VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

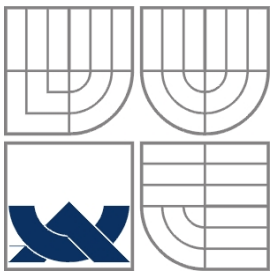
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

Grafické prostředí pro simulaci robotů

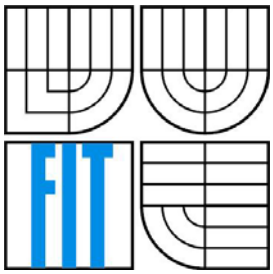
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Michal Malina
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

Grafické prostředí pro simulaci robotů

Graphical Environment for Robot Simulation

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Michal Malina

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. František Zbořil, Ph.D.

BRNO 2008

Abstrakt

Tato práce se zabývá modelováním simulace chování robotů. Nejprve je vysvětlen přínos takovýchto simulací. Dále pak je popsáno obecné schéma robota. Poté výběr konkrétního prostředí, ve kterém bude robot simulován. Je uveden návrh tohoto robota a vysvětlena implementace v konkrétním prostředí. Pak již následuje návrh a realizace rozhranní, přes které je možné s robotem komunikovat. Výsledkem je autonomně se pohybující robot schopný reagovat na překážky a vyhýbat se jim. Je schopen také vytvářet mapu okolního prostředí. S využitím této mapy dokáže najít optimální cestu k zadanému cíli s využitím algoritmu A^* a tohoto cíle dosáhnout. Je možné i využití více robotů ve scéně. Ti pak spolu komunikují a nejbližší z nich se dostane k cíli.

Klíčová slova

Robotika, simulace robota, senzor, Microsoft Robotics Studio, Blender Creator, Dashboard, tvorba mapy, A^*, navigace.

Abstract

This thesis deals with modelling of the robot behavior simulation. At first we discuss a benefit of this kind of simulation. Next we describe the general scheme of robot. After that we select the environment where robot will be simulated in. There is a conception of this robot and explanation of the implementation in chosen environment. We continue with design and realization of the communication interface of this robot. The result of this procedure is robot with independent movement able to response to the obstructions and avoid them. Robot is able to create a map of the environment and use this map to find out the optimal way to the scheduled goal using the A* algorithm and achieve this goal. There is a possibility to use more robots in the scene. These communicate and the nearest one is able to get the goal.

Keywords

Robotics, robot simulation, sensor, Microsoft Robotics Studio, Blender Creator, Dashboard, the map creation, A^*, navigation.

Citace

Malina Michal: Grafické prostředí pro simulaci robotů. Brno, 2008, diplomová práce, FIT VUT v Brně.

Grafické prostředí pro simulaci robotů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Františka Zbořila, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Malina
15.5.2008

Poděkování

Děkuji Ing. Františku Zbořilovi, Ph.D za podněty, rady a pomoc při studiu této problematiky.

Obsah

1 Úvod

Robotika je odpovědí na touhu starou jako lidstvo samo a to pohodlný životní styl. Ovšem nejen pohodlí bylo hnacím motorem vývoje. Také nedostupná prostředí a postradatelnost robota. V dnešní době jsou roboti, ať už velmi jednodušší, téměř na každém kroku. Jejich vývoj kráčí neodvratně kupředu. Jejich senzory a pohybové schopnosti se neustále zlepšují. Proč tedy vědci již dávno nesestrojili robota inteligentního jako člověk? Tedy podle definice s takovou umělou inteligencí, že při komunikaci s ním by člověk nepoznal, že se jedná o umělou inteligenci. Opověď je snadná. Je to způsobeno dokonalostí lidského těla. Ještě dnes není náš mozek do detailů pochopen. Vždyť příroda měla na jeho vývoj miliony let, což jo neporovnatelně delší časový úsek, než ten, po který se my, lidé, umělou inteligencí zabýváme.

Největší problém robotiky je tedy s umělou inteligencí u autonomních robotů. Tedy robotů schopných reagovat samostatně a logicky na vzniklé situace bez zásahu obsluhy. V poslední době jsou výhodou vyšší programovací jazyky, které se pro programování robotů mohou použít. Důležitou vlastností je také schopnost učit se. Poučit se z vlastních chyb. Nejen reagovat na situace, ale ukládat si důležitá data a těch poté vhodným způsobem využít.

Velkou budoucnost také vidím ve využití velkého počtu jednodušších robotů. Jedná se o obdobu včelího nebo mravenčího společenství. Výzkumy ukázaly, že je možné dosáhnout vysoce „inteligentních“ výsledků při použití jednoduchých robotů.

Návrh mobilních robotů patří v současné době mezi témata výzkumu na mnoha univerzitách i na komerčních pracovištích. Jedná se o velmi složitou mezioborovou disciplínu. Je třeba navrhnout mechanické částí, poté elektrotechnická zapojení a v neposlední řadě také dát robotu nějakou „inteligenci“.

I když je návrh velmi kvalitní, vždy se řada nedostatků projeví až po praktických testech, kdy je robot sestrojen. Tomu lze zamezit, nebo alespoň snížit počet nedostatků tím, že se navržený robot sestrojí „virtuálně“. S využitím sofistikovaného softwaru lze virtuálně sestrojit navrženého robota i s odpovídajícími fyzikálními parametry. Za těchto podmínek je pak snadné měnit různé části robota, popřípadě porovnávat různé návrhy. Lze také měnit prostředí, do kterých je robot zasazen.

Příkladem může být simulační metoda, kterou vyvinuli vědci v Ústavu pro výrobní techniku a obráběcí stroje IFW (Institut für Fertigungstechnik und Werkzeugmaschinen) univerzity v Hannoveru [4]. V rámci projektu s názvem Iroprog (Innovative Roboter-Programmierung) vymysleli odborníci z IFW inovativní postup, jak programovat průmyslové roboty. Uživatel má k dispozici software, který mu zobrazuje věrný 3D model takovéhoto robota. Pomocí volně pohyblivého zadávacího kolíčku může uživatel pohybovat ramenem robota, a to po takových drahách, po který se bude rameno pohybovat v reálných podmínkách. Tyto informace si software uloží a navíc takto získaná data vyhladí (korekce třesu lidské ruky). Velkou výhodou u této techniky programování je, že ovládací

kolíček dokáže klást odpor, pokud ovládané rameno narazí na nějakou překážku. U dříve využívaných postupů došlo k detekci srážky ramene s překážkou až po stanovení všech drah. Navíc je možné kolizím předem zabránit, a to s využitím tzv. „pomocné ruky“. Při vedení kolíčku jsou totiž předem definována toleranční pásma a při jejich opuštění začne kolíček klást odpor.

Další výhodou simulace robotů je oproštění se od hardwaru. V dnešní době se cena kvalitních robotů pohybuje velmi vysoko. Proto je pro řadu účelů vhodnější si robota nasimulovat. Jen tak se může o robotiku začít zajímat každý, i ten, který nemá přístup k drahému hardwaru.

Ve druhé kapitole je vysvětlen historický vývoj robotů a definován robot jako takový a jeho typy. Ve třetí kapitole je vysvětleno obecné schéma robota. Dále jsou zde vysvětleny senzory, kterých může robot využívat, typy podvozků a paradigmata robotiky. Ve čtvrté kapitole je vysvětleno, proč byl zvolen právě software od Microsoftu. Dále pak je uvedeno obecné pojednání a způsob implementace robota v prostředí Microsft Robotics Studia. Pátá kapitola popisuje volbu programovacího jazyka a rozhranní, přes které je robot řízen. Dále je zde vysvětlen princip vandrujícího robota. Šestá kapitola osvětluje, co je mapa okolí robota a ukazuje, jak je řešeno mapování v této práci. Sedmá kapitola popisuje algoritmus A^* a následně je zde ukázáno, jak byl tento algoritmus implementován v této práci. V osmé kapitole je vysvětlen pojem navigace a následně je zde ukázáno, jakým způsobem byl robot navigován k cíli v této práci. Devátá kapitola popisuje strukturu souborů této práce a následně ovládání aplikace. V desáté kapitole je zhodnocení dosažených výsledků, vlastní přínos a možnost návaznosti na jiné práce.

2 Robot

2.1 Historie

Snaha o automatické vykonávání práce vedla ke konstrukci automatických zařízení, která se však nepodobala člověku. Nicméně snaha vyrobit umělého člověka provází a zřejmě bude provázet člověka ještě dlouhou dobu [5].

Vzhledem k vývoji techniky měly první napodobeniny člověka, případně zvířete, podobu mechanickou. Známé jsou mechanické napodobeniny člověka (androidy) švýcarských mistrů Piera a Henry Drozů (18. stol.). Ti vytvořili automatického písaře, který byl schopen psát perem několik vět a velmi dobře napodoboval člověka.

Ještě staršího data jsou mechanické napodobeniny zvířat (zooidy). Po věku mechaniky přispěla k vývoji robotů elektrotechnika. Zásadním mezníkem je v robotice rok 1920. Roboti té doby byli stále hříčky používané většinou na výstavách k přilákání pozornosti návštěvníků. Ale 20. století je století velmi racionální a začínají se objevovat první praktické aplikace, které spadají do oblasti robotiky. Jde o teleoperátory pro manipulaci s radioaktivními a jinými nebezpečnými materiály (1940-7). Pak již následuje velmi rychlý vývoj.

- 1949 je zahájen výzkum obráběcích strojů, které jsou řízeny numericky
- 1961 je dán do provozu první průmyslový robot UNIMATE u firmy General Motors. Vývoj tohoto robota je spojen se jmény G. Devol , J. Engelberger a universitou Columbia University U.S.A.
- 1964 jsou otevřeny laboratoře umělé inteligence (UI) na Massachutess Institute of Technology (M.I.T.), Stanford Research Institute (S.R.I.) a dalších institucích v U.S.A. Mají se zabývat m.j. využitím UI v robotice.
- 1968 je postaven na S.R.I. mobilní robot Shakey vybavený viděním
- 1977 dává do prodeje své velmi zdařilé roboty evropská firma ASEA
- v r. 1979 jsou uvedeny na trh roboti koncepce Selective Compliant Articulated Robot Arm (SCARA). Průmysloví roboti se stávají běžným prostředkem automatizace manipulačních operací především v automobilovém průmyslu. Průmysloví roboti jsou masivně používáni pro svařování plamenem, elektrickým obloukem, bodové svařování, jsou používáni pro nanášení barev a všude tam, kde jsou manipulační operace pro člověka nebezpečné a zdraví škodlivé. Počáteční předstih U.S.A. ve výzkumu, ale hlavně ve využití robotů, přebírá Japonsko. Ročenka OSN uvádí v roce 2001 následující počty nasazených průmyslových robotů: 389 000 v Japonsku, 198 000 v Evropské unii a 90 000 v U.S.A.
- po roce 1980 začínají být první průmysloví roboti vybavováni počítačovým viděním, čidly hmatu a dalšími prvky, které zatím spadaly do oblasti výzkumu UI

- v r. 1995 se objevuje první chirurgický robotický systém pro tzv. minimálně invazivní chirurgii
- v r. 1997 je na Marsu vysazen robot Sojourner
- v r. 2000 předvádí firma Honda svého humanoidního robota, ASIMO a SONY předvádí své zooidy AIBO

Je zřejmé, že konečným cílem robotiky by mohlo být postavení stroje, který by téměř nahradil člověka. I když zní tento cíl prozatím nereálně, může mít cesta k tomuto cíli celou řadu podružných a přesto významných výsledků. Za výsledek robotického výzkumu můžeme považovat např. pohybové pomůcky, které mají sloužit zdravotně postiženým lidem.

Jako výsledek robotického výzkumu můžeme označit např. exoskeletony. Jsou to zařízení, která si člověk na sebe obléká a která mnohonásobně zvýší jeho fyzické schopnosti, především pak sílu. Perspektivní použití exoskeletonů je rovněž ve zdravotnictví, a to při manipulaci s nepohyblivými pacienty. K robotickému výzkumu patří také výzkum dálkového řízení robotů na principu teleprezence. Při takovém řízení získává operátor všechny informace o prostředí, ve kterém se robot pohybuje, ve formě vhodné pro smysly člověka (zrak, hmat, sluch, čich, chuť), takže má dojem, že je skutečně v prostředí, které obklopuje robota. Ovládání robota má být stejně dokonalé, operátor prostě provádí totéž co by dělal, kdyby opravdu v prostředí byl. Robot řízený tímto způsobem by mohl významně pomáhat hasičům a záchranářům.

2.2 Definice robota

Robotika [5] je obor, který se zabývá studiem a konstrukcí robotů a jim podobných zařízení. Dosud však neexistuje ustálená definice jak oboru, tak pojmu robot. Všeobecně je robot chápán jako stroj, který vykonává podobné činnosti jako člověk. Především však činnosti pohybové a manipulační. Většinou musí takový stroj získávat informace o prostředí, ve kterém se pohybuje a musí být schopen toto prostředí fyzikálně, především mechanicky, ovlivňovat.

Robot [1] je tedy samostatně pracující stroj vykonávající určené úkoly. Slovo robota bylo známo již v 17. století ve významu otrocká práce podaných. Mírně pozměněné jej poprvé ve významu stroj použil český spisovatel Karel Čapek v divadelní hře R.U.R. Slovo mu poradil jeho bratr Josef Čapek, když se ho Karel ptal, jak umělou bytost pojmenovat. Původně zamýšlený *labor* zněl autorovi příliš papírově. Pro některé druhy robotů se používá přesnější označení:

- manipulátor – stroj, který nemá vlastní inteligenci. Je řízen dálkově.
- droid – jakýkoliv inteligentní a samočinný robot
- android – robot podobný člověku. Obvykle se očekává biologické složení. Roboti v R.U.R. byli podle tohoto dělení androidi
- humanoid – robot podobný člověku principiální stavbou těla a zejména způsobem pohybu

- kyborg (*kybernetický organismus*) – živá bytost obohacená o mechanické či elektronické součástky. V extrémním případě může z původní bytosti zůstat pouze mozek

Termín robot se používá také pro počítačové programy, který za svého majitele provádí opakované činnosti.

Roboti se dají dále obecně dělit na stacionární a mobilní. První z nich znamená, že se nemůže pohybovat z místa na místo. Naopak mobilní robot je schopen se pohybovat, a to buď samostatně (autonomní robot), nebo je řízen pokyny operátora.

Robot by se měl také řídit zákony robotiky, což jsou pravidla chování robotů, definované Isaacem Asimovem v jeho povídkách a později románech. Principy, které tyto zákony představují, jsou považovány za obecná shrnutí základních požadavků na vývoj a používání robotů. V původní podobě existovaly následující tři zákony:

- robot nesmí ublížit člověku nebo svou nečinností dopustit, aby mu bylo ublíženo
- robot musí poslechnout člověka, kromě případů, kdy je to v rozporu s prvním zákonem
- robot se musí chránit před poškozením, kromě případů, kdy je to v rozporu s prvním nebo druhým zákonem

3 Obecné schéma robota

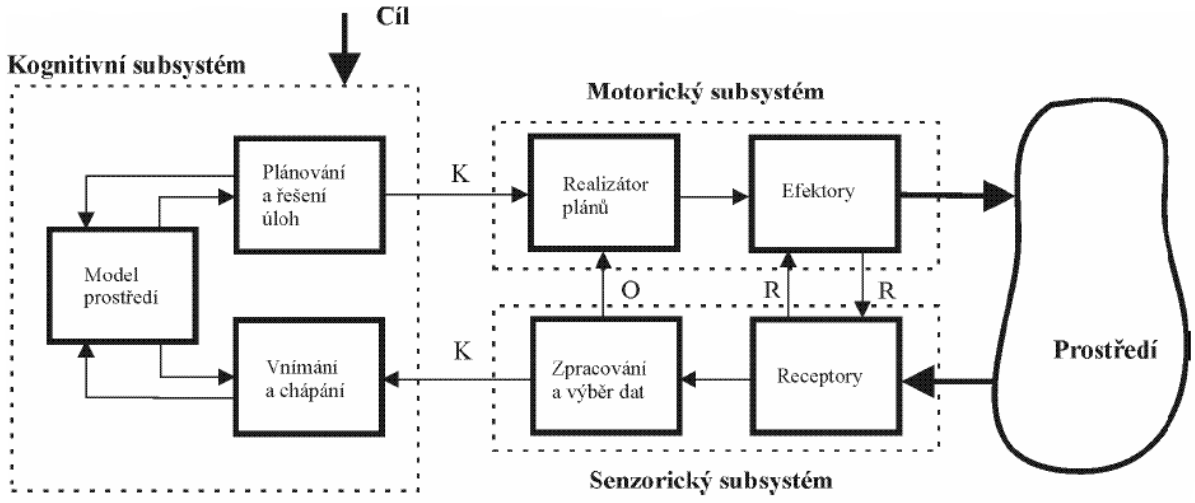
K nahrazení člověka robotem je nutné, aby byl takovýto robot schopen fyzicky ovlivňovat své okolí a pohybovat se v něm [5]. K tomu slouží motorický subsystém, který svými efektory ovlivňuje prostředí. K pohybu robota v prostředí slouží rovněž efektory. Dále pak musí být robot schopen reagovat na prostředí, k čemuž slouží senzorický subsystém. Nad výše uvedenými systémy pracuje kognitivní subsystém. Zde probíhá hlavní rozhodovací a řídící činnost. Je zde ukryta inteligence robota.

