Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Evolvovanie riadenia pohybu mobilného robota v neznámom prostredí

Diplomová práca

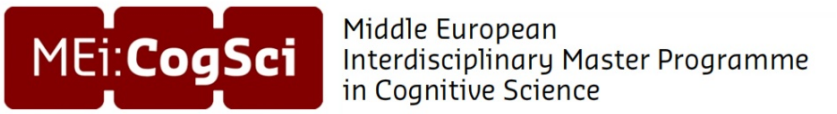
2013 **Ing. Filip Tóth**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra aplikovanej informatiky

Evolvovanie riadenia pohybu mobilného robota v neznámom prostredí

Diplomová práca

Študijný program:	Kognitívna veda
Študijný odbor:	9.2.11 Kognitívna veda
Školiace pracovisko:	Katedra aplikovanej informatiky FMFI UK
Vedúci práce:	doc. Ing. Igor Farkaš, PhD.
Konzultantka:	RNDr. Kristína Rebrová



Bratislava, 2013 **Ing. Filip Tóth**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Filip Tóth
Študijný program: kognitívna veda (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.11. kognitívna veda
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Evolvovanie riadenia pohybu mobilného robota v neznámom prostredi

Cieľ: Navrhnite a implementujte učiaci mechanizmus mobilného robota na báze kombinácie genetického algoritmu a učenia s posilňovaním, ktorý na základe signálov zo senzorov umožní cielený pohyb robota v neznámom priestore. Systém otestujte v simulovanom prostredí.

Vedúci: doc. Ing. Igor Farkaš, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Dátum zadania: 09.11.2010

Dátum schválenia: 15.01.2012

prof. RNDr. Pavol Zlatoš, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

Ďakujem môjmu školiteľovi doc. Ing. Igorovi Farkašovi, PhD. a konzultantke RNDr. Kristíne Rebrovej za študijné materiály, konzultácie a ostatnú odbornú pomoc.

Abstrakt

Výskum v oblasti mobilnej robotiky sa v poslednej dobe stáva objektom intenzívneho záujmu. Úsilie sa venuje vývinu samostatného inteligentného správania robota, špeciálne v neznámom prostredí. Pre študovanie rozličných prístupov riadenia používa výpočtová kognitívna veda resp. umelá inteligencia rôzne prístupy. Hlavným cieľom diplomovej práce je návrh a implementácia učiaceho mechanizmu mobilného robota na báze kombinácií genetického algoritmu a učenia posilňovaním, ktorý na základe signálov zo senzorov umožní cielený pohyb robota v neznámom priestore. Výhodou použitia tohto systému je schopnosť autonómne si vytvárať správanie. Pre tento systém bolo vytvorené simulačné prostredie, kde sa pohybuje virtuálny mobilný robot. Model virtuálneho robota sme navrhli na základe fyzicky existujúceho prototypu. Z hľadiska kognitívnej vedy sú použité algoritmy inšpirované prírodou a tiež veľmi dôležitou schopnosťou u živých organizmov učiť sa. Tu vzniká aj signifikantná potreba správne motivovať učiaci mechanizmus, od ktorého na konci požadujeme určité znalosti. V teoretickej časti uvádzame prehľad súvisiacich prác a použitých metód ako aj biologickú motiváciu pre takéto systémy. Navrhnutý učiaci mechanizmus bol úspešne implementovaný do simulačného modelu a v závere sa nám ho v obmedzenom rozsahu podarilo otestovať aj v reálnom prostredí. Vo výsledku bol mobilný robot riadený týmto systémom schopný sa v neznámom prostredí učiť princípy vyhýbania sa prekážkam a sledovania stien. Navrhnutý algoritmus pracuje v reálnom čase a vykazuje významné učiace atribúty. Pre tento typ úlohy hodnotíme použitý systém ako veľmi vhodný.

Kľúčové slová: robotika, učenie posilňovaním, genetický algoritmus, LCS (učiaci sa klasifikačný systém).

Abstract

The research into mobile robotics has been receiving more and more attention recently. To study various approaches to control, computational cognitive science as well as artificial intelligence use various methods. A lot of effort is put into the development of autonomous intelligent behavior for robots, especially for use in an unknown environment. The goal of this work is the implementation of a learning mechanism for a mobile robot based on a mix of a genetic algorithm and reinforcement learning, which can based on sensor data, enable the movement of a robot in an unknown environment. The advantage of this system is that it enables the autonomous definition of its behavior. A simulation environment was created for this system in which a virtual robot can move. This model is based on an actual existing prototype. The project uses machine nature-inspired algorithms that equip the agent with a very important ability - the ability to learn. This creates a significant need to motivate the mechanism, from which we require certain knowledge, correctly. In the theoretical part we provide a walkthrough through some related work as well as used methods. The designed learning mechanism was successfully implemented in a simulated model and in the end we successfully used it in a real world environment. The designed algorithm allows a real robot to learn to navigate in an unknown environment and to learn the principles of collision avoidance. Designed algorithm operates in real time and it exhibits significant learning properties. We conclude that the used algorithm is appropriate for this type of task.

Keywords: robotics, reinforcement learning, genetic algorithm, learning classifier system.

Obsah

Zoznam obrázkov

Úvod

Robotika prešla od svojho vzniku už viac ako 50-ročným vývojom a nepochybne sa v blízkej budúcnosti stane súčasťou každodenného života mnohých z nás. Ešte v nedávnej minulosti roboty kraľovali hlavne v priemyselnej doméne, kde stále vynikajú svojou donekonečna opakovateľnou precíznou prácou. V súčasnosti robotika postupne preniká do komerčnej sféry a už dávno nie je len výsadou vedeckých pracovníkov. Tento prienik je spôsobený veľkým pokrokom vo všeobecných parciálnych potrebách robotov, ako napríklad výkonnejšie a energeticky nenáročné riadiace systémy, komplexnejšie senzory a efektívnejšie pohony. Pokrok v spomenutých systémoch je priamo úmerný vývoju v oblasti mobilnej robotiky. Mobilné roboty našli uplatnenie všeobecne v prieskumných úlohách alebo aj vo variante robotických vysávačov, čí robotizovaných kosačiek. Okrem veľkého množstva úloh, ktoré musia mobilné roboty zvládať je kľúčovou vlastnosťou ich autonómnosť. Čo je to autonómia a ako sa robot s touto vlastnosťou správa? Autonómnosť pochádza z Gréckeho slova $\alpha\upsilon\tau\phi\nu\phi\mu\phi\sigma$ teda *autos* - sám a *nomos* – zákon (riadiaci sa vlastnými zákonmi) (autonomia, 2013). Z hľadiska robotiky to znamená, že robot má vykonávať všetky určené úlohy a ciele, vzhľadom na okolie samostatne s možnosťou si sám sebe stanovovať „zákony" a to všetko bez zásahu človeka. Základné úlohy, ktoré musí každý mobilný robot vykonávať sú: schopnosť pohybovať sa a orientovať sa v neznámom prostredí. Nasledujúce kapitoly tejto práce sa budú venovať hlavne schopnosti pohybovať sa. Aby bolo možné autonómne spĺňať tieto úlohy, mobilný robot musí mať v každom časovom okamihu odpovede na tieto otázky:

- *Kde som?*
- *Kam chcem ísť?*
- *Ako sa tam dostanem?*

Fundamentálne úlohy, ktoré v robotike odpovedajú na tieto tri otázky sa nazývajú lokalizácia a navigácia (Duchoň, 2012).

„Lokalizácia je pradávna aktivita, ktorá sa objavila všade tam, kde sa človek potreboval orientovať a pohybovať sa v prostredí. Lokalizácia predstavuje súbor

úloh, ktoré vedú k stanoveniu miesta alebo polohy objektu v prostredí“ (Duchoň, 2012). Na snahu zodpovedať otázku „kde je robot“ je možné dať rôzne odpovede. Prvá z nich môže byť takzvaná geometrická lokalizácia. Tento spôsob vyjadrovania lokalizácie robota nie je závislý od charakteristiky prostredia a dáva nám presné geometrické informácie. Súčasťou geometrickej informácie sú aj niektoré ďalšie parametre a vyjadrenie nepresnosti. Táto forma lokalizácie môže byť použitá vo všetkých druhoch prostredia no nie je to spôsob, ktorý používa človek (ani zvieratá), teda z kognitívneho hľadiska nie je táto metóda relevantná. Ďalšou možnosťou je vyjadriť lokalizáciu robota významom vzťahu s prostredím, napríklad: robot je miestnosti A, pred dverami. Tento druh je nazývaný topologická lokalizácia. Táto metóda sa blíži spôsobu, ktorý používajú ľudia. Nevýhodou je fakt, že metóda vyžaduje určitý druh prostredia s bohatou typologickou informáciou s cieľom dosiahnuť nejakú úroveň presnosti. Existujú však jednotvárne prostredia, ako napríklad moria a púšte, pre ktoré je spomínaná metóda zvlášť nevhodná (Duchoň, 2012). Ľudia pravdepodobne používajú kombináciu viacerých metód, so znalosti o prostredí a taktiež majú hrubé odhady o posune, aby vedeli ich presnú polohu v každom momente, napr. robot je v miestnosti A a je blízko dverí. Existujú rôzne riešenia zaoberajúce sa problémom lokalizácie, ktoré možno zhruba rozdeliť na tvrdo výpočtové a ďalšie berúce do úvahy význam a členenie prostredia, ktoré sú z hľadiska kognitívnej vedy biologicky relevantné. Vzhľadom na nedostatky uvedených metód a neexistujúcej generálne použiteľnej metódy, mobilné roboty zvyčajne kombinujú jednu alebo viac metód z každého druhu.

„Navigácia vo všeobecnosti predstavuje súbor úloh, ktoré dávajú objektu pohybujúcemu sa v prostredí odpovede na otázky „Kam chcem ísť?“ a „Ako sa tam dostanem?“ (Duchoň, 2012). Základná schopnosť mobilnej robotiky je navigácia. Pre najambicióznejšie vyjadrenie významu navigácie je možné vykonať nasledujúci test: robot je umiestnený v prostredí, ktoré je pre neho neznáme, veľké, komplexné a dynamicky sa meniace. Po čase, ktorý robot potrebuje na preskúmanie, musí byť schopný ísť na akékoľvek zvolené miesto, pričom sa snaží minimalizovať aktuálne zvolenú fitnes (učelovú) funkciu, napríklad čas, trajektóriu, energiu, atď. Ak zhrnieme tieto (úspešne zrealizovné) vlastnosti môžeme povedať, že robot sa správa optimálne. Pre lepší opis navigácie a problémov s ňou spojených sa zvykne používať takzvaný maximálny navigačný test – MNT (Salichs a Moreno, 2000). Aby robot uspel v MNT, musí vyriešiť niekoľko problémov. V prvom probléme robot musí byt robot schopný pohybu dostatočne rýchlo, kontrolovane a bezpečne, zvládnuť obchádzanie statických a pohyblivých prekážok (motion control problem). Počas pohybu musí byť robot schopný zhromažďovať poznatky o prostredí (world modeling problem) a byť si vedomý svojho umiestnenia (localization problem). Ak sa robot pohybuje po veľkej ploche mal by svoj pohyb plánovať, aby bol čo najefektívnejší (planning problem). Opísané problémy by mali byť považované iba za čiastkové problémy

navigácie. Všeobecným problémom navigácie nie je suma čiastkových problémov, ale interakcie medzi nimi (architecture problem) (Salichs a Moreno, 2000). Existuje mnoho inžinierskych metód navigácie mobilných robotov, no v súčasnosti čoraz viac zaznamenávajú pokroky aj metódy spadajúce do oblasti umelej inteligencie, využívajúce prvky umelej inteligencie, ako je výpočtová inteligencia (neurónové siete, fuzzy logika či evolučné algoritmy) (Arkin, 1989).

V teoretickom úvode uvádzame výsledky niekoľkých prác, ktoré sa týkajú týchto metód. Naša práca pre navigáciu využíva učiaci sa systém na báze kombinácii genetického algoritmu a učenia s posilňovaním. Hlavnú časť tvorí návrh s podrobným technickým aj teoretickým opisom jeho častí. V nami navrhnutom simulačnom prostredí týmto systémom riadime mobilného robota a v závere prezentujeme dosiahnuté výsledky.

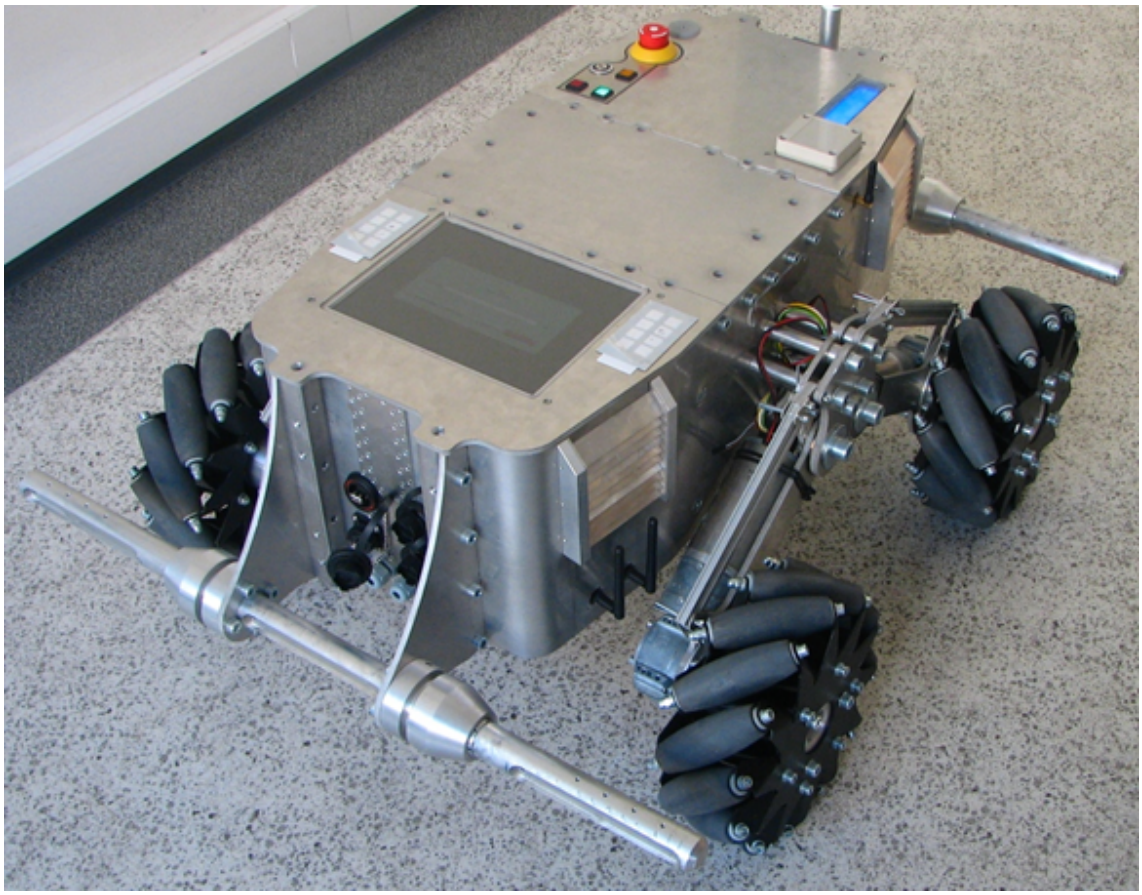
Kapitola 1

Teoretický úvod

1.1 Ciele práce

Cieľom práce je návrh a implementácia učiaceho mechanizmu mobilného robota na báze kombinácií genetického algoritmu a učenia s posilňovaním, ktorý na základe signálov zo senzorov umožní cielený pohyb robota v neznámom priestore. V tejto kapitole sa nachádza aj prehľad súvisiacej problematiky, ktorý napomôže k lepšiemu zvládnutiu zadanej úlohy. Ďalej popisujeme virtuálne prostredie s mobilným robotom, ktoré sa snaží verne modelovať správanie fyzicky existujúceho prototypu (Tóth, 2011) na obrázku 1.1.

Nami vytvorené virtuálne prostredie bude obsahovať nastavovacie prvky, ktoré umožnia rôzne konfigurovať prostredie a inicializovať vlastnosti robota pred štartom. Mobilný robot ovládaný naším učiacim sa mechanizmom by mal byť schopný zvládať základné úlohy navigácie, ktoré sú obchádzanie prekážok, sledovanie ohraničenia priestoru, či dosiahnutie cieľovej pozície. Kľúčový systém, použitý v tejto práci sa volá *Learning Classifier System* (LCS) a je to algoritmus, ktorý spája evolvovanie a učenie s posilňovaním, pochádzajúci s oblasti prírodne inšpirovaných algoritmov (Brownlee, 2011).



Obr. 1.1: Mobilný robotický systém

1.2 Prehľad súvisiacej problematiky

V tejto kapitole je uvedený prehľad výskumu v oblasti navigácie mobilného robota s využitím genetického algoritmu, učenia s posilňovaním a LCS.

1.2.1 Navigácia mobilného robota s genetickým algoritmom

Moderné technológie vyžadujú robotov, ktorí sa dokážu pohybovať aj v dynamicky meniacom sa prostredí. Príkladom takejto aplikácie môže byť využitie robotov v priemysle pre prepravu náradia a ďalších materiálov z jedeného miesta na druhé. V súčasnosti sa takéto roboty bežne používajú vo výrobných halách najmä automobiliek, ale nevyužívajú žiadnu umelú inteligenciu, pretože na svoj pohyb majú vyhradený koridor so značným množstvom magnetického/optického značenia a ľudia tam nemajú povolený vstup. Pretože mnoho robotov v budúcnosti bude pracovať spoločne, je potrebné zabezpečiť voľnú navigáciu pre každého mobilného robota zvlášť.

Funkcionalita a biologické pozadie genetického algoritmu budú podrobne opísané v nasledujúcej v druhej kapitole.

Autori Kala a kol. (2009) boli motivovaní problémom robotickej navigácie a v ich práci riadili dvojkolesového mobilného robota pomocou genetického algoritmu, neurónovej siete a algoritmom A*, a to v každom časovom okamihu cesty robota. Ich riešenie môže byť použité pre vyslanie robota na prieskum, zber dát alebo na určitú špecifickú prácu. Bezkolízny pohyb robota pohybujúceho sa v prostredí s prekážkami môže byť použitý vo svete robotov aj ľudí. Výsledky ukazujú, že všetky tri algoritmy (genetický, neurónová sieť, algoritmus A*) sú schopné riadiť pohyb robota bez kolízii (Kala a kol., 2009). Pre našu pracú je však zaujímavé iba použitie genetického algoritmu.

Predstavme si situáciu s veľa robotmi a prekážkami pohybujúcimi sa neustále a konštantne, ako ľudia v nákupnom centre, či v metre. Problém je riadiť pohyb robota zo štartovacej do finálnej pozície (Ragavan a Ganapathy, 2007). Potrebujeme optimalizovať cestu a taktiež potrebujeme zabezpečiť, aby robot nebol v rozpore s inými prekážkami. Preto Kala a kol. (2009) potrebovali robota, ktorý používa navigačný plán. Ich práca používa genetický algoritmus na to, aby robot zistil optimálnu cestu v každom časovom okamihu. Algoritmus skúša nájsť najlepšie riešenie kombináciou existujúcich riešení. Robot sa snaží nájsť cestu, ktorá ho optimálne dovedie do cieľovej pozície. Na to používajú jednoduchú jednotku pohybu. Na ďalšiu jednotku času je opäť potrebné spustiť algoritmus, ktorý vypočíta ďalší jednoduchý pohyb. Preto v každom časovom okamihu beží algoritmus a určuje ďalší pohyb robotovi. Robot sa fyzicky pohybuje iba podľa týchto výsledkov. Autori vychádzajú z toho, že každý pohyb nezaručuje menšiu vzdialenosť k cieľu no na konci robot cieľ dosiahne (ak je možný). V prípade, ak ho nie je možné dosiahnuť, robot začne byť stacionárny.

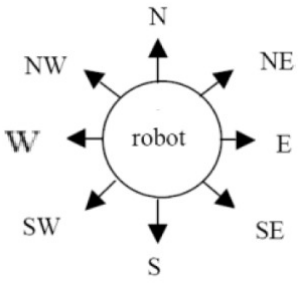
Základný riadiaci model použitý v ich práci zahŕňa tri hlavné cykly. Medzi nimi cyklus vnem–akcia je najčastejšie používaný v mobilných robotoch. Tento cyklus najprv lokalizuje prekážku a potom generuje riadiace príkazy pre nastavenie motorov, teda pre pohyb robota. Druhý cyklus prechádza od vnímania do plánovania stavov, zatiaľ čo tretí zahŕňa všetky možné stavy vrátane vnímania, plánovaní a akcií. Ich robot používa pre orientáciu ultrazvukové senzory a kameru (Kala a kol., 2009).

Autori nadizajnovali robota s dvoma kolesami. Preto sa môže pohybovať vpred/vzad (obe kolesá sa točia v rovnakom smere), alebo sa točia v smere/proti smeru hodinových ručičiek (kolesá rotujú v opačných smeroch). Pre potreby algoritmu bolo kvantizovaných zopár druhov pohybov, ktoré sú platné a môžu byť vykonávané s malým časovým oneskorením:

- pohyb vpred (jednotkový krok)
- pohyb z aktuálnej pozície o uhol 45° a vpred (jednotkový krok)

- otočenie sa v smere alebo protismere h.r. o 45°
- otočenie sa v smere alebo protismere h.r. o 90°

Autori z uvedeného predpokladajú, že sa robot otáča a pohybuje len v násobkoch 45°. Preto môžu pohybovať robotom v smeroch: sever–juh, východ–západ, severovýchod–západ a juhovýchod–západ, tak ako je uvedené na obrázku 1.2.



Obr. 1.2: Výsledné možné variácie pohybov (Kala a kol., 2009)

Pre potreby algoritmu ďalej pohyby očíslovali: S – 0, SV – 1, V – 2, JV – 3, J – 4, JZ – 5, Z – 6, SZ – 7 (Kala a kol., 2009).

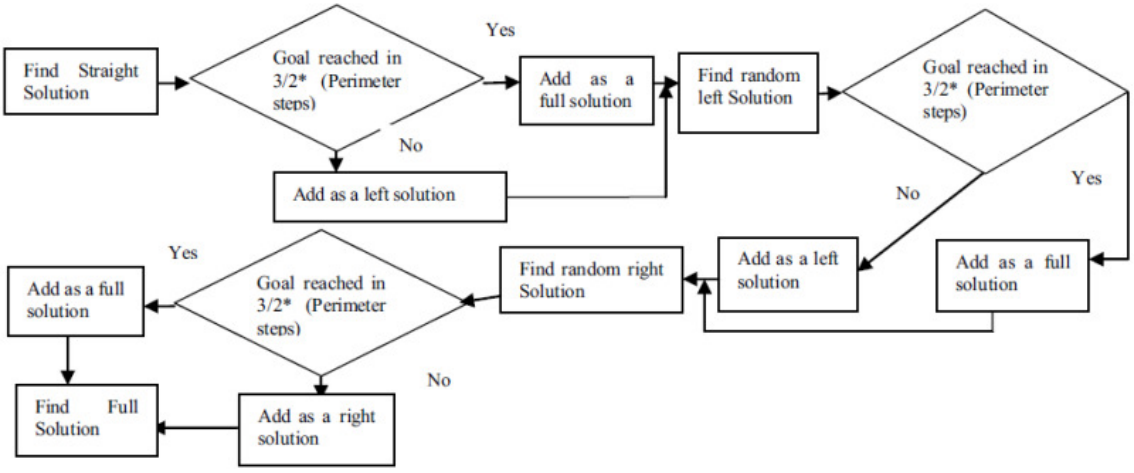
V uvedenej práci autori predpokladajú, že cesta robota je z počiatočného do finálneho stavu zložená z radu bodov, na ktoré sa robot dostane. Každý bod obsahuje tri atribúty (x, y, d), to sú jeho súradnice x, y a smer d, v ktorom robot stojí (od 0 do 7). Finálne riešenie je udané vo forme (x_1, y_1, d_1) (x_2, y_2, d_2) (x_3, y_3, d_3) ... (x_n, y_n, d_n). Každé takéto riešenie je uložené v súbore riešení, pričom existujú tri typy riešení. Prvý typ obsahuje kompletné riešenie a plné prepojenie od štartu k cieľu. Druhý obsahuje začiatok od štartu, ale nie je schopné dosiahnuť cieľ. V treťom absentuje začiatok, no dosahuje cieľ. Každý bod (x, y, d) je reprezentovaný ako uzol v grafe a môže byť spojený s ďalšími uzlami. Ako v svojej práci popisujú autori, existujú len *regulárne prepojenia* jedného uzla s iným. Napríklad, platné prepojenie je (0, 0, 0) s (0, 0, 3) alebo (25, 69, 8) s (25, 69, 6), teda môže meniť smer v tej iste polohe. Ale (0, 5, 0) s (0, 0, 3) alebo (0, 0, 3) s (1, 1, 1) sú neplatné prepojenia, teda toto spojenie nie je možné (Kala a kol., 2009).

Evaulačnou funkciou merajú Rahul a Shukla fitnes chromozómov. Cieľom je nájsť chromozóm s čo najnižšou fitnes. V práci vytvorili nasledovné vzorce:

- *kompletné riešenie*: pre dosiahnutie cieľa nie sú potrebné žiadne kroky
- *ľavé riešenie*: počet krokov potrebných k prejdeniu od zvoleného do posledného bodu + kvadrát vzdialenosti od tohto bodu do cieľa + $R(n)$
- *pravé riešenie*: kvadrát vzdialenosti od zvoleného bodu do prvého bodu + $R(n)$ + počet krokov potrebných k prejdeniu od tohto do posledného bodu.

Pričom $R(n)$ je minimálny potrebný čas pre rotácie v celej ceste robota a cieľom je, aby sa otáčal čo najmenej, čo prispeje aj k plynulejšiemu pohybu.

Samotný algoritmus sa po spustení snaží nájsť počiatočné riešenie. Kompletné riešenie autori vysvetľujú na obrázku 1.3.



