VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DIPLOMOVÁ PRÁCE

Model robota Trilobot

2007 Miroslav Štěpán

Model robota Trilobot

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Hrubého, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Rád bych touto cestou poděkoval Ing. Martinu Hrubému, Ph.D. a Ing. Vladimíru Janouškovi, Ph.D. za odbornou pomoc při tvorbě této práce.

…….

V Brně, dne 19. 1. 2007 Miroslav Štěpán

Abstrakt

Tato diplomová práce popisuje tvorbu modelu pohybu mobilního robota Trilobot, který je následně implementován do jednoduchého simulačního nástroje. Jsou zde popsány experimenty provedené s tímto robotem, stručně je popsán nástroj SmallDEVS s jehož využitím je model v prostředí Squeak Smalltalku implementován.

Motivací této práce je zjednodušení návrhu a testování navigačních algoritmů pro robota Trilobot, který je studentům FIT VUT v Brně k dispozici v robotické laboratoři ÚITS. Tvorbou jednoduchého simulačního nástroje by se tedy dala i částečně snížit závislost na fyzické dostupnosti tohoto robota.

Klíčová slova

mobilní robot, robotika, Trilobot, Player/Stage, Gazebo, Squeak, Smalltalk, SmallDEVS, DEVS, model, simulace, odometrie, navigace

Abstract

This diploma thesis describes creation of motion model of mobile robot called Trilobot. This model is implemented into simple simulation tool. Some laboratory experiments with the robot are described in this paper. There is also some information about SmallDEVS tool and Squeak Smalltalk environment in which the model was implemented.

Motivation of this work is effort to simplify the design and testing of navigation algorithms for Trilobot, which is available for students of FIT BUT in the robotics lab of department of intelligent systems. This simple simulation tool could partially reduce dependence on physical availability of this robot.

Key words

mobile robot, robotics, Trilobot, Player/Stage, Gazebo, Squeak, Smalltalk, SmallDEVS, DEVS, model, simulation, odometry, navigation

Obsah

1. Úvod

V roce 2004 vznikla v rámci grantového projektu FRVŠ MŠMT Robotická laboratoř Ústavu inteligentních systémů FIT VUT v Brně vybavená mj. 10 roboty Trilobot. Jedná se o mobilní roboty určené především pro výukové aplikace a výzkum navigačních algoritmů. Navigace Trilobota je však poměrně nepřesná a tvoří tak překážku při řešení navigačních úloh, kterými je nutné se zabývat na nižší úrovni než by bylo třeba u spolehlivějších mobilních robotů. Protože se nelze spolehnout na přesnost navigované dráhy, bylo by výhodné, kdyby se dal pohyb Trilobota simulovat. Simulace by umožnila rychlejší a snadnější návrh a testování řídících a navigačních algoritmů, protože dovoluje testovat algoritmus v podmínkách, jejichž navození by v reálném světě vyžadovalo časové, prostorové nebo finanční prostředky, které nemusí být k dispozici. K tomu je ale nejdříve potřeba vytvořit simulační model použitelný ve vhodném simulačním prostředí. Právě tvorba takového modelu je cílem tohoto diplomového projektu.

Při modelování je nejdříve nutné důkladně prozkoumat Trilobota, zejména se zaměřit na jeho pohonný systém, aby bylo možné přesněji určit příčiny chyb. Podle analýzy chování lze poté sestavit abstraktní model, který je možné aplikovat do nějakého simulačního prostředí. V oblasti robotiky existuje několik zajímavých simulačních prostředí z nichž nejznámější je nástroj Player/Stage.

Analýza tohoto prostředí byla předmětem předcházející semestrální práce. I když jsem shledal, že je možné model Trilobota v Player/Stage realizovat, komplikovanost řešení, které by umožňovalo využít výhod tohoto prostředí by byla zbytečně velká. Jako schůdnější řešení se tedy nakonec jeví vytvoření vlastního, jednoduššího simulátoru určeného speciálně pro simulaci pohybu Trilobota. Jako návrhové prostředí pro tento úkol jsem zvolil Squeak Smalltalk, jehož koncepce v podstatě odstraňuje závislost na platformě a zajišťuje dokonalou perzistenci objektů, což je pro simulace velice výhodná vlastnost. Navíc jsem v tomto prostředí mohl využít poměrně nového nástroje SmallDEVS určeného pro podporu simulací ve Smalltalku.

Úvodní stránky této práce nás lehce uvedou do oblasti robotiky. První kapitola se věnuje historii tohoto oboru a jednoduchému rozdělení robotů abychom si mohli udělat představu o obsáhlosti robotiky. Následující kapitola se zaměřuje na Trilobota samotného, popisuje zejména jeho pohonný systém. Třetí kapitola navazuje na poznatky z předcházející semestrální práce. Jsou zde shrnuty informace o simulačních nástrojích Player/Stage, prostředí Squeak Smalltalk a popis formalismu DEVS. Je zde také stručně popsána smalltalkovská implementace podporující modelování a simulaci na základě

tohoto formalismu – SmallDEVS, jejímž autorem je Ing. Vladimír Janoušek Ph.D. V závěrečných kapitolách jsou již shrnuty informace získané při řešení tohoto diplomového projektu, výsledky laboratorních experimentů s Trilobotem a popis vývoje abstraktního a simulačního modelu. V poslední kapitole je popsána výsledná implementace využívající tohoto modelu, jednoduchý simulátor pracovně nazvaný TriloSim.

2. Úvod do robotiky

Vzhledem k tomu, že dosud neexistuje ustálená definice tohoto pojmu, nebudu se pokoušet tento obor příliš formálně vymezovat. Obdobně je to i se slovem robot. Oba pojmy byly poprvé použity ve vědecko-fantastické literatuře. Slovo robot poprvé zaznělo ve hře R.U.R. Karla Čapka v roce 1920, když jím pojmenoval uměle vytvořenou bytost. Slovo robotika použil asi o dvacet let později Isaac Asimov v povídce Runaround - i když v trochu jiném významu než ho chápeme dnes. **Robotika** by se dala popsat jako obor zabývající se mj. konstrukcí a studiem robotů, **robot** zase jako stroj, jehož účelem je usnadnit člověku práci (zejména mechanickou), případně ho v tomto smyslu nahradit. V češtině bylo slovo robot původně skloňováno podle neživotného vzoru *hrad* (množné číslo *roboty*), ale později se začalo (zejména pro inteligentní, člověku podobné roboty) používat skloňování podle životného vzoru *pán* (množné číslo *roboti*).

2.1. Historie robotiky

Historie robotiky - i když tak ještě nebyla nazývána - by se dala datovat i do doby před průmyslovou revolucí. Do dob, kdy byly vytvořeny první mechanické stroje napodobující zvíře (zooidy), nebo později člověka (androidy). Tyto výtvory však byly pouze určitou atrakcí, jejich praktický přínos byl v podstatě nulový.

První patent z oblasti robotiky byl podán roku 1954 společností Unimation, která vyrobila historicky prvního průmyslového robota. Tento robot byl určen k jednoduché manipulaci s objekty.

Dnes už robotika zažívá podobný rozmach jako svět počítačů před třiceti lety. Stejně jako se stala realitou vize osobních počítačů, i moderní výdobytky robotiky již nejsou daleko od představ autorů science-fiction. Robotika je multidisciplinární obor, využívající poznatky z elektrotechniky, strojního inženýrství, kybernetiky, počítačové grafiky, umělé inteligence, genetického programování a jiných oblastí.

Vývoj elektroniky jde velkým tempem vpřed, senzory jsou díky novějším výrobním procesům přesnější, levnější a technologicky vyspělejší. Ke slovu přichází i výpočetní technika. Díky výpočetnímu výkonu, kterého je moderní hardware schopen je možné roboty programovat pomocí vysokoúrovňových programovacích jazyků. Roboty jsou schopny zpracovávat obrovské množství informací ze senzorů, analyzovat obrazová data a uplatňovat složité algoritmy z umělé inteligence.

Robotika už není obor kterým se zabývaly pouze specializované vědecké laboratoře nebo strojírenské firmy. Robotika se dostala i do komerční sféry. Existuje mnoho společností, vědeckých laboratoří i nadšenců zabývajících se vývojem a stavbou robotů určených pro zábavu, jednoduché domácí práce ale i speciální aplikace kde také například nahrazují člověka. Existují vyspělí roboti věrně napodobující člověka i robotické hračky nahrazující některá domácí zvířata. Roboty se dostaly i do vesmíru, některé nám zasílají informace získávané na jiných planetách, jiné se dostaly už i mimo naši sluneční soustavu.

Popularizace robotiky roste nejen díky představivosti autorů science-fiction, ale také díky mnoha robotickým soutěžím. Mezi těmito soutěžemi je nejznámější asi DARPA challenge, která je zaměřena na testování zpracování obrazu, strojového učení a jiných problémů nutných k úspěšnému zvládnutí navigačních úkolů. Hlavní zkouškou soutěže je průjezd 142 mil dlouhým úsekem pouště Mojave. Na posledních ročnících této soutěže byl viditelný znatelný nárůst vývoje v této oblasti. Zatímco v roce 2004 nebyl schopen úkol splnit jediný robot, následující rok již v této obtížné zkoušce uspělo 5 strojů, z nichž nejúspěšnější byl schopen bezpečné jízdy rychlostí přes 30km/h.

Budoucnost robotiky tedy tkví nejen v rozvoji elektrotechniky a jiných „podpůrných“ věd, ale i v popularizaci této zajímavé oblasti. V počítačovém světě jde vývoj kupředu stále větším tempem zejména díky soutěživosti (resp. snaze o ovládnutí trhu) největších rivalů, u robotiky tento čas právě nastává.

V současnosti jsou mezi nejznámějšími vývojáři v této oblasti zejména Japonské firmy jako Sony nebo Honda známá svým vyspělým humanoidním robotem Asimo. Mezi povědomí lidí, kteří se o tento obor příliš nezajímají se v poslední době dostala například i společnost iRobot vyrábějící populární „uklízecí“ roboty řady Roomba, ale i specializované roboty do terénu používané například při odstraňování nastražených výbušnin.

2.2. Rozdělení robotů

V tomto rozdělení nejde o rozbor všech možných druhů robotů, ale spíše o přehled nejznámějších kategorií a tříd robotů aby bylo možné se podívat na Trilobota v kontextu moderní robotiky.