Senzorický subsystém se skládá ze dvou základních částí, a to z receptorů a ze systému zpracování a výběru dat. Receptory snímají fyzikální veličiny z prostředí a konvertují je na vhodné vnitřní signály. Z takto získaných signálů pak vybírá systém zpracování a výběru dat informace důležité pro robota.

Motorický subsystém se skládá z efektorů a realizátoru plánů. Jak již bylo v úvodu nastíněno, slouží efektory k zásahu do prostředí. O jejich řízení se stará realizátor plánů. Je propojen se senzorickým subsystémem pomocí operační smyčky (O) a reflexivní smyčky (R). Smyčka O zajišťuje vykonání naplánované úlohy. Smyčka R řeší základní problémy jako reflexy u člověka.

Kognitivní subsystém hlouběji analyzuje informace získané ze senzorického subsystému. Tato analýza zahrnuje vnímání a chápání. Je však nezbytné, aby měl robot vytvořen model prostředí a stanoven požadavek, čeho má dosáhnout. V tomto subsystému se provádí řešení úloh a plán akcí na základě již zmíněné analýzy, modelu prostředí a stanoveném cíli. Kognitivní subsystém je spojen s motorickým a senzorickým subsystémem pomocí vazby K.

Výše popsané schéma je znázorněno na Obrázku 1.



Obrázek 1: Obecné schéma robota [5]

3.1 Senzory robota

Senzor [5,7] je zařízení, které snímá sledovanou fyzikální, biologickou nebo chemickou veličinu a dle určitého definovaného principu ji transformuje fyzikálním převodem na veličinu výstupní. Ta bývá často elektricky kvantitativní. Senzor se skládá ze dvou základních částí, a to čidla, které přímo snímá sledovanou veličinu, a vyhodnocovací logiky.

Vstupní část převádí sledovanou veličinu na veličinu elektrickou. Dále je možné v této části zesilovat a filtrovat elektrické veličiny, linearizovat statické veličiny, chránit proti nežádoucím vlivům, normovat signál atd.

Pomocí výstupní části senzor komunikuje s okolím pomocí sběrnice prostřednictvím integrovaného rozhranní.

Vnitřní část kalibruje elektrické i neelektrické veličiny. Dále pak může provádět číslicovou linearizaci, autodiagnostiku, statistické vyhodnocení získaných dat, hlídání mezí atd.

Senzory se z obecného hlediska dělí na tři generace. Do první generace spadají senzory, které využívají makroskopické, elektromechanické, elektrochemické či mechanické principy. Do této skupiny patří senzory odporové, kapacitní atd. V dnešní době již dosáhly svého maxima.

Druhá generace senzorů využívá elektronických jevů v tuhých látkách (např. piezoelektrický, fotoelektrický jev atd.) a v plynech (např. nárazová ionizace). Nejvýznamnější jsou mikroelektronické polovodičové senzory. Ty jsou velmi snadno slučitelné se vstupními a výstupními obvody. Tyto senzory se nazývají inteligentní.

Třetí generace využívá působení neelektrické veličiny na svazek světelného záření. Tyto senzory jsou označovány jako Optické vláknové senzory (OVS). Bývají konstruovány s využitím principů optoelektroniky a integrované optiky.

S rostoucí kvalitou senzorů je robot schopen přesněji vnímat své okolí a plnit tak těžší cíle. Mezi základní atributy senzorů patří:

- přesnost a rozlišení – přesnost udává, jak moc správná naměřená data jsou. Po jak velkých krocích je senzor schopen měřit udává rozlišení.
- záběr a dosah – záběr senzoru určuje, jak velkou plochu najednou je schopen obsáhnout. Dosah pak určuje maximální vzdálenost, ve které jsou data ještě spolehlivá.
- spotřeba
- hardwarová spolehlivost
- velikost
- výpočetní složitost – udává, kolik kroků provede algoritmus
- spolehlivost interpretace – je třeba brát v potaz, jak je senzor spolehlivý.

3.1.1 Dotykové senzory

Dotykové senzory fungují jakou dvoustavové spínače, které se při nárazu sepnou. Po vhodném rozmístění těchto senzorů po celém obvodu robota lze při nárazu zjistit, která část robota narazila na překážku. Nevýhodou těchto senzorů je fakt, že reagují až na samotný střet s překážkou, což by například u některých reálných aplikací bylo nemyslitelné.

3.1.2 Senzory pro odometrii

Odometrie je proces, který popisuje transformaci dat poskytnutých enkodéry na změnu pozice a orientace robota. Vlastní slovo *odometrie* je složeno ze dvou řeckých slov *hodos* (cestovat, cesta) a *metron* (měřit). Enkodér pracuje tak, že sleduje přítomnost nebo nepřítomnost dobře detekovatelného materiálu na otáčejícím se kolečku. Použít se dají enkodéry mechanické, optické a magnetické. Tímto způsobem je možné zjistit velikost pootočení hřídele, popřípadě lze stanovit přesnou polohu hřídele pomocí speciálních kódů na kolečku.

Dalším možným přístupem měření ujeté vzdálenosti je měření Doplerova jevu. Princip je založen na pozorování změny fáze signálu. Senzor vysílá signál, který se zpět odrazí s určitým fazovým posunem.

3.1.3 Senzory pro měření vzdálenosti

K měření vzdálenosti se prakticky využívají tři principy:

- měření doby letu (Time Of Flight - TOF) vyslaného pulzu energie směrem k objektu. Od něj se odrazí zpět a je zachycen přijímačem.
- měření fázového posuvu, kde je vyslán spojitý signál
- technika podobná měření fázového posuvu, a to technika založená na frekvenčně modulovaném radarovém signálu

Nejčastěji jsou využívány TOF systémy, a to pro svoji nízkou cenu. Jako pulz energie jsou nejčastěji vysílány zvukové vlny, rádiové vlny nebo optické vlny. Pro výpočet vzdálenosti je třeba znát pouze dobu letu pulzu a rychlost šíření pulzu v prostředí.

3.1.4 Senzory snímající akceleraci (Akcelerometry)

Akcelerometr je senzor, který využívá setrvačnosti hmoty pro měření rozdílu mezi kinematickým zrychlením (vzhledem k určitému inerciálnímu prostředí) a gravitačním zrychlením.

Jedná se o jednoduchý mechanismus, který se skládá z pevné základny a pohyblivého tělesa, které je k základně připevněno pomocí pružiny. V klidovém stavu není pružina natažena a nepůsobí tedy žádnou silou. Pokud dojde k posunutí zařízení s určitým zrychlením, pak se pružina natáhne,

popř. smrští. Je možné měřit akceleraci prostřednictvím velikosti natažené pružiny. Zrychlení je přímo úměrné konstantě pružnosti pružiny a velikosti výchylky. Nepřímo pak hmotnosti tělesa na pružině.

3.1.5 Senzory snímající obraz (kamery)

Jedná se o velmi významný senzor. V současnosti jsou především založené na CCD snímačích. CCD je v podstatě posuvný registr, který je vystaven světlu. CCD využívá podobně jako všechny ostatní světlocitlivé součástky fyzikálního jevu známého jako fotoefekt. Tento jev spočívá v tom, že částice světla, foton, při nárazu do atomu dokáže přemístit některý z jeho elektronů ze základního do tzv. excitovaného stavu.

V polovodiči se takto uvolněný elektron může podílet na elektrické vodivosti, respektive je možno ho z polovodiče odvést pomocí přiložených elektrod tak, jak se to děje například u běžné fotodiody. Ta proto po dopadu světla vyrábí elektrický proud. Stejně fungují i fotočlánky, které se používají jako zdroj elektrické energie.

U CCD je ovšem elektroda od polovodiče izolována tenoučkou vrstvičkou oxidu křemičitého SiO_2, který se chová jako dokonalý izolant, takže fotoefektem uvolněné elektrony nemohou být odvedeny pryč.

Nejprve jsou z CCD bez přístupu světla odebrány všechny volné elektrony, čímž je z něj smazán jakýkoliv zbytek předchozího nasnímaného obrazu.

Poté je již možné exponovat nový obraz. Otevře se tedy např. uzávěrka a na CCD se nechá působit světlo. Dopadající fotony excitují v polovodiči elektrony, které jsou pak přitahovány ke kladně nabitým elektrodám (elektroda odpovídá pixelu). Shluky elektronů z jednotlivých pixelů se pak posouvají přes sousední pixely směrem k výstupnímu zesilovači. Tento zesilovač pak zesílí malý proud odpovídající počtu elektronů v jednotlivých pixelech na napěťové úrovně vhodné pro další zpracování obrazu.

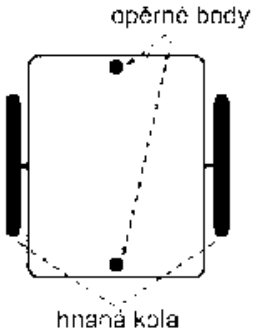
Pokud je robot vybaven tímto senzorem, lze jej navigovat pomocí různých metod rozpoznávání obrazu. Více informací viz [6].

3.2 Typy podvozků robota

Existuje několik základních typů podvozků. V této kapitole budou popsány pouze kolové podvozky. Ostatní typy pohonů nejsou příliš rozšířené a jsou mnohem náročnější z hlediska řízení.

3.2.1 Diferenciální podvozek

Mobilní robot využívající diferenciálně řízeného podvozku (viz Obrázek 2) je zřejmě nejjednodušším a nejpoužívanějším typem mobilního robota. Velmi často je využíván u malých a levných strojů. Je to asi nejoblíbenější řešení u amatérů a nadšenců, především díky jednoduché konstrukci a snadnému řízení. Byl využit i u robota implementovaného v této práci.



Obrázek 2: Diferenciální podvozek [5]

Diferenciální podvozek je složen ze dvou hnaných kol a jedním či více opěrnými body, které slouží k zachování celkové stability robota. Ty jsou tvořeny třecími elementy nebo opěrnými koly (můžou být i otočná). Roboti založení na tomto typu podvozku mají stupeň volnosti roven jedné. Nejpoužívanějším typem pohonů jsou malé stejnosměrné motory s enkodéry nebo krokové motory.

Jako největší výhody diferenciálního podvozku lze zmínit jeho jednoduchou a robustní konstrukci, nízkou cena, velmi jednoduchou a přesnou odometrie pro navigaci robota. Tato výhoda je často používána i u poměrně sofistikovaných robotů. Zde je však odometrie doplňována nějakou další metodou bez chyby s integračním charakterem. Další velkou výhodou tohoto podvozku je velmi dobrá manévrovatelnost. Robot s diferenciálním podvozkem je schopen otočit se na místě. Tato vlastnost bývá ještě posílena tím, že se roboti s diferenciálním podvozkem vytvářejí ve válcovitém tvaru, což zmenšuje pravděpodobnost, že by robot uváznul v rozích místnosti a podobně.

Diferenciální robot je řízen na základě rozdílné rychlosti jednotlivých kol. Pokud se obě kola otáčí stejnou rychlostí a stejným směrem, robot jede rovně. Pokud se bude každé kolo otáčet jiným směrem, pak se bude robot otáčet na místě kolem středu (referenčního bodu) nápravy.

Pokud je rozdíl rychlostí kol nenulový a obě nápravy se otáčí stejným směrem, pak se robot pohybuje po kružnici, jejíž poloměr je dán poměrem rychlostí obou kol jako:

$$R = \frac{b}{2} \cdot \frac{(v_l + v_r)}{(v_l - v_r)}, \quad (1)$$

kde R je poloměr zatáčky, b je rozchod kol robota a v_l (resp. v_r) je rychlost levého (resp. pravého) kola. Úhel natočení robota je úhel, který svírá směrový vektor výchozí pozice robota a směrový vektor cílové pozice robota. Jeho velikost lze vyjádřit ujetou dráhou jako:

$$\varphi = \frac{\Delta s_l + \Delta s_r}{2}, \quad (2)$$

kde φ je úhel, který svírá původní směrový vektor (než byl započat pohyb) s cílovým směrovým vektorem (po ukončení pohybu) a ΔS_l (resp. ΔS_r) je dráha ujetá levým (resp. pravým) kolem. Celkovou ujetou dráhu robota lze popsat jako:

$$s = \frac{s_l + s_r}{2}, \quad (3)$$

přičemž s_l (resp. s_r) je dráha ujetá levým (resp. pravým) kolem. Rychlost referenčního bodu (středu nápravy) lze vypočítat analogicky jako:

$$v_{ref} = \frac{v_l + v_r}{2}, \quad (4)$$

kde v_l (resp. v_r) je rychlost levého (resp. pravého) kola.

3.2.2 Synchronní podvozek

Synchronního podvozek má u každého kola dva stupně volnosti. Typicky tento podvozek obsahuje tři kola uspořádaná do tvaru rovnostranného trojúhelníku. Tvar robota bývá obvykle válcový. Všechna kola se otáčí vždy stejným směrem a zároveň i stejnou rychlostí. Také vždy směřují na stejnou stranu.

Pohyby jednotlivých kol lze synchronizovat elektronicky, a to pomocí vhodně navrženého senzoricko-regulačního systému, nebo častěji pomocí vhodného mechanického uspořádání podvozku. Pochopitelnou nevýhodou v případě elektronické synchronizace je potřeba dvou motorů na každé kolo. Pokud je využito mechanické provázání pohybů kol, bude zde složitější mechanická konstrukce, postačí však pouze dva motory pro libovolný počet kol.

3.2.3 Tříkolové uspořádání

U robotů s tímto podvozkem jsou hnaná pouze zadní kola. Přední kolo slouží jako řídící a pomocí něj dochází k ovládání dráhy robota. Výhodou tohoto uspořádání je jednoduché řízení. Tento podvozek je již možné použít v těžším terénu. Naopak nevýhodou je, že se s takto řízeným robotem nedá otáčet na místě. Robot tedy může uvíznout v úzkých a složitých místech.

V závislosti na natočení řídícího kolečka se robot s tříkolovým uspořádáním pohybuje buď po přímce nebo po kružnici. Ke zjištění aktuální pozice je třeba znát dva údaje, a to orientaci řídícího kolečka a jeho ujetou vzdálenost.

3.2.4 Robot s Ackermanovým podvozkem

Roboti založení na Ackermanově podvozku jsou v podstatě obdobou podvozků u automobilů. V tomto případě jsou zadní kola hnaná a přední kola jsou natáčená každé pod jiným úhlem, protože při zatáčení opisují kola kružnice o různých poloměrech. Existuje i řešení, kdy jsou kola natáčena pod stejným úhlem, ovšem nastávají zde větší prokluzy při zatáčení. Zejména při vyšších rychlostech je tento problém dosti významný.

3.2.5 Podvozky se všesměrovými koly

Jako alternativa k běžně používaným kolům vznikla tzv. všesměrová či složená kola, která umožňují pohyb ve dvou osách. Při vhodném použití takovýchto kol je možné zkonstruovat tzv. všesměrové mobilní platformy. Všesměrová kola jsou v zásadě běžná kola, která mají na svém obvodu řadu pasivních válečků.