Obr. 1.3: Diagram krokov pre generovanie počiatočného riešenia (Kala a kol., 2009)

Diagram môžeme popísať aj nasledujúcimi bodmi:

- nájdi priamu cestu medzi zdrojovým umiestnením a cieľom
- nájdi náhodné „ľavé“ riešenie medzi zdrojom a cieľom
- nájdi náhodné „pravé“ riešenie medzi zdrojom a cieľom
- nájdi kompletné riešenie

V procese kríženia autori kombinujú dve známe riešenia (rodičia) a tým vytvárajú nové. Nové riešenie môže byť lepšie, aké reprezentovali rodičia. Kríženie môže nastať ako výsledok z nasledujúcich dvojíc:

- úplné riešenie a úplné riešenie
- „ľavé“ a úplné riešenie
- úplné a „pravé“
- „ľavé“ a „pravé“ riešenie

Vo výsledku sú dve sekvencie bodov (x, y, d). Jedna sekvencia je vybraná z ľavej časti vyššie uvedeného riešenia druhá z pravej časti. Krížiaci proces je podobný tomu, ktorý sa používa v počiatočnom procese. Napríklad vybraná ľavá sekvencia je: (x_{11}, y_{11}, d_{11}) (x_{12}, y_{12}, d_{12}) ..... (x_i, y_i, d_i) ($x_{1i}+1$, $y_{1i}+1$, $d_{1i}+1$) ..... (x_{1n}, y_{1n}, $d_{|1n}$) a vybraná pravá sekvencia je: (x_{21}, y_{21}, d_{21}) (x_{22}, y_{22}, d_{22}) ..... (x_i, y_i, d_i) ($x_{2i}+1$, $y_{2i}+1$, $d_{2i}+1$) ..... (x_{2n}, y_{2n}, d_{2n}). Spojené kompletné riešenie teda bude: (x_{11}, y_{11}, d_{11}) (x_{12}, y_{12}, d_{12}) ..... (x_i, y_i, d_i) ($x_{2i}+1$, $y_{2i}+1$, $d_{2i}+1$) ..... (x_{2n}, y_{2n}, d_{2n})

Jednoducho povedané bola vybraná ľavá časť z ľavej sekvencie a pravá časť z pravej sekvencie. Konečné riešenie je pridané do množiny riešení, ale iba ak je kríženie možné. Ak existuje viac ako jeden spoločný bod, kríženie sa vykonáva pre všetky body a na konci sú zase pridané do množiny riešení (Kala a kol., 2009). V procese mutácie autori skúšali nájsť náhodné riešenie z riešení, ktoré vznikli krížením. Aby sa vyhli lokálnym minimám, zaviedli autori do výpočtu náhodu. Tento proces prebieha s pravdepodobnosťou 0,02 a v závere je zvolené jedno kompletné riešene. Potom sa vezmú ľubovoľné dva body a pomocou podobného algoritmu sa pokúšajú zistiť cestu medzi nimi. Ak takáto cesta existuje, potom sa nahradí pôvodná novou a ak je viac riešení vyberie sa tá, s najlepším ohodnotením fitnes funkcie (Kala a kol., 2009). Autori v závere zhodnotili, že genetický algoritmus navrhnutý pre problém navigácie, bol schopný doviesť robota zo začiatočnej pozície do cieľa. Algoritmus však potreboval viac času pre vygenerovanie počiatočných riešení, ale generoval dobré výsledky, ktoré sa zlepšovali po každej iterácií.

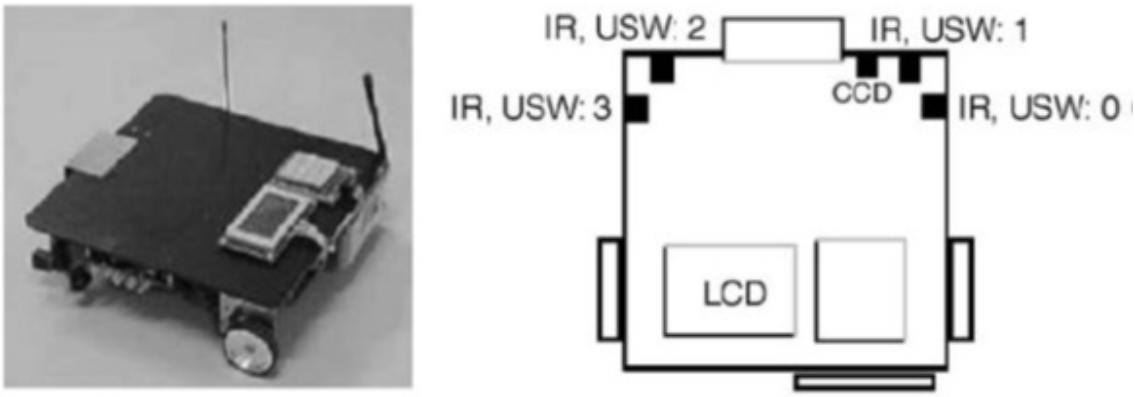
1.2.2 Určovanie parametrov pomocou genetického algoritmu

Práca od autorov Kamei a Ishikawa (2004) sa zaoberá otázkou ako genetickým algoritmom optimálne nastaviť parametre pre učenie s posilňovaním (reinforcement learning - RL), pričom tento algoritmus používa navigácia mobilného robota.

Učenie s posilňovaním je veľmi užitočné pri problémoch robotickej navigácie, ide hlavne o obchádzanie prekážok a plánovanie cesty (Sutton a Barto, 1998). Algoritmus učenia s posilňovaním vyžaduje pre svoju činnosť nastavenie rôznych parametrov a autori sa tieto parametre nastavujú pomocou genetického algoritmu, ktorý samotné učenie urýchli až o 30% a počet akcií mobilného robota potrebných pre dosiahnutie cieľa sa zníži na polovicu (Kamei a Ishikawa, 2004).

V štúdii autori využívajú malého mobilného robota s názvom TRIPTERS, ktorý je vybavený infračervenými a ultrazvukovými senzormi (Obr. 1.4). Ultrazvukové snímače merajú kontinuálne do 0,8 m, ich lúč je charakterizovaný ako úzky. Infračervené majú väčší rozptyl a tak lepšie detegujú prekážky, no ich výstup je však binárny, teda ak je vzdialenosť menšia ako 0,7 m – výstup je log „1“ a ak viac, tak výstup je log „0“. Oproti robotovi, ktorý používali autori Kala a kol. (2009) v predošlej práci

sa tento robot nemôže otáčať na mieste, ale pri pohybe môže len mierne zabáčať doľava alebo doprava (Kamei a Ishikawa, 2004).



Obr. 1.4: Mobilný robot TRIPTERS s infračervenými a ultrazvukovými senzormi (Kamei a Ishikawa, 2004)

.

Algoritmus učenia s posilňovaním generuje optimálnu cestu k cieľu a odpovedá za sekvenciu akcii vyvolávanú interaktívnym učením, pričom agent dostáva z prostredia odmenu. Učenie s posilňovaním využíva rôzne parametre, ako sú:

- *veľkosť kroku* (α), podobne ako rýchlosť učenia s učiteľom
- *diskontná rýchlosť* (γ)
- *"hladná"stratégia* (ε)
- *odmena za akciu* (r)

Učenie s posilňovaním je zamerané na maximalizovanie sumy znížených cien budúcich odmien. Existuje viac druhov RL, no robot používa *Q-learning*, ktoré opakovane aktualizuje hodnotu $Q(s_t, a_t)$, kde Q je dvojica stavu s a akcie a ako

$$Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \alpha[r_{t-1} + \gamma_{max_{\alpha t}} Q(s_{t+1}, \alpha_{t+1}) - Q(s_t, \alpha_t)] \qquad (1.1)$$

kde s reprezentuje umiestnenie a orientáciu robota, α zodpovedá akcii, a r je odmena z okolitého prostredia. V genetickom algoritme, má každý jednotlivec svoj chromozóm, ktorý je hodnotený fitnes funkciou. Chromozóm predstavuje *step-size* parameter α, *greedy* stratégiu parameter ε, a odmenu r. (Greedy stratégia je optimalizačná technika, ktorej cieľom je nájsť najlepšie riešenie). Všetky uvedené parametre sú v lineárnom meradle, ale *discount rate* γ je reprezentovaná logaritmicky:

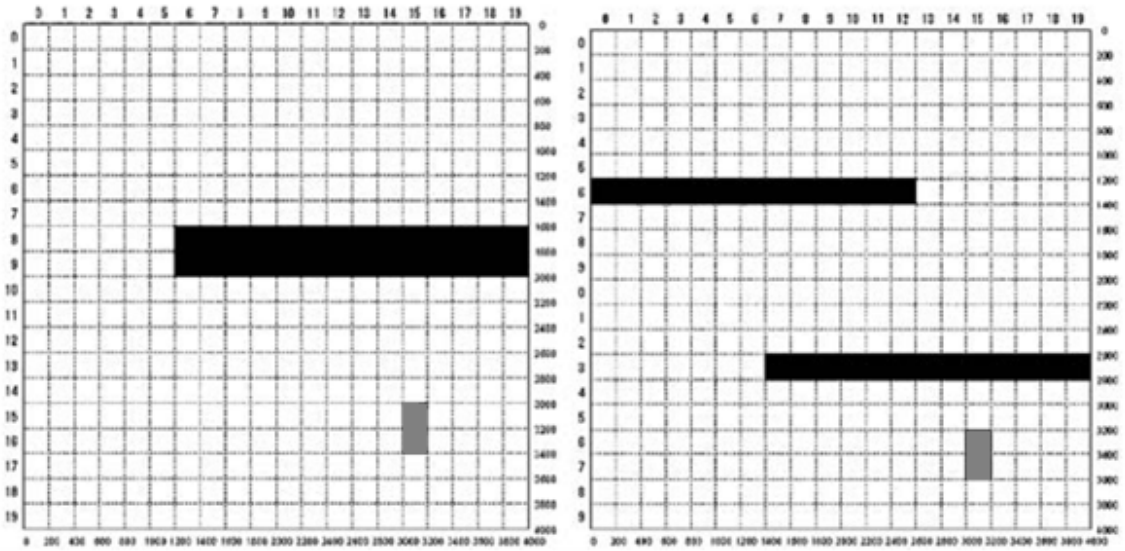
$$\gamma = 1 - 10^{-x} \tag{1.2}$$

kde x je reprezentácia v génovom lokuse (úsek chromozómu obsahujúci jeden gén). Autori určili dĺžku chromozómu na 30 bitov so šiestimi bitmi pre každý parameter. Lokus génu v chromozóme ma hodnotu od 0 do 63. Fitnes funkcia pre ohodnotenie jednotlivca je:

$$f = \left(1 - \frac{N_\alpha}{N_{max}}\right) - (N_e - N_g) \tag{1.3}$$

kde N_a je počet úspešných akcií v epizódach, N_{max} je horná hranica akcií v epizóde vynásobená počtom uspených akcií, N_e je počet všetkých epizód a N_g je počet úspešných epizód (Kamei a Ishikawa, 2004).

Predpokladá sa, že robot pozná svoje umiestenie a orientáciu. V počítačovej simulácii je robot umiestnený náhodne, teda na začiatku každej epizódy RL sú umiestenie a orientácia vyberané náhodné. Epizóda sa končí, ak robot dosiahne cieľ, ak narazí do prekážky, alebo ak počet akcií dosiahne 250, čo je horná hranica. Stav robota je reprezentovaný umiestnením v mriežke 20x20 a uhol natočenia je rozdelený do ôsmich sektorov po 45°, pričom robot môže vykonávať tri akcie ktoré sú: pohyb dopredu, pohyb doprava o 10° a doľava o 10°. Na obrázku 1.5 je znázornené prostredie ktoré, vytvorili autori pri ich experimente. Použitá plocha predstavuje 4 x 4 m a jeden štvorec má rozmer 20 x 20 cm (Kamei a Ishikawa, 2004).



Obr. 1.5: Použité prostredia pre mobilného robota, čierne štvorce sú prekážky a sivé sú ciele (Kamei a Ishikawa, 2004)

Každý lokus génu bol vyberaný s pravdepodobnosťou 10% a následne kríženie tiež prebiehalo s pravdepodobnosťou 10%. Pri inicializácii bolo vygenerovaných 50 jednotlivcov, z ktorých každý je ohodnotený fitnes funkciou, následne bolo vygenerovaných 25 nových. Zo 75 jednotlivcov bolo vybraných 50 s najväčším ohodnotením. Tento postup bol opakovaný 100 generácií. Učenie v 50-tich epizódach štartuje s Q hodnotami, po 20 000 epizódach robí RL s určitým súborom parametrov. Použitý genetický algoritmus, ktorý nastavoval parametre RL sa javil ako veľmi nápomocný, hlavne v zrýchlení učenia o 30%. Tiež suma potrebných akcií robota na dosiahnutie cieľa klesla na polovicu (Kamei a Ishikawa, 2004).

1.2.3 Učenie robota pomocou LCS

Veľmi dobrá práca od autorky (Jabin, 2010) ponúka východiskový teoreticky základ systému LCS ako takého. Je v nej opis historického vzniku LCS ale aj základne matematické ci výpočtové mechanizmy potrebne k LCS ako je genetický algoritmus, Markovovský rozhodovací proces a učenie posilňovaním čo bolo veľmi užitočné pre túto diplomovú prácu. Autorka ďalej vysvetľuje rôzne podskupiny LCS s orientáciou pre reálne robotické aplikácie ako sú *Pittsburgh* a *Michigan* či *ANIMAT* klasifikačný systém.

Kapitola 2

Použité metódy

Základ kognitívneho systému, ktorý predstavujeme v kapitole 4, tvoria evolučné algoritmy. Preto v tejto kapitole uvádzame ich teoretický opis a tiež podrobný opis LCS (Learning Classifier System), ktorý tvorí jadro robotickej navigácie v našom robotickom systéme.

2.1 Evolučné algoritmy

Evolučné algoritmy patria do oblasti evolučného počítania (*evolutionary computation*), ktoré sa zaoberá výpočtovými metódami, inšpirovanými procesmi a mechanizmami biologickej evolúcie. Proces evolúcie je daný prirodzeným výberom a Charles Darwin bol prvý, ktorý odomkol záhadu evolúcie, spočívajúcu v prirodzenom výbere, čiže prežití najschopnejšieho. Pri jeho skúmaniach rôznych druhov zvierat pozorovaním zistil, že v každej populácii sa vyvinuli také znaky, ktoré v danom prostredí najlepšie zabezpečovali prežitie a tieto výhodné znaky sa odovzdávajú ďalšej generácii a jedinec, ktorý sa neprispôsobí, vyhynie (Darwin, 1878). Mechanizmus evolúcie možno opísať tým, že evolúcia v skutočnosti prebiehala modifikáciami a rozširovaním genetického materiálu (Dawkins, 1976). Evolučné algoritmy sa zaoberajú skúmaním výpočtových systémov, ktoré sa podobajú na veľmi zjednodušené mechanizmy a procesy evolúcie s cieľom vytvárať adaptívne systémy. Evolučný algoritmus je heuristicky prístup, ktorý sa snaží nájsť riešenie zložitých problémov, ktoré sú natoľko komplexné, že exaktný algoritmus je nedostačujúci alebo neexistuje. Existuje aj mnoho iných domén spadajúcich do výpočtovej oblasti evolučných algoritmov, ktoré sa snažia využiť vlastnosti z príbuzných oborov, ako je populačná genetika, populačná ekológia, atď.

Evolučné algoritmy zdieľajú vlastnosti adaptácie pomocou iteračného procesu, ktorý kumuluje a zosilňuje pozitívne zmeny prostredníctvom pokusov a omylov. Kandidátske riešenia sú reprezentované členmi virtuálnej populácie. Táto populácia

sa snaží prežiť v danom prostredí, pričom problém je špecifikovaný pomocou účelovej funkcie. V každom prípade, evolučný proces adaptuje populáciu kandidátskych riešení v prostredí, kde typickým procesom evolúcie sú genetické rekombinácie a mutácie (Bäck a kol., 2000).

V súčasnosti sa používa veľa druhov, modifikácii a metód evolučných algoritmov, ako sú napríklad:

- *Geneticky algoritmus* – je inšpirovaný populačnou genetikou vrátane dedičnosti, génových frekvencií a vývoja úrovne populácie, ako sú chromozómy a gény. Jeho hlavné mechanizmy sú rekombinácia a mutácia. Niektorí autori tento algoritmus nazývajú aj ako moderná syntéza evolučnej biológie. Geneticky algoritmus je považovaný za adaptívnu stratégiu a globálnu optimalizačnú techniku, ktorá je základným stavebným kameňom nižšie uvedených bodov. Princíp práce genetického algoritmu je postupná tvorba generácii rôznych riešení daného problému. Pri riešení sa vytvára populácia jedincov, z ktorých každý prestavuje riešenie daného problému. Riešenia sa zlepšujú počas toho, ako populácie prebiehajú evolúciou. Tradičné výpočtové riešenie je reprezentované binárnymi číslami, no používajú sa aj polia, stromy a matice (Brownlee, 2011).

- *Evolučné stratégie* – sú inšpirované teóriou evolúcie, hlavne prirodzeným výberom, konkrétne technikou na makro úrovni evolúcie, ako je fenotyp, dedičnosť a variačnosť. Nezaoberajú sa však genetickými mechanizmami, ako je génom či chromozóm (Brownlee, 2011).

- *Diferenčné evolúcie* – je to stochastické priame vyhľadávanie a globálna optimalizačná metóda (Brownlee, 2011).

- *Gramatická evolúcia* – je inšpirovaná biologickým procesom používaným pri generovaní proteínov z genetického materiálu. Genóm sa skladá z DNA (genetický stavebný materiál v bunkách), ako reťazec stavených blokov, ktorý je zapísaný do RNA (nukleová kyselina). RNA sú postupne preložené do sekvencie aminokyselín a sú použité v proteíne. Výsledný proteín je vo svojom prostredí *fenotypom* (súhrn všetkých dedičných znakov organizmu) (Brownlee, 2011).

- *Genómové programovanie* alebo aj *genetické programovanie* – je inšpirované replikáciou a expresiou DNA molekuly, konkrétne na úrovni génov. Zahŕňa prepis DNA do RNA podobne, ako Gramatická evolúcia (Brownlee, 2011).

- *Učiaci sa klasifikačný systém (LCS)* – cieľom tohto systému je optimalizovať príspevok založený na pôsobení na špecifické prostredie. Toto je dosiahnuté

ohodnocovaním akcií na základe pravidiel, ktoré prechádzajú evolučným procesom. LCS je spojenie evolučného algoritmu a inštancie učenia s posilňovaním (Brownlee, 2011).

- Ďalej ešte existujú napríklad: Nedominantne riadený genetický algoritmus (NSGA), Sila Paretovho evolučného algoritmu a rôzne iné (Brownlee (2011), Bäck a kol. (2000)).

V ďalšej časti nás budú zaujímať základné mechanizmy učiaceho sa klasifikačného systému (LCS), a teda, genetický algoritmus a učenie s posilňovaním.

2.1.1 Genetický algoritmus

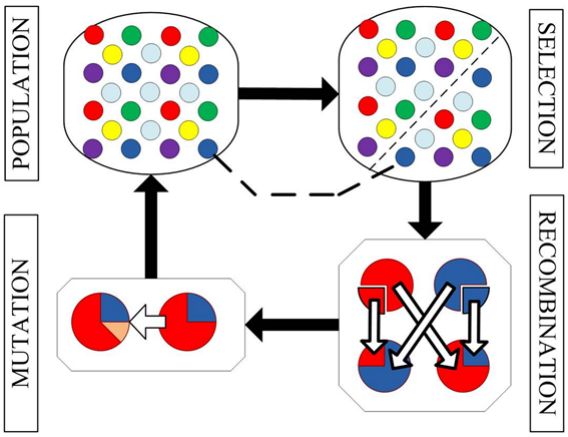
Prvý základný mechanizmus LCS je genetický algoritmus (GA), ktorý je inšpirovaný neo-darvinovskou teóriou prirodzeného výberu (Holland, 1975). Tento algoritmus manipuluje populáciou jedincov reprezentujúcich možné riešenie daného problému. GA sa opiera o analógie fungujúce v biológii:

- používajú kód
- genotyp alebo genóm;
- jednoduché operácie na tomto kóde – genetické operátory;
- proces selekcie pre prežitie najsilnejšieho.

Genetické operátory sú používané pre zavedenie niektorých variácii v genotypoch, kde sa využívajú dve triedy operátorov (Obr. 2.1):

- *kríženie* – vytvára genotypy rekombináciou podčasti genotypov dvoch, alebo viacerých jednotlivcov;
- *mutácia* – náhodne modifikuje genotypy v jednotlivcovi.

V procese selekcie sa extrahujú genotypy, ktoré si zaslúžia byť reprodukované, teda na ktorých sa budú vykonávať genetické operátory (kríženie a mutácia). GA manipuluje súborom ľubovoľne inicializovaných genotypov, ktoré sú vybrané a modifikované generáciu po generácii. Tie, ktoré nie sú vybrané sú eliminované. Pomocné funkcie alebo takzvané fitnes funkcie hodnotia mieru záujmu o fenotyp s ohľadom na daný problém. Prežite zodpovedajúceho riešenia, alebo počet potomkov v ďalšej generácii závisí od hodnotenia pomocných funkcií. Potomkovia jednotlivca sú vytvorení z kópie jeho genotypu, na ktoré sú aplikované genetické operátory (Goldberg a Holland, 1988), (Goldberg, 1989).



Obr. 2.1: Diagram operácii jednoduchého genetického algoritmu (Lancater, 2013)

Výsledok celkového procesu spočíva v iterácii nasledovného cyklu:

1. vyber genotypy podľa vhodnosti (fitness) zodpovedajúcich fenotypov;
2. na tieto genotypy aplikuj genetické operátory pre generovanie potomkov;
3. zostav fenotypy z týchto nových genotypov a vyhodnoť ich;
4. vráť sa na krok 1.

Ďalej už nebudeme rozoberať genetický algoritmus ako taký, ale viac sa zameriame na jeho použitie v učiacom klasifikačnom systéme. Napriek tomu, že genetický algoritmus je koreňom LCS, môžeme GA využívať len v obmedzenej miere a je dôležité opísať niektoré nasledujúce aspekty.

Je potrebné rozlišovať medzi jedno-bodovým operátorom kríženia (ktorý rozdeľuje dva genotypy do dvoch častí na náhodne vybranom mieste a následne tvorí nový genotyp prevracaním podčasti od rôznych rodičov) a mnoho-bodovým operátorom kríženia (ktorý rozdelí rodičovské genotypy na viac kusov). Pri vzniku LCS sa používali iba jedno-bodové operátory kríženia, no v súčasnosti rastie záujem o komplexnejšie stavebné bloky LCS, hlavne z dôvodu väčšej výpočtovej kapacity, čo vedie k potrebe používať aj mnoho-bodový operátor kríženia (vysvetlenie operácie kríženia v bode je vysvetlene časti 2.4.4) (Goldberg a Holland, 1988), (Goldberg, 1989) a (Montague, 1999).

Ďalej je tiež potrebné rozlišovať medzi generačným GA (kde je celá alebo dôležitá časť populácie vynovená predošlou) a ustáleným GA (kde sa menia len jednotlivci, jeden za druhým), teda nedeje sa generačne. Väčšina LCS používa ustálený genetický algoritmus, pretože tento mechanizmus je menej náchylný na šum a vedie k lepšej

súhre medzi evolučným procesom a procesom učenia (Goldberg a Holland (1988), Goldberg (1989)a Montague (1999).

Z pohľadu učiaceho sa klasifikačného systému, ktorý bude opísaný v časti 2.3 nás zaujíma binárna reprezentácia kanonického genetického algoritmu podľa Engelbrecht (2007). Väčšina operátorov kríženia pre binárne reprezentácie je založených na pohlaví, teda kríženie je aplikované z dvoch vybraných rodičov. Ak $x_1(t)$ a $x_2(t)$ označujú dvoch vybraných rodičov, tak potom rekombinačný proces je vyjadrený v pseudokóde 1.. V tomto algoritme, $m(t)$ maska špecifikuje, ktoré časti rodičov by mali byť vymenené pre vytváranie nových potomkov, teda $\overline{x_1}(t)$ a $\overline{x_2}(t)$ (Engelbrecht, 2007).

Algorithm 1 Geneticky algoritmus pre bitovo reprezentované kríženie (Engelbrecht, 2007)

```
Let x̄1(t) = x1(t) and x̄2(t) = x2(t)
if U(0,1) ≤ pc then
  Compute the binary mask, m(t);
  for j = 1, ..., nx do
    if mj = 1 then
      //swap the bits
      x̄1j(t) = x2j(t);
      x̄2j(t) = x1j(t);
    end if
  end for
end if
```

2.2 Učenie s posilňovaním

Druhý fundamentálny mechanizmus potrebný v LCS je učenie s posilňovaním (*reinforcement learning*). Z psychologického hľadiska nás najprv zaujíma učenie ako také. Základom každého učenia je podmieňovanie, kde pri učení podmieňovaním sa vytvárajú asociácie medzi reflexami. Ako prvý objavil a definoval klasický podmienený reflex významný ruský profesor fyziológie Ivan Petrovič Pavlov (Pavlov's Dogs, 2012). Klasické podmieňovanie je podľa Pavlova: „Asociácia podnetu, ktorý vyvoláva merateľnú odpoveď (nepodmienený podnet) s podnetom, ktorý bežne nevyvoláva merateľnú odpoveď (podmienený podnet), pričom podmienený podnet musí predchádzať nepodmienený. Podmieňovanie = párovanie – naučená odpoveď na podmienený podnet spôsobená modifikáciou synaptického prenosu" (Ostatníkova, 2011).

V znázornenom experimente sa páruje pohľad na potravu a zvuk zvončeka (najskôr zvon, hneď potom potrava). Ak pes vidí jedlo, tak na základe vizuálneho vnemu je vyvolané slinenie. Ale ak sa spojí vizuálny vnem jedla a zvukový vnem zvončeka

naraz, tak po určitom počte opakovaní (resp. trénovania), psovi je možné vyvolať slinenie iba zvukovým vnemom (pes sa naučil, že po zvuku dostane potravu) (Ostatníkova, 2011). Základný princíp pri učení sa volá posilňovanie (reinforcement), kde ide o docielenie akcií na podnet a pri učení odmenou a trestom (reinforcement learning) ide o naučenie stratégií (policy) resp. propagovanie ohodnotenia stavov a akcií. Posilňovanie môže byť buď pozitívne, alebo negatívne teda R+ a R-. Jedná sa o hlavný spúšťač, ktorým je organizmus motivovaný učiť sa. Príkladom R+ a R- je experiment so Skinnerovou klietkou (Niederhoffer, 2012). V klietke je zviera, ktoré sa učí, kedy má stláčať páku pre privodenie želanej akcie.

- R+ je asociácia významného podnetu (odmena, jedlo) s motorickou odpoveďou (stlačenie páky pri modrom svetle alebo zvuku).
- R- je asociácia významného podnetu (elektrický šok) s motorickou odpoveďou (stlačenie páky pri červenom svetle alebo zvuku).

Väčšina algoritmov, používaných v umelej inteligencii sa inšpiruje prírodou a vychádza z analógie, ktorá bola vyššie opísaná. To iste platí aj pre učenie s posilňovaním (reinforcement learnig, RL), ktoré používa aj náš LCS. V robotike si môžeme:

- R+ predstaviť ako pozitívnu virtuálnu odmenu (reward), napríklad za správne vykonanie akcie alebo požadovaného cieľa;
- R- si môžeme predstaviť ako niečo negatívne čo spôsobuje virtuálny trest, napríklad robot sa odklonil z požadovanej trajektórie, vykonal nežiaducu akciu alebo je v kolíznom stave s nejakým objektom.