Kdybychom chtěli roboty rozdělit podle účelu, mohli bychom použít následující označení (i když ne všechny roboty bychom mohli do těchto tříd zařadit):

Droid – inteligentní samočinný robot

Android – robot věrně napodobující člověka jak vzhledem, tak chováním

Zooid – robot napodobující zvíře

Humanoid – robot podobný člověku principielní stavbou těla (tělo, ruce, nohy) a zejména způsobem pohybu (chůze po dvou nohách)

Kyborg (kybernetický organismus) – (zatím) pouze záležitost science-fiction, jedná se o androida s implantovaným mozkem živé bytosti

Manipulátor (průmyslový robot) – dálkově řízený stroj bez vlastní inteligence

Roboty lze dále dělit na **mobilní**, které se mohou přemísťovat a **stacionární**, které mají uplatnění hlavně v průmyslu. Mobilní roboty se podle způsobu pohybu rozdělují na kráčející roboty a roboty s kolovými nebo pásovými podvozky. Nejrozšířenější kategorií mobilních robotů jsou roboty s kolovými podvozky dále dělitelné na:

Diferenciální podvozek – dvě samostatně hnaná kola a podpůrné kolo/kola nebo podpěry, nejčastěji užívaná platforma mobilních robotů, do této kategorie spadá i Trilobot

Synchronní podvozek – typicky se třemi koly, každé kolo má dva stupně volnosti (může se otáčet i natáčet)

Trojkolový podvozek – dvě hnaná kola, přední pouze natáčené

Ackermanův podvozek – 4 kola, z toho dvě hnaná a dvě (zpravidla přední) natáčená – podvozek známý z automobilů

Podvozky se všesměrovými koly – kola umožňují pohyb ve dvou osách protože mají po svém obvodu řadu pasivních válců, obvykle v trojúhelníkovém uspořádání

2.3. Navigace mobilních robotů

Navigace zahrnuje postupy určování polohy (lokalizace) a nalezení nejvhodnější cesty k cíli. Techniky navigace čerpají z dávných znalostí i nejnovějších technologií. Použití konkrétní navigační metody závisí zejména na měřítku navigace, které určuje stupeň její přesnosti. Podle měřítka jsou navigace rozděleny do kategorií:

Globální navigace – postačuje nejhrubší měřítko navigace, tedy v rámci celého prostředí – světa. Slouží pro navedení robota do cíle na základě určení absolutní pozice.

Lokální navigace – vztahuje se k blízkému okolí robota, na základě určení relativní pozice pomáhá například při vyhýbání se překážkám.

Osobní navigace – slouží například k bezpečnému navádění manipulátorů nebo jiných částí robota, měřítko této navigace je určeno rozměry robota.

Narozdíl od lokální a osobní nemá navigace v globálním měřítku pro Trilobota vzhledem k jeho pohybovým možnostem smysl (Trilobot je schopen přemisťovat se v řádu metrů, ne kilometrů).

Aby bylo možné robota navigovat, musí být určený také **referenční bod navigace**. Veškerá lokalizace je prováděna ve vztahu k tomuto bodu. Proto je rozumné tento bod volit vzhledem k typu navigace – u osobní navigace má smysl jako referenční bod definovat nějaký pevný bod robota, u lokální zase nějaký pevný bod v jeho okolí.

2.3.1. Dead-reckoning a Odometrie

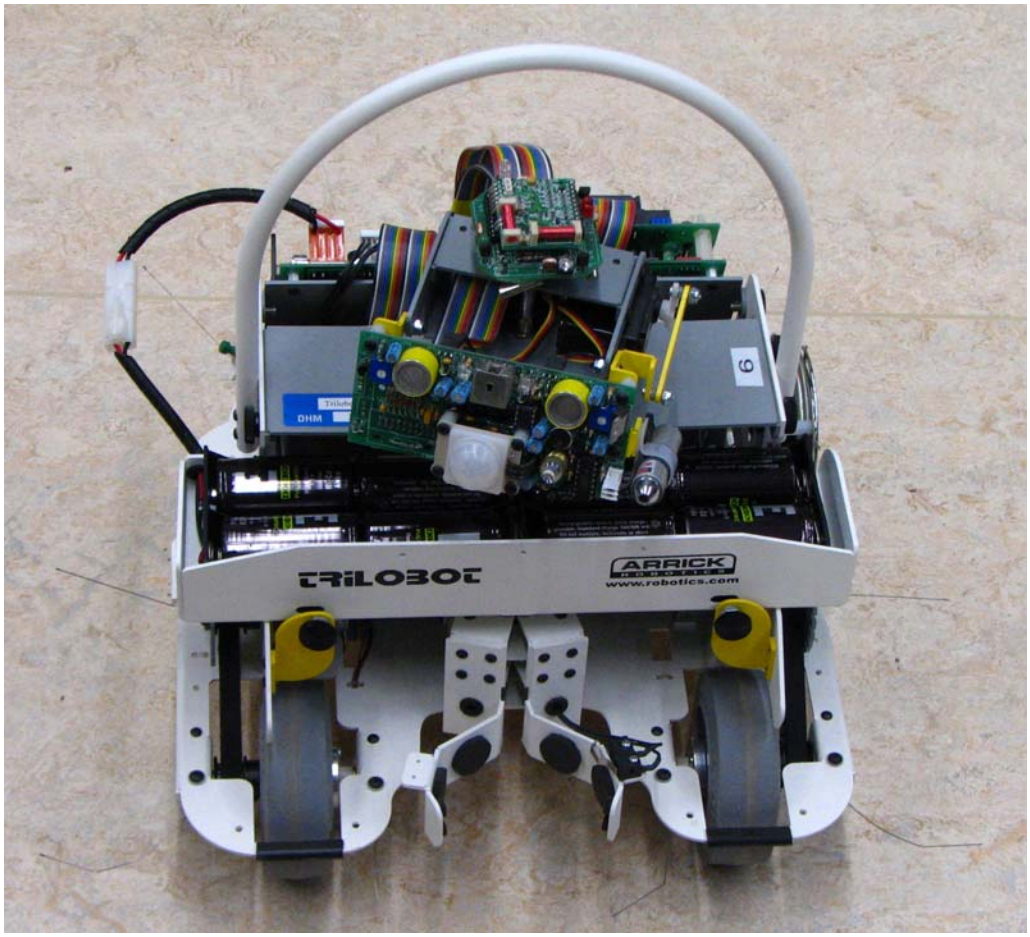
Dead-reckoning (*DR, deduced reckoning*) je nejstarší metoda navigace užívaná dříve zejména pro navigaci na moři. Aktuální poloha je vypočítána na základě známé (nebo předpokládané) výchozí polohy, směru a rychlosti pohybu v daném čase. Nevýhodou tohoto způsobu navigace je zejména akumulace chyby, proto je vhodné používat i jiný způsob lokalizace, aby bylo možné upřesnit polohu bodu z kterého tento výpočet vychází.

Nejjednodušší aplikací tohoto způsobu navigace je **odometrie**. Data potřebná pro výpočet aktuální polohy získáváme v tomto případě prostřednictvím *odometrů*, které zaznamenávají délku ujeté trasy. Nejpoužívanější formou odometru je optický rotační enkodér spojený s koly robota. Nejjednodušší variantou tohoto senzoru je inkrementální odometr, který získává údaje pouze přičítáním impulsů generovaných při otočení kola enkodéru o jednotlivé kroky. Není tedy možné určit absolutní pozici (resp. konkrétní úhel natočení) kola enkodéru. Složitější variantou je absolutní odometr, který umožňuje získat informaci o konkrétním natočení kola.

Přesností fungování odometru je dána také chyba odometrie. Například při použití klasických inkrementálních optických enkodérů není možné detekovat podklouznutí podvozkového kola a vzniká tak rozdíl mezi změřenou a skutečně ujetou vzdáleností. Chyby odometrie se dají rozdělit na systematické a nesystematické. **Systematické chyby** mají vesměs deterministickou povahu (např. průměr kol neodpovídající hodnotě s kterou se počítá), takže odhalením jejich příčiny by mělo být možné je eliminovat. Jejich příčina ale může mít i jinou povahu – důvodem chyby může být také například malá rozlišovací schopnost enkodérů. Oproti tomu **nesystematické chyby** mají zcela nahodilý charakter (např. podklouznutí kola vlivem nerovnosti povrchu apod.).

3. Stručná charakteristika Trilobota

Jedná se o mobilního robota určeného především pro výukové aplikace, testování navigačních algoritmů a výzkum robotiky a umělé inteligence. Výrobcem je americká společnost Arrick Robotics, která ho vyráběla zhruba od poloviny devadesátých let až do roku 2006. Během této doby Trilobot [7] prošel inovací, resp. rozšířením senzorického vybavení a doplněním jednoduché mechanické ruky. Jak je na webových stránkách této firmy vidět, nejedná se o velkou společnost, ale o skupinu několika málo studentů a dalších lidí, kteří v podstatě v domácích podmínkách sestavují „stavebnicové“ roboty, jejichž autorem je zakladatel firmy – Roger Arrick. Tím by se dala vysvětlit konstrukce Trilobota. Nejedná se o výsledek práce vývojového týmu nějaké firmy, ale v podstatě o výtvor robotického nadšence, který ho úspěšně uvedl do hromadné produkce. Trilobot se dostal do univerzitních výzkumných laboratoří, po své inovaci byl úspěšně prodáván i na mnohé střední školy ve Spojených Státech Amerických. Jeho platforma byla využita například i při zajímavé aplikaci naváděcího audio systému na univerzitě v Torontu [9].



Obrázek 3.1: *Čelní pohled na Trilobota s bateriovým držákem.*

3.1. Stavba robota

Robota je možné systematicky rozdělit na tři části. První je podvozek uložený v pevném ocelovém plechu. V jeho přední části je uprostřed „vykrojený“ prostor pro jednoduchou mechanickou ruku (nejedná se o napodobeninu lidské ruky, ale o jednoduchý zachycovač umožňující v určitém rozsahu sevřít čelisti a mírně zvednout uchopený objekt). Střední část robota je tvořena převážně řídící elektronikou, obsahuje i odnímatelný bateriový držák. Vrchní část robota se skládá z otočné hlavy a nepohyblivého „můstku“ na nichž je umístěna většina senzorů. Pokud nebudeme brát v úvahu dotykové senzory umístěné po obvodu podvozkové části, zabírá robot prostor o rozměrech cca 30 x 30 x 30cm. Jeho váha je dle výrobce 5,5kg.