Kdyby byly válečky zablokovány, chovalo by se takové kolo jako běžné kolo s jedním stupněm volnosti. Kdyby byla naopak zablokovaná hlavní osa kola, kolo by se mohlo pohybovat pouze v kolmém směru. Kombinací těchto mechanizmů a vhodným uspořádáním podvozku vzhledem k danému typu kol, je možné dosáhnout libovolného pohybu podvozku, tj. podvozek se může pohybovat libovolným směrem a libovolně rotovat, případně vykonávat oba tyto pohyby současně.

Každé kolo je poháněno jedním motorem, ale celkově je nutné použít alespoň tři motory. Jde o obecný princip. Chceme-li při pohybu měnit tři prostorové souřadnice, potřebujeme k tomu alespoň tři zdroje pohybu. Informace potřebné k výpočtu změny pozice obsahují enkodéry jednotlivých kol.

3.3 Paradigmata robotiky

Pojmem paradigma [5] je míněna filozofie nebo množina předpokladů a technik, které charakterizují přístup k určité třídě problémů. Nikdy nelze stanovit, které z paradigmat je nejlepší. To znamená, že pro různé problémy se hodí různé přístupy k jejich řešení. Více přístupů k problému může být správných, jeden z nich však může být nejlehčí. Tato kapitola se zabývá různými paradigmaty (přístupy) k organizaci inteligence v inteligentních robotech.

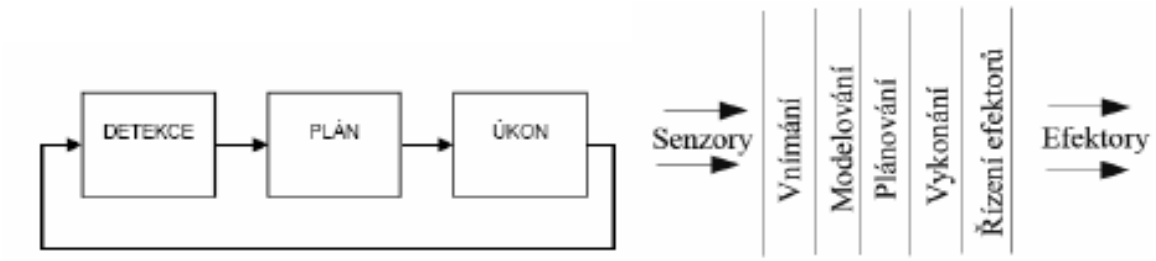
Paradigma v robotice lze definovat dvěma způsoby:

- vztahem mezi třemi základními prvky (detekce pomocí senzorů, plán a úkon (např. pohyb))
- způsobem, jakým jsou data ze senzorů zpracovávána a rozesílána po systému

3.3.1 Hierarchické paradigma

Popis paradigmatu

Hierarchické paradigma (viz Obrázek 3 a 4) je nejstarší metodou, jak organizovat inteligenci v robotech. Objevovalo se nejčastěji od roku 1967 do roku 1980, kdy bylo objeveno reaktivní paradigma.



Obrázek 3: Schéma hierarchického paradigmatu [5]

prvky	vstup	výstup
detekce	Data ze senzorů	Detekované informace
plán	Informace - detekované a/nebo poznané (spočítané)	Pokyny
úkon	Pokyny	Příkazy pro akční členy

Obrázek 4: Schéma hierarchického paradigmatu [5]

Nejprve provede robot detekci okolí a vytvoří globální mapu okolí. Poté přestane brát v úvahu data ze svých senzorů a naplánuje všechny pokyny pro dosažení cíle. Nakonec robot provede první pokyn. Poté, co robot provedl posloupnost detekce-plán-úkon, začíná cyklus znovu. Robot zjistí následek svých akcí a opět se naplánují všechny pokyny pro dosažení cíle. Naplánují se znovu, i když se nezměnily. Poté se znovu provádí.

Data ze všech senzorů se slučují do jedné globální datové struktury, ke které má přístup plánovač. Takováto globální datová struktura se nazývá modelem okolí. Model okolí může obsahovat:

- již dříve ověřenou reprezentaci prostředí, ve kterém robot pracuje (např. mapa budovy)
- detekované informace (např. kde přesně se v rámci budovy nachází)
- všechny další znalosti, které mohou být potřebné pro splnění úkolu (např. souřadnice odkud kam dojet)

U hierarchických systémů je tedy základním znakem využívání všech dostupných údajů (aktuálních i minulých), a to k definici optimálního plánu na dosažení zadaného cíle. Nejčastěji je při implementaci takového systému využita funkční dekompozice řešeného problému. Příkladem může být systém obsahující modul, který zpracovává informace ze senzoru, které jsou vstupem pro modul vytvářející model okolí. Tento model je předán modulu pro plánování. Na základě údajů v globální mapě plánovač rozhodne, co bude robot dělat, aby dosáhl zadaných cílů (ke každému z cílů generuje posloupnosti akcí, jak daného cíle dosáhnout). Systém řízení robota se pak pokouší dané akce vykonat. V anglické literatuře se tento postup nejčastěji označuje jako SPA (Sense-Plan-Act).

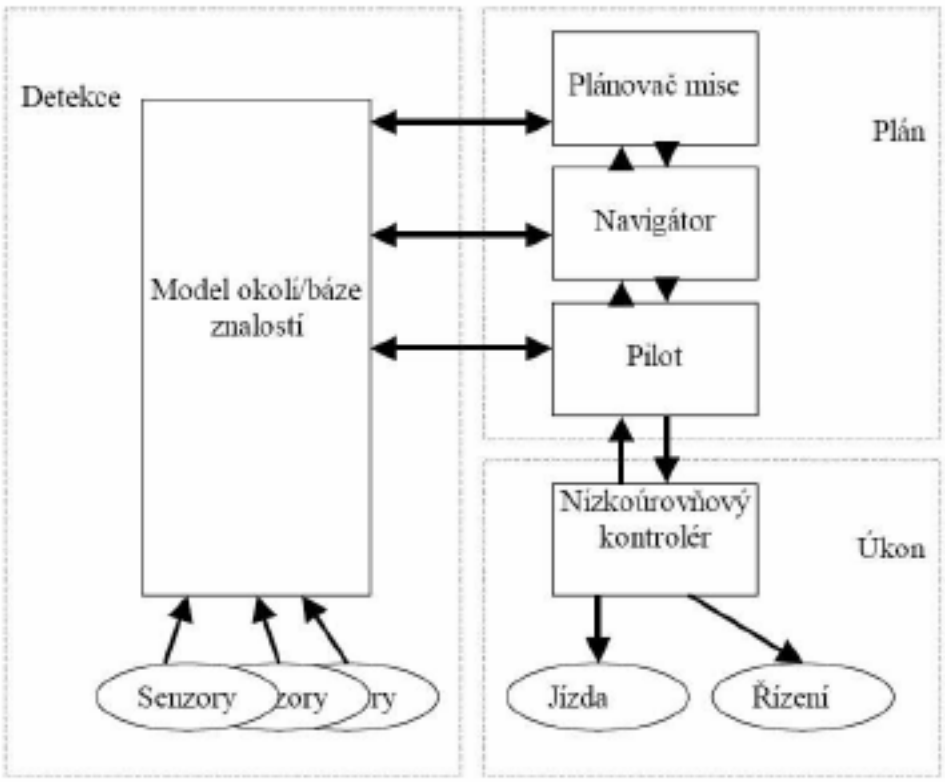
Architektury hierarchického paradigmatu

Architektura je definována jako metoda implementace paradigmatu. Architektura je konkrétní zhmotnění principů. Nejčastěji se vyskytují architektury Nestet Hierarchical Controller (NHC) a NIST Realtime Control System (RCS).

Nestet Hierarchical Controller (NHC)

U metody NHC (viz Obrázek 5) je hlavním příspěvkem dekompozice plánování do třech různých podsystémů. Tyto podsystémy jsou: plánovač mise, navigátor a pilot. Plánovač mise obdrží zadání mise od člověka, nebo si misi zvolí sám. Plánovač mise přeloží cíle mise do jazyka, kterému rozumí ostatní funkce. Poté se plánovač mise podívá do globálního modelu okolí a zjistí, kde se robot nachází a kde je jeho cíl. Navigátor převezme tyto informace a vygeneruje cestu ze současné lokace do cíle. Vygeneruje řadu traťových bodů. Ty se předají pilotovi. Pilot vezme první přímý segment a stanoví, jaké akce má robot provést, aby projel první segment. Nevýhody hierarchického paradigmatu jsou následující. Plánovač může dobře plánovat pouze tehdy, pokud je přesný model okolního světa. Ten ale většinou nelze získat. Jeho procházení a vyhledávání by pak bylo výpočetně neúnosně náročné.

Tento přístup se také nehodí, pokud je přítomna velká míra neurčitosti. Tento přístup je výpočetně velmi náročný také proto, protože vůbec neobsahuje akce typu stimul - odezva. Vždy je přítomna ještě fáze plánování.



Obrázek 5: Nested Hierarchical Controller [5]

Hlavní výhodou koncepce SPA je zejména její relativní průhlednost. Všechna data putují sériově jedním směrem. Jsou tak postupně přetvářena jednotlivými algoritmy, než se dosáhne požadovaného cíle. Tento postup odpovídá nejobecnějším a nejpoužívanějším postupům na zpracování dat a je tedy možné pro jeho design, vývoj a testování použít techniky obecně známé a odzkoušené. Díky tomu typicky získáme modulární systém s dobře definovanými rozhraními a funkcemi. Takto sestavený systém lze také dobře testovat.

I když se tato koncepce osvědčila například u zpracování databázových dat, ukazuje se, že její přímá aplikace na řízení systému v reálném čase, což mobilní roboti bez pochyby jsou, není zcela bez komplikací. Jedním z problémů je právě striktně „dopředný“ tok dat. Systém typicky pracuje po krocích, kdy je aktivní vždy jeden z modulů. Informace postupně putují od senzorového subsystému do modelování světa a do plánovače, ale nikdy ne obráceně. Jedním „cyklem“ systému je potom myšlen čas, který uplyne, než je jeden a ten samý modul znovu aktivován s novými daty. Celkové chování robota potom rozhodujícím způsobem závisí na délce tohoto cyklu. Pokud je cyklus kratší, robot reaguje rychleji na nově příchozí informace ze senzoru o okolí a může tedy rychleji přizpůsobit své akce aktuální situaci. Na druhou stranu je ale limitován časem, který mohou jednotlivé moduly

strávit zpracováváním příchozích dat a přidáním nových údajů do vnitřního modelu. Následně vytvořený plán tedy nemusí být za všech okolností optimálním řešením.

Zvolíme-li delší krok, vystavujeme robota nebezpečím plynoucím z pomalých reakcí na změny okolí. Pokud probíhá plánování příliš dlouho, tak v době vyhotovení plánu nemusí vůbec odpovídat aktuální situaci. A toto řešení je ještě méně optimální než v prvním případě. Další záludností je zpracování dat ze senzoru do podoby nějakého modelu, se kterým by ostatní části mohly pracovat. Dříve se výzkum na poli umělé inteligence zabýval především různými reprezentacemi světa a plánováním nad nimi. Vlastní konstrukce modelu světa z dat senzorů se většinou odsouvala stranou jako problém nepatřící na pole umělé inteligence. Zavedením různých abstrakcí se dařilo úspěšně vyhýbat problémům spojeným s hardwarem robota (senzory i motorickou částí) a vyvíjet algoritmy, které všechny problémy řešily např. v abstraktním světe polygonu. Rané práce v umělé inteligenci se zabývaly především řešením různých her, geometrickými úlohami a podobně.

Je nutné podotknout, že informace, které má robot k dispozici, nejsou nikdy úplné. Právě proto logicky nemůže být úplný ani žádný model na jejich základě postavený. Většina algoritmů je ale definována nad strukturami plně reprezentujícími okolí robota. V případě, že tomu tak není, ztrácíme mnoho výhod, které by nám mohlo použití těchto algoritmů přinést.

Za správný přístup se typicky považuje použití vhodné abstrakce při řešení problému, protože se tím vyjasní nejen vlastní problém, ale i množina možných řešení. Po vyřešení některých problémů v abstraktních doménách vyvstala celkem logicky potřeba aplikace těchto řešení na reálné problémy. Obecně se soudilo, že nejlepší způsob, jak toho dosáhnout, tkví ve vhodné reprezentaci složitější domény tak, aby mohl být daný algoritmus použit skoro v původní podobě. Vzájemně jednoznačné mapování např. mezi skutečným světem a světem konvexních polygonů nebo mezi obecným robotem a bezrozměrným bodem se ukázalo být tvrdším oříškem, než se očekávalo. Přes tyto komplikace je v dnešní době možné realizovat i tímto způsobem řadu úloh, u kterých by to dříve bylo nemyslitelné. Algoritmy, které se před pěti lety zdály nepoužitelné, dnes proběhnou v jednom kroku SPA bez problémů, hlavně díky vyššímu výkonu běžně dostupných počítačů. Přesto však mají tyto systémy tendenci být nedostatečně flexibilní a těžkopádné i pro triviální úlohy.

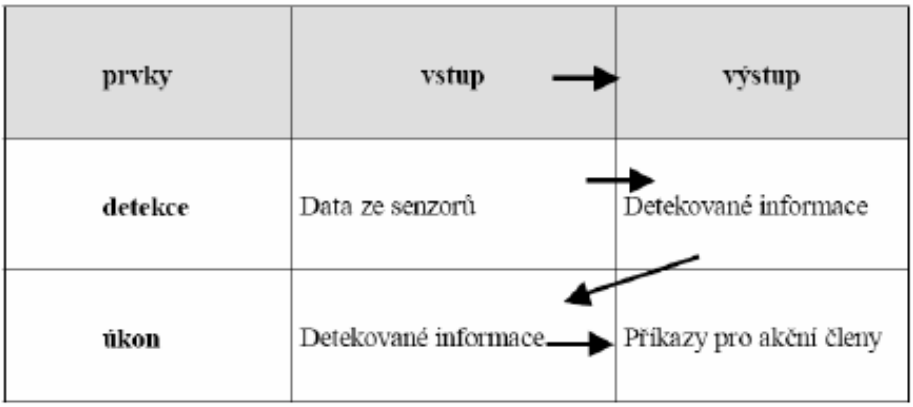
V osmdesátých letech se proto objevilo několik architektur řešících nejbolavější problémy SPA a deduktivních systémů obecně. Tyto architektury se začaly souhrnně označovat jako „reaktivní."

3.3.2 Reaktivní paradigma

Jako základ reaktivního paradigmatu (viz Obrázek 6 a 7) posloužila biologie, a to především chování zvířat v přírodě. V robotice je zvlášť důležité znát principy zvířecí inteligence. Živočichové žijí v otevřeném světě a musí překonat problém „uzavřeného světa". Problém uzavřeného světa nastává u takových robotů, kteří potřebují pro své fungování znát přesně všechny údaje o okolním světě. To lze v uzavřené místnosti, avšak ani přes zvětšující se výpočetní možnosti není možné znát všechny údaje

o celém světě nebo o rozsáhlém prostředí, ve kterém se robot pohybuje. Problém „uzavřeného světa“ dokáží překonat i velice jednodušší živočichové, kteří prakticky nemají žádný mozek (např. hmyz, žáby). V biologické inteligenci je základní element chování, které slouží jako základní komponenta inteligence ve většině robotických systémů. Chování je definováno jako mapování vstupů senzorů na vzor motorických akcí, které jsou použity k dosažení cíle.