Ako už bolo povedané, RL je základným mechanizmom LCS. Pre lepšie pochopenie jeho matematickej funkcie je potrebné predstaviť Markovovský rozhodovací proces (MDP) a tiež Q-učenie (Q-learning resp. algoritmus A^*), ktoré použili autori v časti 1.2.2. Tejto matematickej problematike sa venujú autori (Puterman a Shin, 1978),(Montague, 1999) a (Bull a Kovacs, 2005).

Markovovský rozhodovací proces je definovaný ako množina nasledovných elementov:

- konečná množina S diskrétnych stavov s agenta;
- konečná množina A diskrétnych akcii a;
- prechodová funkcia $P : S \times A \to \Pi(S)$, *kde* $\Pi(S)$ je rodelenie pravdepodobností nad S. Distribúcia pravdepodobnosti $P_r(s_{t+1}|s_t, a_t)$ indikuje pravdepodobnosť dosiahnutia stavu s_{t+1} ak je agent v stave s_t a vykoná akciu s_t.

- funkcia odmeny $R : S \times A \rightarrow IR$, ktorá dáva každému páru (s_t, a_t), jednorozmernú odmenu, ktorú agent prijíma, ak vykonáva akciu a_t v stave s_t.

„Markovský rozhodovací proces, agentom popisuje stochastickú štruktúru problému, ale nehovorí nič o správaní agenta v jeho prostredí. Hovorí len o tom aká bude, v závislosti na aktuálnom stave a akcií, jeho budúca situácia a odmena. Vyššie uvedená definícia prechodovej funkcie P, značí určitý predpoklad o povahe stavu agenta. Tento predpoklad stanovuje pravdepodobnosť uvedeného stavu s_{t+1} iba v závislosti na s_t a a_t, ale nie v závislosti do minulosti agenta. Teda $P(s_{t+1}|s_t, a_t) = P(s_t + 1|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0)$. To znamená, že ak má agent Markovsku vlastnosť, znalosť minulosti neprináša žiadne ďalšie informácie do jeho ďalšieho stavu“ (Jabin, 2010).

Ďalším zakladaným mechanizmom je algoritmus A^*. Algoritmus A^* je prehľadávanie, ktoré má najlepšie vlastnosti - vždy nájde riešenie, a z daných prehľadávacích algoritmov navštívi najmenej zbytočných vrcholov. Funkcia ma predpis:

$$f(x) = h(x) + g(x) \tag{2.1}$$

kde, $g(x)$ je už prejdená cesta (napríklad počet navštívených vrcholov, alebo suma váh hrán grafu, po ktorých už algoritmus prešiel) a $h(x)$ je odhad koľko vrcholov ostáva do cieľa, dôležité je aby cesta nebola nadhodnotená, ak je nadhodnotená tak nie je zaručené, že $h(x)$ naozaj nájde najkratšiu cestu, naopak ak sa odhad veľmi podhodnocuje, môže to viesť k úplnému oslabeniu $h(x)$ teda k $f(x) = g(x)$, co praxi znamená, ze odhad nie je využívaný a algoritmus prehľadáva "klasicky"do šírky. Z vyššie uvedeného vyplýva, že nutná podmienka je nenadhodnocovať, mierna podmienka je nepodhodnocovať.

Pre správny odhad $h(x)$ sa používajú, rôzne techniky rasp. heuristiky, napríklad v klasických bludiskách sa používa tzv. Manhatanský. odhad (lebo máju rovne ulice) čo v praxi znamená, že ak sa chceme dostať z políčka (x_1, y_1) do (x_2, y_2) jednoducho sa vyčíta najkratšia cesta $|(x_1 - x_2)| + |(y_1 - y_2)|$ a ak je umožnený pohyb šikmo (teda po diagonále) počíta sa väčšinou pomocou pytagorovej vety $h(x) = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$.

2.3 Učiaci sa klasifikačný systém (LCS)

Učiaci sa klasifikačný systém (*LCS - Learning Classifier System*) je evolučný algoritmus, ktorý operuje na populácií zloženej z množiny pravidiel, a tieto pravidla sú používané pre klasifikovanie situácie (Brownlee, 2011). Prvý klasifikačný systém vytvoril John Henry Holland v roku 1976, čo bolo rok po tom, ako vytvoril genetický algoritmus (Holland, 1986). V kognitívnej vede je dôležitá multidisciplinarita,

ktorou je charakteristický aj J. H. Holland, pretože bol profesor psychológie a tiež profesor elektrotechniky a informatiky.

LCS sa dokáže učiť interakciou s prostredím, z ktorého dostáva spätnú väzbu v podobe numerickej odmeny. Tento spôsob učenia sa snaží maximalizovať hodnotu odmeny, ktorú môže dostať. Existuje mnoho modelov LCS a mnoho spôsobov ako definovať čo LCS je, no všetky modely, viac menej zahŕňajú štyri hlavné zložky, ktoré sú zobrazené na obrázku 2.2 (Holmes a kol., 2002):

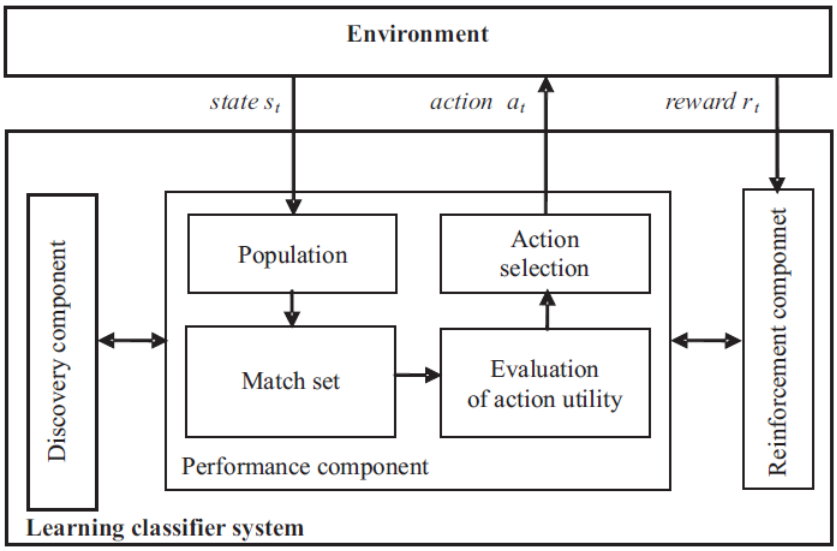
1. konečnú populáciu klasifikátorov, teda pravidiel podmienka–akcia (*condition-action rules*), ktoré predstavujú aktuálne znalosti o systéme;

2. výkonný komponent (*performance component*), ktorý riadi interakciu s prostredím;

3. posilňovací komponent (*reinforcement component*), nazývaný aj ako ohodnocovací komponent, ktorý rozdeľuje odmenu od prostredia do klasifikátorov zodpovedných za získanie domény (Cielecki a Unold, 2007);

4. objavovací komponent (*discovery component*), zužuje alebo vyhľadáva súbor pravidiel, ktoré majú najlepšie hodnotenia a tiež pomocou genetického algoritmu zlepšuje existujúce (Cielecki a Unold, 2007).

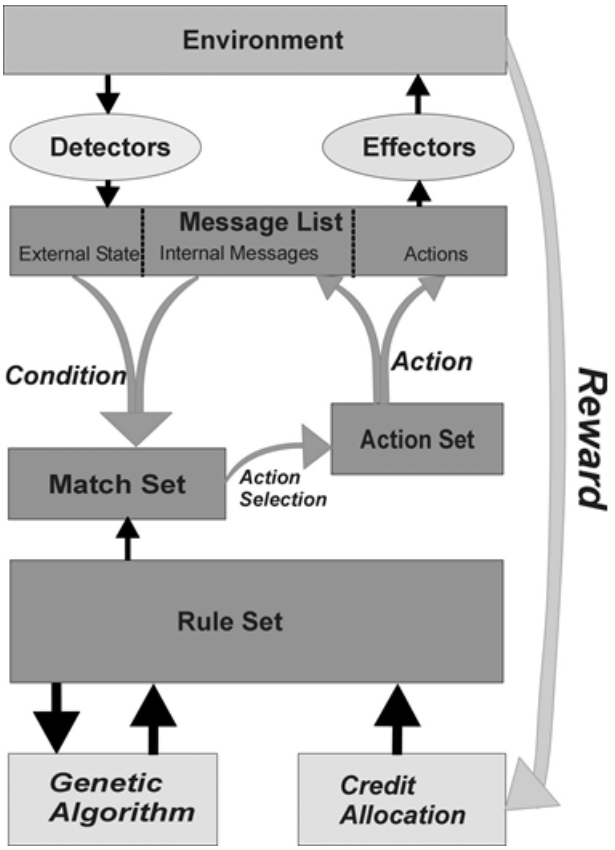
Klasifikátory majú dve súvisiace meradlá a to predikciu a fitnes. Predikcia odhaduje v súvislosti s odmenou, ktorú systém dostane. Fitnes odhaduje kvalitu informácie a je to ohodnocujúci komponent pre správne riadenie evolúcie. Vysoká fitnes znamená, že klasifikátor dáva dobrú informáciu o probléme, preto by mal byť viac uvedený do genetického algoritmu. Naopak, nízka fitnes znamená, že klasifikátor dáva slabú alebo zlú informáciu o probléme, preto nie je vhodným kandidátom do genetického algoritmu to znamená, že si nezaslúži si byť reprodukovaný (Cielecki a Unold, 2007).



Obr. 2.2: Architektúra LCS (Holmes a kol., 2002)

Učiaci sa klasifikačný systém LCS prijíma v každom diskrétnom kroku t vstup v podobe stavu prostredia s_t, pričom sa vytvára párovacia množina, ktoru reprezentuje populácia klasifikátorov. Populácia reprezentuje súčasný stav. V nasledujúcich krokoch systém hodnotí užitočnosť akcií. Potom sa vyberie akcia a_t (ktorá spĺňa určité kritérium) a posiela sa do prostredia kde na neho bude pôsobiť. Systém dostáva odmenu r_t, ktorá je priamym dôsledkom akcie a_t v aktuálnom stave s_t. Odmeny r_t sú rozdeľované do jednotlivých klasifikátorov, ktoré boli úspešné. „Toto je možné implementovať buď algoritmom špeciálne navrhnutým pre LCS (Holland, 1986) alebo algoritmom inšpirovaným tradičným učením s posilňovaním ako je Q-učenie (Wilson, 1995). Objavovací komponent, teda genetický algoritmus náhodne vyberá s pravdepodobnosťou úmernou hodnote fitnes, dvoch klasifikátorov z populácie. Aplikáciou kríženia a mutácie sa generujú dva nové klasifikátory“ (Cielecki a Unold, 2007).

LCS sa vo všeobecnosti považuje za adaptívny systém učiaci sa vyberať najlepšiu akciu vzhľadom na jeho vstup. Vzhľadom na mobilného robota by znamenalo, že urobil takú akciu, za ktorú dostal najväčšiu odmenu. Vstupom sa myslí informácia, ktorá je zaznamenaná senzorom (detektorom), zvyčajne ako vektor numerických hodnôt. Sada dostupných akcií veľmi závisí od kontextu. V našom prípade je to teraz mobilný robotický systém, teda je schopný vykonávať fyzické akcie ako: choď dopredu, dozadu, doboku doľava či doprava, otoč sa atď.. V rámci kontextu klasifikácie, môžu byť k dispozícii aj akcie typu: áno, nie alebo pozitívny či negatívny, atď.



Obr. 2.3: LCS podľa Eiben a Smith (2003)

Všeobecne platí, že LCS je jednoduchý model inteligentného agenta interagujúceho s prostredím. Schéma zobrazujúca množina pravidiel (rule set) a systém správ, kreditného systému a genetického algoritmu, ktoré sú na obrázku 2.3 Táto schéma bola navrhnutá Eiben a Smith (2003) a je veľmi dobre aplikovateľná v robotickom systéme, pretože obsahuje detektory a efektory, cez ktoré interaguje LCS s prostredím (Eiben a Smith, 2003). Informačné toky z prostredia smerujú cez detektory, takže to môžu byť „oči a uši“ robota, technicky povedané, rôzne druhy snímačov, kde sa dekóduje jedna alebo viac konečných správ. Tieto správy z prostredia sú v zozname správ, kde správy môžu aktivovať reťazec pravidiel nazvaný klasifikátor. Správy potom môžu vyvolať iné (ďalšie) klasifikátory alebo môžu vyvolať akciu. Ak systém vyvolá akciu, myslí sa tým prenos do efektorov, to znamená, že mobilný robot má akčné členy ako sú motory, ktorými sa môže pohybovať. Samozrejme to nemusia byť len kolesá, ale môže sa jednať napríklad o robotické rameno, ktoré cielene manipuluje s nejakým predmetom.

Pri LCS môžeme hovoriť o schopnosti vyberať skúsenosťami najlepšiu akciu co je priamym dôkazom o jeho adaptivite. Dôvodom tohoto zlepšovania je posilňovanie teda RL. Ide o formu numerickej odmeny, ktorá je prijímaná z prostredia. V danej

úlohe je potrebné ju nejakým spôsobom naformulovať a to experimentálne, externe, trénerom (učiteľom), ktorý nemusí byť súčasťou LCS. V robotike je to najčastejšie nejaký externý systém, ktorý sleduje robota a na základe prepočtov zo snímačov je priraďovaná odmena, tak aby robot konvertoval k želanému výsledku (Cielecki a Unold, 2007). Môže to byť napríklad lokalizačný systém, ktorý na základe polohy robota generuje odmeny pripadne tresty. Alebo ak takýto systém nemáme je možné použiť aj tabuľkový spôsob.

2.4 Implementácia a práca s klafikátormi

Jadro fungovania LCS tvori okrem učenia s posilňovaním aj práca s klasikátorom, ktorý sa vytvára a priebežne sa modifikuje tým ako robot resp. iný systém interaguje s prostredím. Nasledujúce podčasti sa bud zaoberajú samotným klasifikátorom, aktiváciou a ohodnocavaním klasifikátorov, kde v závere bude konkrétny príklad ako geneticky algoritmus manipuluje s klasifikátorom.

2.4.1 Jadro klasifikátora

Každé pravidlo inak nazvané ako klasifikátor, môžeme prirovnať k podmienkam typu “ak niečo, tak niečo“ (if - then). Ak sú podmienky definované klasifikátorom zhodné zo správami z prostredia teda z detektorov v našom prípade zo snímačov robota, tak systém vykonáva zodpovedajúce akcie, ktoré sú priradené v klasifikátore. Pravidla sú reprezentované bitovým reťazcom, napríklad na obrázku 2.4. Ďalšie uvedené texty ohľadom práce s klasifikátormi sa týkajú iba samotného jadra klasifikátora, pretože iba to vstupuje do genetických operácií. Jeho aktuálne hodnotenie alebo iné parametre sú podla ID klasifikátora uložené na iných miestach v LCS.

Obr. 2.4: Klasifikátor

kde, prvých 6 bitov predstavuje stav prostredia, teda podmienku, ktorá sa práve zhoduje s prostredím. Stav prostredia je vyjadrený hodnotami snímačov. Môže byť vyjadrený čisto binárnym spôsobom, čo znamená, že „1“ predstavuje aktivovaný snímač a „0“ predstavuje deaktivovaný snímač. Toto vyjadrenie je funkčné no pre robotiku väčšinou nevhodné, pretože informácia je veľmi “hrubá“. Pre túto nevýhodu je potrebné použiť iný spôsob reprezentácie hodnôt, teda stavu prostredia. Druhú možnosť predstavuje vyjadrenie spojitej hodnoty pomocou binárneho zápisu. Napríklad v troch bitoch je možné zapísať 23, teda 8 hodnôt. Počet bytov sa dá

rozšíriť až na hranicu presnosti, alebo rozlišovacej schopnosti snímačov. Týmto je možné dosiahnuť akúkoľvek „jemnosť kroku“, ktorá sa volí vzhľadom na danú úlohu, prostredie a možnosti robota. Posledné tri bity predstavujú akciu, ktorá môže byť vykonaná efektormi. V našom prípade mobilného robota sú to povely pre jeho pohyb. Tieto akcie sú pevne dané pohybovými schopnosťami robota a tiež ich môžeme variabilne meniť, ako v prípade snímačov. Vo všeobecnosti, akcií býva menej ako stavov prostredia. Príkladom môže byť súbor rôznych pohybov mobilného robota. Tiež to môžu byť ich rôzne kombinácie. V prvej podmienkovej časti sa okrem „0“ a „1“ nachádza aj mriežka „#“. Mriežka vyjadruje stav „nezáleží“, teda inak povedané, na tomto mieste môže byť„0“ alebo „1“. Táto hodnota sa používa iba v podmienkovej časti, preto sa pri vykonávaní akcie mriežka nevyužíva. V našom prípade si môžeme predstaviť robota a jeho snímače opísané v časti 3.2.1. Ak sa robot pohybuje vpred, zaujímajú ho hlavne prekážky pred ním, na ktoré reagujú hlavne predné snímače a tiež občas aj bočné, ale hodnoty zadných snímačov sú v takomto prípade potrebné len málo kedy. LCS časom prirodzene zistí, že tieto hodnoty nie sú pre neho zaujímavé a tak binárne hodnoty „0“ a „1“ nahradí mriežkou (Rajakaruna, 2003).

2.4.2 Aktivácia a vyber klasifikátorov

LCS používa súbor, v ktorom sa nachádzajú zoznamy podmienok a akcií. Z tohoto zoznamu sa vyberajú potenciálne klasifikátory, ktorých akcie môžu byť posielané na vykonávanie do prostredia. Následne je tvorená populácia rôznych klasifikátorov a v nej sa bude nachádzať vždy viac ako jeden potenciálny klasifikátor - teda taký klasifikátor, ktorý je nejakým spôsobom výhodný. V danú chvíľu je však zaujímavý len jeden. Z tohto dôvodu sa v pri výbere z potenciálnych klasifikátorov používa súťaž alebo aukcia, ktorá tento výber realizuje.Každá spárovaná dvojica podmienky a akcie klasifikátora predkladá hodnotu úmernú fitnes funkcii a tiež jeho špecifickosť. Špecifickosť klasifikátora udáva počet bitov iných, ako mriežka. Systém potom musí na základe špecifickosti a fitnes vybrať najlepšieho z pomedzi potenciálnych klasifikátorov. Víťazný klasifikátor už nebude takmer nikdy vybraný. Je to z toho dôvodu, že menej špecifický klasifikátor môže byť lepší resp. úspešnejší, ako silnejšie a viac špecifické pravidlá víťazného klasifikátora (Rajakaruna, 2003).

2.4.3 Ohodnocovanie klasifikátorov

Pod pojmom ohodnocovanie, myslíme odmenu a trest v numerickom podaní. Výkon alebo efektívnosť učiaceho sa klasifikačného systému (LCS) sa môže zlepšovať iba jeho pôsobením na prostredie. Je to dôležité pre jeho schopnosť modifikovať populáciu klasifikátorov. Proces modifikácie je riadený na základe tohto kreditného

systému, kde cieľom je maximalizovať odmenu a minimalizovať trest. Proces odmeny a trestu spôsobuje zvýšenie alebo zníženie hodnoty fitnes klasifikátorov, čo koreluje s ich výkonnosťou (Rajakaruna, 2003). Existuje viac spôsobov, ako a od koho prijímať odmenu. V našom prípade je však prijímanie odmeny buď priame, kde si pevne zadefinujeme (tabuľkový spôsob) za čo dostane robot odmenu (prípadne trest), čo je zvlášť vhodné i užitočné pri „oživovaní“ a ladení LCS. V druhom prípade je odmena prijímaná priamo z prostredia. V reálnom svete (prostredí) to môže byť nejaký samostatný inteligentný systém. V simulačnom prostredí je to blok programu, ktorý sleduje robota, kde na základe jeho správania prideľuje odmenu, prípadne trest. Na princípe tohto ohodnocovacieho systému sa LCS snaží vyberať také klasifikátory alebo akcie, ktoré vedú k pozitívnemu výsledku. Naopak, minimalizuje sa výber takých klasifikátorov, ktoré vedú k negatívnemu výsledku. Voľba stupňa (ich pomeru) odmeny a trestu je dôležitá pre správny výkon alebo aj efektívnosť LCS. V niektorých prípadoch je vhodné voliť pomer pozitívnej odmeny menší od pomeru negatívnej odmeny. Takýto prístup má za následok rýchlejšie odstránenie klasifikátorov, ktoré nemajú pozitívny vplyv na prostredie. V našom prípade sme dosiahli dobré výsledky aj len s pozitívnou odmenou, pričom trestom bolo prideľovanie nulovej odmeny.

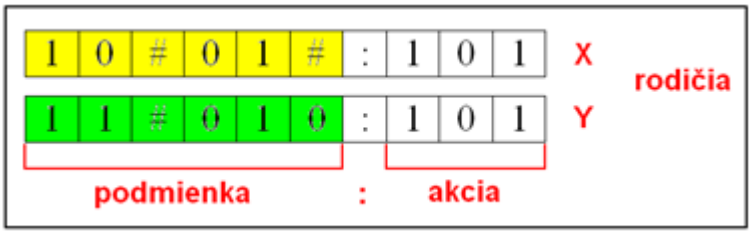
2.4.4 Genetický algoritmus v LCS a jeho časti

Genetický algoritmus operuje na populácii klasifikátorov, kde ich podrobuje evolučnému procesu. Včlenenie genetického algoritmu do LCS je vyjadrené v nasledovnej sekvencii:

1. „Inicializácia štartovacej populácie.
2. Vykonaj obvyklé operácie a umožní zodpovedajúce rozlišovanie klasifikátorov.
3. Aktivuj genetický algoritmus, ktorý bude vytvárať nové sady pravidiel.
4. Vyber daný počet klasifikátorov z populácie s pravdepodobnosťou zodpovedajúcou fitnes funkcii alebo výkonom každého klasifikátora.
5. Aplikuj operáciu kríženia a operáciu mutácie na vytvorenie nových klasifikátorov z vybraných párov v kroku 4.
6. Vyber jednotlivcov z populácie, ktorí budú nahradení (alebo doplnení) novou generáciou.
7. Vráť sa na krok 2. a vykonávaj nasledujúce kroky v slučke“ (Rajakaruna, 2003).

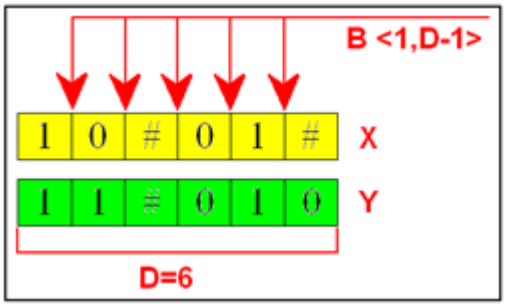
Tak ako je uvedené v sekvencii siedmich bodov, vytváranie nových množín klasifikátorov sa skladá z niekoľkých kľúčových operátorov, ktorými sú: selekcia vhodných klasifikátorov pre ich reprodukciu, kríženie klasifikátorov, mutácia klasifikátorov a vloženie nových potomkov do populácie.

V nasledujúcom texte sa budeme zaoberať jednotlivými operátormi podrobnejšie. Prvý operátor sa bude týkať selekcie. Táto metóda je inšpirovaná Darwinovou teóriou prirodzenej selekcie, ktorá bola opísaná v časti 2.1, kde prežívajú len tí najsilnejší, v našom prípade len tie najlepšie ohodnotené klasifikátory. Hlavný cieľ je podporovať rast tých najlepších, ale na druhej strane je potrebné zabezpečiť, aby sa opakovane nevyberali tie isté, preto sa výber vykonáva náhodne. Po vhodnom výbere klasifikátorov operáciou selekcie, nasleduje operácia kríženia slúžiaca pre vytvorenie nového potomstva klasifikátorov. Operácia kríženia vytvára nové potomstvo klasifikátorov s vlastnosťami podobnými s ich rodičom. Každý klasifikátor je zložený z pevnej dĺžky šiestich bitov v reťazci. Dĺžku reťazca sme označili D a bod operácie kríženia ako B. Bod B je vyberaný náhodné v rozmedzí od 1 do D-1. Proces sa vzťahuje na menenie všetkých bitových znakov reťazca, ktoré sa vykonáva na základe náhodné zvoleného krížiaceho bodu B, za účelom vytvárania nového reťazca. Majme nasledovný príklad. Z populácie sú pomocou selekcie vybrané dva klasifikátory, teda rodičia budúcich potomkov. Ako je vidieť na obrázku Obr. 2.5, sú vybrané dva klasifikátory X a Y, ktoré sú tvorené podmienkou a akciou. Keďže akcia je pevne daná, nemalo by význam, aby vstupovala do procesu evolvovania. Preto budeme pracovať iba s podmienkami X a Y rodičov (Rajakaruna, 2003).



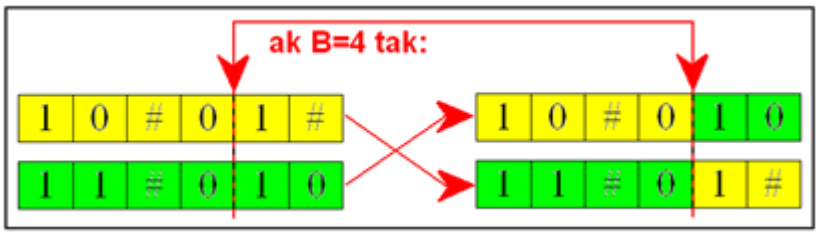
Obr. 2.5: Selekcia klasifikátorov

Reťazec podmienky má dĺžku šiestich bitov, teda D = 6 (Obr. 2.6). Z dĺžky D vyplýva aj bod, v ktorom bude prebiehať operácia kríženia, ktorého deka môže byť od 1 do 5. Hodnota bodu sa vyberá náhodným spôsobom.



Obr. 2.6: Proces pred operáciou kríženia

Nasleduje samotný proces kríženia. V našom príklade bola náhodne zvolená hodnota B=4, ako je vidieť na obrázku Obr. 2.7. Z rodičov X a Y vznikli procesom kríženia dvaja potomkovia, ktorí zdedili ich vlastnosti.



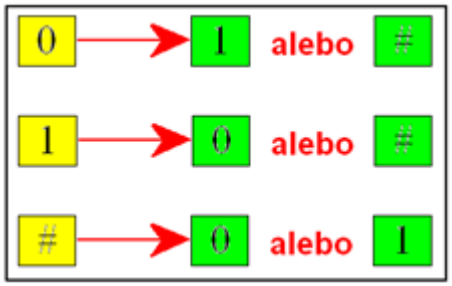
Obr. 2.7: Operácia kríženia

Podľa Engelbrecht (2007) z časti práce 2.1, noví potomkovia $\overline{x}(t)$ a $\overline{y}(t)$ sú teraz nosičmi vlastnosti rodičov $x(t)$ a $y(t)$. Vzniknuté klasifikátory sú vyobrazené na obrázku Obr. 2.8.