3.2. Výbava a senzory robota

Trilobot má velice široké spektrum senzorů, které by teoreticky měly zajišťovat podporu pro inteligentní navigaci v neznámém prostředí, komunikaci s ostatními roboty nebo dokonce s člověkem. Vše je však třeba brát s rezervou. Například právě pro potřeby navigačních úloh je vybavení Trilobota poměrně chudé. Informace o geometrii vnějšího prostředí totiž zprostředkovávají pouze dva jednoduché sonary a dotykové senzory. Ostatní senzory (jako tepelný senzor, detektor vlhkosti apod.) jsou pro tuto aplikaci zbytečné. Bylo by vhodnější použít kvalitnější senzory potřebné pro inteligentní navigaci protože ta je u mobilního robota klíčová. U elektronických senzorů bohužel výrobce neuvádí jejich typové označení, nebo alespoň přesné parametry. Stejně tak chybí popis některých funkcí – například fungování přepočtu otáček motorů při zatáčivém pohybu robota. Uživatel tento pohyb může specifikovat pomocí tří parametrů, ale klíčový parametr pro zatáčení je definován pouze výčtem možných hodnot – ne jejich přesným významem.

Trilobot je také osazen jednoduchým uchopovacím zařízením, které však podle mého názoru nenajde velkého uplatnění. Zařízení umožňuje sevřít pouze předměty specifických rozměrů, které však jen stěží projdou mezi dotykovými senzory rozmístěnými po celém obvodu robota – tedy i před uchopovacím zařízením.

Trilobot má v základní výbavě konkrétně tyto senzory:

- 8 nezávislých dotykových senzorů
- elektronický kompas s přesností 2 stupně
- sonar (dva sonarové snímače umístěné na otočné hlavě)
- pasivní infračervený detektor pohybu
- 4 světelné senzory (možnost určit směr a intenzitu světla)
- tepelný detektor

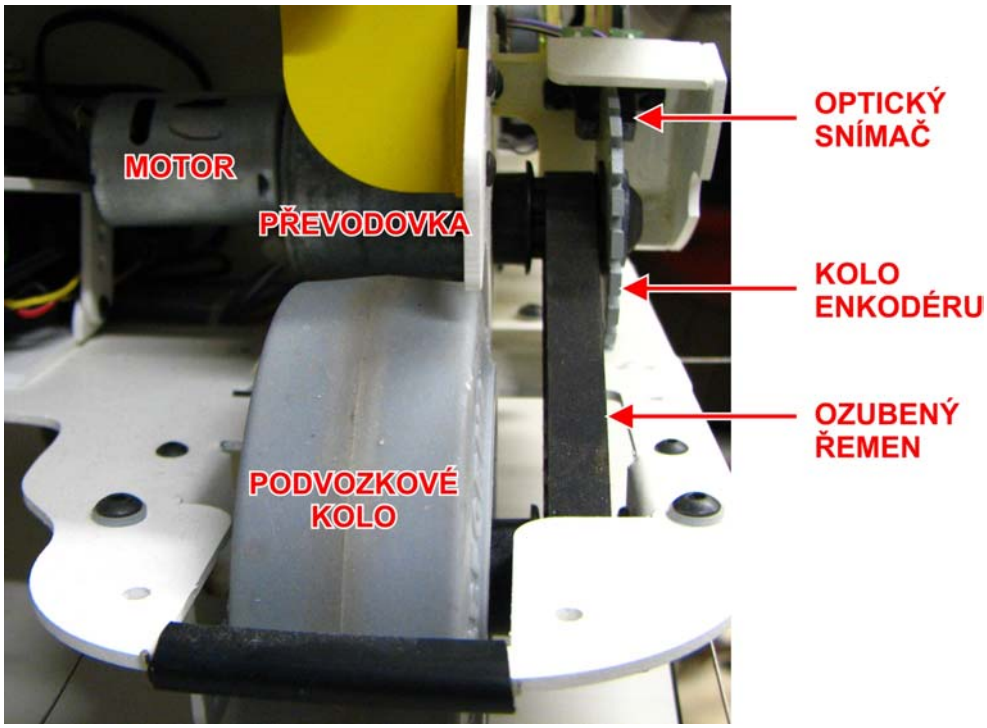
- detektory naklopení ve všech směrech
- detektor vlhkosti (resp. „detektor louží“ – umožňuje detekovat, zda robot nejede vodou)
- audio vybavení (možnost detekce a nahrání zvuku)
- optické enkodéry pro odometrii a určení rychlosti motorů
- sledování napětí baterie
- infračervený přijímač a vysílač pro příjem signálu z dálkového ovládání a komunikaci s ostatními Triloboty

Vzhledem k tomu, že se v dalším popisu budu zabírat některými vlastnostmi robota detailněji, upozorňuji, že popisuji Trilobota označeného v laboratoři ÚITS číslem 6. Je možné, že jiný exemplář Trilobota by mohl mít tyto parametry mírně odlišné.

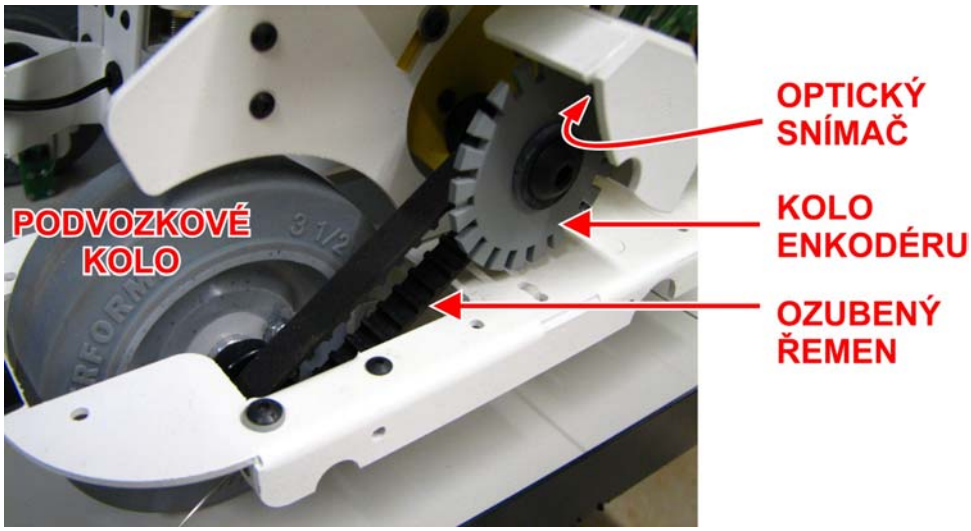
3.3. Podvozek a pohonné jednotky

Podvozek diferenciálního typu je tvořen dvěma hnanými koly o průměru 89mm, která jsou umístěna v přední části robota na pevných osách, a jedním kolem o průměru 51mm umístěným na otočné ostruze. Obě hlavní kola mají vlastní pohonnou jednotku (stejnosměrný motor s převodovkou) a inkrementální optický enkodér. Otočné kolo má pouze podpůrný účel, není nijak řízeno. Je umístěno na podélné ose robota v jeho zadní části. Všechna kola jsou vyrobena z gumy a nemají žádný vzorek. Pouze po obvodu hlavních kol vystupuje podélný šev vzniklý při výrobě od něhož se průměr kol k jejich okraji lehce snižuje. Kola tedy nemají kontakt s podložkou po celé své tloušťce, ale pouze po cca 7mm širokém pásu (tento údaj samozřejmě záleží na míře opotřebení kol). Tato vada však zřejmě nezpůsobuje vážné problémy při pohybu – na první pohled se nejeví, že by robot při jízdě podkluzoval. Pouze při měření obvodu kola jsem došel k hodnotě 281mm – kdežto podle udávaného průměru by obvod měl činit 279,6mm. Tento rozdíl by mohl hrát roli při úkonech, kdy robot počítá dobu pohybu v závislosti na obvodu kol – tedy např. při zatočení o specifický úhel. Rozdíl je však minimální a mohl by se projevit až při velkých vzdálenostech.

Kroutící moment od motorků je veden přes převodovku na ozubené kolečko, které je přes ozubený řemen spojeno se stejným kolečkem na ose podvozkového kola. Na ose prvního ozubeného kolečka je připevněno i kolo optického enkodéru s 22 zářezy, jehož otáčení je snímáno optickým senzorem umístěným na krytu optického enkodéru. Jedno otočení kola enkodéru tedy signalizuje jedno otočení podvozkového kola.



Obrázek 3.2: *Detailní pohled zepředu na pohon levého kola.*



Obrázek 3.3: *Detailní pohled zboku na pohon levého kola a optický enkodér.*

I když má kolo enkodéru 22 zářezů, optický senzor zřejmě zaznamenává obě **hrany zářezu**, takže jedno otočení kola enkodéru je zaznamenáno jako 44 implusů. Pokud má tedy podvozkové kolo obvod 281mm, můžeme uvažovat jeden krok enkodéru jako ujetí vzdálenosti 281 / 44 = 6,39mm. Podle manuálu [8] robota by čtyři kroky enkodéru měly odpovídat ujetí vzdálenosti jeden palec – tedy 25,4mm, při přepočtu ze známého obvodu ale dojdeme k hodnotě 6,39mm x 4 = 25,55mm. Pokud bychom tedy spoléhali na tvrzení výrobce, docházelo by k chybě navigace zhruba o 1mm na 4 kroky enkodéru.

3.4. Ovládání robota

Robota je možné ovládat přes sériovou linku, pomocí joysticku nebo dálkového infračerveného ovladače. Další možností je vložit jednoduchý program přímo do paměti robota.

Ovládání pomocí infračerveného ovladače je stejně jako ovládání přes joystick omezené jen na několik základních povelů. Daleko mocnější je ovládání pomocí textových příkazů přes sériovou linku popř. náhradou sériového kabelu bezdrátovým transceiverem.