Obrázek 6: Reaktivní paradigma [5]

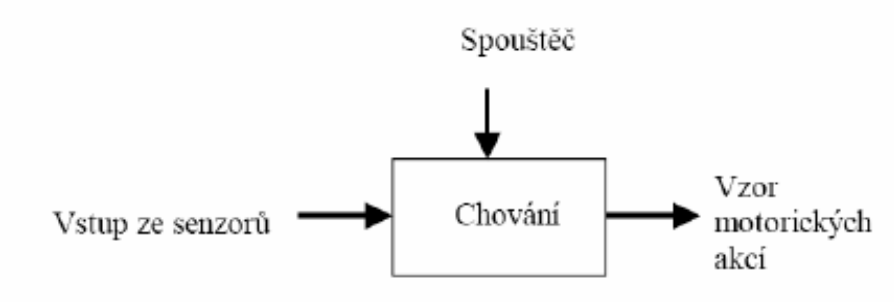


Obrázek 7: Reaktivní paradigma [5]

Chování je možné rozdělit na tři druhy:

- reflexivní chování a chování stimul - odezva: stimul je zde přímo spojen nervovým spojením s odezvou. Je tak možné dosáhnout nejkratší doby odezvy, např. ucuknutí při pocitu bolesti.
- reaktivní chování: je to chování, kterému se musí naučit, tedy není vrozené. Spouští se bez vědomých myšlenek. Např. řízení motocyklu. Reaktivní chování lze změnit vědomými myšlenkami. Např. pokud při řízení motocyklu jedeme přes úzký most.
- vědomé chování: chování předem naplánované a zcela řízené našim vědomím např. sestavování různých částí stavebnice

Konkrétní chování spouští odpovídající spouštěč (viz Obrázek 8). Spouštěč je specifický stimul, který spustí stereotypní vzor akcí. Spouštěč se chová jako ovládací signál, který aktivuje chování. Pokud není chování spuštěno, vstupy senzorů nevytváří motorické výstupy. Např. živočich, který nemá hlad, nejí jídlo, i když má tu možnost.



Obrázek 8: Spouštění chování [5]

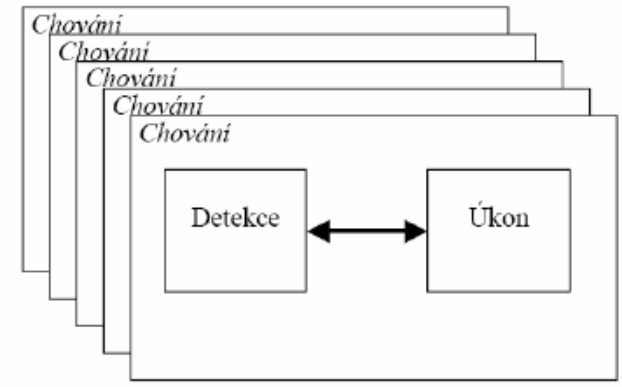
Jako spouštěč slouží logická proměnná. Může se jednat i o kombinace logických proměnných. Chování se mohou navíc i kombinovat. Spouštěčem se může stát vnitřní stav (motivace) nebo stimul z okolního prostředí. Vnímání lze použít dvěma způsoby. Vnímání může být použito jako spouštěč chování, nebo jako vstup v chování, nebo i oběma způsoby najednou. Např. ryba uvidí světlo (spouštěč), toto světlo vidí a plave k němu (vstup ze senzorů).

Vznik reaktivního paradigmatu byla reakce na hierarchické paradigma a vedlo k velkému pokroku v robotice. V současnosti se stále používá, je však znát tendence k přechodu k hybridnímu paradigmatu. Reaktivní paradigma vzniklo také jako reakce na zkoumání dějů v biologii.

Z reaktivního paradigmatu zcela zmizelo plánování. Jeho organizace je pouze detekce - úkon. Zatímco v hierarchickém paradigmatu se úkon (pohyb) provádí na základě plánu, v reaktivním paradigmatu se úkon provádí pouze na základě přímých čtení ze senzorů (detekce). Takto by však mohl robot provádět pouze jednu činnost. Proto robot obsahuje více dvojic detekce - úkon.

Dvojice detekce - úkon se nazývají chování (viz Obrázek 9). Chování je proces, do kterého vstupují lokálně detekovaná data, a který provede nejlepší úkon nezávisle na ostatních procesech. Chování se mohou kombinovat.

Chování je definováno jako mapování vstupů senzorů na vzor motorických akcí, které jsou použity k dosažení cíle. Z matematického hlediska je chování funkce. Chování, které se v této práci popisuje, obsahuje alespoň jedno motorické schéma a jedno schéma vnímání.



Obrázek 9: Chování u reaktivního paradigmatu [5]

Detekce pomocí senzorů je lokální pro každé chování. Jedno chování neví co přijímá jiné. Více chování však může přijímat data z jednoho senzoru. Systémy, které pracují pomocí reaktivního paradigmatu, jsou velice rychlé, výpočetně nenáročné, nepotřebují téměř žádnou paměť.

Vrstvová architektura

Na těchto základech byla postavena vrstvová architektura (SA - Subsumption Architecture) a je zřejmě nejznámější z reaktivních architektur. Nejdůležitějším rozdílem oproti dřívějším architekturám je rozsazovaná úkolová dekompozice řídícího systému namísto tradiční funkční úkolové dekompozice a nepoužívání žádných stavových informací.

Řídící systém by se měl podle SA skládat z několika vrstev postupně složitějších a složitějších modulů. Každý modul implementuje některé konkrétní úkoly nebo dovednosti robota. Jejich výstup se pak v SA nazývá chování (behaviour). Moduly by měly v maximální míře využívat lokální informace o okolí a řídit robota reaktivně - například při vyhýbání se překážce není třeba plánovat celou cestu kolem, ale stačí se pouze pohybovat podél překážky a vyčkávat, až bude mimo dosah.

Rozdělením systému podle požadovaných dovedností lze docílit toho, že při implementaci jednoho chování nemusíme brát v úvahu všechny možné senzorické stimuly, ale jen ten typ, který má přímý vztah k implementované dovednosti. Pokud tedy omezíme vstupní domény, můžeme design modulu značně zjednodušit.

Aby bylo možné zaručit nezávislost jednotlivých modulů, je snahou minimalizovat vzájemnou komunikaci mezi nimi. Propagována je spíše komunikace přes okolní prostředí, kdy se robot pod kontrolou jednoho modulu dostane do takového vnějšího stavu, kdy je jiný modul aktivován na základe údajů ze senzoru. Tímto způsobem klesá kromě potřeby komunikace i potřeba uchovávání stavové informace, která je schována v konfiguraci robota a stavu prostředí v jeho blízkosti.

To, že modul nemá žádný vnitřní stav, v sobě skrývá ještě jednu výhodu. Díky tomu, že se moduly rozhodují podle nejnovějších údajů ze senzorů, nehrozí zde nebezpečí, že by svá rozhodnutí mohly zakládat na zastaralých informacích. Také nehrozí nebezpečí, že by se vnitřní stav neshodoval s realitou. Navíc si není třeba dělat starosti s globální konzistencí takového stavu (jestli má mapa správnou topologii, jestli není geometricky nesmyslná a pod.). Problémem však zůstává rozhodnutí, který modul by měl v daném čase kontrolovat robota.

Problém reaktivní architektury

Ve své době dosáhla SA nevídaných výsledků na poli bezkolizní navigace. Zatímco roboti založení na hierarchické architektuře stavěli své modely a plány na výkonných počítačích, roboti založení na SA se rychle prohánělí po laboratořích. Přetrvával obecný názoru, že vyšší rychlost znamená vyšší inteligence. Reaktivní architektura tak vypadala jako zásadní průlom v řízení robotů. Mnoho robotů bylo poté postaveno na jejích základech. Brzy se však začaly projevovat některé její nedostatky.

Jedním z velkých problému SA je paradoxně její heslo „svět je svůj nejlepší model“. Obtížný problém modelování světa je v SA vyřešen tak, že se zkrátka žádný model nevytváří. Každý modul je sám zodpovědný za interpretaci a uchovávání senzorických údajů v libovolné formě. SA se spoléhá na absolutní kvalitu senzoru a na jejich 100% spolehlivost. Na senzory jsou pak kladeny nerealistické nároky a robot jimi musí být dostatečně vybaven, což zbytečně zvyšuje náklady na jeho stavbu. Například těžké závěsy jsou díky dobrému pohlcování zvuku pro robota používajícího pouze sonary prakticky neviditelné. Pokud by robot mohl mít mapu, kde by toto jedno místo bylo vyznačeno, pak by se jim dokázal vyhnout. Stačilo by, aby věděl, kde se nachází.

Pravdou je, že svět je svůj nejlepší model, ale rozhodně to není model, který by byl snadno přístupný. Navíc robot může ze své pozice přímo zjišťovat pouze údaje lokální. Pokud se rozhoduje jen na jejich základě, hrozí nezanedbatelné nebezpečí, že zůstane zachycen v některém z lokálních minim.

Na první pohled by se mohlo zdát, že jsou moduly vzhledem k omezené komunikaci na sobě velmi nezávislé. Každý modul by také měl mít značně omezenou doménu působnosti, a to díky velké specializaci. Například na vyhýbání se překážkám stačí jejich vzdálenost od robota a není třeba jejich barva a textura, která může být třeba na dosažení jiného cíle. Neměl by tedy být velký problém moduly samostatně navrhnout a samostatně otestovat. Ve většině případu tomu skutečně tak je.

Obtíže začnou nastávat až při sestavování množiny nezávislých modulů do podoby kompaktního kontroléru spolehlivě řídícího robota za všech okolností. Ukazuje se, že při použití pevných priorit a podobných jednoduchých postupů na řešení konfliktu výstupů modulů není možné vyvíjet moduly jako jednotlivé entity. Je tedy nezbytné vyvíjet celý systém jako celek již od začátku a to částečně degraduje výhodu vrstvené architektury.

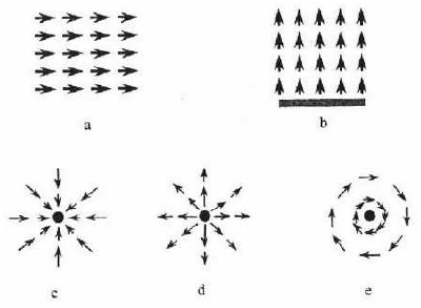
Metoda potenciálních polí

Dalším stylem jak implementovat reaktivní architekturu je pomocí potenciálních polí. Tento styl používá vektory a součty vektorů. Součtem vektorů z jednotlivých chování je výsledný vektor, který reprezentuje výsledné chování. Tato metodologie tedy předpokládá, že motorické akce chování jsou reprezentovány potenciálovým pole. Potenciálové pole je pole vektorů. Každá buňka pole reprezentuje část prostoru, vektor v buňce reprezentuje sílu, která působí na robota. Velikost vektoru reprezentuje velikost síly.

Na Obrázku 10 jsou znázorněny základní typy potenciálových polí:

a) uniformní
b) kolmé
c) přitažlivé
d) odpudivé
e) tečné

Každé chování má svoje potenciálové pole, které reprezentuje motorické chování.

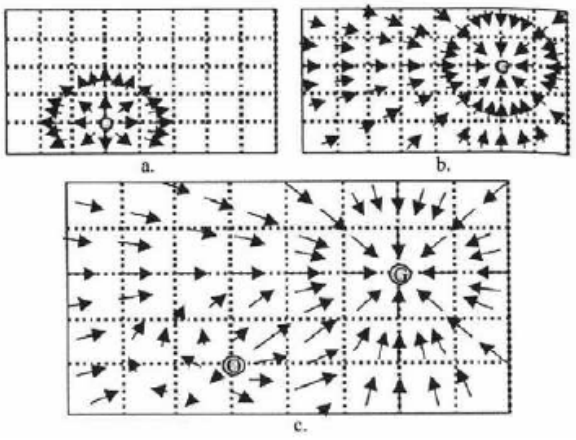


Obrázek 10: Základní typy potencionálových polí [5]

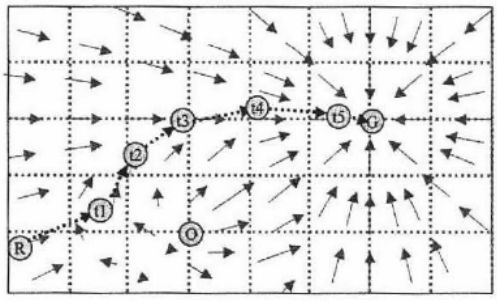
Na Obrázku 11 je popsán příklad kombinace dvou různých chování. Na mapě se nachází překážka a cíl. Jednotlivé části lze popsat následovně:

a) reprezentuje odpudivé pole od překážky
b) reprezentuje přitažlivé pole k cíli
c) je pak kombinace těchto dvou chování

Pokud bychom měli plánovat cestu robota z některého místa mapy do cíle, pak by výsledná cesta byla přímo ovlivněna působením jednotlivých potenciálových polí. Obrázek 12 ukazuje výslednou cestu.



Obrázek 11: Kombinace dvou různých chování a jejich kombinace [5]

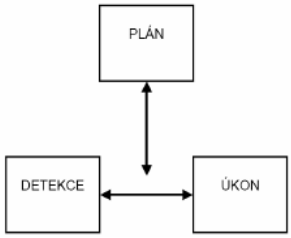


Obrázek 12: Výsledná cesta [5]

Nevýhodou reprezentace pomocí potenciálových polí je vlastnost, že součtem vektorů jednotlivých chování můžeme dostat nulový vektor a robot se pak ocitne v pasti.

3.3.3 Hybridní paradigma

Nevýhody reaktivní architektury můžeme vyřešit využitím složitějších algoritmů na rozhodnutí, který modul má v daném okamžiku robota řídit. Složitost takového systému je však značná (typicky vetší než u jednotlivých řídících modulů), a proto se jeho použitím dostáváme k další kategorii řídících systému, a tou jsou kombinované systémy (hybridní paradigma) viz Obrázek 13.



Obrázek 13: Schéma hybridního paradigmatu [5]

Princip paradigmatu

Ke konci osmdesátých let byl trend navrhovat u robotů umělou inteligenci pomocí reaktivního paradigmatu. Novým úkolem bylo opětovné začlenění plánování a uvažování do robotů, aniž by byly ztraceny výhody reaktivního paradigmatu. Z tohoto důvodu bylo chování zachováno a bylo určeno pro provádění nízkoúrovňových operací. Nad vrstvu chování byly přidány další vrstvy.

Systémy postavené na hybridním paradigmatu typicky rozdělují kontrolér do tří vrstev. Vrstva nejnižší odpovídá reaktivnímu kontroléru implementovanému množinou modulů. Každý z nich dodává robotovi určitou schopnost. Vrstva nejvyšší svou architekturou naopak odpovídá deduktivním systémům. Mezi těmito vrstvami leží vrstva zodpovědná za interpretaci plánu dodaných plánovačem v posloupnost aktivování modulu z vrstvy nejnižší tak, aby se robot choval podle navrženého plánu.

Pokud bude na těchto základech postaven kontrolér, bude schopen řešit nejtíživější problémy předchozích architektur. Při tvoření detailního plánu v hierarchické architektuře vždy hrozilo několik základních nebezpečí:

- vytváření plánu může probíhat tak dlouho, že se okolí robota mezitím změní natolik, že se vytvořený plán nedá použít
- během plánování se robot nevěnuje dostatečně svému okolí a muže se tedy dostat do potenciálně nebezpečných situací (pro sebe i okolí)
- interní model prostředí není synchronizovaný se skutečností, není konzistentní, není úplný nebo je jinak nedostatečný pro detailní naplánování všech potřebných úkonů

Výhodou je, že je robot stále pod kontrolou nejnižší reaktivní vrstvy, a to bez ohledu na to, co právě dělají vyšší vrstvy. Pokud tato poskytuje základní funkce pro přežití robota, nemůže se stát, že by nezaregistroval nebezpečí a díky tomu se dostal do nebezpečné situace.

Vrstva plánování zahrnuje veškeré rozhodování a modelování okolního světa, nezahrnuje tedy pouze plánování cesty. Robot tedy pracuje tak, že nejprve naplánuje, jak dosáhnout cíle a poté vytvoří nebo aktivuje množinu chování (Detekce - Úkon), které plán nebo část plánu provedou. Chování jsou spuštěna, dokud není plán proveden, poté plánovač vygeneruje další množinu chování atd. Plánování pokrývá dlouhodobý horizont a vyžaduje globální znalosti, je odděleno od zpracování v reálném čase. Plánování vytvoří cíle a vybere metody pro jejich dosažení, není však určeno pro podrobné, jemné, opakující se rozhodování.