Obr. 2.8: Noví potomkovia

Po operácii kríženia nasleduje proces mutácie, ktorý pracuje podobným spôsobom, ako mutácia v prírode. Cieľom operácie mutácie je zabezpečiť, aby potenciálne užitočný genetický materiál nebol stratený v reprodukcii. Mutačný proces náhodne vyberá bitové pozície, ktorých hodnotu buď prevracia alebo invertuje (zamení za

opačnú hodnotu). Menenie hodnôt sa vykonáva s ohľadom na abecedu, v našom prípade je to 0, 1 a #, podľa obrázku Obr. 2.9.



Obr. 2.9: Proces mutácie

Pravdepodobnosť zmeny ktoréhokoľvek bitu v klasifikátore je rovnaká, aby bolo zaistené nulové uprednostňovanie pre ostatnými bitmi. Proces mutácie sa takmer vôbec nezapája do procesu zmien v klasifikátoroch, takže z pomedzi genetických operátorov má na výsledok najmenší vplyv. V prírode sa vykonáva zámena genetického materiálu mutáciou len s veľmi malou pravdepodobnosťou. V našom prípade v klasifikátore zmutuje iba každý tisíci bit, čo je pravdepodobnosť rovnajúca sa jednej tisícine. Po všetkých operáciách genetického algoritmu prichádza na rad zmena populácie klasifikátorov. Rozhodovanie o výbere klasifikátorov, ktoré budú nahradené vo vnútri populácie má významný vplyv na úspešnú budúcnosť celej populácie. Jednoduché odstránenie klasifikátora s nízkou hodnotou fitnes nestačí, pretože by mohlo dôjsť k zníženiu potenciálu súboru pravidiel, čo môže byť prekážkou výkonnosti LCS samotného. Aby bola zaistená veľká variácia pravidiel respektíve rôznorodosť klasifikátorov v populácii, vstupuje do výberu kandidátov pre nahradenie nových potomkov náhodný proces (Rajakaruna, 2003).

Kapitola 3

Prostredia

Na splnenie stanovených cieľov diplomovej práce bolo potrebné rozhodnúť sa, ako a v čom bude LCS testovaný. Do úvahy prichádzali v zásade dve možnosti. Prvá je implementovanie algoritmov do fyzicky existujúceho mobilného robota a druhá možnosť použiť virtuálne (simulované) prostredie, ktoré je porovnateľné s reálnym. Táto kapitola sa venuje virtuálnemu a reálnemu prostrediu, pričom reálne prostredie je doplnené o opis mobilného robota.

3.1 Virtuálne prostredie

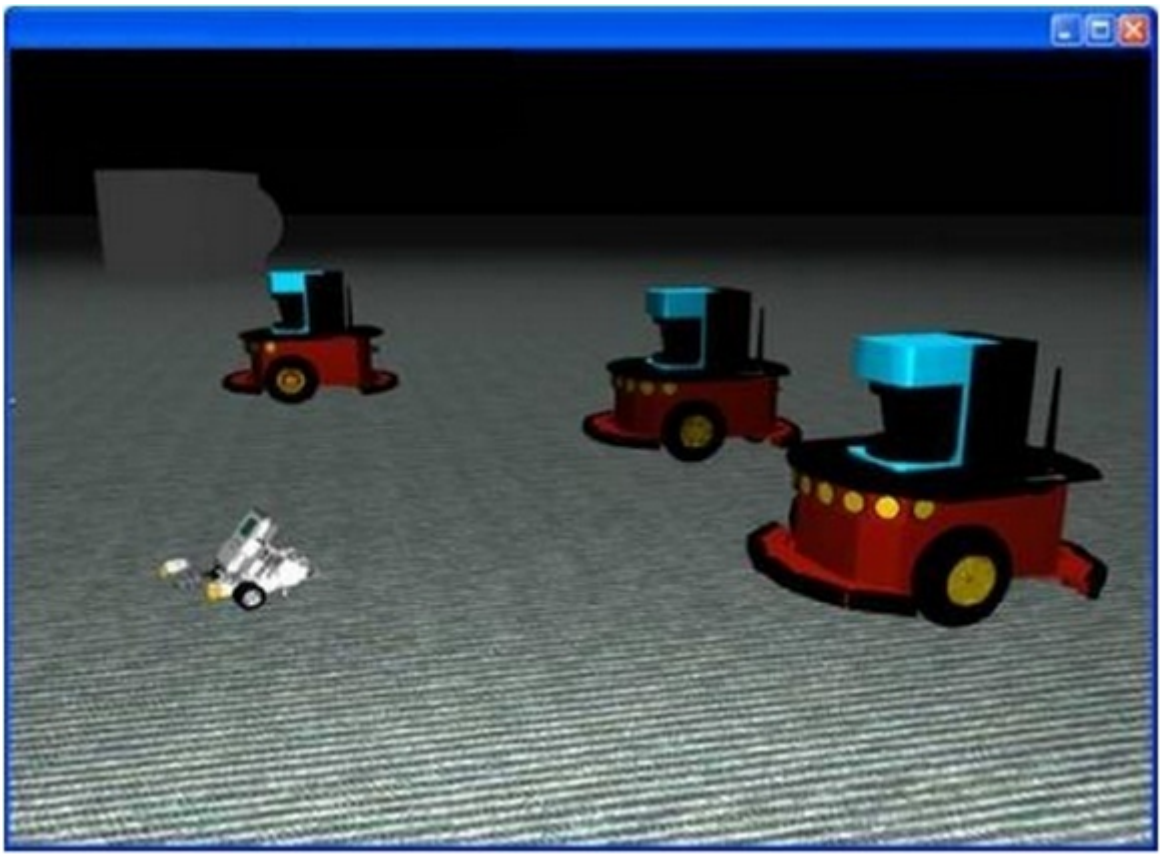
Robotika a kognitívna veda sú vedné odbory, v ktorých sa pri výskume a exaktnom dokazovaní veľmi často používajú experimenty (pokusy). V rámci týchto dvoch vedných odborov časom vznikol jeden previazaný a tým je kognitívna robotika.

„Kognitívna robotika je v súčasnosti jednou z najdynamickejších a najpopulárnejších oblastí v poli kognitívnych vied. Jej základom je tzv. konštruktivistický prístup, ktorého ústrednou myšlienkou je skúmanie rôznych kognitívnych procesov pomocou ich modelovania vo fyzických a simulovaných robotoch“ (Rebrová a Farkaš, 2011). Roboty poskytujú nové chápanie vyšších ľudských kognitívnych funkcií vytvorených umelým spôsobom (Asada a kol., 2009).

Prvotné experimenty v oblasti kognitívnej robotiky sú väčšinou robené menej nákladným spôsobom, a to simulovaním na počítačoch. V prvom rade nehrozí žiadne poškodenie technológii, ohrozovanie ľudí alebo porušovanie bezpečnostných noriem. V druhom rade sú takéto experimenty viac-menej jednoducho opakovateľné a šetria nemalé finančné prostriedky. V kognitívnej vede, napríklad v oblasti skúmania inteligencie davu sa veľmi často robia pokusy na robotoch (*swarm robotics*) (Sharkey, 2005). Mnohé z týchto experimentov sa z počiatku končia neúspechom, väčšinou kvôli zlyhaniu hardvérovej časti alebo nedokonalou prípravou experimentu. Pre „hrubé“ skúmanie je vhodné v takýchto experimentoch používať simulačné prostre-

die.

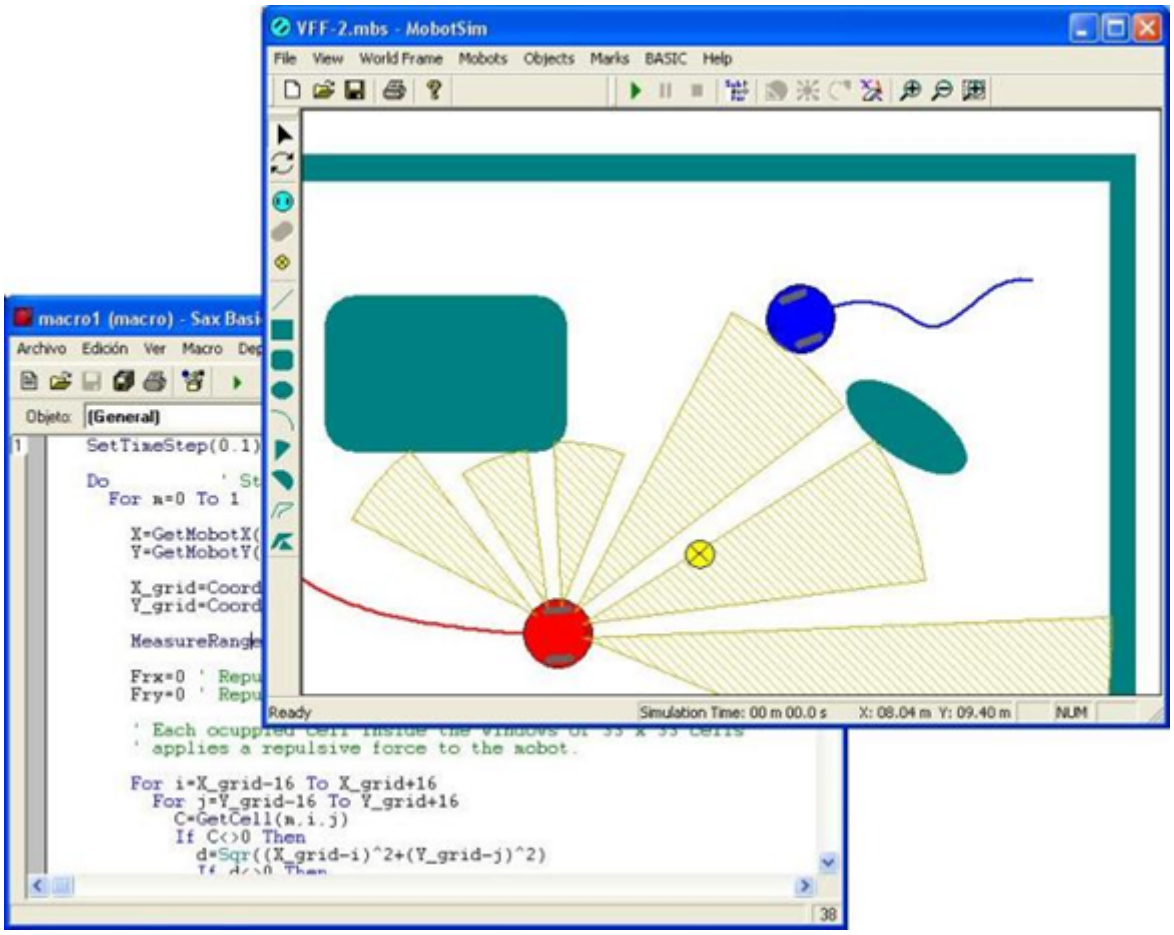
V našom prípade je použitie simulačného prostredia zvlášť vhodné, pretože pri ladení LCS môžu vznikať rôzne problémy s nastavením konštánt, zlyhania programu atď. Pre naše potreby simulácii prichádzali do úvahy nasledujúce dve možnosti. Prvá je možnosť použiť voľne dostupné virtuálne softvérové simulátory a druhá naprogramovať vlastný simulátor.



Obr. 3.1: Simulácia mobilných autonómnych robotov od LogicDesign (2012)

Robotická simulácia sa vzťahuje na metódy vytvárania aplikácii pre reálny výkon robota. Robotická simulácia je veľmi užitočná v koncepčných, dizajnových, výrobných a testovacích fázach robotických aplikácii. Kľúčový aspekt simulácie je možnosť používať robota v zapnutom stave. Termín robotická simulácia môže referovať na niekoľko rozdielnych aplikácii robotických simulátorov. Napríklad v mobilnej robotike sú to aplikácie založené na správaní robota, kde si užívateľ môže vytvoriť jednoduchý svet (prostredie) zložený z pevných objektov a programovať roboty pre interakciu s týmto svetom 3.1. Mobilným robotom sa vo všeobecnosti myslí zariadenie, ktoré nie je fixne umiestnené a má schopnosť pohybovať sa v jeho prostredí. Prostredím sa rozumie svet ako celok, s ktorým môže robot interagovať (LogicDesign, 2012).

Prvý dostupný robotický simulátor, stiahnuteľný v trial verzii, ktorý by mohol byť vhodný pre náš výskum bol *MobotSim* (*Mobile robot simulator*), teda simulátor mobilného robota (obrázok 3.2) od spoločnosti (MobotSoft, 2012).



Obr. 3.2: MobotSim – Mobile robot simulator od spoločnosti MobotSoft (2012)

MobotSim je 2D simulátor pre mobilné roboty. Všetky roboty, ktoré sa v ňom dajú simulovať sú diferenciálneho typu, čiže majú diferenciálny podvozok (jeden virtuálny motor pre ľavé a jeden pre pravé koleso). Poskytuje grafické rozhranie, ktoré reprezentuje prostredie, kde je možné ľahko si vytvárať, nastavovať a editovať robotov aj objekty. Aby bolo možné pohybovať a riadiť robotov, tento simulátor má BASIC Editor, teda editor, v ktorom si môže užívateľ písať makrá s využitím špecifických funkcií a získať informácie o robotových súradniciach, hodnotách snímačov, nastavovať rýchlosť jazdy, atď. Teda je možnosť využívať všetky vlastnosti jazyka Basic pre navigačné techniky robota. Tento softvér bol vyvinutý pre vedcov, výskumníkov, študentov a nadšencov robotiky, ktorí chcú na navrhovať a testovať

navigačné techniky, vyhýbanie sa prekážkam, umelú inteligenciu a iné MobotSoft (2012). Ak by sme mali zhrnúť kľúčové vlastnosti, môžeme ich usporiadať do nasledovných bodov:

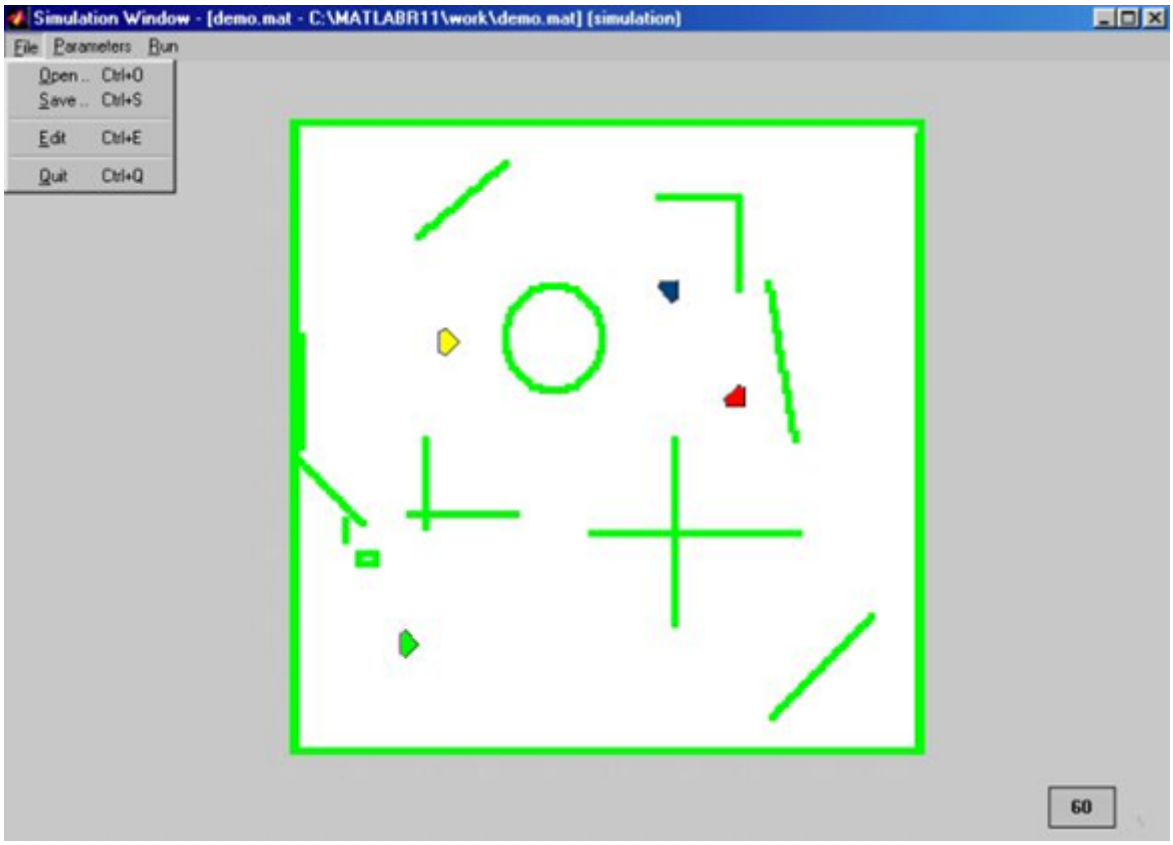
- „neobmedzený počet robotov a prekážok;
- niekoľko tvarov prekážok (čiara, obdĺžnik, okrúhly obdĺžnik, kružnica, elipsa,...) a tiež je možnosť nakresliť si vlastnú prekážku;
- mobilné roboty majú diferenciálny pohon;
- simulácia senzorov pre meranie vzdialenosti (ultrazvukové snímače);
- flexibilná konfigurácia robotov, teda je možné nastaviť priemer platformy, priemer kolies, vzdialenosť medzi kolesami, počet senzorov a uhol medzi nimi;
- konfigurácie rozsahu senzorov – snímací kužeľ, rozsah, skreslenie;
- každý robot ma konfigurovateľnú mriežku pre mapovanie prostredia, pomocou senzorových funkcii;
- vytváranie simulácie cez rýchle a jednoduché písanie makier, kompatibilne s VisualBasic;
- kontrola toku programu – debugovanie;
- zobrazovanie dát, ako sú snímače, čas – vhodné pre debugovanie;
- jednoduchá integrácia tretích strán, možnosť pridania špecifických nástrojov napríklad pre využitie fuzzy logiky, genetického algoritmu, neurónových sieti a podobne“ (MobotSoft, 2012).

Druhý dostupný robotický simulátor, celý voľne stiahnuteľný, ktorý by mohol byť vhodný pre náš výskum bol *Simulátor autonomních mobilních robotů* vytvorený Ústavom automatizace a měřicí techniky, Fakulty elektrotechniky a informatiky, Vysokého učení technické v Brně (Simulátor, 2012).

Uvedený simulátor bol vytvorený ako toolbox pre Matlab 5, teda nejedná sa o samostatný program (Obr. 3.3). Tento toolbox umožňuje simuláciu jedného alebo viacerých mobilných robotov, pohybujúcich sa vo virtuálnom prostredí. Výhodou je, že každý robot môže byť riadený vlastným algoritmom. Matlab obsahuje množstvo iných toolboxov, ako sú fuzzy logika či neurónové siete, čo je veľkou výhodou. Oproti predošlému MobotSimu má tento program okrem ultrazvukových senzorov k dispozícii aj laserový skener. Autori uvádzajú, že toolbox pozostáva z dvoch samostatných častí, ktoré sú: Editor a samotný Simulátor. Editor slúži k vytváraniu simulácii,

vytváranie mapy, umiestňovanie robotov, programovanie riadiacich algoritmov. Simulátor môže byť spúšťaný priamo z editora alebo z príkazového riadku. Simulátor umožňuje pohyb robotov vo virtuálnom prostredí, tiež so simuláciou senzorov a s vyhodnocovaním riadiacich algoritmov.

Ďalšia funkcia simulátora je replay, ktorá umožňuje prehrávať priebeh simulácie a taktiež priebehy ukladať. Z uloženého priebehu je následne možné vykresľovať trajektórie robotov (Simulátor, 2012).



Obr. 3.3: Okno simulátora v Simulátor (2012)

Zhrňme teda kľúčové vlastnosti simulátora, môžeme ich usporiadať do nasledovných bodov:

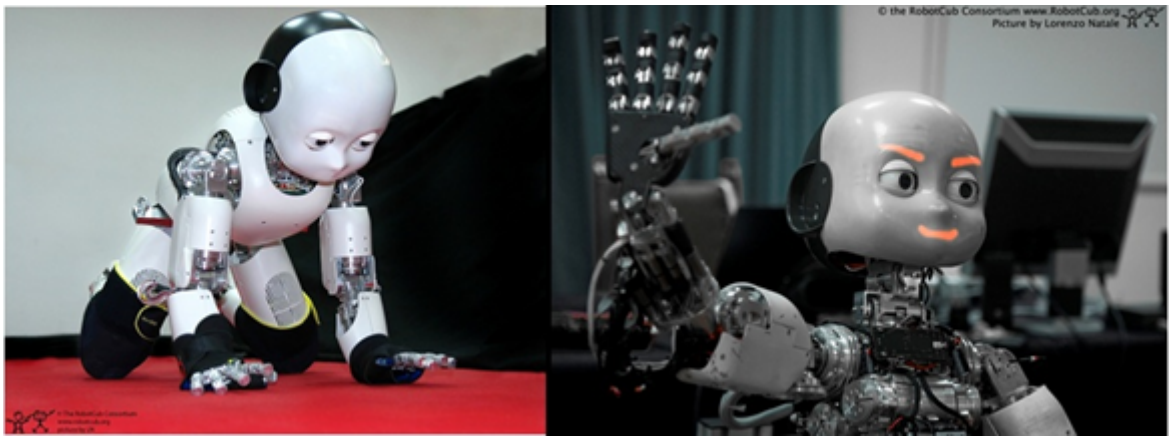
- „toolbox je vybavený GUI (graphic user interface);
- GUI obsahuje: menu pre Editor a Simulátor, kontextové menu pre roboty, editor senzorového subsystému, atď.;
- možnosť simulácie viacerých robotov súčasne;
- možnosť využitia iných toolboxov ako sú fuzzy logika, neurónové siete;

- virtuálne ultrazvukové senzory a laserové skenery;
- virtuálny svet obsahuje statické a dynamické prekážky;
- priebeh simulácie je ukladaný s možnosťou ju zase prehrať.“ Simulátor (2012).

Ak by sme mali zhodnotiť MobotSim a Simulátor autonomních mobilních robotu, tak MobotSim má lepšie prepracovaný vzhľad GUI a tiež nie je závislý od nadradeného programu. No druhý simulátor má väčšie možnosti, pretože je implementovateľný do MATLABu, kde je možné využiť jeho nesporne ohromné možnosti a komplexnosť.

3.2 Reálne prostredie

V kognitívnej robotike je asi najznámejší robot iCub (Obr. 3.4) a je to výsledok projektu s názvom RobotCub. Robot iCub je humanoidný robot, ktorý je stavbou tela vernou kópiou dva a pol ročného dieťaťa. Jeho výška je 90 cm, hmotnosť 23 kg a precízna mechanika obsahuje 53 stupňov voľnosti (Tsagarakis, 2006).



Obr. 3.4: Humanoidný robot iCub (`http://www.robotcub.org`)

Reálne telo robota spolu s virtuálnym softvérovým simulátorom (postaveným na báze fyzikálneho enginu ODE) je ideálnym nástrojom pre kognitívnu robotiku, ktorá sa snaží porozumieť ľudskej kognícii, čo je jedným zo základných cieľov kognitívnej vedy ako takej.

V našom prípade sa reálnym prostredím myslí robotický hardvér v reálnom prostredí. Jedná sa hlavne o vnútorné prostredie (indoor), teda prostredie laboratória alebo jednoduchej miestnosti. Náš hardvér predstavuje mobilný robot s holonómnym

typom podvozku opísaného v časti 3.2.1. Reálne prostredie obsahuje množstvo javov a náhodných faktorov, ktoré v simulačnom prostredí neuvažujeme, resp. môžeme ich tam doplniť, ale stále tým plne nenahradíme fyzický svet. Jedná sa napríklad o premenlivé trenia medzi robotom a podložkou (medzi kolesami a podlahou) – teda nepredvídateľné zmeny mechanických parametrov, rôzne oneskorenia na rozmedzí hardvéru a softvéru, nedokonalosti snímačov, (ktoré v simulačnom prostredí fungujú vždy na 100 %), atď.. Jedným z najmarkantnejších rozdielov medzi reálnym a simulačným prostredím sú časové súvislosti. Pokiaľ ide o simuláciu, môžeme si dovoliť časové oneskorenia alebo zrýchlenie simulácie, prípadne neobmedzený počet iterácii či experimentov vo veľmi krátkom čaše. Preto v hlavne v kognitívnej robotike simulačný – virtuálny experiment predchádza tomu reálnemu, vo fyzickom svete. V reálnom prostredí musí byť všetko naplánované a prispôsobené reálnemu času. Okrem vývoja znalostí riadiaceho algoritmu je vo všeobecnosti značný čas vynaložený na technické detaily, ako je zber dát zo snímačov, posielanie dát do akčných členov, obmedzenie výpočtového výkonu, energetická sebestačnosť robota, jeho komunikačné technológie a mnoho iných.

Spomenutý mobilný robot bol skonštruovaný pre rôzne druhy experimentov pričom poskytuje komfortné ovládanie a na mobilné zariadene disponuje pomerne veľkým výpočtovým výkonom, ktorý nevyžaduje žiadne úsporné riešenia a obmedzovanie sa. Uvedené vlastnosti sú veľmi dôležité v oblasti kognitívnej robotiky, pretože po prvotnom prekonaní hardvérových problémov sa výskumník sústredí už len kognitívnu časť experimentu, teda na jadro jeho výskumnej úlohy.