3.4.1. Textové příkazy

Robot se řídí pomocí krátkých textových zpráv s pevně daným formátem. Prvním znakem zprávy je vždy znak „!“. Druhým znakem je identifikační číslo kontroléru (pro případ zřetězení více robotů na sériové lince) – při obvyklém způsobu zapojení jednoho robota k jednomu portu sériové linky je tedy na tomto místě „1“. Následuje vlastní příkaz, který se skládá ze dvou znaků. První určuje typ příkazu:

G (get) –> získání informací od robota

P (put) –> zaslání instrukce robotovi

O (other) –> ostatní příkazy

Druhým znakem příkazu je identifikátor požadované operace následovaný požadovanými parametry (pokud je třeba). Parametry se zadávají v hexadecimálním formátu (buď jako hexadecimální bajt, nebo jako hexadecimální slovo).

Návratová hodnota je v případě příkazu typu GET hexadecimální číslo, znak „A“ (acomplished) v případě úspěšného zpracování zprávy nebo „B“ (bad) v případě neúspěchu.

3.4.2. Konkrétní příkazy týkající se pohybu:

PM (**P**ut Driver **M**otor Control)

Určuje jakou rychlostí, na jakou vzdálenost a jakým směrem se má robot pohybovat. Příkaz je tedy specifikován třemi parametry:

První parametr je hexadecimální bajt určující rychlost a může nabývat hodnot 01 - 07 (Trilobot se tedy může pohybovat sedmi různými rychlostmi)

Druhý parametr je hexadecimální slovo určující vzdálenost, resp. počet kroků optického enkodéru.

Třetí parametr udává poměr rychlostí obou kol. Možné hodnoty tohoto parametru:

00	rovně vpřed (obě kola stejnou rychlostí)
01-0F	vpravo vpřed s různými poloměry zatáčky (levé kolo vyšší rychlostí)
11-1F	vlevo vpřed s různými poloměry zatáčky (pravé kolo vyšší rychlostí)
20	rovně vzad
21-2F	vpravo vzad s různými poloměry zatáčky
31-3F	vlevo vzad s různými poloměry zatáčky
40	pravá rotace na místě (obě kola stejnou rychlostí, ale opačným směrem)
41	levá rotace na místě

Například příkaz !1PM02010000 bude interpretován jako jízda rychlostí č. 2 (02), na vzdálenost odpovídající 256 kroků enkodéru (0100h) přímo vpřed (00).

Konkrétní specifikace hodnot 01-0F apod. však nejsou v manuálu k Trilobotovi uvedeny.

GM (**G**et Drive **M**otor Information)

Parametr tohoto příkazu může nabývat hodnoty 1 nebo 2.

Příkaz !1GM1 vrátí aktuální počet impulsů napočítaných optickým enkodérem jako hexadecimální slovo.

Příkaz !1GM2 vrátí aktuální rychlost jako hexadecimální bajt.

4. Analýza vhodného simulačního nástroje pro řešení práce

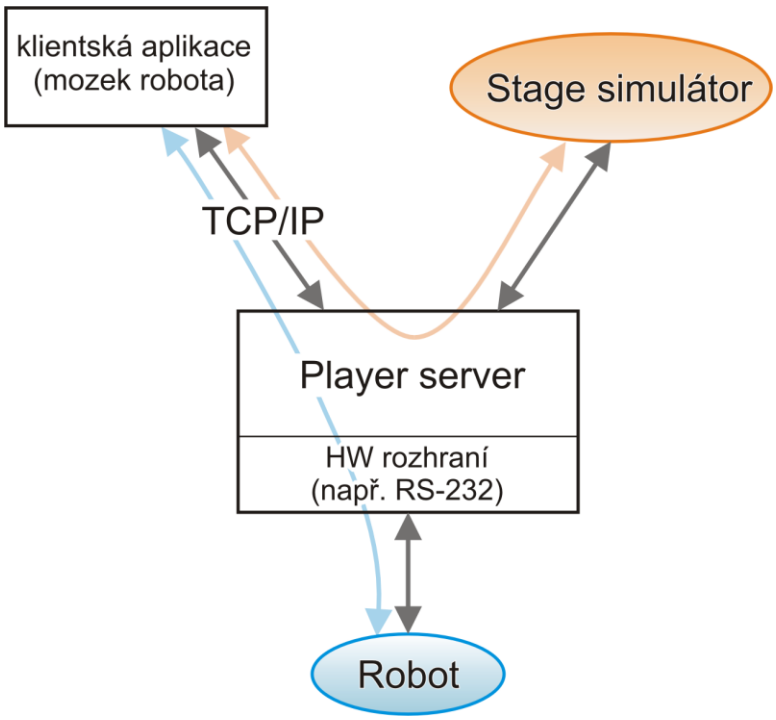
První simulační prostředí, které se jevilo jako výhodné pro zkonstruování modelu Trilobota je Player/Stage. Toto prostředí je zřejmě nejznámější prostředí pro simulaci robotů jak v 2D (nástroj Stage) tak i v 3D prostoru (nástroj Gazebo). Jeho vývoje, který probíhá pod licencí GNU, se aktivně účastní vývojáři z několika zemí. Použití tohoto prostředí by bylo výhodné také díky rozhraní vytvořeném ve Squeak Smalltalku diplomantem D. Bajgerem v roce 2005.

4.1. Projekt Player/Stage a simulátor Gazebo

Cílem projektu Player/Stage [2] je podpora výzkumu v oblasti robotických a senzorických systémů. Výsledkem jsou SW prostředky Player, Stage a Gazebo které jsou stále vyvíjeny pod licencí GNU.

Player je server poskytující rozhraní pro řízení robota a spojení s jeho senzory. Stage je multiagentní simulátor pro Player server. Gazebo má stejnou funkci jako Stage, je však rozšířen do 3D.

Koncepce serveru umožňuje využití jakéhokoliv programovacího jazyka poskytujícího prostředky pro komunikaci přes TCP sockety. Klientský program má prostřednictvím TCP socketů přísup k informacím ze senzorů robota a může zadávat příkazy jeho efektorům. Z pohledu klientského programu se tedy nerozlišuje zda se jedná o robota připojeného fyzicky k player serveru nebo jeho simulaci poskytovanou např. simulátorem Stage. Sada příkazů je stejná. Tato koncepce tedy umožňuje vyvinout algoritmus (resp. program) při jeho snadné a rychlé simulaci a v nezměněné podobě ho použít na řízení skutečného robota.



Obrázek 4.1: *Schématické znázornění dvou alternativ použití prostředí Player/Stage. Fyzické připojení robota může nahrazovat simulátor Stage.*

4.1.1. Player server

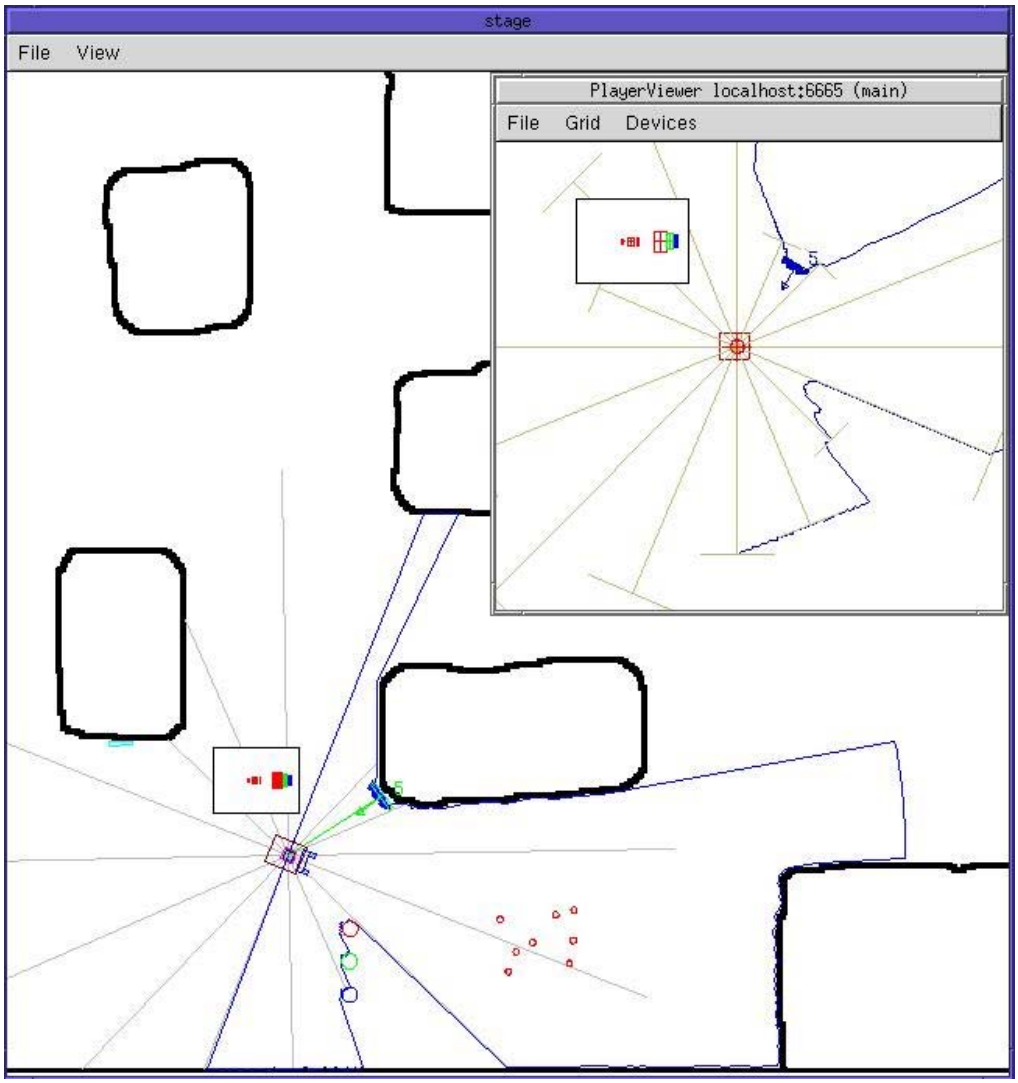
Player server je hlavní součástí prostředí. Je to vlastně síťový server, který poskytuje rozhraní pro komunikaci klienta s robotem. Pro komunikaci s klientem využívá protokol TCP/IP. Na straně druhé je možné na Player připojit robota přes fyzický port poskytovaný serverem (většinou RS 232) nebo virtuálně – na simulátor Stage nebo Gazebo.