U hybridního paradigmatu je organizace detekce složitější než u reaktivního. V chováních zůstává detekce stejná jak u reaktivního paradigmatu. Je tedy lokální a specifická pro chování. Avšak plánování a rozhodování vyžaduje globální model světa. Plánovací funkce tedy musí mít do tohoto modelu přístup. Model je konstruován procesy, které jsou nezávislé na detekci v chováních. Tvoření modelu a chování však mohou sdílet stejné senzory. Při tvorbě modelu se mohou používat senzory, které nejsou použity v žádném chování. Organizace detekce, plán, úkon je rozdělena do reaktivní a rozhodovací části.

Generování plánu je již mnohem méně náročné, než u např. hierarchické architektury, protože zde nemusí být plán tak detailní a nemusí tedy končit až na úrovni rychlostí jednotlivých motorů. Stačí, když obsahuje dostatek informací, aby z nich prostřední vrstva byla schopna vyvodit pořadí aktivování jednotlivých dovedností robota. To v důsledku znamená, že plán může být jen rámcový s tím, že některé konkrétní údaje si do něj doplní nižší vrstvy v době, kdy budou dostupné a hlavně aktuální. Tím se také snižuje závislost plánu na konkrétním prostředí (stejný plán může díky své neúplnosti fungovat v různých situacích) i na dokonalosti a úplnosti jeho symbolické reprezentace.

Problémy a nedostatky čistě reaktivní architektury jsou řešeny v kombinovaném systému. Zadávání složitých úloh má za úkol nejvyšší deduktivní vrstva a na základě jejích pokynů rozhoduje vrstva střední, který modul má být kdy aktivován. Vysoká závislost na kvalitě senzoru je omezena hlavně využitím doplňkových stavových informací ve vyšších vrstvách.

Nastávají však komplikace při řešení problému s překážkami, které robot není schopen detekovat svými senzory. Vlastní hardware robota je většinou pod kontrolou reaktivní vrstvy, která nemá potřebné informace na to, aby je mohla zohlednit při řízení. Možná řešení se nabízejí dvě:

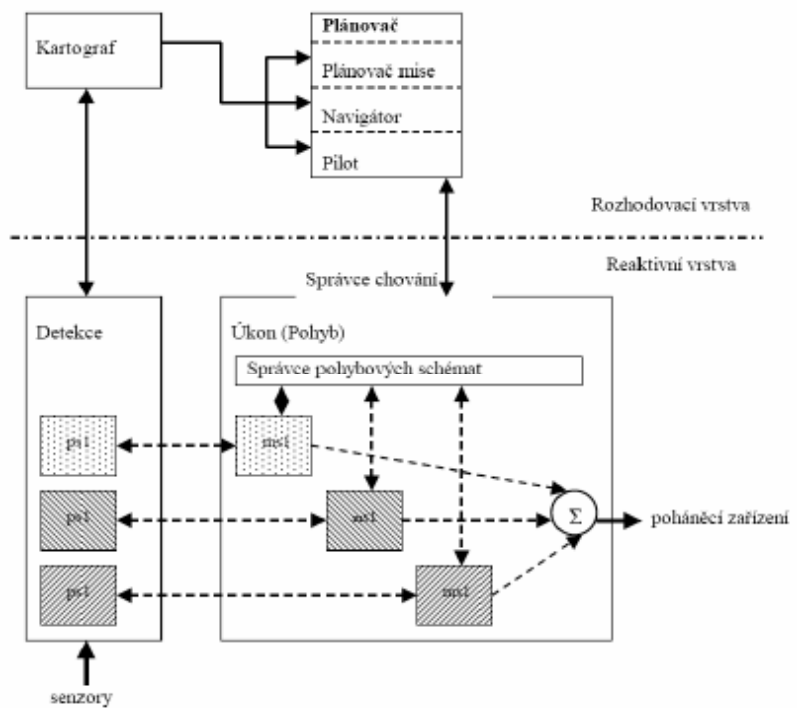
- vyřešit tento problému ve vyšších vrstvách. Toto však není správná volba, protože se zkomplikuje část již tak dost složitá, a to navíc zcela nesystémovým způsobem, protože bychom řešili problém, který svou podstatou patří do vrstvy reaktivní.
- do reaktivní vrstvy dodat potřebná data. Zde vyvstává komplikace jiného druhu. Reaktivní systém pracuje s informacemi o překážkách v lokálních souřadnicích, zatímco v mapě jsou k dispozici v souřadnicích absolutních. Z toho vyplývá, že na to, aby robot mohl tuto informaci využít, potřebuje vědět, kde se nachází.

Architektura hybridního paradigmatu

Hybridní architektura se obvykle skládá z modulů. Typické moduly hybridní architektury jsou:

- řadič - je to modul, který generuje množinu chování, určuje jejich posloupnosti nebo podmínky aktivace. Jeho reprezentace bývá většinou pomocí konečného automatu.
- správce zdrojů - alokuje zdroje pro chování. Např. pokud robot obsahuje více senzorů pro měření vzdálenosti k nějakému předmětu, přiřadí určitému chování ten nejvhodnější.
- kartograf - je odpovědný za tvorbu, uložení a udržování mapy a prostorových informací. Často obsahuje globální model světa a přídavné informace.
- plánovač mise - komunikuje s člověkem, převádí příkazy do termínů, kterým rozumí robot
- monitorování výkonu a řešení problémů - umožňuje zjistit robotovi, jestli při jeho snaze po dosažení cíle dochází k nějakému pokroku

Na Obrázku 14 je znázorněn konkrétní příklad hybridní architektury. V rozhodovací části jsou umístěny dva systémy - kartograf a plánovač. Kartograf je zodpovědný za tvorbu mapy a za funkce, které čtou údaje potřebné pro navigaci. Může mu být také předána již existující mapa. Plánovač je rozdělen na tři části. Plánovač mise je rozhranním mezi člověkem a programem robota. Navigátor spolupracuje s kartografem, vytvoří cestu robota a rozdělí ji na části. Pilot vezme první část a vygeneruje chování a předá je správci pohybových schémat. V reaktivní části jsou dva systémy, detekce a pohyb. Základem principu hybridního paradigmatu je vzájemná komunikace jednotlivých modulů.



Obrázek 14: Architektura hybridního paradigmatu [5]

4 Simulace chování robota

Jak již bylo v kapitole 1 uvedeno, je simulace chování robota velmi užitečnou záležitostí. Je tedy třeba vybrat pouze onen sofistikovaný software. V této práci se využívá služeb Microsoft robotics studia [2]. Jedná se o nový software od společnosti Microsoft a vyznačuje se velmi obecným návrhem. Je tedy jedno, jestli řídící program řídí skutečného robota nebo simulovaného. Velkou výhodou je také řada jazyků, ve kterých je možné robota programovat. Obsahuje též již předdefinované roboty, ze kterých je možné vycházet. Proto bylo v této práci zvoleno právě Microsoft Robotics Studio.

4.1 Microsoft robotics studio

Microsoft Robotics Studio nabízí jednotné API pro ovládání robotů. Jedná se o nadstavbu nad libovolnou verzí Visual Studia 2005. Kromě reálných robotů nebo stavebnic je možné ovládat též roboty v emulátoru. Programovací jazyk je pro tuto práci zvolen C#.

Před samotnou implementací robota je třeba navrhnout a implementovat scénu, ve které se bude pohybovat. Jednou z možností je vložit vzhled scény přímo do zdrojového souboru aplikace. Lze pomocí jednoduchých příkazů přímo vkládat jednoduchá primitiva, jako např. koule, kvádr atd. Jejich poloha ve scéně se zadává přímo při vytváření konkrétní primitivy. Při simulaci je manipulace s objekty možná, a to tak, že lze přepnout simulaci do tzv. editačního módu. Poté lze zvolit libovolný objekt ve scéně a měnit u něj všechny atributy (pozice, rotace, barva atd.). Takto definovaná scéna se dá prohlížet v jediném souboru XML, kde jsou uloženy všechny atributy všech objektů. Tento soubor je možné uložit a zachovat tak scénu. Při dalším zpuštění je pak možné tento soubor načíst.

Takto vytvořenou scénu je pak možné prohlížet v několika módech. Microsoft Visual Simulation Environment umožňuje visuální, drátový, fyzikální a kombinovaný pohled. U visuálního pohledu lze vidět grafický vzhled entit tak, jak jim byl vymodelován v některém grafickém studiu. U drátového modelu si lze prohlédnout předchozí model, a to tak, jak byl vytvořen. Lze tedy pozorovat jednotlivá primitiva, ze kterých byl sestaven. Fyzikální model poskytuje náhled na fyzikální stavbu entity, která nemusí mít nic společného s grafickou podobou entity. Jsou zde patrné primitiva, ze kterých je entita složena. Dále pak směr síly a těžiště. Kombinovaný náhled pak slouží k porovnání grafického a fyzikálního modelu entity.

Jednoduchou možností, jak vytvořit řídící program robota, je Visual Programming Language (VPL – vizuální programovací jazyk). VPL pracuje na principu funkčních bloků, které reprezentují různé servisy. Funkční bloky stačí umístit na pracovní plochu a jejich propojením vzniká aplikace řídící robota. Dále je možné celé schéma „zabalit“ do jedné komponenty a tu opětovně využít v dalším schématu.

Pokud je robot simulován, využívá se 3D prostředí s fyzikálními vlastnostmi. Robot tedy není pouze grafická entita, ale platí pro něj fyzikální zákony a sám má fyzikální vlastnosti (hmotnost, tření atd.).

K interakci s roboty je využito klasického okna nebo webového prohlížeče. Lze tedy snadno například využít html a javaScript ke sledování senzorů robota.

Důležitou knihovnou je CCR, což je zkratka od Concurrency&Coordination runtime (souběžný a koordinovaný běh). Tato knihovna řeší u robotů koordinaci mezi jednotlivými aktivitami, které robot může dělat (program je tedy co nejvíce asynchronní). Knihovna je založená na portech (třída Port), pod kterými si lze představit objekt, který může obsahovat frontu zpráv nebo dat nějakého typu. V případě robotů jsou porty všechny senzory a motory (ze senzorů může aplikace data číst a do portů pro motory zapisovat).

Druhou důležitou třídou v CCR je třída Arbiter, pomocí které se dají vytvářet handlery událostí (pokud přijde zpráva do portu) a zároveň se pomocí ní dají handlery reagující na události z portů spojovat dohromady (například pokud chceme, aby robot začal nějakým způsobem reagovat až poté, co se ve dvou různých portech objeví zpráva) - v terminologii CCR se toto propojení nazývá *join*. Další možností jak spojit události je pomocí propojení *choice*, které vyjadřuje, že robot má čekat na jeden z několika povelů a poté začít vykonávat reakci.

4.2 Simulace chování robota v Microsoft Robotics Studiu

Vytvoření robota v Microsoft Robotics studiu se skládá z několika fází. Je třeba robota vymodelovat graficky. Vytvořit jeho vzhled, který ovšem nemá nic společného s tím, jak se bude chovat vůči jiným objektům ve scéně. Dále pak vytvořit fyzikální model. V poslední fázi se fyzikálně namodelují senzory a spustí se služba, která umožňuje čtení dat z těchto senzorů.

V této práci byl navržen a vytvořen robot, který svým chováním a vzhledem odpovídá robotu používaném při soutěžích v robotickém fotbale. Jeho tvar odpovídá krychli. O jeho pohyb se starají dvě hnaná kola s nastavitelným směrem otáčení a rychlostí otáčení pro každé kolo. Jedná se tedy o diferenciální řízení, jak již bylo vysvětleno v kapitole 3.2.1.

4.2.1 Grafická tvorba robota

V této prácí bylo využito programu Blender Creator 2.4 [3]. Tento program je open-source 3D modelář, ve kterém je možné vytvářet animace, provádět střih a kompozici video sekvencí nebo dokonce vytvářet interaktivní hry. Vlastní rozhranní je vykreslováno pomocí knihovny OpenGL. Využití této knihovny poskytuje nejen hardwarovou akceleraci, ale především snadnou přenositelnost

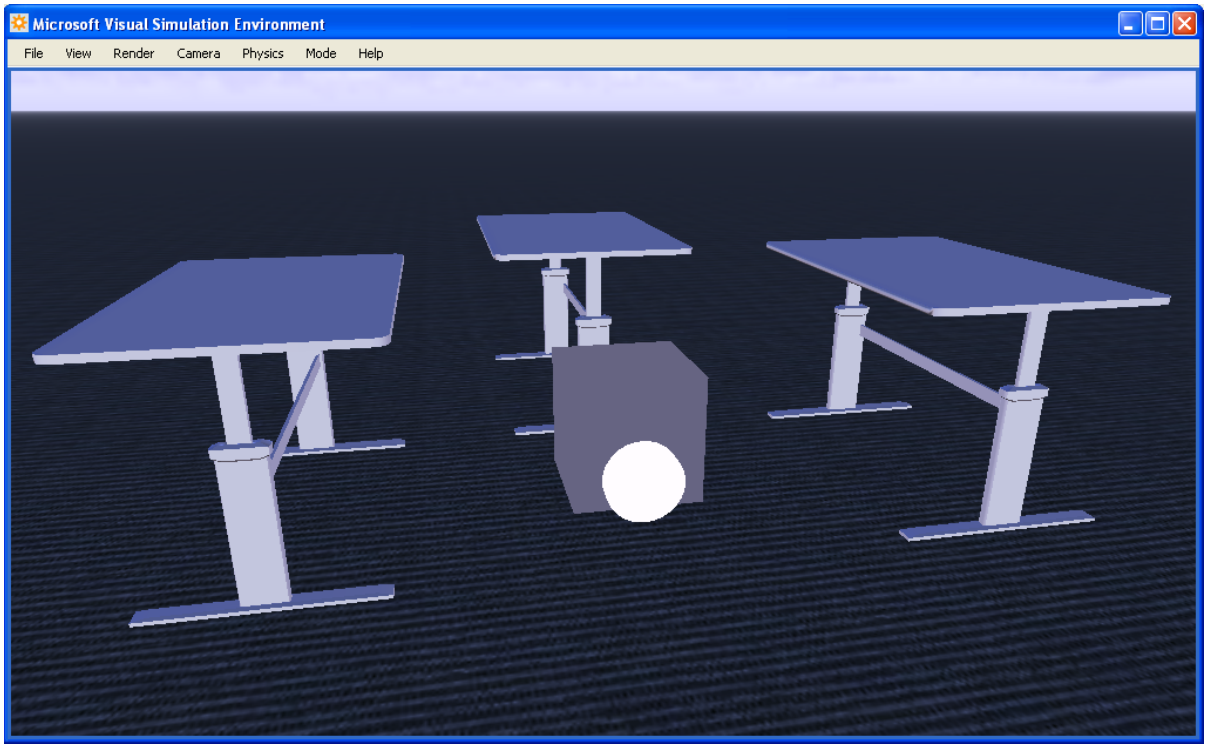
na ostatní platformy. Blender je dostupný na mnoha platformách a operačních systémech (FreeBSD, IRIX, GNU/Linux, Microsoft Windows, Mac OS X a Solaris).

Modelovací schopnosti jsou především zaměřeny na práci s reprezentací těles pomocí malých ploch. S oblibou jsou uživateli využívány tzv. subsurf plochy. Podporuje také práci s parametrickými plochami a křivkami a implicitními plochami.

Blender poskytuje uživatelsky příjemné rozhraní a dokáže vymodelovanou scénu exportovat do souborů s příponou *.obj a *.mtl. Ty jsou podporovány Microsoft Robotics Studiem a je možné je přímo načíst do scény.

Takto vytvořený model však musí alespoň přibližně odpovídat fyzikálnímu model, jinak by chování robota neodpovídalo tomu, co by bylo možné vidět v grafickém výstupu.

Na Obrázku 15 je zachycen grafický vzhled robota. Je umístěn mezi třemi překážkami, které tvoří stoly.