Ďalšia dôležitá vec je bezpečnosť okolia a samotného zriadenia – robota. Okrem kľúčového algoritmu je potrebné zaoberať sa rôznymi bezpečnostnými opatreniami a úpravou algoritmov v prípade ich zlyhania. Mnoho krát sú spomenuté opatrenia riešené úplne na tej najspodnejšej hardvérovej úrovni, do ktorej nemá prístup žiaden algoritmus. V tejto súvislosti je vhodné obohatiť problematiku o filozoficky veľmi bohatú tému a tou sú zákony robotiky. Zákony robotiky sa zaoberajú pravidlami alebo správaním sa robotov, a je jedno, či sa jedná o mobilného robota s kolesami, alebo o robota humanoidného charakteru. Tieto zákony boli definované *Isaacom Asimovom* v roku 1942, kedy zákony odzneli v jeho poviedke *„Hra na naháňačku (Runaround)“* (Hecht, 1979). Tieto princípy boli ešte pred nedávnom považované za hranice, ktoré by sa nemali prekračovať pri vývoji a používaní robotov. Žiaľ v súčasnosti tomu tak nie je. *Isaacom Asimovom* boli tieto zákony naformulované nasledovným spôsobom (Železný, 1988):

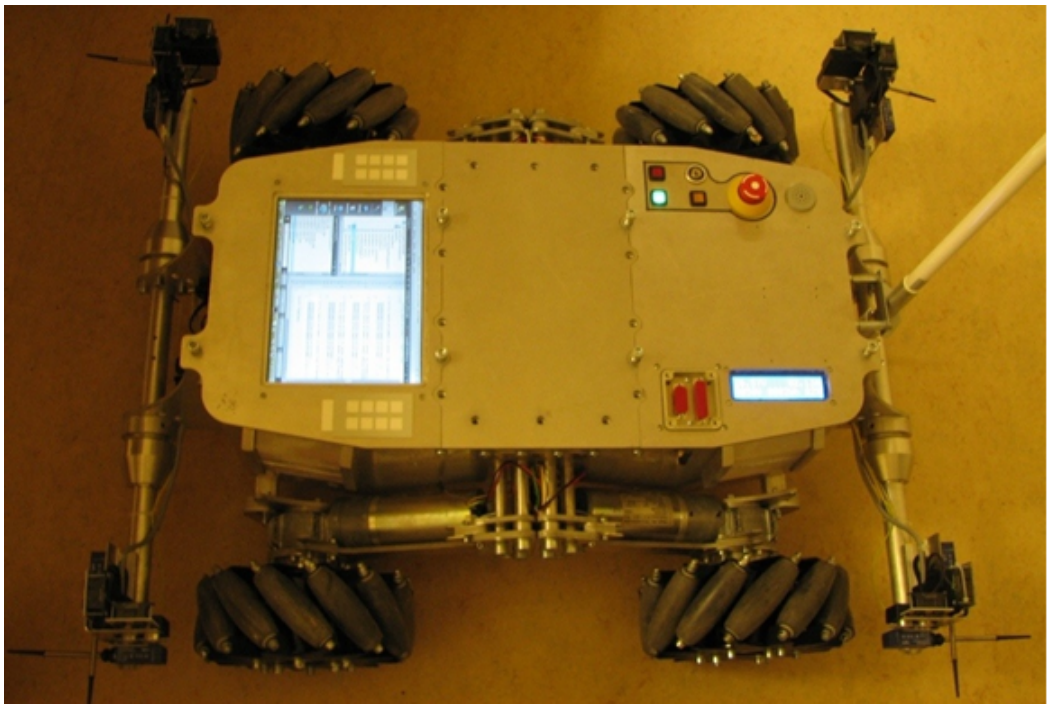
1. „Robot nesmie ublížiť človeku alebo svojou nečinnosťou dopustiť, aby bolo človeku ublížené.

2. Robot musí poslúchať človeka, okrem prípadov, kedy je to v rozpore s prvým zákonom.

3. Robot sa musí chrániť pred poškodením, okrem prípadov, kedy je to v rozpore s prvým alebo druhým zákonom.“ (Isaac Asimov 1942)

3.2.1 Mobilný robot

Všetky simulačné prostredia, ktoré sa používajú pre výskum v kognitívnej robotike sú väčšinou postavené na rozumnom reálnom základe. Tak, ako je simulátor humanoidného robota iCub naprogramovaný s ohľadom na fyzického robota aj naše simulačné experimenty majú reálny podklad v podobe mobilného robota. Pre vytvorenie čo najvernejšieho simulátora alebo použitie voľne dostupného, aspoň trochu podobného je veľmi dôležité poznať kľúčové vlastnosti mobilného robotického systému, použitého v našom prípade. Najprv sa budeme venovať všeobecnému popisu mobilného robota a potom prejdeme na detaily, ktoré sú považujeme za dôležité. Jedná sa mobilného robota určeného predovšetkým pre rôzne druhy experimentov tohto typu, teda riešenie problémov lokalizácie a navigácie mobilného robota (Obr. 1.1 a Obr. 3.5). Samotný mobilný robot je masívnej konštrukcie s rozmermi 93 x 73 x 35 cm (dĺžka, šírka, výška) a hmotnosťou presahuje 80kg. Má štyri špeciálne samostatné elektricky poháňané kolesá, vstavaný informačný systém s LCD displayom, veľkokapacitné lítiové akumulátory, rôzne druhy snímačov a tiež obsahuje WiFi komunikačné technológie (Tóth, 2011).

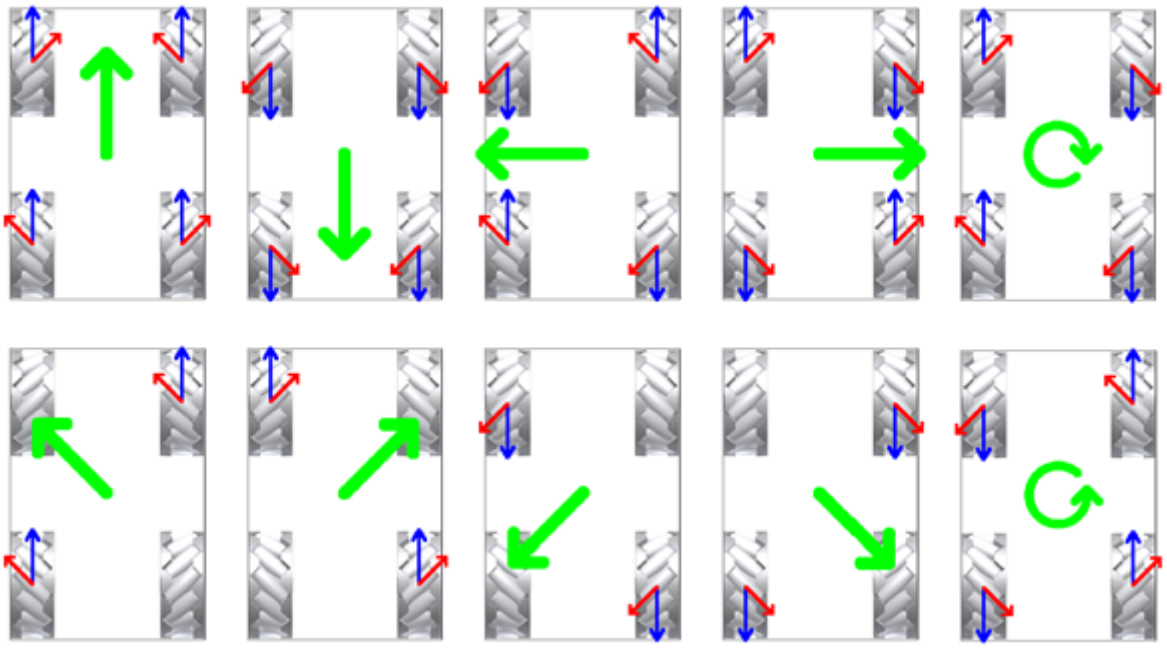


Obr. 3.5: Mobilný robotický systém

Prvá veľmi dôležitá časť pre potreby simulácie je pohonný subsystém, teda pohybové schopnosti robota a niektoré parametre. Opisovaný robot má špeciálny Mecanum (holonómny) podvozok. Mobilná platforma však nepoužíva konvenčné kolesá.

Hlavnou súčasťou holonómneho (Mecanum) podvozku sú špeciálne kolesá typu Mecanum. Mecanum koleso je tiež známe ako švédske koleso a bolo patentované v roku 1975. Tvoria ho obvodové valčeky pod uhlom 45° a bežne sa používajú v usporiadaní, kde sú kolesá umiestnené súbežne v rohoch pomyselného štvorca alebo obdĺžnika. Mecanum kolesový podvozok má oproti diferenciálnemu o jeden stupeň voľnosti naviac (teda spolu tri stupne voľnosti). Umožňuje pohyb po priamkach vo všetkých smeroch a robot sa môže pohybovať do boku aj otáčať sa okolo vlastného stredu. O Mecanum kolesovom podvozku môžeme povedať, že je holonómny (z anglického *holonomic*), umožňuje nezávislé riadenie všetkých troch stupňov voľnosti. V tomto type podvozku sa jedna o transláciu v osi x, transláciu osi y a rotáciu v osi z (Tóth, 2011).

Z uvedeného vyplýva, že robot môže v rovine vykonávať akékoľvek pohyby teda všetky kombinácie pohybu po priamke a po kružnici (rotácie). Základne varianty pohybov znázorňuje obrázok Obr. 3.6, pričom sa ich dá ľubovoľne kombinovať. Maximálnu rýchlosť, ktorú môže robot vyvinúť je 0.7 ms^{-1} s možnosťou ju plynule meniť od nulovej rýchlosti, pričom takou istou rýchlosťou je schopný isť aj v reverznom chode.



Obr. 3.6: Základne varianty pohybov mobilného robota (Tóth, 2011)

Druhá taktiež dôležitá časť pre potreby simulácie je senzorový subsystém. Robotický systém disponuje hneď niekoľkými druhmi senzorov:

- taktilné senzory;
- ultrazvukové senzory vzdialenosti;
- infračervené senzory vzdialenosti.

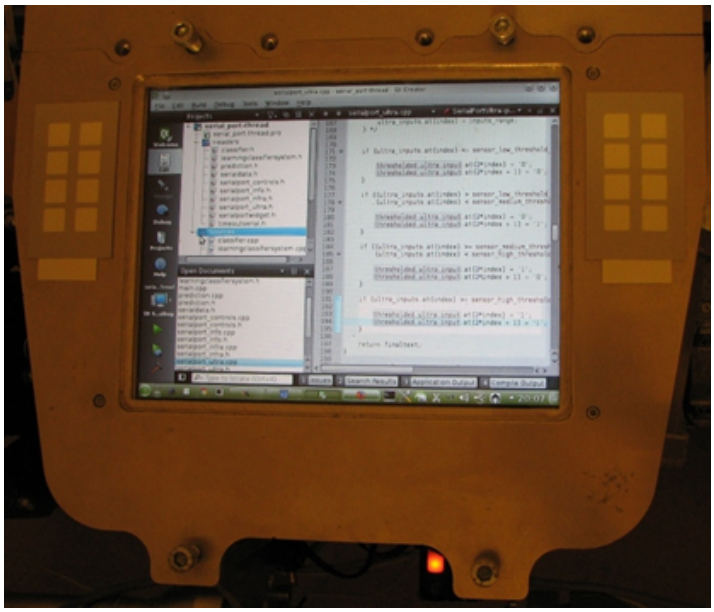
Senzory sú umiestnené v každom rohu robota v pároch vzájomne natočených o 90°, takže robot vie snímať vzdialenosti vždy pred sebou a na boku. Keďže senzory sú každom rohu v páre, spolu je ich osem kusov z každého druhu. Je ich možné vidieť na obrázku Obr. 3.5, kde sú v trojiciach (taktilný, ultrazvukový a infračervený).

Taktilné senzory sa inak nazývajú ako aj antikolízne senzory. Môžeme ich brať ako najposednejšiu informáciu o kolízii v prípade, ak by zlyhali ostatné (ultrazvukové a infračervené) a pracujú na mechanickom princípe s binárnym výstupom. Okrem programového prístupu k hodnotám snímačov sú hardvérovo napojené k elektronike pohovov a v prípade kolízie táto elektronika deaktivuje pohony robota.

Ultrazvukový senzor sa veľmi často používa v aplikáciách mobilnej robotiky. V našom prípade senzor môže merať vzdialenosti až do 13 metrov, čo je pomerne veľký rozsah. Do hlavného počítača sú priebežne posielané namerané hodnoty v podobe reťazcov označených indexom, ktorý udáva ID senzora. Sú v celku spoľahlivé, no ich presnosť záleží od mnohých externých parametrov a tiež ich snímací uhol sa s vzdialenosťou zväčšuje, čím rastie ich citlivosť.

Posledné senzory sú infračervené, tiež často používané na poli mobilnej robotiky. Ich merací rozsah nie je veľký, v našom prípade je to 1,5 metra. Sú však prenesené a do hlavného počítača sú posielané dáta v podobe vektora ôsmich hodnôt, všetkých snímačov naraz. Nie sú vhodné do každého prostredia, napr. ostré lúče slnka znemožňujú meranie. Výskumník, ktorý programuje senzorový systém si sám volí, ktoré snímače je vhodné v danej situácii používať, samozrejme so snahou navoliť bezpečnostné zóny robota tak, aby nedochádzalo k núdzovým zastaveniam robota.

Tretí dôležitý subsystém je výpočtový systém. Mnoho komerčných robotov používa veľa druhov minimalistických počítačov, ktoré nútia užívateľa obsluhovať len to najdôležitejšie a použitie náročnejších aplikácii si vyžaduje externé zariadenie (PC alebo notebook). Náš informačný systém pozostáva z „klasických“ komponentov bežného stolového PC doplneného o komunikačné rozhranie, ktoré je schopné prijímať dáta zo snímačov a vysielať akčné zásahy do pohonov. Pozostáva z ITX matičnej dosky, Intel procesora i5, 64 GB SSD disku, 4 GB DDR3 RAM pamäte, 2.4 a 5 GHz WiFi a 10″ LCD (Obr. 3.7). Počítač je možné ovládať pomocou bezdrôtovej klávesnice (myši) alebo cez sieť, či dokonca cez internet.



Obr. 3.7: Detail používania vstavaného počítača pri programovaní

Výdrž akumulátorov robota je pri plnom zaťažení až 12 hodín, čo úplne postačuje. Opísaná konfigurácia dimenzovaním postačuje na drvivú väčšinu výpočtov v oblasti kognitívnej robotiky.

Tento mobilný robot by bolo možné opísať oveľa podrobnejšie, ale to už nie je relevantné z hľadiska kognitívnej robotiky a uvedené informácie postačujú pre výber, alebo naprogramovanie simulačného prostredia.

3.3 Zhrnutie

V tejto kapitole sme rozoberali reálne a simulačné prostredia z hľadiska potrieb pre vykonávanie našich experimentov v oblasti kognitívnej robotiky. Rozhodli sme sa, že prvotné experimenty budeme realizovať v simulačnom prostredí. Z dôvodu záujmu budúcej implementácie týchto algoritmov aj do reálneho mobilného robota je potrebné použiť verné simulačné prostredie a zachovať tak, akú – takú podobnosť medzi simulačným a reálnym prostredím. Uvedené simulátory sú na vysokej úrovni, ale napriek tomu majú aj nevýhody, pre ktoré ich nie je vhodné použiť. Napríklad robot je buď kruhovitého tvaru, alebo je robot zobrazovaný len ako hmotný bod (resp. trojuholník), usporiadanie snímačov je iba v rámci kruhových výsekov a čo je najdôležitejšie, ich model vôbec neumožňuje také pohyby (kinematiku), akými disponuje náš reálny model. Z týchto dôvodov si vytvoríme vlastný simulátor, ktorý bude čo najvernejšie zohľadňovať reálneho mobilného robota s danými vlastnosťami. Kompletný návrh simulátora a implementácia LCS bude opísaný v štvrtej kapitole.

Kapitola 4

Vlastný návrh a implementácia

V tejto štvrtej kapitole predstavujeme implementáciu LCS do nami navrhnutého robotického simulátora. Hlbšie popíšeme, ako funguje LCS v implementácii s mobilným robotom teda v našom prípade hlavne s robotickým simulátorom. Jadro algoritmu LCS pochádza z knihy Brownlee (2011) a inšpirácia ohľadne implementácie je prevažne od Rajakaruna (2003).

4.1 Učiaci sa klasifikačný systém

Ako už bolo spomenuté v teoretickom úvode, základ učiaceho sa klasifikačného sa systému (Learning Classifier System) je tvorený evolučným algoritmom a učením s posilňovaním. LCS je možné aplikovať vo viacerých oblastiach a vo všeobecnosti existujú dva štýly LCS. Prvý je štýlu Pittsburgh, ktorý optimalizuje celý klasifikátor a druhý je štýl Michigan, ktorý optimalizuje množina pravidiel. Štýl Michigan je používaný oveľa častejšie a delí sa dve hlavné verzie. Prvou je ZCS (zeroth-level classifier system) a XCS (accuracy-based classifier system) (Brownlee, 2011). Náš LCS je štýlu Michigan a XCS verzie (Butz a Wilson, 2002).

Hlavným cieľom LCS je optimalizovať odmenu alebo prínos založený na pôsobení robota na prostredie, v ktorom sa nachádza. Toto je dosiahnuté riadením kreditného systému, ohodnocujúceho užitočné pravidlá, ďalej je to vyhľadávanie nových pravidiel a už existujúcich variácii pomocou evolučného procesu.

Hlavní aktéri LCS su detektory, efektory, správy, spätná väzba a klasifikátory. Efektory riadia akcie, v našom prípade sú riadené pohony mobilného robota, a tým robot vykonáva cielené pohyby v priestore. Detektory sú používané v systéme pre vnímanie stavu prostredia, teda nameraných hodnôt snímačov, kde sú použité hlavne ultrazvukové snímače. Správy predstavujú súhrn dátových rámcov diskrétnych informácií z detektorov, ktoré smerujú do systému. LCS systém neustále vníma prostredie cez spomínané detektory, ktoré pomáhajú získavať spätnú väzbu v hlavne v podobe

numerickej odmeny, prípadne trestu. Asi najdôležitejším a kľúčovým prvkom v LCS sú klasifikátory, ktoré sú zostavené z pravidiel podmienka-akcia, ktoré sú filtrom pre správy. Ak senzory robota namerajú také hodnoty, ktoré splnia podmienkovú časť klasifikátora, tak je spustená priradená akcia. Správy majú dopredu stanovené pevne pevné dĺžky a sú zložené z binárnych reťazcov. Klasifikátor je definovaný ako trojica kombinácie reťazca z možnej abecedy $\in \{1, 0, \#\}$, kde # reprezentuje „nezáleží“, to zanemená, že mriežka # môže reprezentovať buď 0 alebo 1. Táto problematika sa bude bližšie opisovať v nasledujúcich častiach.

V LCS systéme sa správy z prostredia umiestňujú do zoznamu správ, ďalej sa kontrolujú podmienky každého klasifikátora kde je kladený dôraz ci je splnená aspon jedna podmienka v danom zozname správ. Všetky klasifikátory, ktoré sa podieľajú v “súťažení” a teda tie, ktoré boli úspešné, zapisujú svoje akcie do zoznamu správ. Všetky správy zamerané pre pohonný subsystém mobilného robota sú vykonávané – akcie sa nechajú pôsobiť na prostredie. Všetky správy (v zozname správ) z predchádzajúceho cyklu sa rušia (vymazávajú) – správy pretrvávajú iba v jednom cykle - to znamená ze LCS sa dokáže dynamicky prispôsobovať zmene prostredia.

LCS algoritmus pozostáva z množstva voliteľných parametrov, no všetky pokusy o radikálne zmeny dopadli neúspešne, preto sme sa rozhodli ponechať pôvodne nastavenia(Brownlee, 2011), ktoré vyplývajú z práce Butz a Wilson (2002):

- Rýchlosť učenia β pre klasifikátor vzhľadom na na fitness je v rozsahu $\beta \in (0,1; 0,2)$.
- Frekvencia spúšťania genetického algoritmu θ_{GA} by mala byť v rozmedzí $\theta_{GA}(25; 50)$.
- Diskontný faktor $\gamma \approx 0,71$.
- Hodnota pravdepodobnosti kríženia v genetickom algoritme $\chi \in (0,5; 1,0)$.
- Hodnota pravdepodobnosti mutácie v genetickom algoritme $\mu \in (0,01; 0,05)$.
- Počiatočné nulové hodnoty sa nastavujú pre odmenu, chybu a fitnes.
- Pravdepodobnosť výberu náhodnej akcie $p_{exp} \approx 0,5$ pre účely prieskumu.

Naša inšpirácia LCS a verzie XCS pochádza z programu opísaného v (Brownlee, 2011) a naše znenie pseudokódu 2. je nasledovné:

Algorithm 2 LCS (XCS) pseudokód

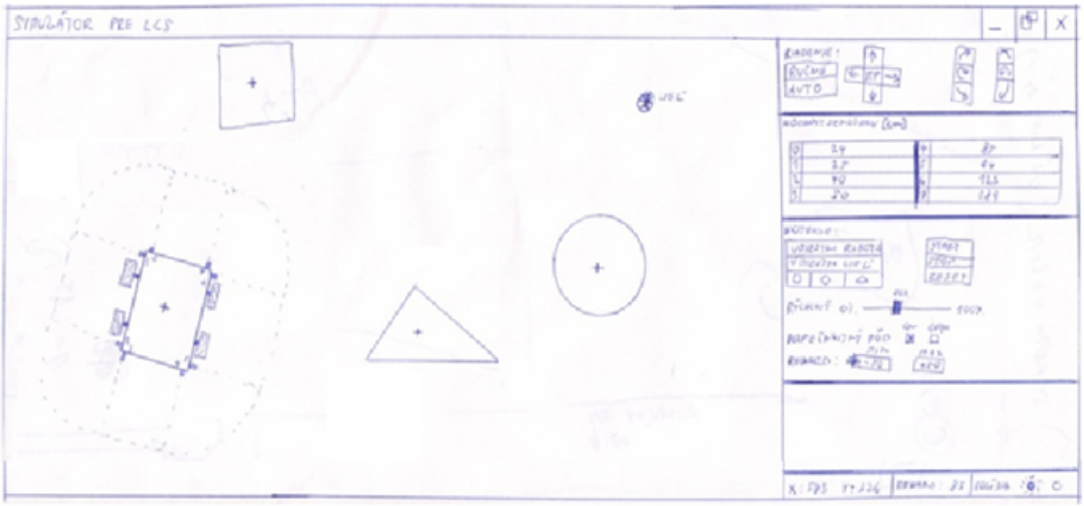
```
initialization
while not goal do
    read_inputs_from_sensors_(RS232)
    inputs_quantization
    explore_mode⇐is_even_iteration;
    generate_match_set
            (input_from_environment, possible_actions_set)
    generate_predictions
    select_action
    if not explore_mode then
        send_action_to_actuators_(RS232)
        execute_action
    end if
    calculate_reward(input, action)
    if explore_mode then
        generate_action_set
        update_fitness
        if can_run_genetic_algorithm then
            note_last_gen_for_action_set
            run_genetic_algorithm(action_set, input)
        end if
    else
        calculate_performance_from_predictions
        calculate_errors
        calculate_accuracy
    end if
    next_iter
end while
```

4.2 Robotický simulátor s LCS

Simulátor s LCS je aplikácia určená pre simuláciu robota, jeho pohybov v rámci scény, medzi prekážkami, ako sú trojuholník, štvorec a kruh. Kompletný printscreen aplikácie je vyobrazený na obrázku Obr. 4.2 a jeho prvotný návrh na obrázku Obr. 4.1. Robota je možné ovládať automaticky (LCS) alebo pomocou manuálneho riadenia. V užívateľskej časti simulátora sú zobrazované hodnoty ôsmich senzorov vzdialeností robota a aktuálna pozícia ťažiska robota na scéne. Ďalej je možné nastavovať viaceré atribúty robota ako rýchlosť, polomer otáčania, bezpečnostný mód, odmenu za vzdialenosť od cieľa, trest za nevhodné akcie, ďalej je možné pridávať na scénu prekážky, mazať ich, presúvať. Programové prostredie, kde bol simulátor naprogramovaný bude opísané v ďalších častiach tejto kapitoly.

4.2.1 Qt prostredie

Qt je jedna z najpopulárnejších multiplatformových knižníc pre vytváranie programov a aplikácii s grafickým užívateľským rozhraním. Prostredie Qt funguje na platformách Windows, Linux, Mac a iných (Qt-Homepage, 2013). Zdrojový kód aplikácie robotického simulátora je napísaný v Qt C++ a je pod licenciou LGPL v 2.1.



Obr. 4.1: Prvotný náčrt dizajnu simulátora

GNU Lesser General Public License (GNU LGPL, či jednoducho LGPL) je voľnejší variant licencie pre voľne dostupný softvér GNU GPL. Licencia umožňuje spojovanie s licencovaným kódom kódom a tým je vhodnejšia napríklad pre knižnice.

Väčšina licencií pre softvér a iné diela sú navrhnuté tak, aby obmedzovali voľnosť jeho zdieľania a upravovania. GNU LGPL naopak zaručuje slobodu zdieľania a upravovania všetkých verzií programu, aby bol softvér voľný pre všetkých jeho používateľov.

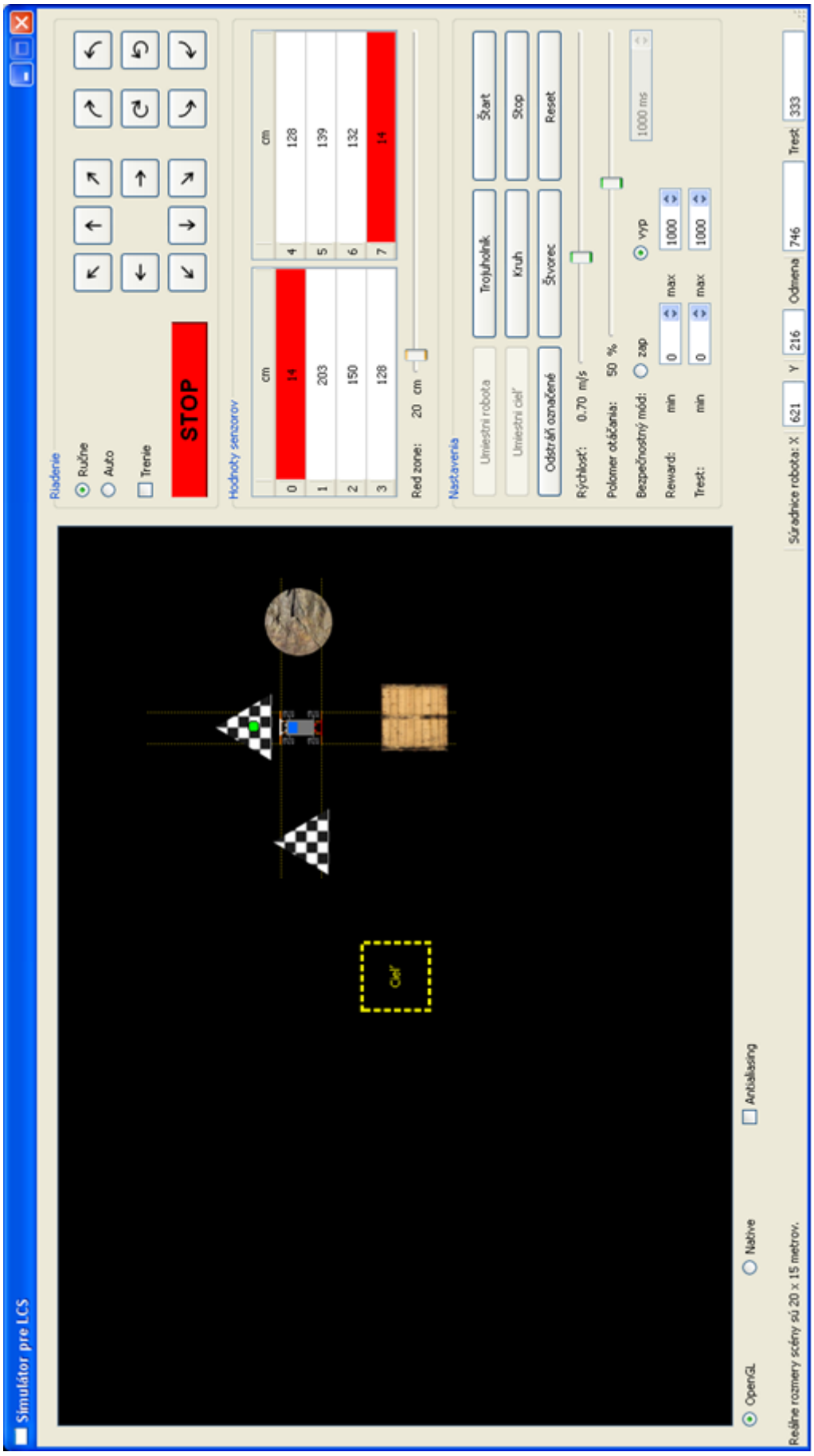
Pre ďalšie upravovanie softvéru je potrebné pokračovať v licencii LGPL v2.1 alebo novšej. Jedná sa o licenciu, ktorá definuje zrieknutie sa akejkoľvek záruky a obmedzenie zodpovednosti. Viac dostupných informácií o licencii GNU LGPv2.1 sa nachádza na web stránke GNU Operating System (GNU-OS, 2013).