Aby bylo možné využít přímého připojení robota k Player serveru, je nutné mít ovladače k HW robota. Player obsahuje ovladače k několika nejrozšířenějším typům robotů (Pioneer, AmigoBot, Khepera…) a k množství senzorického HW (laserové skenery, sonary, dotykové senzory, PTZ kamery, GPS přijímače…). Vzhledem k tomu, že player je psán v jazyce C/C++ je nutné v případě psaní ovladačů pro některé další typy zařízení použít stejný programovací jazyk.

4.1.2. Simulátor Stage

Stage simuluje vstupy pro Player a zastupuje tak robota i prostředí v kterém se pohybuje. Hlavními vstupy pro Stage jsou konfigurační soubory *.world, *.cfg a *.inc. Tyto soubory obsahují nastavení Stage, specifikace simulovaného světa a robotů. Simulace (jak se zdá) probíhá podle zjednodušeného modelu, robot je uvažován jako soubor senzorických zařízení s určitými parametry a nastavením.

Konfigurační soubor definující simulované prostředí obsahuje základní parametry jako jsou rozměry světa. Umožňuje načíst rastrovou definici překážek. Překážky je dále možné definovat jako samostatné modely se speciálními vlastnostmi, např. zvýrazněná viditelnost pro určitý typ senzorů a podobě.



Obrázek 4.2: *Okno simulátoru Stage a Player serveru (vpravo nahoře). V okně Playeru jsou znázorněny informace získané senzory robota (v tomto případě laserovým scannerem).*

Nicméně použití tohoto prostředí je vázané na existenci ovladače simulovaného robota. I když ovladačů pro Player/Stage stále přibývá – bohužel se to týká pouze známých robotů (např. v poslední době přibyl mezi podporované roboty [3] Playeru i komerčně úspěšný uklízecí robot Roomba). Proto nepředpokládám, že by se ve světě mezi zanedbatelným množstvím uživatelů Trilobota našel nadšenec, který by se pokusil vytvořit poměrně komplikovaný ovladač tohoto robota pro Player/Stage.

Tato komplikace nakonec vedla k rozhodnutí raději zkonstruovat vlastní – zjednodušený simulátor přímo pro Trilobota. Pro realizaci simulátoru jsem zvolil Squeak Smalltalk, který poskytuje platformě nezávislé objektově orientované prostředí. Navíc je možné využít simulační nástroj SmallDEVS, který poskytuje třídy pro práci s diskrétními simulacemi navrženými dle formalismu DEVS.

4.2. Squeak Smalltalk

Squeak [14] je moderní otevřené prostředí založené na jazyce Smalltalk, který mu dává naprosto unikátní vlastnosti. Je to čistě objektově orientovaný jazyk, takže narozdíl od ostatních „objektově orientovaných“ jazyků v něm bez výjimky platí pravidlo, že *všechno* je objekt. Smalltalk není jenom programovací jazyk, ale rozsáhlé prostředí, které se některými vlastnosti podobá samotnému operačnímu systému.

Velkou výhodou prostředí Smalltalku je to, že je celé napsáno ve Smalltalku - tedy samo v sobě. Díky tomu že obsahuje i svůj kompilátor, umožňuje naprosto revoluční přístup, takže je možné za běhu velmi jednoduše měnit jakoukoliv jeho část a tím upravovat jeho chování. Tento volný způsob programování tedy neomezuje aplikace v tom, aby měnily například samy sebe přímo za běhu.

Fungování systému je primárně zajištěno souborem *Image* – objektové paměti, díky které je zajištěna dokonalá perzistence všech objektů a *virtuálním strojem*. Ten zajišťuje interpretaci bytekódu, který vzniká dynamickou kompilací smalltalkovského kódu. Platformní nezávislost je díky této architektuře omezená jenom možnostmi portování virtuálního stroje. Po přenesení konkrétní image z jedné platformy na druhou je tedy možné (pokud máme k dispozici virtuální stroj pro danou platformu) spustit systém založený na této image v naprosto stejném stavu, v jakém byl na původní platformě. V současnosti již existují virtuální stroje Squeaku určené pro běh pod nejrůznějšími operačními systémy, takže se toto prostředí dá považovat v podstatě za platformě nezávislé. Běh virtuálního stroje ale nemusí být omezen pouze na operační systémy – Smalltalk je schopen „běžet i na holém železe“.

4.3. Formalismus DEVS

Formalismus DEVS (*Discrete event system specification* – specifikace systému diskrétních událostí) je univerzální modelovací formalismus, který umožňuje definovat simulace řízené diskrétními událostmi jako systém hierarchicky uspořádaných komponent.

DEVS je struktura:

M = ⟨X , S , Y , δ_{int} , δ_{ext} , λ , ta⟩ kde:

- X je množina všech vstupů do systému, $X = \times_{i=1}^{m} X_i$; *m* je počet vstupních portů, každý je identifikován svým indexem *i*
- S je množina stavů, $S = \times_{i=1}^{n} S_i$
- Y je množina povolených výstupů ze systému, $Y = \times_{i=1}^{l} Y_i$; *l* je počet výstupních portů, každý je identifikován svým indexem *i*
- δ_{int} : S → S je funkce interního přechodu,
- δ_{ext} : Q x X → S je funkce externího přechodu, kde:

  Q = {(s,e) | s ∈ S, 0 ≤ e ≤ ta(s)} je množina všech stavů, *e* je doba od vykonání posledního přechodu,
- λ : S → Y je výstupní funkce,
- ta : S → $\Re_{0,\infty}^{+}$ je množina reálných čísel z intervalu 0 až ∞

4.3.1. Popis fungování simulace podle formalismu DEVS

Systém zůstává ve stavu $s \in S$ tak dlouho, dokud se na vstupu nevyskytne externí událost, nebo dokud neuplyne doba určená *ta(s).* Po uplynutí této doby nejdříve dojde k provedení výstupní funkce, která definuje výstupní hodnotu a poté interního přechodu (tedy změně stavu) – tímto pořadím je zajištěna dostupnost výstupů systému ještě před změnou stavu systému. Doba *ta(s)* tedy určuje jak dlouho zůstane systém ve stavu *s*. Pokud bude mít nulovou hodnotu, nemůže dojít k případu, že by byl přechod systému vyvolán externí událostí. Pokud bude mít naopak hodnotu rovnou nekonečnu, zůstane systém ve stávajícím stavu, dokud externí událost nepřijde. V případě výskytu externí události je aktivována externí přechodová funkce, která může převést systém do nového stavu (podobně jako interní přechodová funkce v reakci na uplynutí doby *ta*).

4.3.2. Atomický a spojovaný DEVS

Tato základní varianta formalismu se nazývá **atomický DEVS** (*atomic DEVS*). Pro pohodlnější specifikaci složitějších modelů je však vhodné využít více jednoduchých modelů definovaných zmíněným způsobem a zajistit jejich interakci pomocí určitých vazeb. Právě pro tento účel slouží **spojovaný DEVS** (*coupled DEVS*), který definuje způsob jakým je možné tyto komponenty zapojit do složitější sítě. Z pohledu spojovaného DEVS modelu se přitom tyto komponenty (ať už atomické nebo jiné spojované DEVS modely) jeví jako „černé krabičky“ u kterých nás zajímají pouze jejich vstupy a výstupy. Tímto

způsobem je tedy možné systém popsat na více úrovních abstrakce, přičemž funkční bloky jsou na nejnižší úrovni – ve formě atomických DEVS modelů. Lze přitom dokázat, že každý spojovaný DEVS model může být nahrazen ekvivalentním modelem specifikovaným pomocí atomického DEVS formalismu.

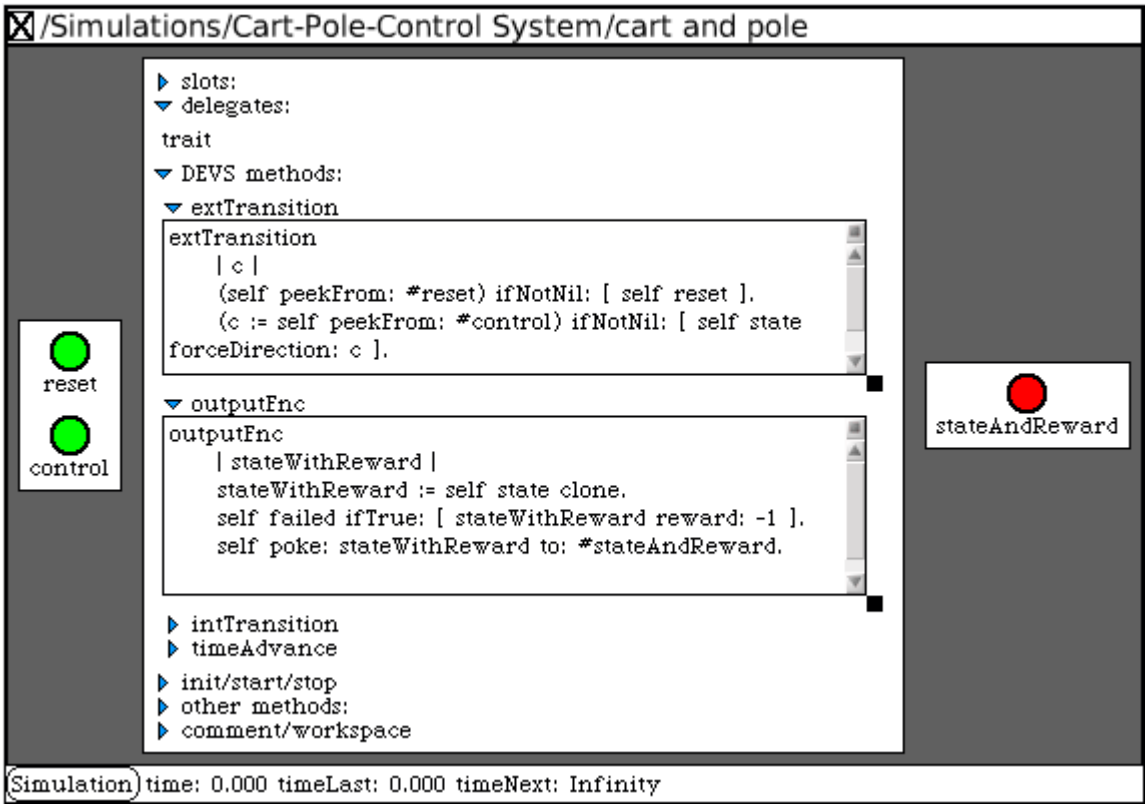
4.4. Nástroj SmallDEVS

Podpora pro modelování a simulaci dle formalismu DEVS již existuje v mnoha programovacích jazycích jako C++, Java, Python apod. V roce 2003 vznikla na naší půdě nová implementace [13] ve Squeak smalltalku. Autorem této knihovny tříd je Ing. Vladimír Janoušek Ph.D. O dva roky později bylo dokončeno také grafické uživatelské rozhraní, jehož autorem je Ing. Elöd Kironský.