Obrázek 15: Robot ve scéně

4.2.2 Fyzikální model robota

Dále je třeba vytvořit fyzikální model. V této práci byl použit model robota Pioneer3DX. Fyzikální model robota je složen ze dvou primitiv, a to kvádru a kruhu. Kvádr definuje tělo robota a dotykové senzory. Kruh pak odpovídá oběma kolům.Všechny jeho fyzikální vlastnosti jsou uloženy v souboru entities.cs a lze je tedy velmi jednoduše měnit. Je zde uložena jak definice robota, tak i jeho senzorů.

K vytvoření samotného těla robota byla použita třída class Pioneer3DX, která dědí ze třídy DifferentialDriveEntity. Při tvorbě objektu této třídy lze zadat pouze pozici entity. Zbývající hodnoty jsou pevně nastaveny v souboru entities.cs a zde je možné je měnit. Jsou to tyto atributy:

- MASS – hmotnost zadávaná v kilogramech
- CHASSIS_DIMENSIONS – trojrozměrný vektor, který udává velikost šasi robota v metrech
- CHASSIS_CLEARANCE – posunutí šasi nad zemí udávaná v metrech
- FRONT_WHEEL_RADIUS – poloměr řízených kol v metrech
- CASTER_WHEEL_RADIUS – poloměr opěrného kolečka v metrech
- FRONT_AXLE_DEPTH_OFFSET – vzdálenost hřídele od centra robota udávaná v metrech
- MotorTorqueScaling – točivý moment motoru

Po vytvoření řídící základny robota je možné osadit robota senzory. K tvorbě dotykových senzorů byla využita třída class BumperArrayEntity, která dědí ze třídy VisualEntity. Nejprve je nezbytné vytvořit dva objekty typu BoxShape (kvádr). První z nich bude reprezentovat přední dotykový senzor a druhý zadní. Poté již stačí vytvořit objekt třídy BumperArrayEntity a jako parametr mu zadat tyto dva objekty.

Při tvorbě laserového dálkoměru byla použita třída LaserRangeFinderEntity, která dědí ze třídy VisualEntity. U vytvoření objektu třídy LaserRangeFinderEntity můžeme zadat pouze jeho pozici. Další vlastnosti jsou uloženy opět v souboru entities.cs. Jsou to tyto atributy:

- SCAN_INTERVAL – interval, který udává počet skenování za sekundu
- IMPACT_SPHERE_RADIUS – poloměr dosahu v metrech

4.2.3 Zasazení robota do scény

Pokud vytvoříme grafický a fyzikální model, je možné robota umístit do scény. Ještě před samotným vložením robota do scény je nutné vytvořit zemi. Jinak by došlo k tomu, že by se robot propadl do nekonečné „díry“. Po vytvoření země a popřípadě oblohy vytvoříme robota. Umístění jakékoliv entity přímo ze zdrojového kódu do scény probíhá tak, že se použije příkaz:

```
SimulationEngine.GlobalInstancePort.Insert(entita);
```

Robot je vytvořen tak, že se vytvoří nový objekt třídy Pioneer3DX. Data o fyzikálních vlastnostech robota jsou načítána ze souboru entities.cs. V tomto souboru je popsán fyzikální vzhled a fyzikální vlastnosti, jako hmotnost, tření atd. Dále pak dojde k vytvoření dvou hnaných kol a jednoho opěrného. Po těchto úkonech se robotovi nadefinuje vzhled, a to pomocí přímého zadání barvy, nebo načtením ze souboru.

Robot prozatím neobsahuje žádné senzory. Jsou tedy přidány dva senzory, a to dotykový senzor a laserový dálkoměr. V této práci se vychází s již předdefinovaných tříd. Pro laserový dálkoměr byla využita třída LaserRangeFinderEntity a pro dotykové senzory třída BumperArrayEntity. Podrobněji jsou tyto senzory popsány v kapitole 3.1.

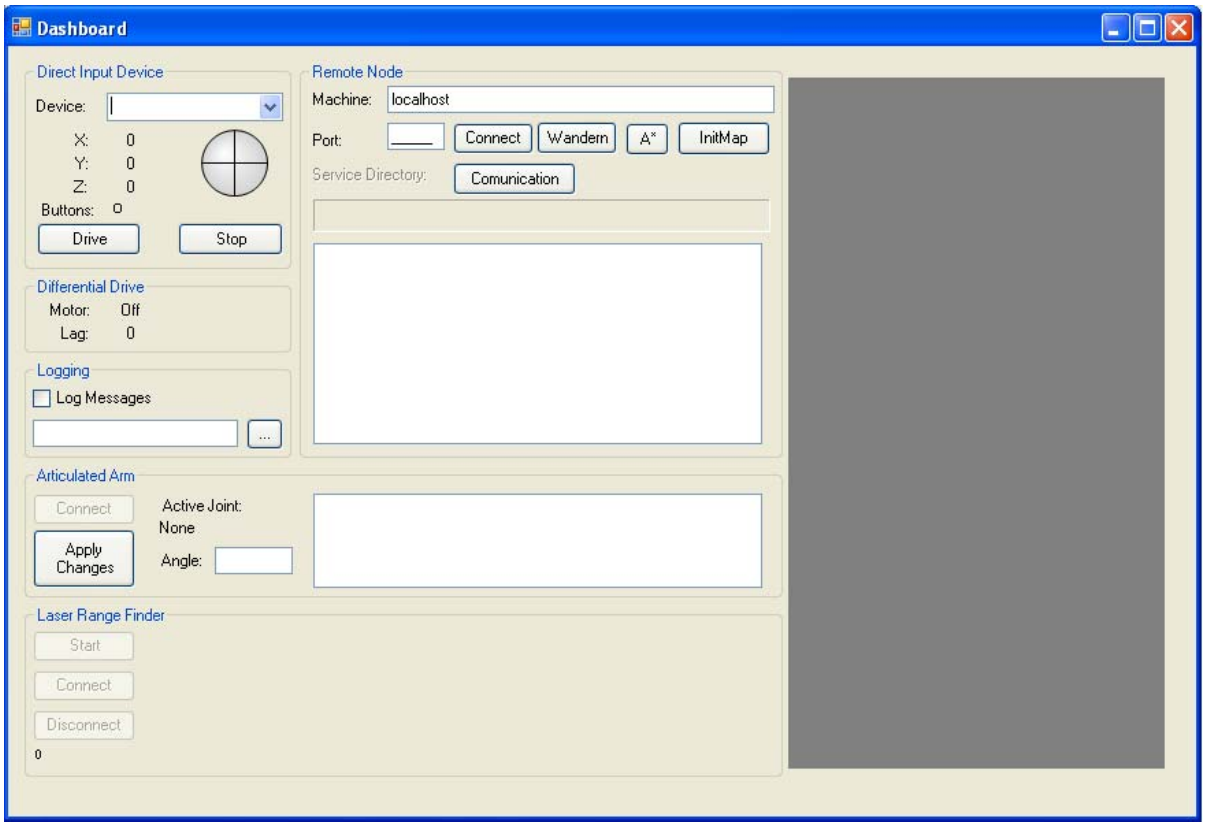
Po těchto úkonech je dále nezbytné vytvořit servis pro komunikaci s nově vytvořenými entitami (motor, laserový dálkoměr a dotykové senzory). Příklad vytvoření servisu pro motor:

```
drive.Contract.CreateService(ConstructorPort,
            Microsoft.Robotics.Simulation.Partners.CreateEntityPartner(
             "http://localhost/" + robotBaseEntity.State.Name)
);
```

5 Rozhranní a jazyk pro řízení robota

Jazykem pro řízení robota je v této práci zvolen C#. Jedná se o moderní, objektově orientovaný jazyk, který byl odvozen od jazyka C++. Tento jazyk také odstranil nebezpečné rysy jazyka C/C++, a to ukazatele, přímé adresování paměti a byla doplněna typová kontrola proměnných. Také byla doplněna automatická správa paměti (Garbage collection). Jazyk C# je postaven nad platformou .NET a může tak využívat její bohatou zásobu funkcí.

Jako rozhranní pro řízení robota byl zvolen panel DashBoard. Jedná se o kontrolní panel, ve kterém lze ovládat pohyb robota a sledovat graficky výstup jeho laserového dálkoměru. Tento panel je možné velmi snadno modifikovat a lze tak dosáhnout libovolného ovládání. Jeho vzhled je znázorněn na Obrázku 16.



Obrázek 16: Kontrolní panel pro ovládání robota

Tento panel funguje tak, že je schopen pracovat s již rozběhnutými službami. Do textové pole Machine se tedy napíše localhost (pokud chceme pracovat se servisy běžícími na daném PC). Poté dojde k vypsání všech entit, se kterými lze nějakým způsobem pracovat.

K ručnímu ovládání robota stačí pouze zvolit simulaci diferenciálního řízení a stisknout tlačítko Connect. Poté zvolit zaškrtávací tlačítko Drive. Nyní je možné robota ovládat pomocí joysticku, který

je umístěn nahoře. Pokud by bylo ve scéně umístěno více robotů, lze ovládat pouze jednoho, a to výběrem konkrétního diferenciálního řízení.

Dále je možné se připojit k laserovému dálkoměru, a to podobným způsobem jako u výše uvedeného diferenciálního řízení. Dole se objeví grafický výstup těchto senzorů. Výstup je dostupný i ve vektoru hodnot, kde každá hodnota značí konkrétní vzdálenost v konkrétním bodě obrazu.

Pro účely této práce bylo přidáno zaškrtávací tlačítko Wandern. Při vybrání tohoto tlačítka se robot rozjede a začne se zcela autonomně vyhýbat překážkám. Používá k tomu oba senzory, které byly uvedeny výše. Pokud není připojen laserový dálkoměr, je robot odkázán pouze na dotykové senzory. Po kontaktu s překážkou se robot zastaví, natočí se na předem zvolenou stranu a pokusí se pokračovat v jízdě. Po připojení laserového dálkoměru již robot do překážek nenaráží. Pokud zaznamená překážku ve vzdálenosti menší než je předem stanovená konstanta, pokusí se jí vyhnout obdobným způsobem, jak bylo uvedeno u dotykových senzorů.

Přesněji funguje „Vandrující robot“ tak, že aplikace čte data ze senzoru. Při změně stavu dotykového senzoru se zavolá obsluha této události BumperUpdateNotification. V obsluze události se zjistí, zda je dotykový senzor stisknut. Pokud ano, je zavolána příslušná funkce pohybu. K pohybu se vytvoří událost OnMove, která má dva parametry, a to rychlost levého a pravého kola. K zastavení robota slouží událost OnEStop. Pokud je navíc aktivní laserový dálkoměr, dochází k neustálému čtení dat z tohoto senzoru. Pokud není některý z naměřených bodů menší než práh, pak je volána událost OnMove a dojde k pokračování směrem vpřed. Pokud dojde k detekci překážky, pak robot couvá. Navíc má při couvání každé kolo jinou rychlost, dochází tak navíc k mírnému vytáčení od překážky. Tato navigace funguje jen za předpokladu, že není stisknut dotykový senzor. Pokud stisknut je, dochází k pozastavení navigace pomocí laserové dálkoměru, dokud se robot od překážky neoprostí.

Dále pak byla přidána tlačítka A*, InitMap a Comunication, jejichž účel bude přesněji popsán v odpovídajících kapitolách.

Algoritmus řízení „Vandrujícího robota“

1. Jeď stále rovně
2. Pokud je aktivní laserový dálkoměr, sleduj stále vzdálenost nejbližší překážky
3. Pokud dotykové senzory zaznamenaly překážku, pak zastav a natoč se o 20º doprava. Poté proveď bod 1.
4. Pokud laserový dálkoměr detekuje překážku, pak zastav a natoč se o 20º doprava. Poté proveď bod 1.
5. Proveď bod 1

6 Tvorba mapy okolního prostředí

Mapou prostředí je myšleno zakreslení překážek, které na své trase robot potkává. V této práci jsou do bitmapy zakreslovány výstupy z laserového dálkoměru robota. Tento algoritmus je implementován v projektu SimpleDashboard v souboru DriveControl.cs.

Pro správnou tvorbu mapy je tedy nutné nejprve připojit k motoru robota odpovídající laserový dálkoměr. Jak simulace motoru, tak i laserový dálkoměr jsou spuštěny jako samostatné služby. Pokud by tedy bylo ve scéně umístěno více robotů, bylo by možné připojit motor jednoho robota a laserový dálkoměr druhého. To by samozřejmě vedlo k chybné tvorbě mapy a je tedy nutné vždy připojit sobě odpovídající služby.

Poté je již možné začít s vytvářením samotné mapy. Tvorbu mapy lze provádět automaticky, nebo manuálně. Pro automatickou tvorbu mapy stačí spustit ovládání založené na principu Vandrujícího robota. Robot se bude náhodně pohybovat v prostředí a případné překážky zakreslovat do mapy. Tato metoda je vhodná pouze do uzavřených scén, protože chování robota je velmi jednoduché. Pokud narazí na překážku, otočí se a vydá se jiným směrem. Pokud před sebou nebude mít žádnou překážku, bude udržovat kurz a opustí místo s překážkami. Je tedy vhodné, aby překážky „odrážely“ robota stále zpět.

Další možností je manuální tvorba. Stačí se pouze připojit k robotovi a pomocí joysticku v panelu Dashboard s ním pohybovat v simulačním prostředí. Lze tedy manuálně zmapovat ty části prostředí, které nás zajímají. Tento postup je naopak velmi výhodný v otevřených scénách. Pokud je však scéna rozsáhlá, je tato metoda časově velmi nevýhodná pro uživatele.

Je také možná kombinace obou výše uvedených řešení, a to nechat zmapovat okolí náhodně se pohybujícím robotem a zbývající části zmapovat manuálně.

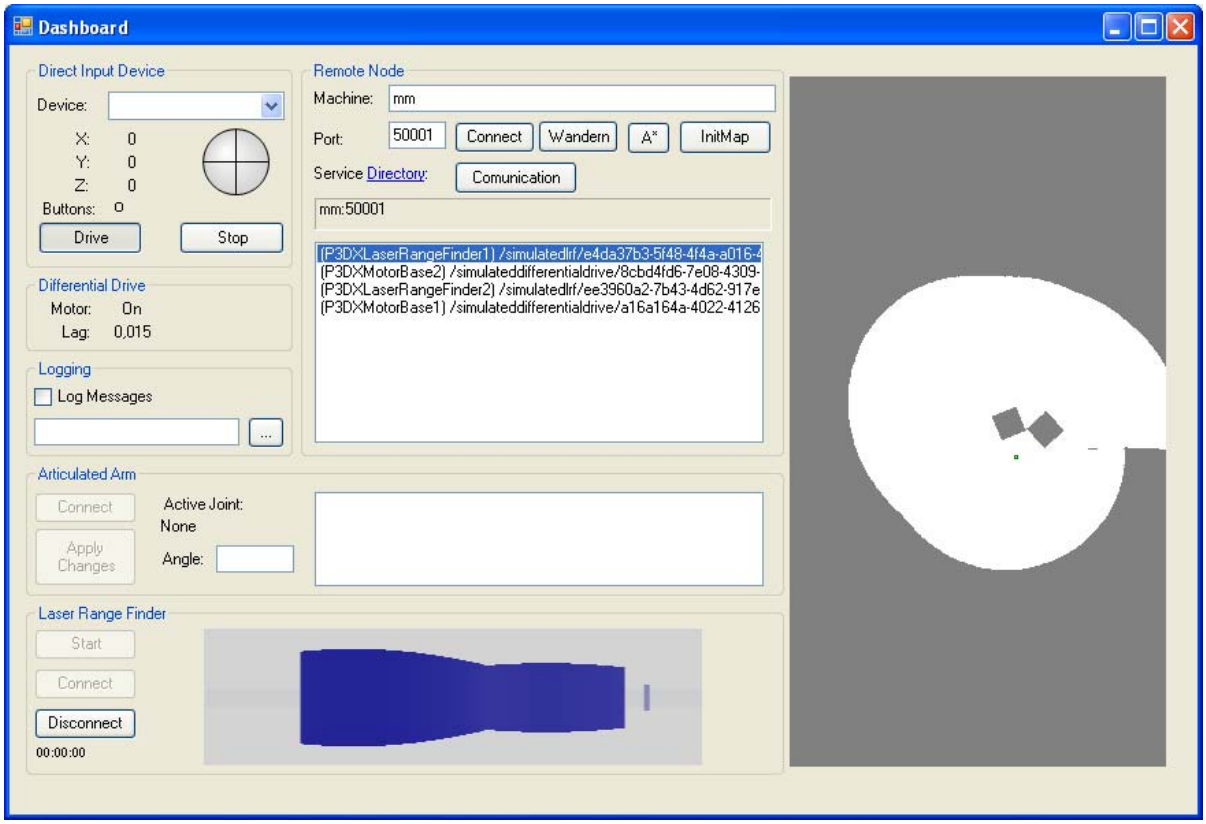
6.1 Algoritmus pro tvorbu mapy prostředí

K tvorbě mapy je nutné nejprve zjistit aktuální souřadnice robota, tedy jeho pozici a natočení. Pozici lze spočítat ze znalosti pozice jednotlivých kol. Pro souřadnici X a Y se spočte součet dvou souřadnic levého a pravého kolo a vydělí se dvěma. Takto lze získat přesný střed robota. Takto získanou pozici musíme dále přepočíst tak, aby byl střed souřadného systému uprostřed bitmapy, do které se mapa zakresluje. Ke každé souřadnici se přičte střed bitmapy. Natočení robota lze ze simulačního prostředí zjistit přímo.