4.2.2 OpenGL a vykresľovanie

Samotná scéna má podporu OpenGL, v prípade chýbajúcej podpory OpenGL systémom je možné prepnúť sa do natívneho vykresľovania (Obr. 4.3).

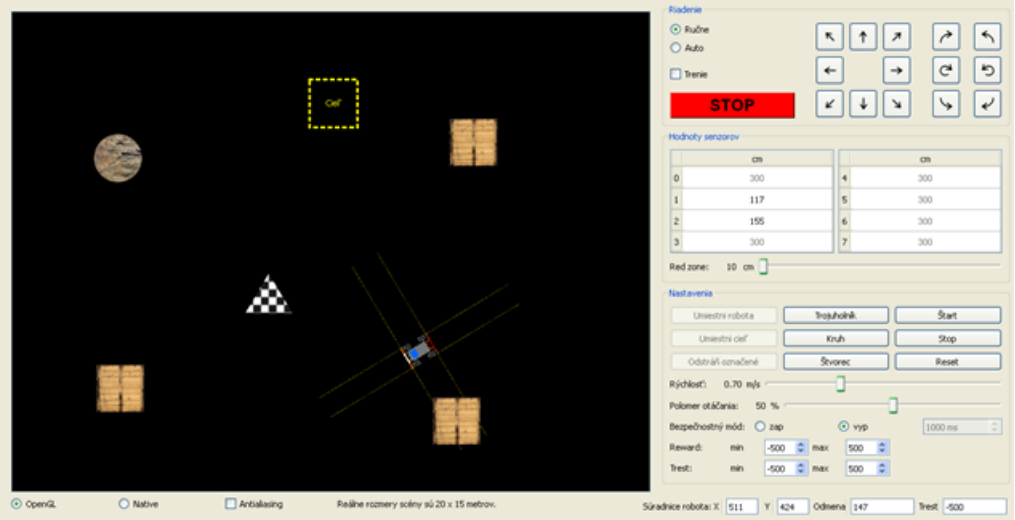
Podpora OpenGL predstavuje to, že scéna, ktorá zobrazuje pohyby robota bude vykresľovaná pomocou grafickej karty. Grafické karty zväčša podporujú OpenGL, ale podpora môže chýbať na strane systémových ovládačov, v tom prípade nie je možné využívať pre účely vykresľovania scény výkon grafickej karty, ale bude použitá natívna podpora. Natívna podpora znamená, že samotný procesor bude vykonávať všetko, čiže kresliť objekty na scénu.

Z hľadiska zaťaženia systému je vždy lepšie využívať podporu OpenGL, lebo sa jedná o hardvérovú akceleráciu, ktorá je pre tento typ operácií určená, a tým nie je



Obr. 4.2: Robotický simulátor s LCS

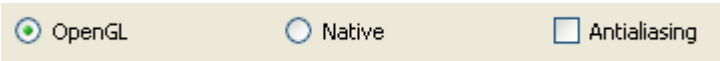
zbytočná záťaž namierená na procesor. Preto môže procesor vykonávať iné operácie, ktoré sú pre neho určené. V prípadoch, keď nie je možné využívať OpenGL je použitá natívna podpora v tomto prípade je možné na slabších procesoroch zaznamenať náznaky spomalenia chodu aplikácie, nižšie FPS (*frame per second* - počet obrázkov za sekundu).



Obr. 4.3: Okno aplikácie robotického simulátora

Vykresľovacia scéna programu, kde sa zobrazuje samotný robot, prekážky a cieľ je s podporou OpenGL, v prípade chýbajúcej podpory OpenGL je k dispozícii natívna podpora. Medzi OpenGL a natívnou podporou sa dá prepínať pomocou tlačidiel typu Radio Button. Je možné zapnúť aj jednoduché vyhladzovanie (Obr. 4.4).

Vyhladzovanie (antialiasing) znamená zaoblenie hrán objektov scény. Je to náročnejšie na vykresľovanie, ale v prípade OpenGL by nemalo byť badať žiadne zmeny ohľadne výkonu. V prípade natívnej podpory je možné postrehnúť u slabších procesorov zníženie výkonu, menšie FPS (frame per second).

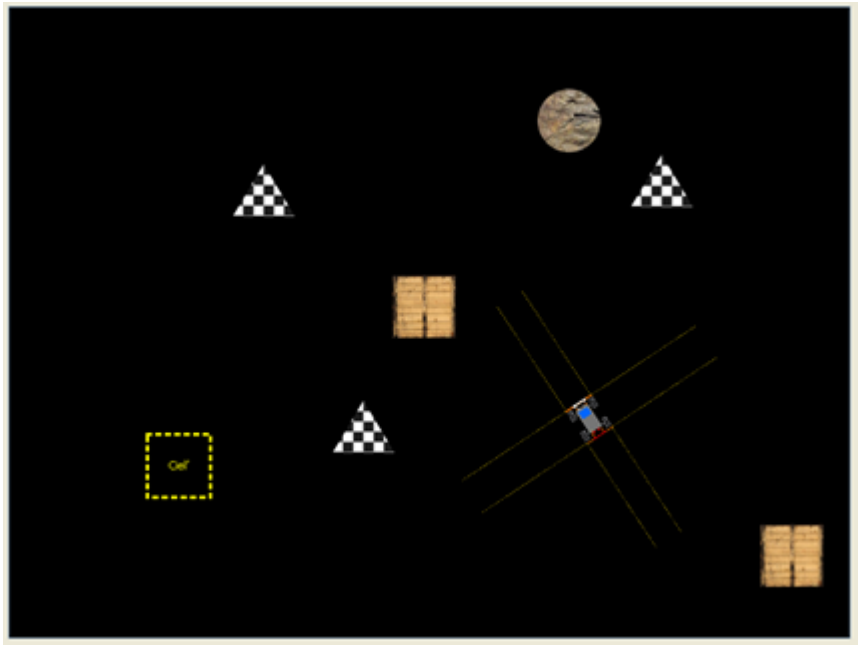


Obr. 4.4: Prepínanie OpenGL, Native, Antialiasing

Po spustení je preferované vykresľovanie pomocou OpenGL, ak je podporované systémom a ak nie je prepnuté na Native.

4.2.3 Virtuálna scéna

Scéna má rozmery 800 x 600 pixelov, v skutočnosti predstavuje 20 x 15 metrov (Obr. 4.5). Scéna je interaktívna, reaguje na akcie myši. Pomocou myši sa presúvajú zobrazené objekty a tiež označujú. Pri prejdení myšou ponad objekt, alebo označení (ľavým tlačidlom myši) objektu sa zobrazí v strede objektu zelený krúžok, v prípade ak je označený iba robot tak sa zvýrazní rozsah snímania senzorov taktiež zelenou farbou (Obr. 4.6).

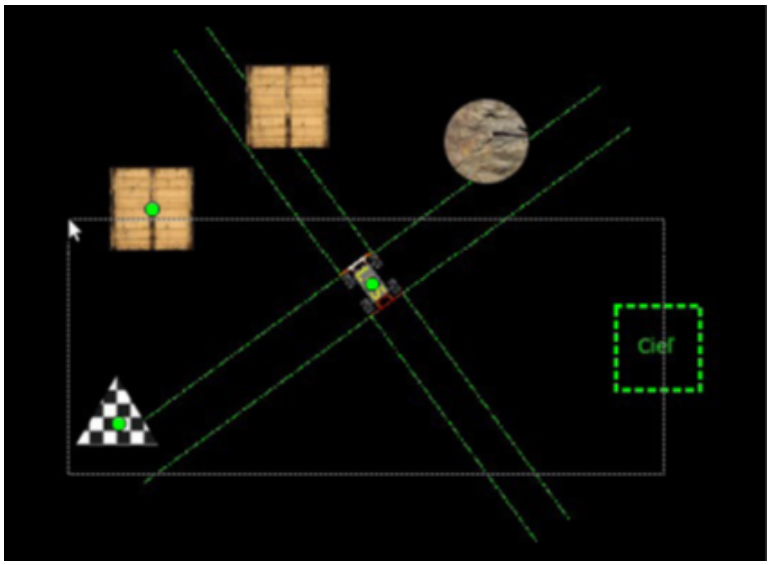


Obr. 4.5: Virtuálna scéna s mobilným robotom a prekážkami

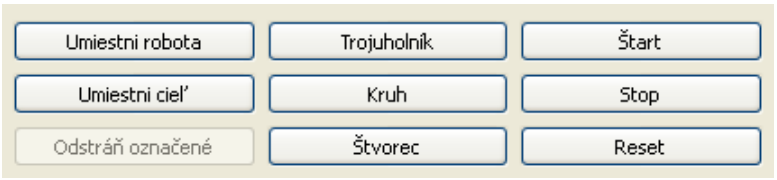
Do scény sa pridávajú objekty pomocou tlačidiel v pravej časti v sekcii nastavenia. Na výber máme pridanie robota, cieľa, trojuholníka, kruhu a štvorca. Na scénu sa dá pridať len jeden robot a len jeden cieľ. Po pridaní robota alebo cieľa sa dané tlačidlá správajú ako neaktívne, po odstránení sú zase aktívne. Prekážok môže byť na scéne ľubovoľný počet. Ak sú nejaké objekty scény označené, tlačidlo *Odstráň označené* je dostupné. Celkové vymazanie scény sa vykoná tlačidlom *Reset*. Tlačidlá Štart a Stop sú určené pre spustenie, alebo zastavenie pohybu robota.

4.2.4 Objekty scény

Medzi objekty scény simulátora patrí robot, cieľ, trojuholník, kruh, štvorec (Obr. 4.8) a medzi prekážky patrí aj samotné ohraničenie virtuálneho priestoru.



Obr. 4.6: Označenie objektov vo virtuálnej scéne

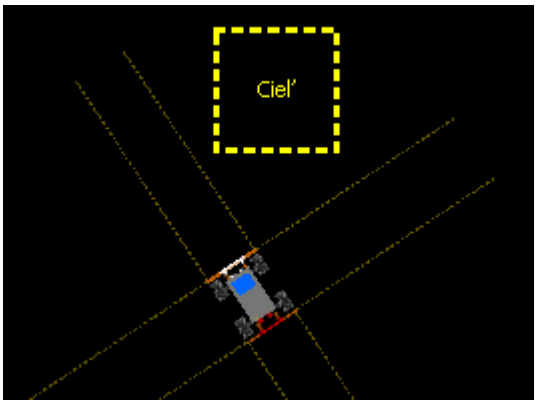


Obr. 4.7: Správa scény

K dispozícii sú tri typy prekážok, trojuholník, kruh a štvorec (Obr. 4.8). Pridávajú sa na do scény pomocou tlačidiel v sekcii nastavenia a to v ľubovoľnom počte (Obr. 4.7). Po pridaní sa automaticky zobrazia v strede scény. Potom ich je možné presúvať myšou, ľavým tlačidlom, buď jednotlivo alebo viacero. Odobrať je možné označené prekážky pomocou tlačidla *Odobrať označené* alebo všetky pomocou *Reset*.

Obr. 4.8: Tri druhy prekážok

Kruh má polomer 30 pixelov, čo predstavuje 75 cm a je zobrazený s textúrou kameňa. Trojuholník je rovnostranný s dĺžkou strany 60 pixelov, 150 cm, s textúrou šachovnice. Štvorec s dĺžkou strany 60 pixelov, 150 cm a textúrou drevenej debny.



Obr. 4.9: Cieľ robota

Objekt *Cieľ* je miesto, kam sa má robot dostať. Pridáva sa na scénu tlačidlom *Umiestni cieľ*. Je možné pridať len jeden cieľ na scénu. Po pridaní sa zobrazí v strede scény, dá sa presúvať pomocou myši. Odstrániť ho je možné pomocou *Odstrániť označené* alebo *Reset*. Rozmery cieľa sú 60 pixelov, čiže 150 cm. Je zobrazený v tvare štvorca, žltými obrysmi (Obr. 4.9).

4.2.5 Robot - detektory a efektory

Jedným z cieľov robotického simulátora bolo, aby verne napodobňoval kinematické schopnosti robota opísaného v časti 3.2.1. V režime ručného riadenia sa môže robot pohybovať pomocou série preddefinovaných tlačidiel, ktoré sú zobrazené na obrázku Obr. 4.10. Predefinované tlačidlá umožňujú dosiahnuť akýkoľvek pohyb zo všetkých možností uvedených na obrázku Obr. 3.6.

Na obrázku Obr. 4.11 je možné meniť rýchlosť robota v $m.s^{-1}$, ktorá je braná ako vstupný parameter do voleného pohybu. Hneď pod ňou sa nachádza ďalší posuvník pre nastavenie polomeru otáčania, aby bolo exaktne povedané ako veľmi sa má robot otáčať. Nasledujúca časť kódu dokumentuje pohybové schopnosti robota (Algoritmus 4.1).

Algoritmus 4.1: Pohybové schopnosti robota

```
switch(robot_directions)
    {
    case Robot::Left:
        robot_item->moveLeft();
        break;
    case Robot::Right:
        robot_item->moveRight();
        break;
    case Robot::Up:
```

Obr. 4.10: Znázornenie nameraných hodnôt senzorov a možnosti riadenia mobilného robota

```
        robot_item->moveUp();
        break;
    case Robot::Down:
        robot_item->moveDown();
        break;
    case Robot::Rotate_Left:
        robot_item->rotateLeft(rot_step);
        break;
    case Robot::Rotate_Right:
        robot_item->rotateRight(rot_step);
        break;
    case Robot::Up_Left:
        robot_item->moveUpLeft();
        break;
    case Robot::Up_Right:
        robot_item->moveUpRight();
        break;
    case Robot::Down_Left:
        robot_item->moveDownLeft();
        break;
    case Robot::Down_Right:
        robot_item->moveDownRight();
        break;
    case Robot::Cross_Down_Left:
        robot_item->moveCrossDownLeft();
        break;
    case Robot::Cross_Down_Right:
        robot_item->moveCrossDownRight();
        break;
    case Robot::Cross_Up_Left:
        robot_item->moveCrossUpLeft();
        break;
    case Robot::Cross_Up_Right:
        robot_item->moveCrossUpRight();
        break;
    }
```

Uvedená časť kódu (Algoritmus 4.1) zabezpečuje samotný pohyb robota. Nachádza sa v metóde *MainWidget::moveRobot()*, táto metóda sa volá pri každom tiknutí časovača, čo je každých 33 ms a premenná *robot_directions* si uchováva aktuálny smer pohybu robota, po skontrolovaní smeru sa robot pohne daným smerom.

Presnosť pohybov tohto typu robotického podvozku je známa svojou silnou závislosťou od kvality povrchu (nerovnomerného rozloženia trenia), na ktorej sa robot pohybuje. Aby sme sa priblížili reálnemu modelu, simulátor umožňuje aktivovať položku Trenie, ktoré sa nachádza pod položkami *Auto* a *Ručne ovládanie* (Obr. 4.10). Trenie vnáša do žiadaného pohybu náhodné pohybové deje, ktoré fungujú v automatickom aj ručnom režime. Nasledujúca časť kódu dokumentuje simuláciu trenia reálneho robota v prostredí(Algoritmus 4.2).

Algoritmus 4.2: Tvorba náhodných pohybov - trenie

```
    if (is_fraction)
    {
        if (rand() \% NOISE_LEVEL == 0)
        {
            switch(rand() \% 6)
            {
            case 0:     // pohyb vpred
                robot_item->moveUp();
                break;
            case 1:     // pohyb vzad
                robot_item->moveDown();
                break;
            case 2:     // mecanum vpravo
                robot_item->moveRight();
                break;
            case 3:      // mecanum vlavo
                robot_item->moveLeft();
                break;
            case 4:      // otacanie na mieste v smere hodinovych ruciciek
                robot_item->rotateRight(0.5);
                break;
            case 5:     // otacanie na mieste proti smeru hodinovych ruciciek
                robot_item->rotateLeft(0.5);
                break;
            }
        }
    }
```

Pre simuláciu trenia sa kontroluje pri každom tiknutí časovača, či je premenná *is_fraction true* alebo *false*, a ak je *true*, tak sa náhodne rozhodne o vykonaní náhodného pohybu.

Algoritmus 4.3: Kontrola zrážky s objektom

```
if (item != target_item && item->collidesWithItem(robot_item))
       {     qDebug() << "MainWidget::moveRobot() -> kolizia";
            stopAndBack();
        }
```

V uvedenej časti kódu (Algoritmus 4.3) sa kontroluje či robot nekoliduje s nejakým iným objektom na scéne a volá sa pre každý objekt, ktorý je rôzny od cieľa, robota a jeho senzorov. Keď koliduje tak sa zastaví časovač a pohne o krok späť.

Robot má osem senzorov, ktorých dosah je zobrazovaný žltou prerušovanou čiarou. Aktuálne nastavený rozsah je do troch metrov v rozlíšení 1cm. Hodnoty senzorov sú zobrazované v položke *Hodnoty senzorov* (Obr. 4.10). Vľavo je vždy zobrazený index snímača od 0 do 7 a tiež aj jeho vypísaná (nameraná) hodnota.

V uvedenej časti kódu (Algoritmus 4.4) sa sprostredkúva meranie vzdialenosti, kde používa pomocné štruktúry uvedené na začiatku kódu. Dosah senzora je tiež objekt na scéne a pri každom tiknutí časovača sa kontroluje, či rozsah senzora nekoliduje s nejakým iným objektom na scéne ako je robot. Ak áno tak sa pomocou pomocnej štruktúry, ktorá je dočasný senzor, začne merať vzdialenosť. Meria prebieha tak, že dočasný senzor sa postupne zväčšuje od minimálnej vzdialenosti až pokiaľ nedosiahne na objekt alebo svoju maximálnu vzdialenosť. Potom sa aktuálna hodnota senzora zapíše do externej premennej *g_ultra_inputs*, ktorý reprezentuje vektor hodnôt senzorov.

Algoritmus 4.4: Meranie vzdialenosti

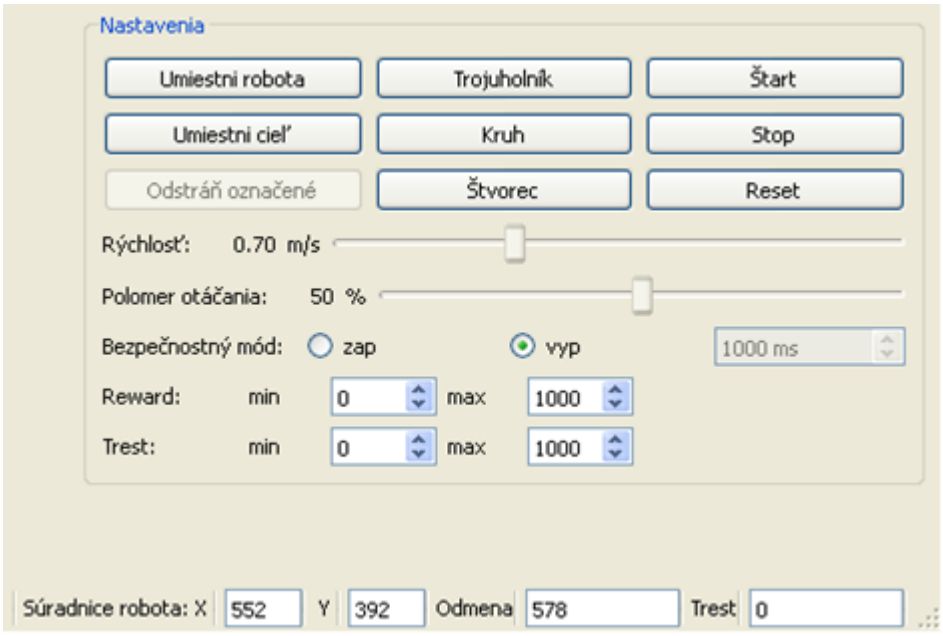
```
QGraphicsItem *item;
QGraphicsLineItem *sensor;
QLineF sensor_line; // uchovanie si linu senzora
QLineF tmp_line; // docasna lajna ktora zisti dlzku
unsigned char collided_sensor_flags = 0x00;
// priznaky, ze ktory senzor naposledy meral
int i; // poradie senzora
qreal j; // vzdialenost senzora 0 po 1
std::vector<int> distance_vector(8, SENSOR_RANGE);
foreach(item, scene_view->scene()->items())
{
   // ked je item == sensor alebo sam robot tak nezistujeme koliziu
   if (item == robot_item || sensor_list->contains(item))
   {
           continue;
   }
   // zistovanie vzdialenosti, ci nejaky rozsah senzora uz dociahne na objekt
   // indikuje zaciatok merania
       for(i = 0; i < 8; i++)
   {
       if (sensor_list->at(i)->collidesWithItem(item))
       {
           // nastavenie priznaku merania aktualneho senzora
           collided_sensor_flags |= (unsigned char)(pow(2, i));
           // vytvorim si smernik na senzor
           sensor = (QGraphicsLineItem*) sensor_list->at(i);
           sensor_line = sensor->line();
           tmp_line = sensor_line;
           #ifndef NO_DISTANCE_DEBUG
           qDebug() << "MainWidget::moveRobot() sensor measuring detected " << i
               ;
           #endif
```

```
            // od zaciatku merania sensora kontrolujem
                     kedy dosiahne ciel, a to je vzdialenost
             for (j = 0.003; j <= 1; j += 0.003)//0.002)
             {
                 // postupne sa zvecujuca lajna
                 tmp_line.setP2(sensor_line.pointAt(j));
                 sensor->setLine(tmp_line);
                 // zistim ci uz po zveceni dociahne na
                 //objekt, ak hej posle cislo senzora a vzdialenost
                 if (item != target_item && sensor->collidesWithItem(item))
                 {
                     // pozre ci nebol nejaky objekt blizsie
                     //ak hej tak necha jeho hodnotu
                    if (distance_vector[i] > j * SENSOR_RANGE)
                      {
                         distance_vector[i] = j * SENSOR_RANGE;
                      }
                     #ifndef NO_DISTANCE_DEBUG
                     qDebug() << "MainWidget::moveRobot() senzor " << i << "
                         vzdialnost od objektu " << j;
                     #endif
                     // pokracujem na dalsi senzor ak moze merat
                      reak;
                 }
             }
             sensor->setLine(sensor_line);
         }
     }
     // samotna kolizia robota s objektom
     if (item != target_item && item->collidesWithItem(robot_item))
     {
         qDebug() << "MainWidget::moveRobot() -> kolizia";
         stopAndBack();
      }
}
int min_distance = SENSOR_RANGE;
// posle len minimalne vzdialenosti
for (i = 0; i < 8; i++)
{
     sensors->setSensorValue(i, distance_vector[i]);
     // najde minimalnu vzdialenost
     if (min_distance > distance_vector[i])
     {
         min_distance = distance_vector[i];
     }
     std::string tmp;
     int tmp_value = distance_vector[i];
     std::string buffer = QString().setNum(tmp_value).toStdString();
     // priprava retazca pre LCS
     tmp += "0";
     tmp += (i + '0');
     tmp += ",";
     if (tmp_value < 10)
     {
   tmp += "00" + buffer;
     }
     else if (tmp_value < 100)
     {
         tmp += "0" + buffer;
     }
```

```
    else
    {
        tmp += buffer;
    }
    tmp += ",0000000000";
    // koniec pripravy retazca pre LCS zapis retazca do externej premennej
    mutex.lock();
    g_ultra_inputs[i] = tmp;
    mutex.unlock();
}
```

Pod hodnotami senzorov sa nachádza posuvník *Red zone* (hraničnej zóny), ktorý reaguje na hraničné (nebezpečné) priblíženie sa robota k objektu. Ak senzor zaznamená hodnotu menšiu ako je *red zone*, tak sa pozadie nameranej hodnoty zmení na červenú farbu. Táto funkcia simuluje taktilné snímače, ktoré majú v prípade robota iba binárny výstup.

V druhej kapitole boli opísané rôzne druhy ohodnocovania systému LCS, teda poskytovanie odmeny alebo trestu za akcie, ktoré vykonáva robot. Simulátor sleduje, kde sa nachádza robot a kde cieľ. Na základe toho sa vypočítava veľkosť odmeny, ktorá je odovzdávaná systému LCS vo forme numerickej reprezentácie. To isté sa deje aj s trestom teda so zápornou odmenou. Tá je priamo prepojená s funkciou *red zone*. K dispozícii je možnosť nastaviť maximálnu a minimálnu odmenu, prípadne trest. Položky sú zobrazené na obrázku Obr. 4.11.



Obr. 4.11: Ďalšie nastavenia a funkcie simulátora

V uvedenej časti kódu (Algoritmus 4.5) sa zabezpečuje prepočítavanie odmeny a trestu, teda ak sa na scéne nachádza cieľ tak sa zisťuje vzdialenosť robota od cieľa.

Algoritmus 4.5: Prepočítavanie odmeny vzhľadom na vzdialenosť od cieľa

```
if (target_item)
    {
        int distance = sqrt(pow(target_item->pos().x()
                            - robot_item->pos().x(), 2)
                           + pow(target_item->pos().y()
                           - robot_item->pos().y(), 2));
        emit scoreChanged(distance < reward_range ?
        k * distance + reward_max : reward_min);
        qDebug() << "MainWidget::moveRobot distance
from target -> " << distance;
    }
    // zmena trestu
    emit penaltyChanged(min_distance < red_zone ?
   k_penalty * min_distance + penalty_max : penalty_min);
```

Odmena sa mení lineárne na základe vzdialenosti. Čím bližšie k cieľu tým väčšia odmena. Minimálna odmena je v prípade vzdialenosti rovnej veľkosti uhlopriečky simulačného prostredia. Trest sa prepočítava pri každom pohybe robota a tiež sa jedná o lineárnu funkciu. Vyberaná je minimálna vzdialenosť meraných hodnôt, z ktorej sa vypočíta trest. Čím bližšie od objektu sa robot nachádza, tým väčší trest dostane. Minimálny trest robot dostáva vtedy, keď je vzdialenosť od objektu väčšia ako nastavená červená zóna.