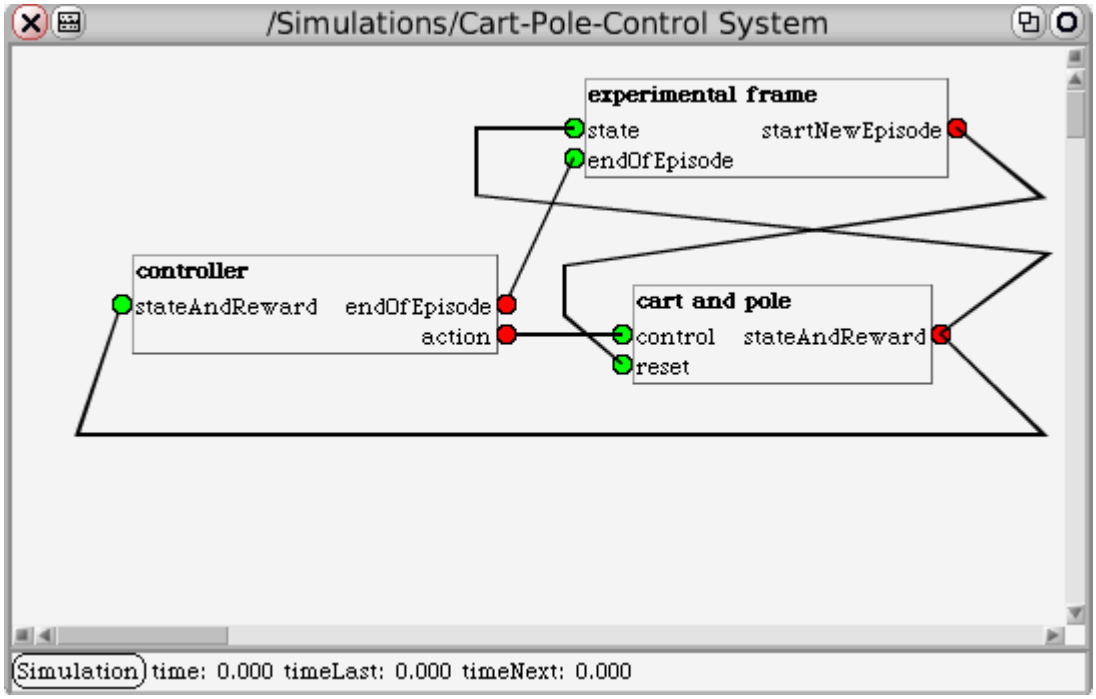
Oproti ostatním objektově orientovaným jazykům, kde je každý objekt striktně vázán na svojí třídu, můžeme u SmallDEVSu využít i univerzálnějšího beztřídního přístupu založeného na **prototypech**. Tento přístup umožňuje mnohem flexibilnější zacházení s modelem během simulace. Navíc díky koncepci Smalltalku, narozdíl od staticky kompilovaných jazyků, není nutné definovat veškeré chování už v čase kompilace. Tato výhoda umožňuje modelovat evoluční i jiné systémy vyžadující možnost úpravy modelu během simulace. Pro úpravu vlastností prototypového objektu totiž není nutná úprava třídy. Tuto funkčnost dodává ve své definici třída *PrototypeObject*, v které je implementován protokol dovolující úpravu každého prototypového objektu aniž by bylo třeba pro něj vytvářet novou třídu. Pro tvorbu DEVS modelů založených na prototypových objektech lze využít tříd *AtomicDEVSPrototype* a *CoupledDEVSPrototype*. Pokud by naopak bylo potřeba aby objekty chování sdílely, poskytuje SmallDEVS třídu *AtomicDEVSTrait*, jejíž objekty mohou definovat vlastnosti a chování stejným způsobem jako jiné atomické DEVS modely s tím, že je možné je přiřadit (pomocí metody *addDelegate: withValue:*) k prototypovému objektu, který z nich bude dynamicky dědit. Tímto způsobem je tedy možné upravovat chování různých objektů najednou i individuálně.

Persistence objektů může být zajištěna běžným smalltalkovským způsobem – vazbou na proměnné workspace. Lepší řešení je ale využitím *MyRepository* - hierarchicky uspořádané struktury umožňující uchovávání modelů a simulací.

Grafické uživatelské rozhraní (GUI) poskytuje komfortnější způsob vytváření a úpravy modelů a simulací. Interaktivní vizualizace navrhovaného modelu umožňuje přidávat/odstraňovat modely, pracovat s jejich porty i kontrolovat simulaci. GUI ve stylu prostředí jazyk Self také umožňuje přístup ke všem parametrům a metodám DEVS modelů i řízení simulace.



Obrázek 4.1: *Grafické uživatelské rozhraní SmallDEVSu umožňující přístup k metodám atomického DEVS modelu.*



Obrázek 4.2: *Grafické uživatelské rozhraní SmallDEVSu. Znázornění schématu simulačního modelu úlohy Cart-Pole*

5. Zpráva z laboratorních měření

Abych blíže prozkoumal způsob fungování pohonného systému robota a zjistil potřebné vlastnosti pro jeho model, musel jsem provést některá měření, jejichž popis je dále rozveden v následujících kapitolách. Všechna měření byla provedena na Trilobotovi označeném laboratoří ÚITS číslem 6. Konkrétní údaje získané při experimentech jsou v příloze na konci této práce.

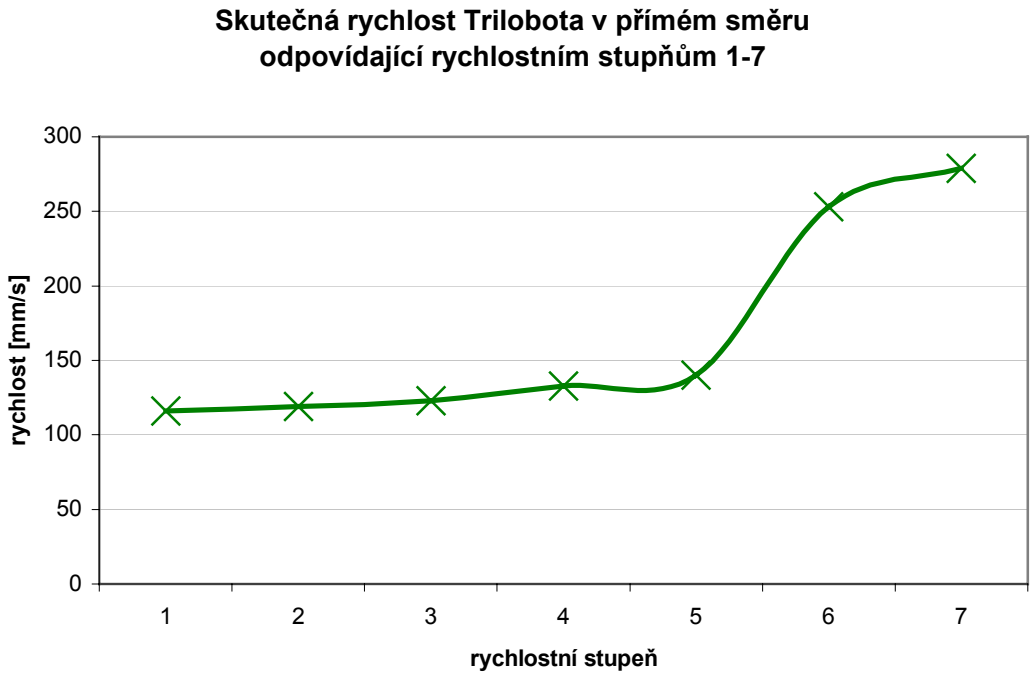
Pro upřesnění uvádím, že pojmem „chyba navigace“ je v celé této práci míněna nepřesnost vzniklá nesprávnou interpretací navigační úlohy (resp. příkazu týkajícího se změny polohy robota). Z pohledu vzniku chyby tedy zřejmě jde o *chybu řízení* nebo *navádění*. Podle mého názoru je však tento termín poněkud zavádějící. Termín „chyba odometrie“ zde zastává svůj pravý význam (viz. termín odometrie v kapitole 2.3.1).

5.1. Analýza rychlosti jízdy

Důležitým vlastností modelu Trilobota je rychlost jeho jízdy. V parametrech příkazu PM je možné rychlost definovat pouze na základě nic neříkající číselné hodnoty (rychlostní stupeň od jedné do sedmi). Ani v manuálu robota není řečeno jaké rychlosti odpovídají těmto hodnotám. Navíc se podle běžného pohledu jeví, že rozdíl mezi jednotlivými rychlostmi není rovnoměrný – resp. že průběh hodnot rychlostí příslušících různým rychlostním stupňům není lineární.

Měření jsem prováděl tak, že jsem robotovi zaslal příkaz k ujetí vzdálenosti danou rychlostí, v různých směrech na různou vzdálenost a čas pohybu robota změřil na stopkách. Nicméně přesnost tohoto měření ovlivňovala nepřesnost ručního měření času - prodleva způsobená reakční dobou člověka.