Jako další krok je spuštěn cyklus, který postupně vykreslí do bitmapy jednotlivé paprsky laserového dálkoměru. Laserový dálkoměr má rozsah přibližně 90° na obě strany od středu robota. Známe tedy natočení paprsku vůči středu robota. Poté se úhel tohoto paprsku přepočítá na globální souřadnice, a to pomocí znalosti globálního úhlu natočení robota. Následně se pro každý paprsek

zakreslí do mapy úsečka, která má jeden bod ve středu robota a druhý bod je dopočten pomocí goniometrických funkcí sinus a cosinus. Přesněji od souřadnice x středu robota je odečtena hodnota délka * cos(úhel), kde délka je vzdálenost k překážce pro daný paprsek a úhel je globální natočení paprsku ve scéně. Pro souřadnici y jde o obdobný postup, ovšem k souřadnici y středu robota je zde přičtena hodnota délka * sin(úhel), kde jednotlivé proměnné značení ty samé hodnoty, jak již bylo uvedeno výše. Výsledná mapa pak může vypadat např. jako na Obrázku 17. Zde byly do scény vloženy dvě krychle, které jsou z mapy jasně patrné. Pozice robota je zde znázorněna malým čtvercem pod krychlemi. Jak je z Obrázku 17 patrné, lze tímto způsobem docílit velmi přesného zmapování prostředí. Každý pixel mapy odpovídá jednomu decimetru v simulačním prostředí, mapa tedy přesně odpovídá simulačnímu prostředí a je tedy možné podle ní přesně navigovat robota, což bude podrobněji popsáno v dalších kapitolách. Mapu je také možné vyčistit a začít s novým mapováním, a to pomocí tlačítka InitMap.

Je zde také možnost umístit do scény více robotů a tvořit mapu pomocí nich. Je to možné, ovšem v omezeném rozsahu. Všichni roboti budou zakreslovat do stejné bitmapy, ovšem vždy v daný čas pouze jeden z nich. Lze se mezi nimi přepínat a volit, který z nich bude tím aktivním. Nesmí však být zvolena sobě neodpovídající dvojice motor – laserový dálkoměr. To by vedlo ke špatné tvorbě mapy a taková mapa by byla posléze nepoužitelná.



Obrázek 17: Mapa okolního prostředí robota

7 Výpočet trasy pomocí algoritmu A*

V této práci je pro navigaci robota v prostředí využit algoritmus A*. Jako vstup mu slouží mapa, kterou je nutné vytvořit pomocí výše uvedené kapitoly. Tento algoritmus je implementován v projektu SimpleDashboard v souboru DriveControl.cs.

7.1 Algoritmus A*

Algoritmus A* [8] je nejznámější a nejpoužívanější metodou pro řešení úloh prohledáváním stavového prostoru. Nejprve je však nutné vysvětli algoritmus Best First Search. Jde o algoritmus založený na výběru nejlépe ohodnoceného stavu. V informovaných metodách musí cena cesty obsahovat i odhad ceny z uzlu *n* do uzlu cílového. Obecně pak hodnota ohodnocující funkce *f(n)* uzlu *n* je dána součtem dvou složek: ceny cesty *g(n)* z počátečního stavu do uzlu *n* a odhadu ceny cesty z uzlu *n* do uzlu cílového *h(n)*. Je zřejmé, že cena cesty v počátečním uzlu je nulová ($g(0) = 0$), a že nulová musí být i hodnota heuristické funkce v cílovém uzlu ($h(G) = 0$). Ceny všech přechodů musí být kladné - musí platit vztah $g(s_{n+1}) > g(s_n)$. Funkce *h(n)* se nazývá heuristickou funkcí, nebo krátce heuristikou. Čím přesnější je tato funkce, tím méně stavového prostoru je nutné prohledávat. V extrémním případě, pokud je heuristika přesným odhadem, tj. pokud vlastně známe řešení, tak samozřejmě nemusíme prohledávat žádný stavový prostor. Tuto metodu lze algoritmicky vyjádřit následovně:

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel včetně jeho ohodnocení.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s nejlepším (nejnižším) ohodnocením.
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému (vrací se posloupnost stavů, nebo operátorů). Jinak pokračuj.
5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky, kteří nejsou jeho předky, umísti do seznamu OPEN, a to včetně jejich ohodnocení. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponech pouze uzel s nejlepším ohodnocením, ostatní ze seznamu OPEN vyškrtni, a vrať se na bod 2.

A* je pak algoritmem Best First Search, jak byl popsán výše, kde heuristická funkce $h(n)$ musí být tzv. spodním odhadem skutečné ceny cesty od ohodnocovaného uzlu k cíli. Taková

heuristika se pak nazývá přípustnou heuristikou (resp. heuristickou funkcí). Metoda A* je úplná a pro přípustné heuristické funkce je i optimální.

Algoritmus A* expanduje pouze uzly *x,* pro které platí $f(s_x) \leq f_o$, kde f_o je cena optimální cesty. Časovou a prostorovou náročnost proto výrazně ovlivňuje použitá heuristika – pokud se blíží 0 (to je jistě spodní odhad skutečné ceny), pak složitost se blíží exponenciální složitosti, pokud je použitá heuristika dobrým spodním odhadem skutečné ceny, pak jsou expandovány pouze uzly kolem optimální cesty (je-li přesným odhadem, pak jsou expandovány pouze uzly na optimální cestě).

7.2 Implementace algoritmu A*

Nejprve musí být stanoveno cílové místo, kam se má robot za úkol dostat. Cílové místo se zadává kliknutím do pictureBoxu umístěném v ovládacím panelu Dashboard a musí být vybráno zaškrtávací tlačítko A*. Aby bylo cílové místo akceptováno, musí splnit několik podmínek:

1. Cíl je vzdálen v dostatečné vzdálenosti od překážky
2. Místo, kde je umístěn cíl, musí být zmapováno
3. Místo, kde je umístěn cíl, musí být dostatečně vzdáleno od okraje bitmapy, jinak bude uživatel varován, že zadal cíl mimo rozsah mapy

Pokud jsou splněny všechny výše uvedené podmínky, lze přistoupit k samotnému výpočtu trasy. Robot však nesmí být v tu chvíli v dotyku s nezmapovanou částí okolního prostředí. Musí kolem něj být dostatečně velký rozsah zmapovaného okolí, který odpovídá přibližně velikosti robota.

Výpočet začne tak, že se vytvoří množina, do které je umístěn startovací prvek. Ten obsahuje aktuální pozici robota a ohodnocení této pozice. Ohodnocení je spočteno jako součet odhadované vzdálenosti k cíli a počtu kroků, které bylo třeba vykonat, aby se robot do tohoto místa dostal. U startovacího prvku je počet kroků roven nule.

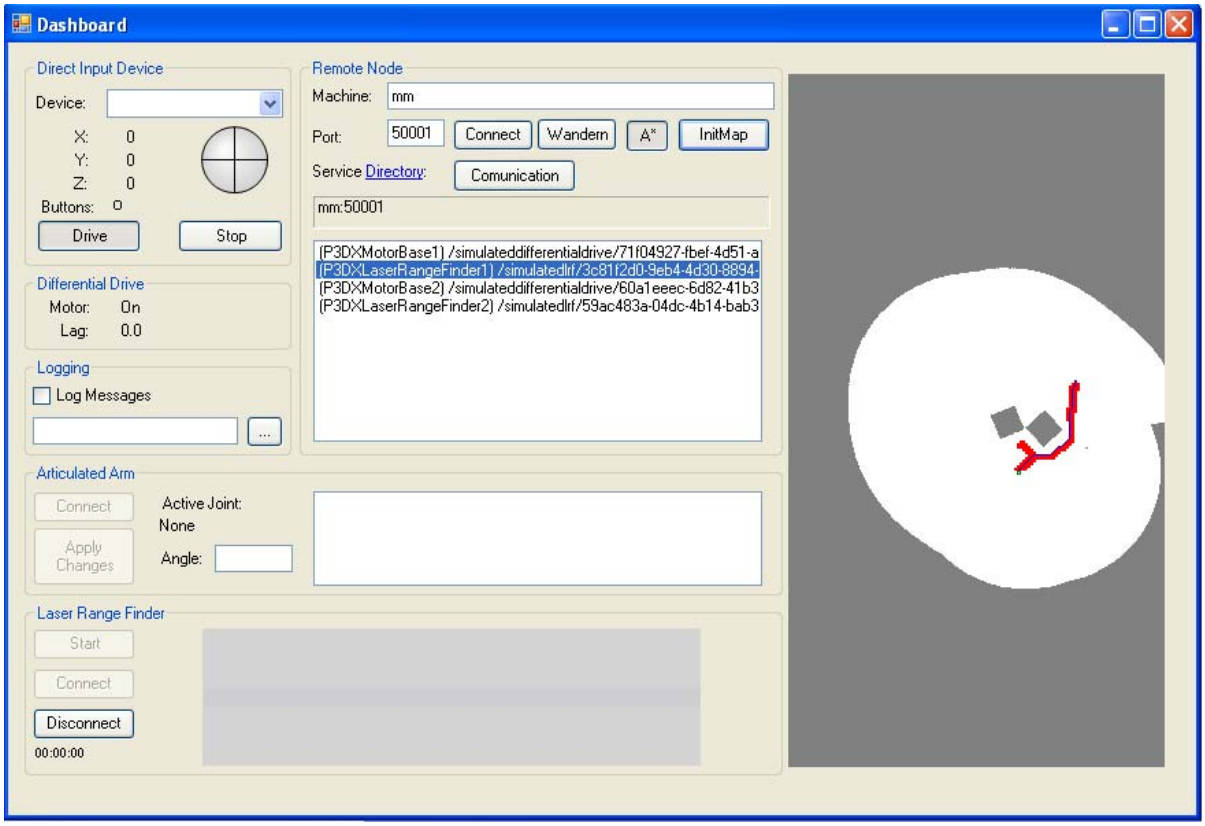
Po inicializaci je z množiny vybrán prvek s nejlepším (nejmenším) ohodnocením. Dojde k jeho rozgenerování ve všech směrech, tedy vezme se množina všech jeho prvků a každý z jeho potomků k ní přidá navíc jeden z okolních bodů. V této práci je využito 8 – okolí, které se pro praktické řešení úlohy hodí nejlépe. Poté rozgenerování je prvek s nejlepším ohodnocením z množiny odstraněn.

Každý z potomků však do množiny přidán není, protože se provádí kontrola několika vlastností tohoto prvku. Je nutné zjistit, zda jeho poslední přidaný bod v množině bodů není příliš blízko překážky nebo příliš blízko okraje bitmapy. V takovém případě prvek nebude přidán. Dále se zjistí, zda tento poslední přidaný bod není již obsažen v množině bodů tohoto prvku. Pokud ano, znamená to, že tento prvek cykluje a taktéž nebude do množiny přidán. Jako poslední krok je třeba zjistit, zda se již v množině prvků nevyskytuje jiný prvek, který by končil v tomtéž bodě. Pokud takový prvek existuje, bude ponechán ten z prvků, který má lepší ohodnocení.

Tento postup se bude stále opakovat, dokud nebude vzdálenost robota od cíle v rámci tolerance, nebo se množina s prvky nevyprázdní. První případ znamená, že se robot úspěšně přesunul k cíli do dostatečné vzdálenosti. Druhý případ značí, že se nelze do zadaného cíle dostat.

Během výpočtu dochází k vykreslování každého bodu, který je aktuálně rozgenerováván. Lze tedy na mapě pozorovat, jak probíhá výpočet trasy pomocí algoritmu A*. Po dokončení výpočtu je vykreslena nalezená optimální cesta, která je tedy dána množinou dvojic souřadnic x a y, které znamenají posloupnost bodů, kterými musí robot projet, aby se dostal do stanoveného cíle. Výsledek může vypadat např. jako na Obrázku 18, kde červeně je znázorněna množina bodů, která byla rozgenerovávána a modře pak nalezená optimální cesta.

Při umístění většího počtu robotů do scény lze zvolit zaškrtávací tlačítko Comunication. Poté budou před započetím výpočtu roboti spolu komunikovat a zjistí, který z nich je k cíli nejblíže. Ten, co je nejblíže, bude aktivován a právě on se pokusí dostat k cíli.



Obrázek 18: Nalezení cesty pomocí A*

8 Navigace pomocí trasy získané A*

Pojem navigace [5] značí souhrnný název pro postupy, pomocí nichž lze kdekoliv na zeměkouli, moři či ve vzduchu stanovit svou polohu (nebo polohu jiného přemisťovaného objektu). Dále pak nalézt nejvhodnější či nejkratší cestu k cíli.

U mobilních robotů navigace pokrývá obrovské spektrum různých systémů, potřeb a řešení. Je třeba pro různé situace volit vhodná řešení, tedy vhodný způsob navigace. Samozřejmě se bude lišit přístup při vývoji servisního robota pohybujícího se po předem daných drahách nebo vesmírného robota pracujícího samostatně ve zcela neznámém prostředí.

Fyzická měřítka se dají pro potřeby navigace definovat podle míry přesnosti potřebné pro daný typ aplikace robota. Můžeme tedy mluvit o rozlišení navigace. Potřeby přesnosti (rozlišení) jsou různé v různých aplikacích, prvním přiblížením měřítka navigace robota může být jeho velikost. Jiné potřeby bude mít venkovní robot pohybující se po silnicích a jiné bude mít chirurgický nanorobot pohybující se v krevním řečišti člověka. Každá autonomní mobilní jednotka musí být schopna korektně určit svoji pozici vzhledem ke zvolenému rozlišení.

Na konci měřítka jsou roboti, kteří mají velikost pouze několik centimetrů a pracující v relativně monotónním prostředí. Tito roboti budou potřebovat velmi přesnou navigaci na malé vzdálenosti (mimo jiné i kvůli omezenému zdroji energie). Na druhé straně jsou obří letadla a transatlantické tankery pohybující se v obrovských vzdálenostech v méně příznivých přírodních podmínkách, které potřebují přesnost na desítky metrů.

Definujme základní kategorie navigace podle měřítka navigace:

- globální navigace: určuje absolutní pozici robota v celosvětovém měřítku (případně i menším, dle potřeb) a dokáže ve své doméně navigovat robota do cílové pozice
- lokální navigace: určuje relativní pozici robota v prostředí ve vztahu ke stacionárním nebo mobilním objektům v jeho okolí a umožňuje korektní interakci (nejen) s těmito objekty
- osobní navigace: určuje polohu různých částí robota ve vztahu k sama sobě, ale i ve vztahu k okolnímu prostředí, a umožňuje manipulovat s objekty

8.1 Navigace založená na mapách

Navigace založená na mapách (někdy nazývaná „map matching“) je technika, v níž robot využívá svých senzorů k tvorbě mapy svého lokálního okolí. Poté se porovná lokální mapa okolí s globální mapou uloženou v paměti. Pokud je nalezena podobnost lokální mapy s některou částí mapy globální, pak lze stanovit polohu robota (musí však znát svoji polohu a orientaci v lokální mapě). Uložená globální mapa může být v některém standardním formátu nebo může mít libovolnou, námi

definovanou, formu. Důležité je, aby mapa umožňovala vyhledání podobnosti oblastí, tedy vyhledání lokální mapy robota v mapě globální.