4.2.6 Systém odmien pre reálne prostredie

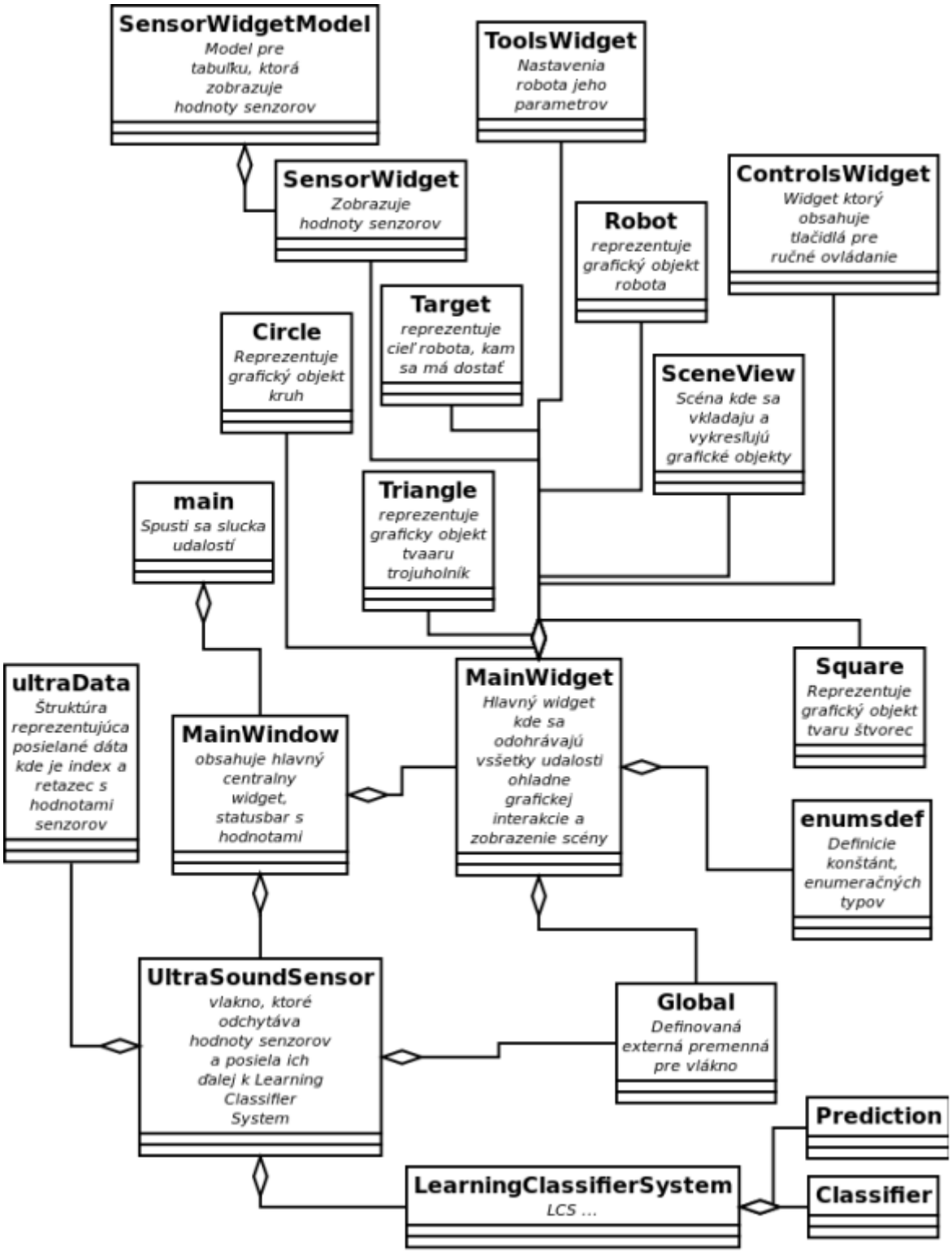
Pri experimentoch v reálnom prostredí bol navrhnutý tabuľkový systém odmien znázornený v Tab. 4.1. Tento systém bol použitý, pretože v reálnom prostredí neboli k dispozícii iné globálne senzory, s pomocou ktorých by bolo možne vytvoriť iný komplexnejší druh odmeňovania. Prvý stĺpec tabuľku znázorňuje rôzne strany robota (predná, zadná, práva, ľavá a všetky rohy). Ďalej nasleduje osem stĺpcov rôznych hodnôt senzorov, kde sa používa osem ultrazvukových senzorov pre meranie vzdialenosti. Pretože, LCS vo svojim procese používa diskrétne hodnoty, bolo potrebne rozdeliť merací rozsah na viacero zón, napríklad: H - veľká, M - stredná, L - malá. Posledný stĺpec znázorňuje druh pohybu robota podobne ako tlačítka v hrach v hrach W - S - A - D a pohyby robota do bokov Q - E.

Tabuľka 4.1: Systém odmien v LCS - XCS

side of the robot	0	1	2	3	4	5	6	7	Move
right side	X	X	X	X	X	M	M	X	W
right side	X	X	X	X	X	H	M	X	A
right side	X	X	X	X	X	M	H	X	D
right side	X	X	X	X	X	M	L	X	Q
right side	X	X	X	X	X	L	M	X	Q
right side	X	X	X	X	X	L	L	X	Q
left side	X	M	M	X	X	X	X	X	S
left side	X	H	M	X	X	X	X	X	A
left side	X	M	H	X	X	X	X	X	D
left side	X	M	L	X	X	X	X	X	E
left side	X	L	M	X	X	X	X	X	E
left side	X	L	L	X	X	X	X	X	E
front	M	X	X	X	X	X	X	M	Q
front	H	X	X	X	X	X	X	M	D
front	M	X	X	X	X	X	X	H	A
front	M	X	X	X	X	X	X	L	S
front	L	X	X	X	X	X	X	M	S
front	L	X	X	X	X	X	X	L	S
rear	X	X	X	M	M	X	X	X	E
rear	X	X	X	H	M	X	X	X	A
rear	X	X	X	M	H	X	X	X	D
rear	X	X	X	M	L	X	X	X	W
rear	X	X	X	L	M	X	X	X	W
rear		X	X	L	L	X	X	X	W
front right corner	M/H	X	X	X	X	M/H	M/H	M/H	Q
front left corner	M/H	M/H	M/H	X	X	X	X	M/H	S
rear right corner	X	M/H	M/H	M/H	M/H	X	X	X	E
rear left corner	X	X	X	M/H	M/H	M/H	M/H	X	W

4.2.7 Štruktúra programového vybavenia simulátora

Program Simulátora s LCS poskytuje grafické prostredie pre simuláciu robota. Robota je možné ovládať ručne alebo automaticky pomocou LCS. Ručné ovládanie sa vykonáva pomocou tlačidiel, ktoré vysielajú signály po kliknutí (*ControlsWidget*) a sú zachytávané a spracovávané pomocou slotov v triede *MainWidget* a odovzdávajú ich ako povely pohybov robota (Obr. 4.12). Pre automatické riadenie je určené LCS, ktoré sa spúšťa v osobitnom vlákne a spracováva hodnoty senzorov robota a potom posiela vygenerované výsledky ako signály pre robota, ktoré sú tiež zachytené a spracované pomocou slotov v triede *MainWidget*. Posúvajú sa ďalej ako povely pre pohyby robota. Hodnoty senzorov sú ukladané do externej premennej *g_ultra_inputs*, čo je vektor definovaný v triede *global.cpp*. Pri zapisovaní a čítaní je premenná chránená pre zachovanie konzistencie dát pomocou mutexu.



Obr. 4.12: Bloková schéma nášho programu robotického simulátora s LCS

Kapitola 5

Výsledky experimentov

Táto posledná kapitola sa venuje hlavne grafickému znázorneniu dát, získaných počas simulácií. V nasledujúcich častiach sa nachádzajú vykreslené priebehy polohy ťažiska robota v simulačnom prostredí, ktorého virtuálny priestor mal veľkosť 800 x 600 pixelov, čo v reálnom svete zodpovedá hodnote 20 x 15 metrov.

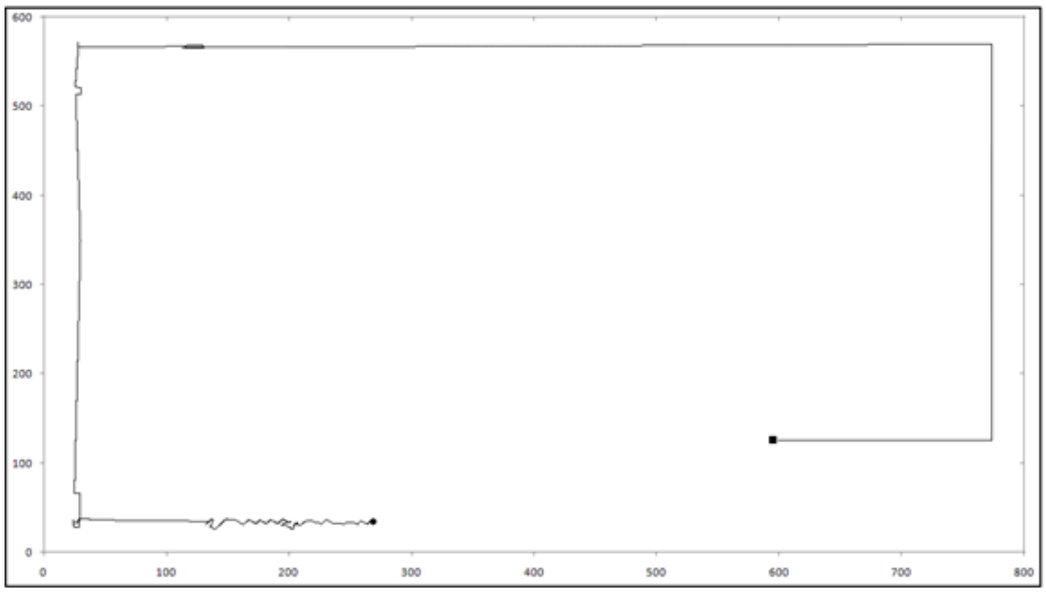
Prvou úlohou robota bolo sledovanie stien a druhou úlohou obchádzanie prekážok. Rôzne podmienky robota v simulačnom prostredí zabezpečovali dva parametre, ktoré majú signifikantný vplyv aj v reálnom prostredí. Prvý parameter udáva rýchlosť robota v, ktorá sa pohybovala v rozsahu od 0,5 $m.s^{-1}$ do 1,5 $m.s^{-1}$. Druhý parameter, buď pridáva alebo nepridáva do pohybu robota náhodné pohybové javy, ktoré sú spôsobené nerovnomerným trením medzi robotom a podložkou (funkcia Trenie opísaná v časti 4.2.5). V závere kapitoly je opísaný experiment v reálnom prostredí, kde úlohou robota bolo sledovanie stien.

5.1 Sledovanie stien

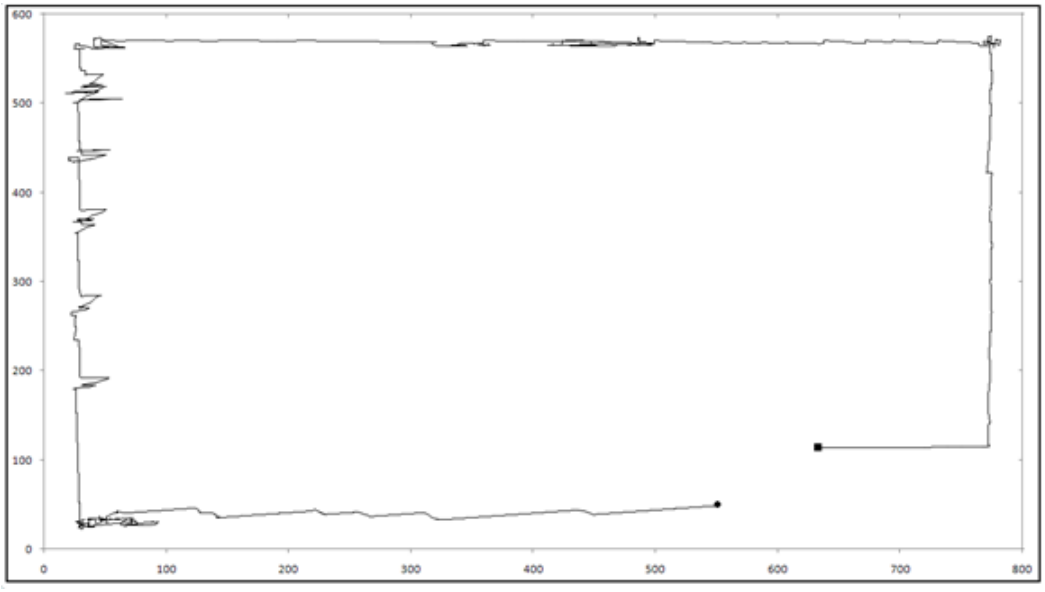
Prvou úlohou robota bolo sledovanie stien. Robot bol umiestnený do počiatočnej polohy, v obrázku označenej čiernym štvorcom. Finálna poloha robota, kedy bola simulácia prerušená je označovaná čiernou kružnicou. Na obrázku Obr. 5.1 sa robot po spustený snaží nájsť stenu, pri ktorej následne uplatní nové pravidlo. Rýchlosť robota je $v = 0,5\ m.s^{-1}$ a podmienky sú ideálne, teda nemusí kompenzovať žiadne nežiaduce premenlivé trenia. Robot nemal problém sledovať stenu a zadanú úlohu zvládol.

Na ďalšom obrázku Obr. 5.2 bola rýchlosť totožná, ale trecie podmienky sa zmenili a jeho pohyb už nie je taký ideálny ako predošlom prípade. Na tomto príklade vidíme, že robot aktívne kompenzoval tieto náhodné javy, pričom výsledkom LCS bola nutnosť použiť viac akcií v kratšom čase. Prvý problém nastal v pravom hornom

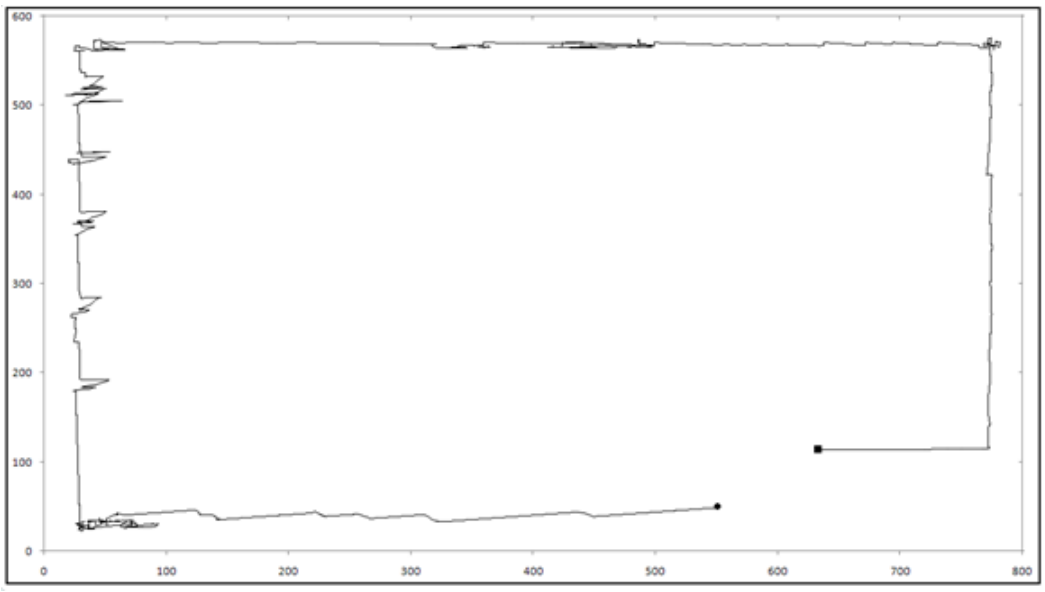
rohu, kedy robot nedosiahol požadovaný odstup na prvý krát. Nasledujúce trajektórie majú tiež menej kvalitný priebeh. Robot však splnil aj tuto úlohu, no s väčším počtom akcií.



Obr. 5.1: Sledovanie stien, ideálne podmienky, $v = 0,5\ m.s^{-1}$

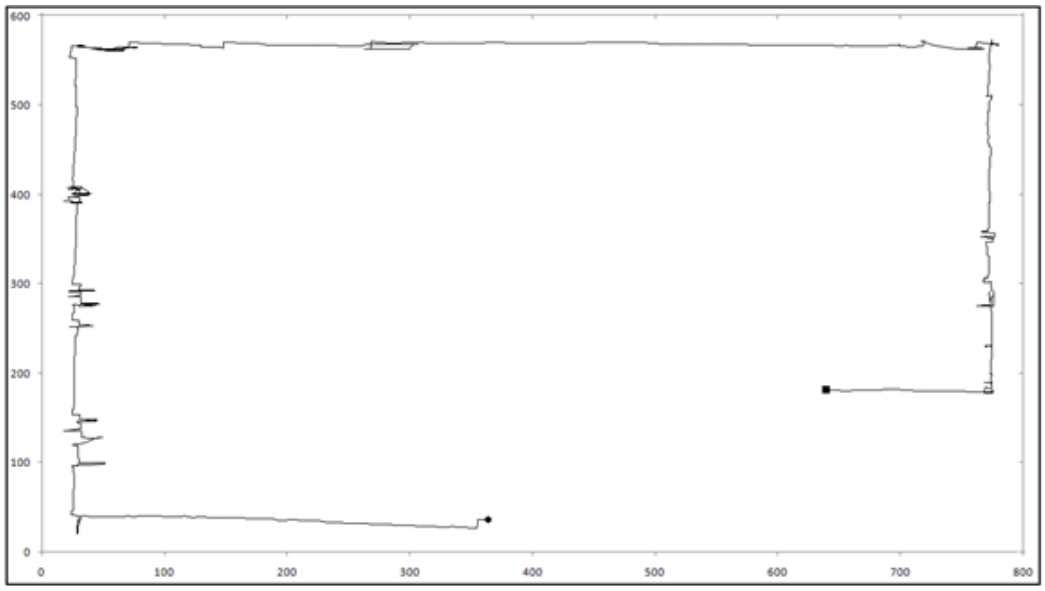


Obr. 5.2: Sledovanie stien, reálne podmieky, $v = 0,5\ m.s^{-1}$



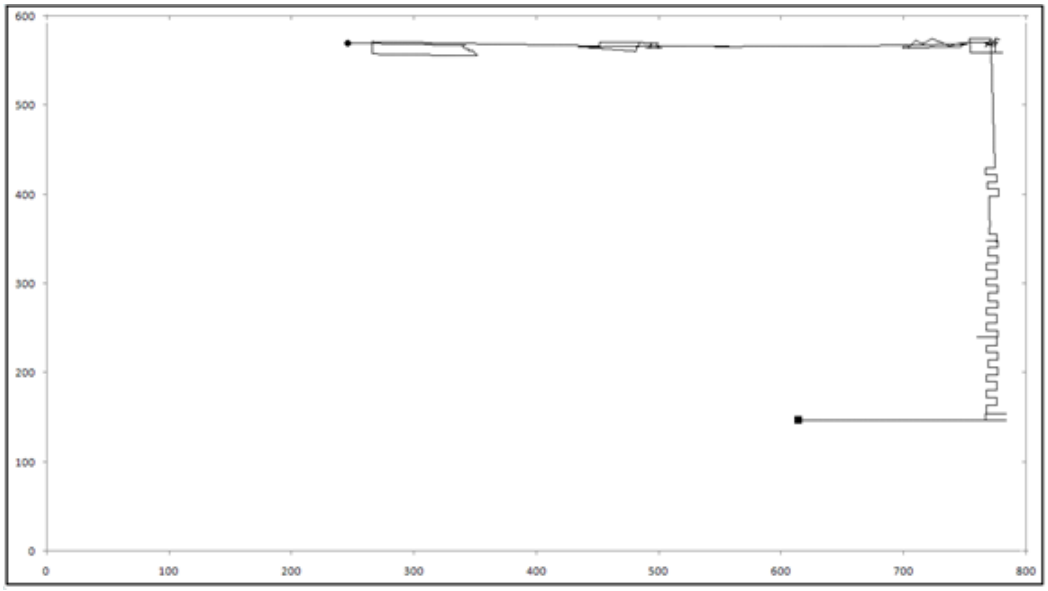
Obr. 5.3: Sledovanie stien, ideálne podmieky, $v = 0,7\ m.s^{-1}$

V pokuse na obrázku Obr. 5.3 sú podmienky veľmi podobné ako to bolo v prvom experimente ale celková rýchlosť robota bola zväčšená na $v = 0,7\ m.s^{-1}$. Zo začiatku je jeho pohyb plynulý ako to bolo v prvom prípade, no v rohu nastáva prvý problém, pretože odozvy z prostredia sú rýchlejšie, a tým je na skúšanie nových akcii menej času. Tento problém sa prejavil aj vo viac zakmitaných rovných úsekoch.



Obr. 5.4: Sledovanie stien, reálne podmieky, $v = 0,7\ m.s^{-1}$

Ďalší pokus na obrázku Obr. 5.4 je trochu paradoxom. Pozorovateľ by po pridaní náhodných faktorov spôsobených nedokonalým trením, očakával zhorenie kvality trajektórie. Ak si tento priebeh porovnáme zo situáciou v rohoch simulátora na obrázku Obr. 5.3, priebeh je plynulejší. Ale pri prvotnom nájdení steny je trajektória menej kvalitná ako to bolo vo všetkých predošlých prípadoch.



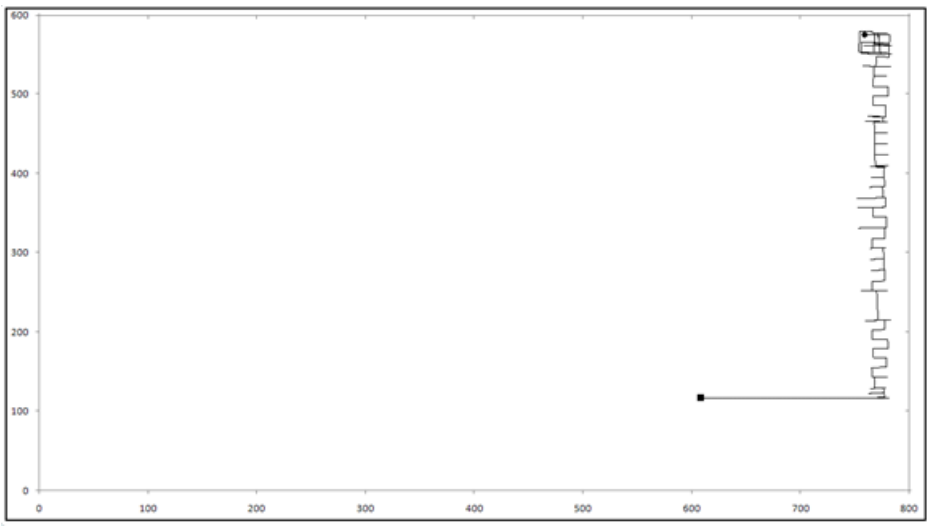
Obr. 5.5: Sledovanie stien, ideálne podmieky, $v = 1\ m.s^{-1}$

V predposlednej úlohe na obrázku Obr. 5.5 už nemalo význam priadavať trenie, pretože oproti rýchlosti robota bolo zanedbateľné. Rýchlosť robota je už dvojnásobná oproti prvému experimentu, teda $v = 1,0\ m.s^{-1}$. Už pri úvode sledovania, vykazoval robot značné problémy, ktoré sú spôsobené veľmi rýchlou interakciou s prostredím, kde robot často „prepínal“ medzi akciami. V obrázku aj vidno, že sa robot niekoľko krát vzdiali od steny natoľko, že ju chvíľu nevedel detegovať. Podmienky tohto experimentu môžeme považovať na hranici únosnosti a schopnosti robota.

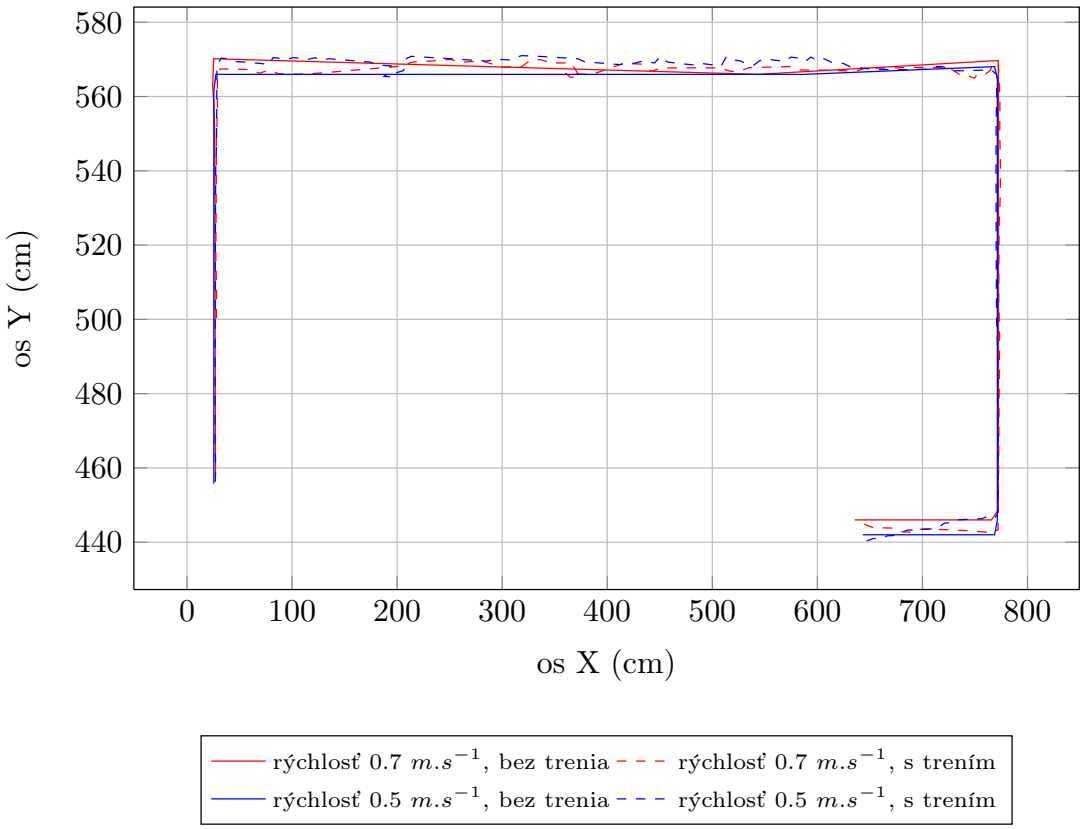
V poslednom experimente na obrázku Obr. 5.6 v úlohe sledovania steny, robot značne kmital a nevedel zaujať monotónny priebeh. Jeho rýchlosť predstavovala trojnásobok oproti prvému experimentu, teda $v = 1,5\ m.s^{-1}$. Tento pokus hodnotíme ako nezvládnutý, pretože robot sa nedokázal dostať z prvého rohu a tak nemalo význam testovať náročnejšie podmienky.

