodu, že se současným simulovaným prostředím nemá autor dobré zkušenosti a jeho vyladění by zabralo příliš mnoho času.

Literatura

[Altman(2011)] ALTMAN, P. *Jádro řídicího systému a virtualizační modul pro robotický fotbal.* Bakalářská práce, Západočeská univerzita v Plzni, 2011.

[Lepic(2011)] LEPIČ, J. *Multiagentní systém pro plánování herní strategie robotického fotbalu.* Bakalářská práce, Západočeská univerzita v Plzni, 2011.

[Valis(2011)] VALIŠ, J. *Modul elementární inteligence robotického fotbalu.* Bakalářská práce, Západočeská univerzita v Plzni, 2011.

[Sokrates(2004)] SCHOUTE, A a POEL, M. *Dynamic motion planning of soccer playing mini-robots.* University of Twente, 2004.

[Alpaydin(2010)] ALPAYDIN, E. *Introduction to Machine Learning.* London, England : The MIT Press, 2010.

[Beitelsaicher] BEITELSAICHER, J. *Applying Reinforcement Learning to Obstacle Avoidance.*

[Hinton(2012)] HINTON, G. *Neural Networks for Machine Learning* [online], 2012. Dostupné z: www.coursera.org.

[Ng(2011)] NG, A. *Machine Learning* [online], 2011. Dostupné z: www.coursera.org.

[Smola(2008)] SMOLA, A. *Introduction to Machine Learning.* London, England : Cambridge University Press, 2008. ISBN 0-521-82583-0.

[Rojas(1996)] ROJAS, R. *Neural Networks.* Berlin, Německo , 1996.

[Sutton, Barto(2005)] SUTTON, R. S. a BARTO, A. G. *Reinforcement Learning: An Introduction.* London, England : The MIT Press, 2005.

[Thrun(2012)] THRUN, S. *Artificial Intelligence for Robotics* [online], 2012. Dostupné z: www.udacity.com.

[Barber(2012)] BARBER, D. *Bayesian Reasoning and Machine Learning*, 2012.

[Stone(2005)] STONE, P. a SUTTON, R. S. a KUHLMANN, G. *Reinforcement Learning for RoboCup Soccer*. 2005.

[Pan, Chitta, Manocha] PAN, J. a CHITTA, S. a MANOCHA, D. *Probabilistic Collision Detection between Noisy Point Clouds using Robust Classification*.