Hlavními výhodami navigace založené na mapách jsou:

- využívá přirozené struktury okolí k určení globální pozice, nezasahuje tedy a nemodifikuje své okolí
- metodu je možné využít k aktualizaci mapy prostředí. Mapy prostředí jsou významné pro plnění úkolů, které robot má (například plánování cesty - případ překážek nezanesených v mapě).
- umožňuje robotovi naučit se znát své okolí a zlepšovat informaci o své poloze zkoumáním lokálního okolí

Mezi nevýhody tohoto typu navigace patří potřeba:

- dostatečného množství stacionárních objektů s jednoduše rozlišitelnými, unikátními příznaky vhodnými pro porovnávání s globální mapou
- dostatečné přesnosti senzorů k tomu, aby byl tento způsob vůbec použitelný (nepřesné senzory způsobí vytvoření nepřesné mapy, což znepřesní nebo znemožní lokalizaci, samozřejmě záleží také na požadovaném úkolu robota a na požadované přesnosti)
- dostatečného množství výpočetní síly k tvorbě mapy a jejímu porovnání s globální mapou (která též potřebuje dostatečnou kapacitu)

Problém tvorby mapy je velice úzce spjat se senzorickými schopnostmi robota. Tento problém lze tedy definovat tak, že je známa pozice robota a množina měření senzorů a nás zajímá, co vlastně senzory vidí.

Analýza pravděpodobnosti a měření chyby hrají velkou roli. Je nezbytné vzít v potaz určité množství nejistoty a zahrnout je do tvorby lokální mapy (například modelování chyb rozdělením pravděpodobnosti). Reprezentace mapy musí poskytovat způsob integrace nově zjištěných informací do stávající mapy. Také by měla poskytovat nezbytné informace nutné pro plánování cesty a vyhýbání se překážkám.

Základními kroky zpracování dat senzorů za účelem tvorby mapy jsou:

- extrakce příznaků z přímých senzorických dat
- fúze dat ze senzorů různých typů
- automatické generování abstraktního modelu prostředí

Porovnání lokální mapy s globálním vzorem je nejsložitější částí celého procesu navigace podle map. Obecně porovnání je docíleno extrakcí příznaků následovanou nalezením korespondence těchto příznaků s uloženým modelem (mapou).

Navigace založená na mapách využívá dva základní typy reprezentace map: geometrickou a topologickou. Geometrické mapy representují objekty absolutní polohou bodů objektu a jejich absolutní vzájemný geometrický vztah. Může to být mapa založená na mřížce nebo abstraktnější polygonová nebo čárová mapa.

Topologický přístup mapování je založen na popisu relativních vztahů příznaků objektů ve vztahu k určitému relativnímu rámci. Narozdíl od geometrických map mohou být mapy topologické vytvořeny a udržovány bez další specifikace polohy robota. Výsledkem tohoto přístupu je možnost integrace malých celků (lokálních map) do jediného velkého celku, neboť všechny vztahy jednotlivých významných uzlů jsou relativní a je možné je vzájemně spojit.

Navigace založená na mapách se stále vyvíjí. V současnosti je tato technika využívána spíše v laboratořích výzkumných skupin než v praxi. Dobré výsledky podává tento druh navigace v případě, že je prostředí robota kvalitně strukturované a dobře analyzovatelné. Zatím je těžké určit, jak se bude chovat robot vyvinutý v laboratoři v reálném prostředí. Souhrnná charakteristika navigace podle map je následující:

Potřebuje dostatečný výpočetní výkon a dostatečný počet kvalitních senzorů (tedy kvalitní senzorická měření).

Zpracování dat může být velmi náročné v závislosti na použitém algoritmu.

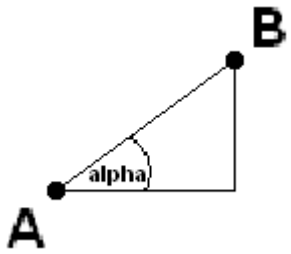
Je nezbytné přesně stanovit počáteční polohu robotu (například odometricky), abychom redukovali počáteční chybu určení polohy a tím i následující možné chyby.

Kritický je výběr senzorů, přesnost a spolehlivost algoritmů a dobrý model chyb senzorů a pohybu robota. Na tyto oblasti výzkumu je potřeba se zaměřit, aby bylo možné navigace podle map prakticky využít.

8.2 Řešení navigace v této práci

Navigační algoritmus je implementován v projektu SimpleDashboard v souboru DriveControl.cs. Jako vstup je dán navigačnímu algoritmu seznam bodů, kterými musí robot projet. Úlohu lze tedy rozdělit na konečný cyklus, kde se v jednotlivých cyklech snažíme z bodu A (aktuální pozice robota) dostat do bodu B. V dalším kroku se z bodu B stane bod A a nový bod B bude získán jako další prvek z množiny bodů. Cesta k cíli je tedy složena z úseček. Je tedy nezbytné vždy robota natočit tak, aby směřoval na požadovaný bod. Dále dojde k vypočtení vzdálenosti k cíli. Pak je již robot vyslán, aby ujel zadanou vzdálenost.

První část problému, a to natočení robota směrem k cíli, je řešena pomocí goniometrické funkce tangens. Známe dva body A a B. Každý bod se skládá ze dvou složek, a to x a y, kde např. A.x značí x-ovou složku bodu A. Úhel, který svírají dva body označíme alpha (viz Obrázek 19). Pak lze úhel alpha vypočíst jako alpha = arctg(|B.y – A.y| / |B.x – A.x|). Poté již zbývá pouze zjistit, ve kterém kvadrantu se tato úsečka nachází a přepočítat úhel do globálních souřadnic.



Obrázek 19: Úhel alpha

Samotné natočení na cíl je řešeno tak, že se robotu zašle požadavek, aby rotoval na tu stranu, kde rychleji dosáhne požadovaného úhlu. Následně se provádí kontrola, zda již dosáhl požadovaného natočení.

Výpočet vzdálenosti, která je mezi body A a B, se řeší pomocí rovnice vzdálenost = Odmocnina($(B.y - A.y)^2 + (B.x - A.x)^2$). Pak se již zadá robotovi vypočtená vzdálenost, kterou má urazit.

Pokud by během cesty po předem definované trase robot narazil na překážku, zastaví se a zneplatní mapu. Znamená to, že ve zmapované oblasti došlo k přesunu některé z překážek a je tedy nezbytné znovu vypočítat trasu k cíli.

9 Popis aplikace

K tvorbě aplikace bylo využito Visual Studia 2005 a programovacím jazykem byl zvolen C#. Dále pak s využitím API Microsoft Robotics Studio byla vytvořena scéna a probíhala zde simulace robotů.

9.1 Soubory programu

Aplikace je tvořena osmi samostatnými projekty. Prvním z nich je projekt uložený v adresáři Bumper. Tento projekt je využit k získávání stavů dotykového senzoru. Definuje třídu SimulatedBumperService, kde jsou uloženy metody nezbytné ke správnému ovládání těchto senzorů.

DifferentialDrive je projekt sloužící k řízení motoru. Pro potřeby této práce byl mírně upraven, aby bylo možné získat pozici a natočení robota ve scéně. Definuje třídu SimulatedDifferentialDriveService, která obsahuje metody pro nastavování parametrů a ovládání motoru.

LaserRangeFinder je využit k simulaci laserového dálkoměru. Obsahuje třídu SimulatedLRFService, která poskytuje metody pro navázání spojení a získávání dat z laserového dálkoměru. Využívá raycastingu, tedy vyšle paprsek do scény a zjistí, jak daleko je překážka pro každý konkrétní bod rozsahu .

Mapováni prostoru je hlavní projekt, který vychází ze simulačního tutoriálu číslo 2. Pomocí něj je vytvořena scéna a potřebné servisy pro komunikaci s robotem a jeho senzory. Více informací viz [2].

SimpleDashboard slouží ke komunikaci s robotem a zobrazuje i výstup laserového dálkoměru. Pro potřeby této práce byl doplněn tlačítky ke spuštění vandrujícího robota, spuštění navigace pomocí A* a ke komunikaci mezi roboty. Dále pak je zde umístěn pictureBox, kde je možné sledovat jak vytvořenou mapu, tak i nalezené řešení A*. Obsahuje třídu SimpleDashboardService, která obsahuje metody jak pro řízení motoru, tak i pro získávání událostí ze senzorů. Dále pak třídu DriveControl, která zajišťuje ovládání joysticku, výpočet A^* a navigaci. Je zde uložen i samotný vzhled ovládacího panelu Dashboard.

SimulatedWebcam obsahuje třídu simulatedWebcamService, jejíž metody jsou využity k přístupu k webové kameře.

Posledním projektem je WebCam, který slouží pouze k zachytávání obrázků z webové kamery

Každý z těchto projektů je klasickým projektem ve Visual Studiu. Obsahuje tedy zdrojové texty, soubory ke spuštění projektu, již přeložené soubory atd. Zdrojové soubory pak obsahují definici tříd a samotné ovládání. Je tedy možné kterýkoliv z projektů samostatně zkompilovat a spustit.

9.2 Popis ovládání

K úspěšnému zpuštění je třeba nahrát celou aplikaci nazvanou Mapování prostoru (obsahuje všech osm výše zmíněných projektů) do rootovského adresáře Microsoft Robotics Studia. Dále pak nahrát vzhled robota (RobotSoccer.obj) do adresáře /store/media. Nyní je možné aplikaci spustit dávkovým souborem (MapovaniProstoru.bat), který ovšem musí být nahrán také v rootovském adresáři.

Po spuštění aplikace se objeví celkem tři okna. Prvním z nich je command line okno. Do něj jsou vypisovány aktuální stavy výpočtů. Po zadání cíle do něj robot vypisuje stavy, ve kterých se nachází (zda počítá A^* nebo se již snaží dostat k cíli, popřípadě zda již cíle nedosáhl). Chybová hlášení jsou rovněž vypisována do tohoto okna.

Další okno zobrazuje aktuální stav scény. Je zde tedy vidět např. pohyb robota, který se snaží dosáhnout zadaného cíle. Je možné se scénou pohybovat s využitím šipek na klávesnici a pomocí myši. Scénu lze také editovat vybráním položky Mode a následně Edit. Lze zde měnit vzhled, pozici a rotaci všech entit. Lze také entity přidávat a mazat. V tomto okně je možné přepnout kameru volbou Camera v menu okna, pokud je jich ve scéně umístěno více (typicky hlavní kamera, kamera robota).

Posledním oknem, které se po spuštění zobrazí, je Dashborad. Slouží k ovládání robota a zadávání úkolů. Taktéž zobrazuje aktuální stav mapy, popřípadě výsledek výpočtu navigace pomocí A^*. Nejprve je třeba zadat stroj (Machine). Pokud pracujeme v rámci jednoho počítače, pak zde můžeme zadat localhost a zvolíme Connect. Nyní se o něco níže vypíší všechny služby pro ovládání/získávání dat z motorů a laserových dálkoměrů. Poté se připojíme k jednomu z motorů, který odpovídá konkrétnímu robotovi. Poté lze po zaškrtnutí Drive ovládat robota přímo pomocí joysticku. Můžeme také připojit laserový dálkoměr, který však musí odpovídat danému robotovi. Při připojení k laserovému dálkoměru jiného robota, než je momentálně připojen, by aplikace nepracovala správně. V této práci je každá dvojice motor – laserový dálkoměr jednoznačně rozlišena stejným číslem na konci jejich názvu. Pokud se chceme připojit k jinému robotovi, musíme dodržet následující postup:

- odpojit se od stávajícího laserového dálkoměru
- připojit se k motoru námi zvoleného robota
- připojit se k odpovídajícímu laserovému dálkoměru

Tlačítko Wandern zde slouží ke spuštění algoritmu vandrujícího robota, A^* ke spuštění navigace pomocí tohoto algoritmu, InitMap k inicializaci mapy a komunikace k nalezení nejbližšího robota u cíle.

Pokud je zaškrtnuto tlačítko A^*, dojde po kliknutí do pictureBoxu k výpočtu trasy pomocí algoritmu A^* a následně k navigaci k cíli. Dokud robot nedorazí do cíle, není možné jej nijak ovlivnit a ovládací panel Dashboard se bude jevit neaktivní.

Podrobnější informace k ovládání této aplikace budou umístěny na přiloženém CD a budou součástí uživatelské příručky.

10 Závěr

Vlastní přínos této práce je v návrhu a implementaci robota. Dále pak volba vhodného rozhraní, pomocí kterého robot v současné době komunikuje a lze jej přes toto rozhranní řídit. Je vytvořen i jednoduchý algoritmus „Vandrujícího robota“. Ten je schopen se samostatně pohybovat v terénu a řešit střety s překážkou. Popřípadě se překážce zcela vyhnout pomocí laserového dálkoměru.

Dále byl implementován algoritmus A^*. Byla také navržena a implementována navigace robota na základě výsledků získaných právě algoritmem A^*. Je možné i využití většího počtu robotů umístěných ve scéně.

Tato práce měla hlavně ukázat případným zájemcům, jak se pracuje s robotickým studiem, jaká jsou úskalí. Jsou zde vyřešeny matematické aparáty pro tvorbu mapy a navigaci robota. Dále je zde vidět, jakým způsobem lze napojit algoritmus A^* do praktické navigace. Do budoucna by tedy bylo vhodné např. rozvinout grafické uživatelské rozhranní a dát uživateli možnost nastavit více parametrů. Dále pak naimplementovat sofistikovanější způsob řízení, vyzkoušet například neuronovou síť. Také by bylo vhodné, aby si každý robot tvořil lokální mapu svého okolí.

Je také možná návaznost na bakalářskou práci Petra Binka, který se zabýval aplikací, která je schopná vymodelovat scénu a uložit ji do formátu XML, který lze přímo načíst z Microsoft Robotics Studia. Poté je možné pomocí této práce nechat robota mapovat takové prostředí a nechat jej navigovat na cílová místa.

U této práce byly splněny všechny body zadání. Tato práce je rovněž vhodná k demonstračním účelům ve výuce. Je zde možné demonstrovat výpočet A^* popřípadě demonstrovat autonomního robota.

Literatura

[1] Robot, 2008. Dokument dostupný na URL http://cs.wikipedia.org/wiki/Robot

[2] Microsoft Robotics Studio, 2007. Dokument dostupný na URL http://msdn2.microsoft.com/en-us/robotics/default.aspx(leden 2008).

[3] Tkadlčík, M. Mr. Blender, 2003. Dokument dostupný na URL http://www.sweb.cz/blender/ (leden 2008).

[4] Kab. Simulace usnadňuje programování robotů, 2006. Dokument dostupný na URL http://www.automatizace.cz/article.php?a=1024 (leden 2008).

[5] Orság, F. Robotika – studijní opora, 2006. Dokument dostupný na URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ROB-IT/texts/ROB-StudijniOpora.pdf (leden 2008).

[6] Šonka, M., Hlaváč, V., Boyle, R. Image processing, analysis, and machine vision. New York, PWS Publishing, 1999.

[7] Drahanský, M. Senzory – 1. přednáška, 2006. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/SEN/private/01_SEN_Prednaska.pdf (leden 2008).

[8] Zbořil, F. Základy umělé inteligence – studijní opora, 2006. Dokument dostupný na URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IZU-IT/texts/oporaIZU-ESF-4.pdf (duben 2008).

Příloha A: Obsah datového CD

Na datovém CD jsou uloženy následující soubory:

- tato práce
- programová dokumentace k aplikaci
- zdrojové soubory, které jsou napsány s využitím Microsoft Visual Studia 2005 v jazyce C#. Dále je využito Microsoft Robotics Studia.