Změřený průběh rychlostí je patrný z grafu 5.1



***graf 5.1**: hodnoty rychlosti odpovídající jednotlivým rychlostním stupňům (definovaným prvním parametrem příkazu PM) 01 – 07. I když to vypadá nelogicky, z naměřených hodnot vyplývá, že jejich průběh není lineární ani exponenciální.*

Další částí analýzy rychlosti jízdy bylo měření určující závislost rychlosti jízdy na velikosti zatočení robota. Provedl jsem několik měření času ujetí jedné vzdálenosti při různých stupních zatočení. Průměry naměřených hodnot jsou shrnuty v tabulce 5.1.

poměr zatočení	čas [s]
00	2,8
08	3,2
0F	5,2

***Tabulka 5.1:** délka jízdy rychlostí 01 na vzdálenost 48 kroků enkodéru při různých stupních zatočení*

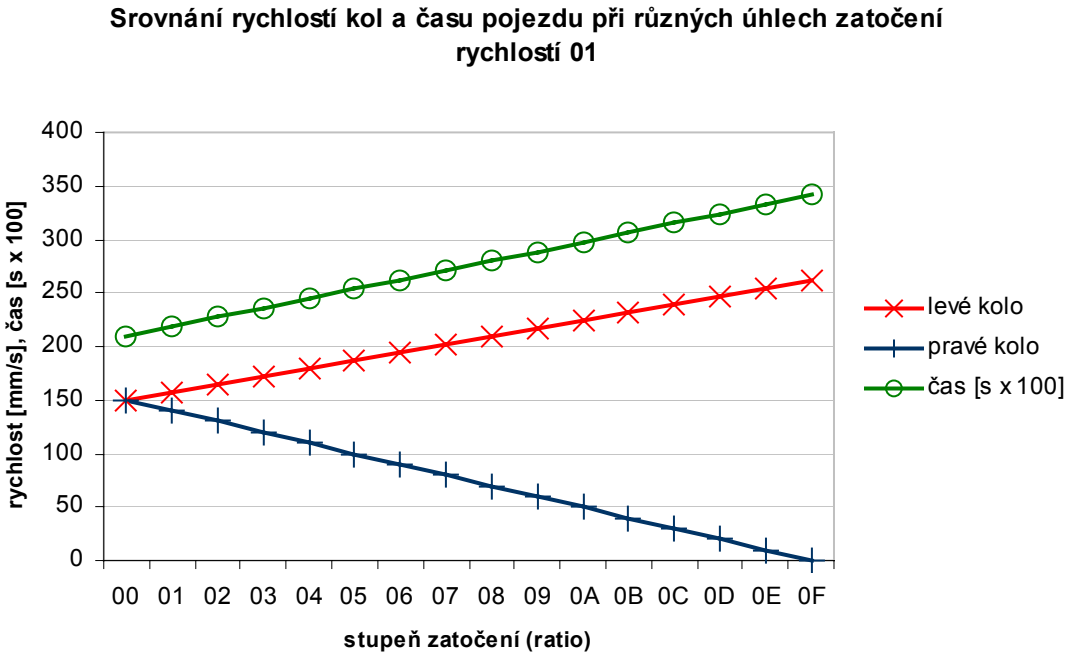
5.2. Způsob určování otáček motorů při zatáčení

Další otázkou byl způsob využití parametru „ratio“ (poměr) v rámci příkazu PM (Put Driver Motor Control). V manuálu robota tento parametr není přesně definován pro hodnoty odpovídající různým stupňům zatočení. Měření, měla objasnit výpočet otáček jednotlivých motorů na základě zmíněného poměru a vzdálenosti, kterou má robot ujet.

Jednoduché zjištění údajů pomocí příkazů vracejících aktuální rychlost nebo ujetou vzdálenost ale není možné, protože návratovou hodnotou je pouze jeden souhrnný údaj pro obě kola. Opět nevíme jak je tento údaj určen.

Proto bylo potřeba experimentálně zjistit tyto údaje zvlášť od každého kola. Pomocí digitální kamery jsem zaznamenal počet otáček kol obou optických enkodérů při vykonávání příkazu ujetí vzdálenosti s různými poměry rozložení otáček na jednotlivá kola. Kvůli obtížné realizaci záznamu kol enkodérů jsem při tomto měření umístil Trilobota na podložku tak, aby nedocházelo ke kontaktu kol s podložkou. Tímto způsobem bylo možné zaznamenat přesné záběry kol enkodérů. Nicméně právě proto, že robot nebyl zatížen, neodpovídaly navozené podmínky realitě. Při běžném provozu musí totiž robot při rozjíždění nejprve překonat tření s podložkou a při brždění zase ujetou vzdálenost ovlivňuje setrvačnost. Nicméně, podle mého názoru, by případný rozdíl v otáčkách měl postihnout obě kola stejně, a míra chyby tohoto měření by měla být přijatelná. Dalším problémem tohoto měření byla rychlost otáčení kol se zářezy, které jsem zaznamenával. Při použití snímání frekvencí 30 snímků za sekundu bylo nemožné ze záznamu spočítat otáčky kol při vyšších rychlostech. Proto jediné údaje získané z tohoto měření se vztahují pouze na nejnižší rychlost Trilobota a malé stupně zatočení (čím větší stupeň zatočení, tím vyšší rychlost vnějšího kola). Ani tento problém nepovažuji za významný, protože předpokládám, že způsob, jakým se „rozdělují“ otáčky jednotlivým motorům je nezávislý na rychlostním stupni.

Ze získaných záznamů jsem poté mohl vyhodnotit přesný čas vykonávání příkazu i počet kroků enkodérů. Z těchto informací je možné vypočítat rychlosti jednotlivých kol, které jsou shrnuty v grafu 5.2 a tabulce 5.2.



graf 5.2: *Hodnoty rychlosti kol v závislosti na třetím parametru příkazu PM v rozsahu 00 - 0F, graf je pouze lineární interpolací naměřených dat, skutečný průběh by mohl být nelineární.*

ratio	rychlost v % rychlosti v přímém směru	
	L	R
00	100,0	100,0
01	105,0	93,3
02	110,0	86,7
03	115,0	80,0
04	120,0	73,3
05	125,0	66,7
06	130,0	60,0
07	135,0	53,3
08	140,0	46,7
09	145,0	40,0
0A	150,0	33,3
0B	155,0	26,7
0C	160,0	20,0
0D	165,0	13,3
0E	170,0	6,7
0F	175,0	0,0

tabulka 5.2: *Procentuální určení rychlosti kol v závislosti na třetím parametru příkazu PM v rozsahu 00 – 0F*

Dalším pokusem, který mohl částečně osvětlit tuto problematiku bylo experimentální zjištění vzdálenosti (resp. druhého parametru příkazu PM), která by odpovídala při maximálním zatočení (ratio = 0F) devadesátistupňovému obratu. Je zřejmé, že při maximálním zatočení robota má kolo na vnitřní straně zatáčky nulovou rychlost. Na základě tohoto faktu a rozchodu kol (200mm) tedy můžeme spočítat dráhu vnějšího kola ze změny orientace robota při (maximálním) zatočení. Z dráhy vnějšího kola ($2\pi r/4$ = 314,2mm) a dráhy odpovídající jednomu kroku enkodéru (6,39mm) můžeme vypočítat, že ujetá vzdálenost vnějšího kola odpovídá 314,2 / 6,39 ≅ 49 krokům enkodéru.

Experimentálně jsem zjistil, že Trilobot se natočí o devadesát stupňů při vykonání příkazu !1PM0100160F – tedy při žádosti o ujetí 22 kroků enkodéru při maximální pravé zatáčce. Pokud tedy vnější kolo muselo vykonat počet otáček odpovídající 49 krokům, můžeme určit, že Trilobot **při maximálním zatočení** vypočítá počet kroků vnějšího kola jako 49/22 = **2,2 násobek kroků enkodéru v přímém směru**.

5.3. Měření odchylky v navigované a skutečné vzdálenosti

Další měření se zabývalo přesností s kterou je robot schopen ujet zadanou vzdálenost. Toto měření spočívalo v zadání příkazu pro ujetí různých vzdáleností různými rychlostmi v přímém směru, následném zjištění ujeté vzdálenosti pomocí příkazu GM (Get Drive Motor Information) a změření skutečně ujeté vzdálenosti pomocí ručního metru. Tímto způsobem je možné určit odchylku mezi vzdáleností zamýšlenou a skutečně ujetou (chyba navigace). Rozdíl mezi vzdáleností skutečnou a tou, kterou robot indikuje pomocí optických enkodérů je označen jako chyba odometrie. Vyhodnocení těchto chyb je vidět v tabulce 5.3.

Příkaz	Chyba odometrie		Chyba navigace [mm]
	[mm]	kroky odometru	
!1PM01008000	14	3	17
!1PM01008000	-1	0	32
!1PM01008000	-1	0	32
!1PM0100F000	-2	0	21
!1PM0100F000	-2	0	27
!1PM0100F000	0	1	25
!1PM07010000	-92	-14	118
!1PM07008000	-81	-12	112
!1PM05010000	-81	-12	113
!1PM05010000	-84	-13	110
!1PM02010000	-60	-9	73

tabulka 5.3: *Výpočet chyby navigace a chyby odometrie při rovné jízdě vpřed*

5.4. Měření odchylky ze směru navigované trajektorie

V tomto testu jsem se zaměřil na schopnost udržení přímé trajektorie jízdy. Robot byl před každým pokusem nastaven podél rovné linie, ostruhové kolečko otočeno do směru jízdy, aby nemohlo způsobit zbytečný odpor, který by robota vychýlil z požadovaného směru. Poté byl robotovi zaslán příkaz ujet určitou vzdálenost v přímém směru, po jehož vykonání byla změřena odchylka robota od linie – tedy od očekávané trajektorie.

Přesnost těchto pokusů však podle mého názoru nebyla dostatečná vzhledem k míře chyby kterou může způsobit nepatrná odchylka v počátečním nasměrování robota. Při běžném pohybu Trilobota přitom hraje natočení ostruhového kolečka významnou roli, protože může způsobit svým nevýhodným natočením „rozkývání" Trilobota, který se poté odchýlí od původní trajektorie. Takže hodnoty získané z tohoto měření jsou stále jenom malou veličinou v porovnání chyby způsobené natáčením ostruhového kolečka a hrají roli pouze v případě přímočarého pohybu bez zatáčení.