Na obrázku Obr. 5.7 je možné vidieť porovnanie trajektórií robota v dvoch rôznych rýchlostiach a tiez ako ne vplyva aktivovane trenie, ktoré spôsobí nerovnomerný pohyb robota hlavne v bočných smeroch.

V ďalšej časti sa budeme venovať podobnému experimentu , kde úlohou robota bude obchádzanie prekážok.



Obr. 5.6: Sledovanie stien, ideálne podmieky, $v = 1,5\ m.s^{-1}$



Obr. 5.7: Porovnanie trajektórií robota v simulovanom prostredí

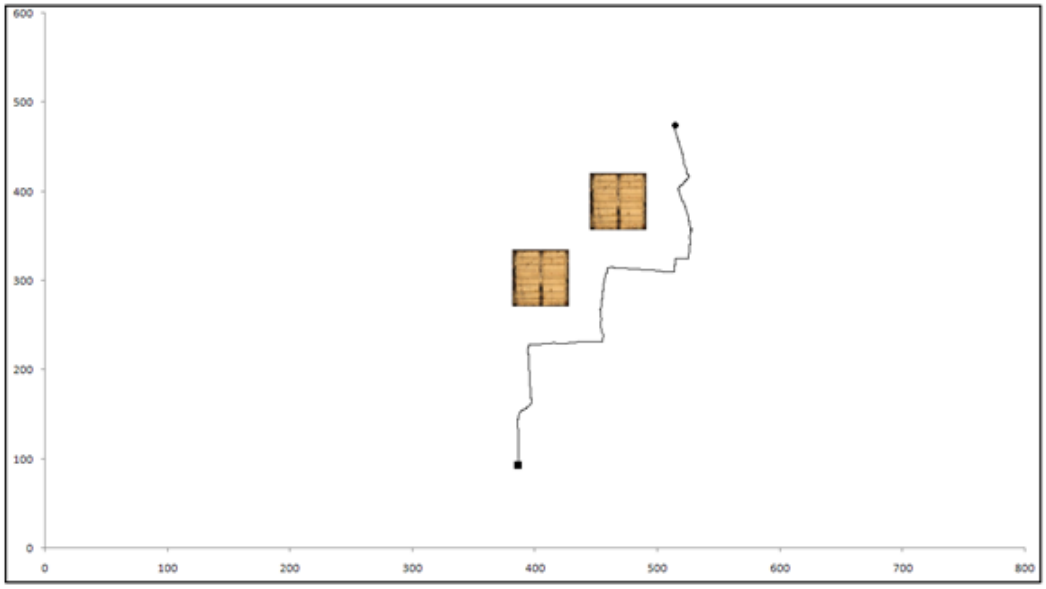
5.2 Obchádzanie prekážok

Druhou úlohou robota bolo obchádzanie prekážok. Robot bol umiestnený do počiatočnej polohy, v obrázku označenej čiernym štvorcom, podobne ako to bolo v pokusoch opísaných v časti 5.1. Finálna poloha robota, kedy bola simulácia prerušená je označovaná čiernou kružnicou. Pre modelovú situáciu sme použili dve prekážky typu štvorec s textúrou dreva.

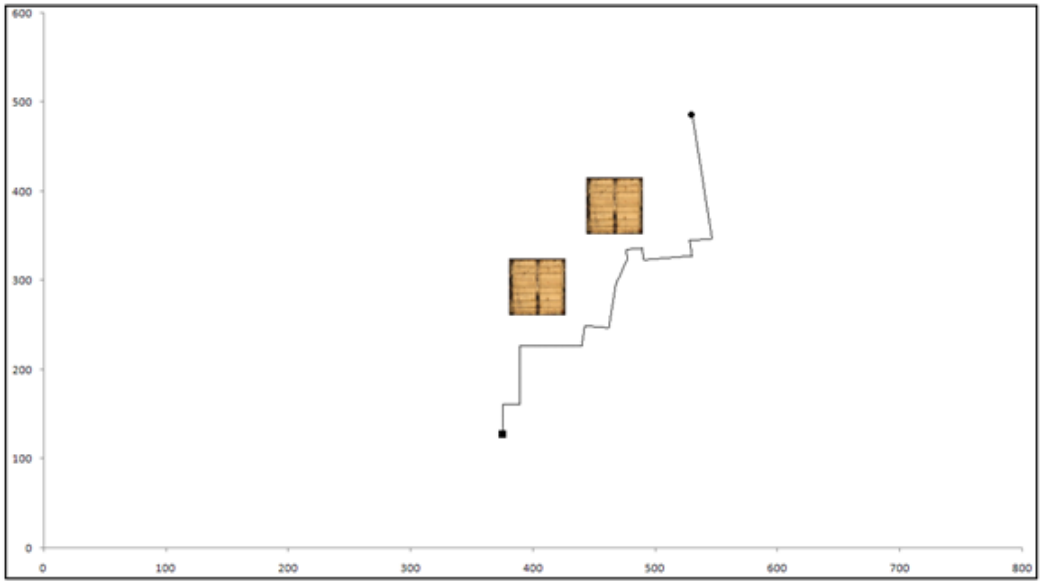
Na obrázku Obr. 5.8 sa robot po spustený pohybuje rovno vpred a pri nájdení prekážky uplatni nové pravidlo, ktoré zabezpečí bezkolízny stav. Rýchlosť robota je $v = 0,5\ m.s^{-1}$ a podmienky sú ideálne, teda nemusí kompenzovať žiadne nežiaduce premenlivé trenia. Podľa uvedenej situácie, robot obišiel prekážky takmer ideálne. Trajektória, ktorá vpravo od objektu pripomína tvar "malých schodíkov" je spôsobená tým, že sa robot rozhodol obísť prekážku skôr ako to bolo možné. Senzory robota následne zistili, že sa prekážka nachádza bližšie ako je požadované, a tak zvolil ďalšie akcie, ktoré mu pomohli dostať sa od prekážky ďalej. Robot nemal problém obchádzať prekážky a zadanú úlohu zvládol skoro ideálne.

Obr. 5.8: Obchádzanie prekážok, ideálne podmieky, $v = 0,5\ m.s^{-1}$

Na ďalšom obrázku Obr. 5.9 bola rýchlosť totožná ale trecie podmienky sa zmenili a jeho pohyb už nie je taký ideálny ako v predošlom prípade. Na tomto príklade vidíme, že robot aktívne kompenzoval tieto náhodné javy, pričom výsledkom LCS bola nutnosť použiť viac akcií v kratšom čase. Oproti predošlému príkladu je trajektória robota horšej kvality, ale tento pokus hodnotíme podobne ako prvý, teda robot splnil aj tuto úlohu, no s väčším počtom akcii.



Obr. 5.9: Obchádzanie prekážok, reálne podmieky, $v = 0,5\ m.s^{-1}$



Obr. 5.10: Obchádzanie prekážok, ideálne podmieky, $v = 1,0\ m.s^{-1}$

V pokuse na obrázku Obr. 5.10 sú podmienky veľmi podobné ako to bolo v prvom experimente ale celková rýchlosť robota bola zväčšená na $v = 1,0\ m.s^{-1}$. Zo začiatku je jeho pohyb plynulý ako to bolo v prvom prípade, no pri obchádzaní druhej prekážky nastáva menší problém, pretože odozvy z prostredia sú rýchlejšie, a tým je na skúšanie nových akcii menej času. Robot zadanú úlohu zvládol.

Obr. 5.11: Obchádzanie prekážok, reálne podmieky, $v = 1,0\ m.s^{-1}$

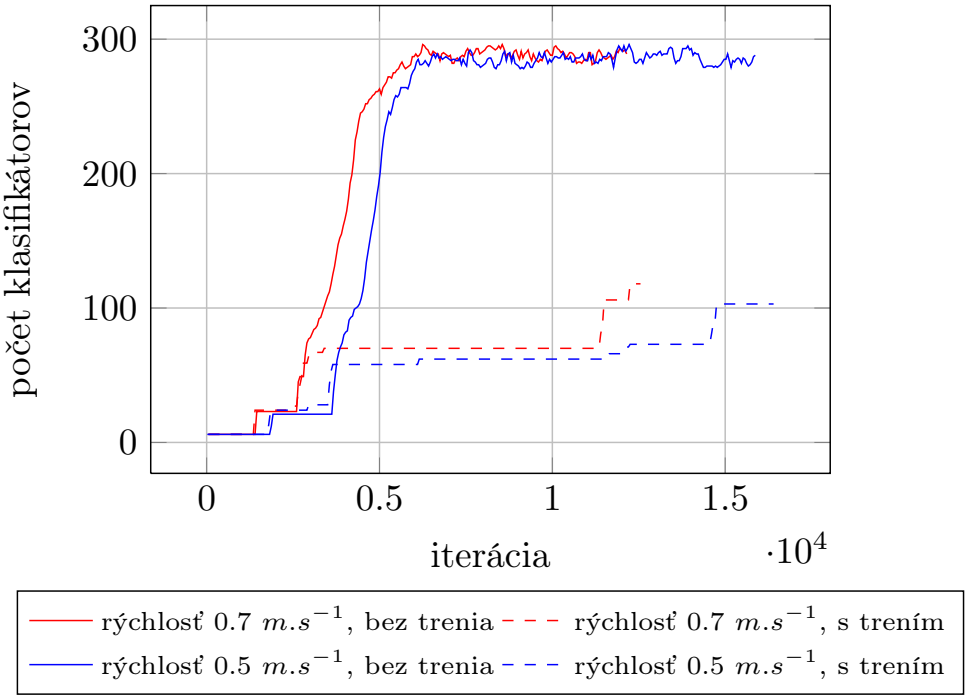
Podmienky ďalšieho pokusu na obrázku Obr. 5.11 sú trochu iné. Rýchlosť robota bola totožná ale trecie podmienky sa zmenili, no jeho pohyb je porovnateľný s predošlým prípadom. Na tomto príklade vidíme, že robot aktívne kompenzoval tieto náhodné javy, pričom výsledkom LCS bola nutnosť použiť ešte viac akcií v kratšom čase. Pri obchádzaní druhej prekážky robot mierne zaváhal no zadanú úlohu hodnotíme tiež ako splnenú.

Obr. 5.12: Obchádzanie prekážok, reálne podmieky, $v = 1,5\ m.s^{-1}$

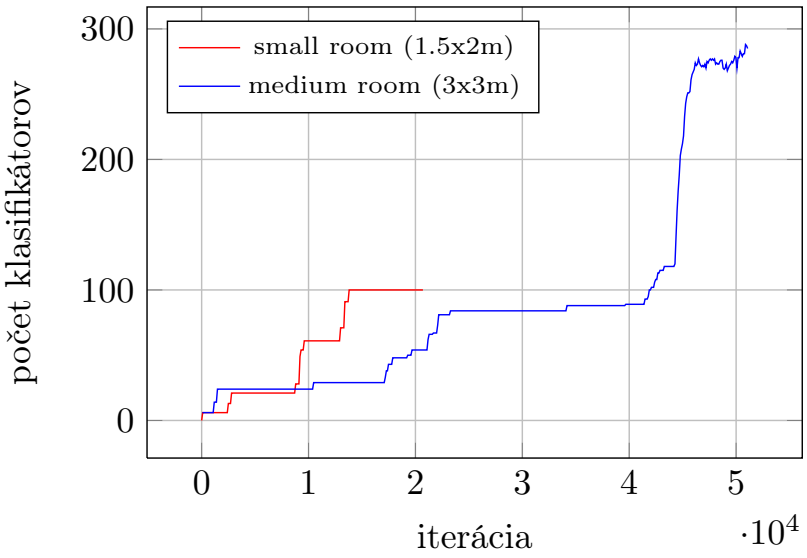
V poslednom experimente na obrázku Obr. 5.12 v úlohe obchádzania prekážok, robot začal vykonávať akcie, ktoré neboli žiaduce a v závere nevedel zaujať monotónny priebeh. Jeho rýchlosť predstavovala trojnásobok oproti prvému experimentu, teda $v = 1,5\ m.s^{-1}$ a do jeho pohybov boli pridané nežiaduce trenia. Prvú prekážku obišiel veľmi rýchlo a prekvapivo bez problémov. Pri druhej prekážke už náhle potreboval kvôli rýchlosti vykonať viac akcii, ktoré už nebol schopný vhodne aplikovať. Tento záverečný pokus hodnotíme ako nezvládnutý, a tak nemalo význam testovať náročnejšie podmienky.

5.3 Počet klasifikátorov

V našom programe s LCS je možné obmedziť maximálny počet klasifikátorov, na ktorých operuje evolučný proces. Po naplnení maximálneho počtu začína LCS vyraďovať nepotrebné alebo inak neefektívne, aby bolo možné množinu priebežne doplňovať o nové a úspešnejšie. Zaujímavé je sledovať ako sa vyvíja ich počet po spustení programu vzhľadom na interakciu s prostredím. Na obrázku Obr. 5.13 je graf znázorňujúci počet klasifikátorov v čase pri simulovanom prostredí a na Obr. 5.14 v reálnom prostredí.



Obr. 5.13: Vývoj počtu klasifikátorov v čase pre robota v simulovanom prostredí



Obr. 5.14: Vývoj počtu klasifikátorov v čase pre robota v reálnom prostredí

5.4 Sledovanie stien v reálnom prostredí

V tejto časti by sme chceli nadviazať na experimenty vykonané s reálnym robotickým systémom, ktorý bol opísaný v časti 3.2.1. Jednalo sa o semestrálny projekt, ktorý bol riešený v rámci študijného pobytu Erasmus na Viedenskej univerzite (Universität Wien), kde sa nám podarilo v obmedzenom rozsahu otestovať tento učiaci sa klasifikačný systém v úlohe sledovania stien (Zaťko (2012) a Tóth (2012)).

Príprava experimentu bola omnoho náročnejšia, pretože na rozdiel od simulačného prostredia bolo potrebne zaoberať sa množstvom hardvérových a softvérových problémov. Keďže sa robot pohyboval v reálnom prostredí, nedokonalosti trenia na podložke spôsobovali veľké výchylky už pri malých rýchlostiach. Na druhej strane sme si nemohli dovoliť kolízny priebeh, kde pri prvom dotyku robota so stenou bol experiment automaticky prerušený. LCS algoritmus bol implementovaný do počítača, ktorého výpočtový výkon je výrazné vyšší než výkon notebooku používaného pri simuláciách a preto sme si mohli dovoliť nastaviť menšie hodnoty časovačov. Napriek tejto skutočnosti bola rýchlosť robota obmedzená len na jednu desatinu maximálnej rýchlosti, čo bolo $v = 0,07\ m.s^{-1}$. Robot zadanú úlohu zvládol výborne, čo dokumentujú videa nachádzajúce sa v prílohe na CD médiu, ale aj na internete v nasledujúcich odkazoch (Zaťko, 2012) a (Tóth, 2012):

- `http://youtu.be/rYwCiSdkUaQ` (pohyb vo väčších priestoroch)
- `http://youtu.be/DRwLNKD4zQo` (pohyb v menšom priestore)
- `http://youtu.be/a86OUAvzkz0` (aktívny LCS pri deaktivovaných pohonoch)

Záver

Ciele diplomovej práce sa nám podarilo splniť. Úspešne sme implementovali učiaci mechanizmus do mobilného robota, postavený na báze kombinácií genetického algoritmu a učenia s posilňovaním, čo je prístup nazývaný aj ako učiaci sa klasifikačný systém (*Learning Classifier System* LCS). LCS na základe signálov zo senzorov umožnil cielený pohyb robota v neznámom prostredí. Výhodou použitia LCS je schopnosť autonómne si vytvárať správanie, a to v reálnom čase.

V úvode sme rozoberali problematiku základov robotickej navigácie a lokalizácie a tiež sme načrtli niektoré problémy s nimi súvisiace. Rozhodli sme sa použiť algoritmy, ktoré sú inšpirované prírodou a tiež veľmi dôležitou schopnosťou živých organizmov učiť sa, čo je z hľadiska pohľadu kognitívnej vedy obzvlášť dôležité. V teoretickom úvode sme čitateľa oboznámili so súvisiacou problematiku, kde sa autori zaoberali navigáciou mobilného robota (len v simulačnom prostredí) pomocou evolučných algoritmov a tiež aj pomocou učenia s posilňovaním, pričom pre nastavovanie parametrov použili genetický algoritmus. Druhá kapitola sa hlbšie zaoberala dvoma kľúčovými metódami LCS, ktorými sú geneticky algoritmus a učenie s posilňovaním. Okrem algoritmického a matematického pohľadu sme uviedli aj ich biologické základy (biologickú motiváciu) s konkrétnymi príkladmi. Ďalej už nasledoval samotný LCS, ktorý operuje na populácií zloženej z množiny pravidiel, kde sú tieto pravidlá používané pre klasifikovanie situácie. Tretia kapitola zaoberajúca sa prostrediami poskytuje pohľad na problematiku experimentov z vyššej perspektívy, pretože nás nútila zamyslieť sa nad možnosťami implementácie LCS. Kapitola sa venuje virtuálnemu a reálnemu prostrediu, pričom reálne prostredie je doplnené o opis nášho mobilného robota, s ktorým sme tiež pracovali. V závere tejto časti sme zhodnotili, že je potrebné vytvoriť vlastné virtuálne prostredie, kde sa bude pohybovať virtuálny mobilný robot. Model virtuálneho robota bol navrhnutý na základe fyzicky existujúceho prototypu. Po tejto teoretickej časti sme nadviazali na získané znalosti a vytvorili sme simulátor mobilného robota s automatickým režimom riadenia, ktorým je LCS. Táto štvrtá kapitola v určitej miere dokumentuje funkcionalitu simulátora a obsahuje informácie, ktoré napomôžu zoznámiť sa možnosťami simulátora. Tu sme zahrnuli aj ukážky zdrojových kódov, ktoré demonštrujú pohybové a percepčné schopnosti robota. Posledná kapitola prezentuje dosiahnuté výsledky,

ktoré zobrazujú vývoj polohy mobilného robota v 2D súradniciach. V tejto časti nadväzujeme na semestrálny projekt, ktorý bol riešený v rámci študijného pobytu *Erasmus* na Viedenskej univerzite, kde sa nám podarilo v obmedzenom rozsahu otestovať tento učiaci sa klasifikačný systém aj v reálnom prostredí, teda s reálnym mobilným robotom, ktorý bol inšpiráciou pre vytvorenie simulačného prostredia. Na základe uvedeného bol LCS úspešne implementovaný do simulačného aj reálneho prostredia. Vo výsledku bol mobilný robot riadený týmto systémom schopný sa v neznámom prostredí učiť princípy vyhýbania sa prekážkam a sledovania stien.

Literatúra

Arkin, R. C. (1989). Motor schema-based mobile robot navigation. In *IEEE - The International journal of robotics research.*

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., Yoshida, C. (2009) Cognitive developmental robotics: A survey. In *IEEE Transactions, Autonomous Mental Development.*

Bäck, T., Fogel, D. B. a Michalewicz, Z. (2000). Evolutionary computation 1: Basic algorithms and operators. Institute of physics publishing Bristol and Philidelphia (p. 339).

Booker, L. B., Goldberg, D. E., a Holland, J. H.(1988). Classifier systems and genetic algorithms. Artificial intelligence, 40(1-3), 235-282.

Brownlee, J. (2011). Clever algorithms: Nature-inspired programming recipes. (1st. Edition) Lulu Enterprises.

Bull, L. a Kovacs, T. (2005). Foundations of learning classifier systems : An introduction. (L. Bull a T. Kovacs, Eds.) Learning, 17(9), 1-17.

Butz, M. V. a Wilson, S. W. (2002). An algorithmic description of XCS. Soft computing a fusion of foundations methodologies and applications, 6(3-4), 144-153.

Cielecki,L. a Unold, O. (2007). Real-valued GCS vlassifier system. (Int. J. Appl. Math. Comput. Sci.), 2007, Vol. 17, No. 4, 539–547.

Darwin, C. (1878) On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. London (pp. 204-208).

Dawkins, R. (1976). The selfish gene. Book (Vol. 8, p. 368). v *Oxford University Press.*

Duchoň, F.(2012). *Lokalizácia a navigácia mobilných robotov do vnútorného prostredia.* Slovenská technická univerzita v Bratislave.

Eiben, A. E. a Smith, J. E. (2003). Introduction to evolutionary computing. (A. E. Eiben a J. E. Smith, Eds.) Evolutionary Computation (Vol. 12, p. 299). 18. Marca 2013. Dostupné [online] z URL `http://www.cs.vu.nl/~gusz/ecbook/ecbook-illustrations.html`.

Engelbrecht, A. P. (2007). Computational intelligence: An introduction. IEEE Transactions on Neural Networks. Wiley

GNU-operating-system(2013). GNU lesser general public license, version. 2. Apríla 2013 Dostupné [online] z URL `http://www.gnu.org/licenses/lgpl-2.1.html`

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison Wesley (p. 432).

Goldberg, D. E. a Holland, J. H. (1988). Machine learning, 3(2), 95-99.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. (R. S. Michalski, J. G. Carbonell, a T. M. Mitchell Eds.), Machine learning an artificial intelligence approach. Vol. 2, pp. 593-623

Holland, J. H. (1975). Adaptation in natural and artificial systems. (J. H. Holland, Ed.) Ann arbor MI University of Michigan press (Vol. Ann Arbor, pp. 1-200). University of Michigan press.

Holmes, J., Lanzi, P. L., Stolzmann, W. a Wilson, S. (2002). Learning classifier systems: New models, successful applications. Information Processing Letters, 82(1), 23-30

Hecht, F. (1979) In memory yet green. The autobiography of Isaac Asimo. 1924-1953. In *The American Journal of Human Genetics,* 31(4), 522.

Jabin S. (2010). Robot learning using learning classifier systems approach.

Kala, R., Shukla, A., Tiwari, R., Rungta, S. a Janghel, R. R.(2009). Mobile robot navigation control in moving obstacle environment using genetic algorithm, artificial neural networks and A* algorithm. In *WRI World Congress on Computer Science and Information Engineering*, 705-713.

Kamei, K., Ishikawa, M. (2004). Determination of the optimal values of parameters in reinforcement learning for mobile robot navigation by a genetic algorithm. In *International Congress Series*, 1269 (2004), 193– 196

Lancater University (2013). Optimisation of collector from and response. 12. Marec 2013. Dostupné [online] z URL `http://www.engineering.lancs.ac.uk/lureg/group_research/wave_energy_research/Collector_Shape_Design.php`.

Lanzi, P. L. (2002). Learning classifier systems from a reinforcement learning perspective. Soft computing a fusion of foundations methodologies and applications. 6(3-4), 162-170.

Leccos-Autonomie (2013). 21. Apríl 2013. Dostupné [online] z URL `http://leccos.com/index.php/clanky/autonomie`.

Logic Design Inc. (2012). Softvare: introduction to robotics simulation. 20. Apríl 2012. Dostupné [online] z URL `http://www.robologix.com/robotics_simulation_intro.php`

MobotSoft (2012). Professional sofware for mobile robots. MobotSim – mobile robot simulator. 21. Apríl 2012. Dostupné [online] z URL `http://www.mobotsoft.com/`

Montague, P. R. (1999) Reinforcement learning: An introduction, by Sutton, R.S. and Barto, A.G. (Trends in Cognitive Sciences) 3(9), 360.

Niederhoffer V. (2006). The Web Site of Victor Niederhoffer and Lurel Kenner. 19. Apríl 2012. Dostupné [online] z URL `http://www.dailyspeculations.com/vic/Operant.htm`

Ostatníkova, D. (2011) Anatomické koreláty a molekulová báza pamäti. (Prednáška, Lekárska fakulta - Univerzita Komenského).

Pavlov's Dogs(2012). Pavlov's dogs (classical conditioning) 19. Apríl 2012. Dostupné [online] z URL `http://psychonthis.com/experiments/pavlovs-dogs-classical-conditioning/`.

Puterman, M. L. a Shin, M. C. (1978). Modified policy iteration algorithms for discounted markov decision problems. Management Science, 24(11), 1127-1137.

Qt Homepage(2012). 2. Apríla 2013 Dostupné [online] z URL `http://qt.digia.com/`

Ragavan, S. V., Ganapathy, V. (2007). A unified framework for a robust conflict - free robot navigation. In *World Academy of Science Engineering and Technology*, 21(January), 88-94.

Rajakaruna, C. (2003) Obstacle avoidance in a mobile robot using the learning classifier. School of Information Technology and Elecrical Engineering, The University of Queensland, 2003. 56 s.

Rebrová K. a Farkaš I. (2011). Neurálne modely v kognitívnej robotike: porozumenie a pomenovávanie akcií. In *Kelemen J., Kvasnička V. (eds.),* Kognice a umělý život XI, Slezská univerzita, Opava. 231-238. 2011.

Salichs, M.A. a Moreno, L. (2000). Navigation of mobile robots: open question. In *Robotica 2000.*

Sharkey, A. J. C. (2005). Swarm Robotics. (E. Şahin a W. M. Spears, Eds.) Connection Science, 3342(3), 245-260. In *Springer Berlin Heidelberg.*

Sigaud, O. a Wilson, S. W. (2007). Learning classifier systems: a survey. Soft Computing. 11(11), 1065-1078.

Sutton, R. S. a Barto, A. G. (1998). Reinforcement Learning. Cambridge, MA: MIT Press, 1998. In *Journal of Cognitive Neuroscience*, 11(1), 126-134.

Tóth, F. (2011). Komplexný návrh a realizácia mobilného robotického systému. Diplomová práca, Bratislava: STU FEI, 2011. 103 s.

Tóth, F. (2012) *Final project report* University of Vienna and Vienna University of Technology, (Erasmus - winter semester 2011/2012)

Tsagarakis, N., Sinclair, M., Becchi, F., Metta, G., Sandini, G. a Caldwell, D. (2006) Lower body design of the "iCub" a human-baby like crawling robot. in *2006 6th IEEERAS International Conference on Humanoid Robots* 450-455. Institute of Electrical and Electronics Engineers (IEEE).

Ústav automatizace a měřicí techniky, Fakulty elektrotechniky a informatiky, Vysoké učení technické v Brně (2012). Simulátor autonomních mobilních robotu pro Matlab 5. 23. Apríl 2012. Dostupné [online] z URL `http://www.uamt.feec.vutbr.cz/robotics/simulations/amrt/simrobot_cz.html#popis`

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. Evolutionary Computation. 3(2), 149-175.

Zaťko, G. (2012) Final project report. University of Vienna and Vienna University of Technology, (Erasmus - winter semester 2011/2012)

Železný, I. (1988) Roboti a androidi. Praha : Svoboda, 1988. Kapitola O autorech, (Isaac Asimov), s. 605

Dodatok A

Príloha – CD médium

Na priloženom CD je uložený text tejto diplomovej práce v elektronickej podobe. Okrem toho sú na ňom uložené zdrojové kódy použité pri implementácii a vyhodnocovaní neurálneho modelu.