5.5. Měření založená na metodice Square-path test

Squeare-path test je názorná zkouška pro měření míry chyb navigace. Pro správné provedení tohoto testu se robot naprogramuje tak, aby se pohyboval po čtvercové dráze. To znamená, že po splnění úkolu by se robot měl nacházet ve výchozím bodě jeho cesty. Odchylka původní a koncové pozice potom znázorňuje chybu.

Když jsem chtěl tento test realizovat, otestoval jsem nejdříve základní příkazy pro pohyb po čtvercové dráze. Klíčovým se v tomto případě ukázal pohyb natočení robota o 90 stupňů. Celkem jsem realizoval asi 20 pokusů, z nichž naprostá většina byla pouhým okem rozhodnutelná jako neúspěšná. Většinou Trilobot setrvačností překonal vyměřený úhel o cca 30 stupňů. V několika případech dosáhl požadované, nebo menší změny směru. Důvod zatočení o znatelně menší úhel se mi však nepodařilo objasnit, nevšiml jsem si ani protočení hnaného kola.

Shledal jsem tedy, že Square-path test nemá smysl realizovat, protože výsledky z něj vyvozené by měly nahodilý charakter a pro jejich další uplatnění by bylo třeba realizovat velké množství experimentů pro určení pravděpodobnostního rozložení chyby navigace. Proto jsem přistoupil ke zjednodušeným testům, které neodpovídají Square-path testu, ale fungují na podobném principu. Jednalo se o dvojnásobné projetí stejné přímé dráhy (tedy „tam a zpátky"). Výsledky těchto pokusů vykázaly viditelnější chyby navigace až při vzdálenostech překračujících 70 cm. Nicméně, jak je vidět z následující tabulky 5.4, chyba odometrie je paradoxně vyšší u vzdáleností kratších.

vzdálenost [mm]	odchylka navigace [mm]	odchylka odometrie [mm]
357	-5	-5
357	2	2
332	-1	5
754	-20	1
850	-10	2

tabulka 5.4: *Zjištěné chyby navigace a chyby odometrie při několika pokusech založených na podobném principu jako square-path test*

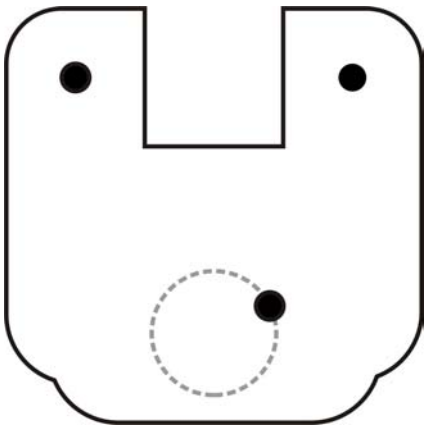
5.6. Závěry laboratorních testů

Tyto experimenty neměly za cíl kompletně proměřit a zmapovat pohybové vlastnosti Trilobota. Šlo hlavně o zjištění základních parametrů, které jsou potřeba pro návrh modelu, ale nejsou uvedeny v manuálu (např. rychlost v závislosti na rychlostním stupni atd.). Další měření měla nastínit povahy chyb v navigaci a odometrii.

Základní otázkou byl způsob rozdělení otáček jednotlivých motorů v závislosti na třetím parametru příkazu PM. I když vzhledem k náhodné povaze chyby bylo provedeno malé množství experimentů a z naměřených hodnot tedy nevyšly zcela přesné výsledky, bylo by možné určitým způsobem definovat fungování řídícího systému Trilobota. Nicméně výsledky měření s využitím videozáznamu si, zřejmě v důsledku navození umělých podmínek (ztráta kontaktu s podložkou), odporovaly s výsledky jiných experimentů. Proto jsem při určování funkce navigačního systému přihlížel spíše k výsledkům z ostatních experimentů.

Hodnota základní rychlosti Trilobota v závislosti na rychlostním stupni má poněkud zvláštní průběh (viz. Graf 5.1), proto jsem naměřené hodnoty akceptovat jako pevně daný parametr, který jsem se nesnažil aproximovat nějakou funkcí. Naopak rozdělení otáček motorů jsem na základě několika naměřených hodnot aproximoval jednoduchou lineární funkcí. Z odchylek navigované a skutečně ujeté vzdálenosti (i směrových odchylek) jsem nevyvozoval žádné konkrétní závěry. Měření těchto chyb mělo pouze nastínit míru chyby navigace.

Při jednom pokusu se náhodou projevilo určité omezení podvozku Trilobota. Zadní podpůrné kolečko se při zatáčení robota stáčí tak, aby kladlo co nejmenší odpor. Protože je umístěno na ostruze (dlouhé 45mm), dostává se **při rotaci Trilobota na místě** blíže ke středu podvozku (a tedy i blíže k těžišti) a snižuje tím stabilitu platformy - situace je znázorněna na obrázku 5.1. Zřejmě malá nerovnost povrchu způsobila, že se v jednom případě robot naklonil dozadu a jedno hnané kolo ztratilo kontakt s podložkou a protáčelo se „naprázdno“. Kdyby kola podvozku byla opatřena odpružením, mohl by být zaručen kontakt kol i na méně rovném povrchu. Problém přemísťování opěrného bodu by mohlo vyřešit například nahrazení všesměrovým kolem, které by nemuselo být umístěno na otočné ostruze. Nicméně tento problém považuji za nepodstatný, běžný pohyb Trilobota vůbec neovlivňuje.



Obrázek 5.1: *Umístění kol na podvozku Trilobota. Horní dvě černé tečky znázorňují hnaná kola. Čárkovaná kružnice znázorňuje dráhu po které se může pohybovat ostruhové kolečko v závislosti na rotaci robota.*

6. Návrh modelu robota

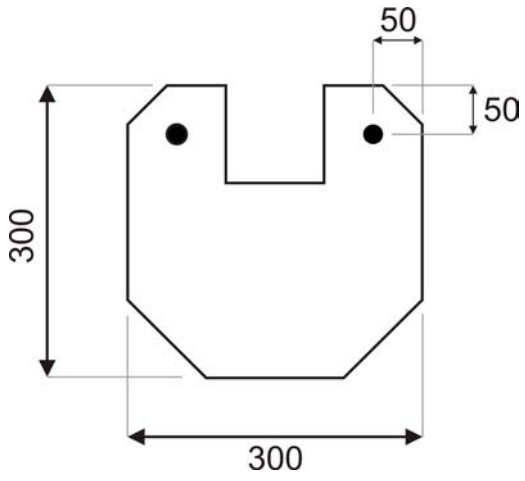
První fází je specifikace abstraktního modelu – tedy popis vlastností, které jsou významné pro potřeby simulace. Tento model se poté převede do simulačního modelu, který už je možné využívat v simulačním prostředí. Převod z abstraktního modelu na programové struktury simulačního modelu by měl být důsledně verifikován, aby výsledný model splňoval vlastnosti původně specifikované v abstraktním modelu.

6.1. Abstraktní model

Abstraktní model formuluje chování a vlastnosti modelovaného systému, které si přejeme dále zkoumat a využívat prostřednictvím simulace. Můžeme tedy shrnout vlastnosti skutečného robota a postupně abstrahovat ty „nepotřebné“. Nicméně můžeme postupovat obráceně. Nejdříve specifikujeme vlastnosti, které definují základní model pohybu robota. Po jeho úspěšné implementaci je poté možné model dále rozšiřovat o ostatní vlastnosti, které budou model zpřesňovat a přibližovat realitě.

6.1.1. Specifikace abstraktního modelu

Základem pro tuto specifikaci může být zjednodušený popis podvozku robota vycházející ze studie Trilobota – tedy platforma o rozměrech 300 x 300mm znázorněná na obrázku 6.1. Robot je poháněn dvěma koly umístěnými 50mm od okrajů předních rohů podvozku. Pro potřeby simulace pohybu můžeme ostruhové kolečko zanedbat a uvažovat podvozek jako objekt, který po podlaze klouže s nulovým odporem. Jeho účelem je pouze zprostředkování pevného spojení mezi oběma hnanými koly.



Obrázek 6.1: *Znázornění modelovaného robota s vyznačením polohy kol. Zadní podpůrné kolo je pro potřeby simulace abstrahováno*

Robot bude řízen stejnými příkazy (ve stejném formátu), kterým rozumí Trilobot s tím, že pro základní navigaci nám postačí příkazy PM a GM popsané výše v této práci. Dle výsledků laboratorních experimentů můžeme specifikovat zpracování těchto příkazů.

6.1.2. Zpracování příkazu PM

Při zpracování tohoto příkazu se nejdříve určí doba jeho vykonávání – ta je závislá na všech třech parametrech příkazu. Z naměřených hodnot, které jsou znázorněny v grafu 5.1 můžeme určit základní rychlost pohybu robota v závislosti na prvním parametru příkazu. Tyto hodnoty jsou shrnuty v tabulce 6.1.

rychlostní stupeň	rychlost [mm/s]
1	116
2	119
3	123
4	133
5	140
6	253
7	279

Tabulka 6.1: *Závislost rychlosti pohybu robota v přímém směru na rychlostním stupni (první parametr příkazu PM)*

Z této základní hodnoty rychlosti je možné určit dobu kterou zabere uražení určité dráhy (dané druhým parametrem příkazu) v přímém směru. Po interpolaci hodnot z tabulky 5.1 můžeme určit funkci podle které se tato doba prodlužuje v závislosti na stupni zatočení (tedy třetím parametru příkazu) takto:

$$T = T_P + \frac{R \cdot T_P \cdot 5{,}6}{100} \tag{6.1}$$

T je celková doba vykonávání příkazu

$\boldsymbol{T_P}$ je doba vykonávání stejného příkazu, ale v přímém směru

R je hodnota druhého znaku třetího parametru příkazu (*ratio*)

Abychom mohli určit rychlost otáčení jednotlivých kol, je potřeba znát ještě dráhu, kterou mají ujet. Ze závěrů měření uvedených v kapitole 5.2 můžeme opět pomocí lineární interpolace určit funkci, která definuje počet kroků enkodéru jednotlivých kol v závislosti na třetím parametru příkazu.

$$S_L = S_P + \frac{R \cdot S_P \cdot 8}{100} \tag{6.2}$$

$$S_R = S_P - \frac{R \cdot S_P}{15} \tag{6.3}$$

S_L (resp. ***S_R***) je počet kroků enkodéru levého (resp. pravého) kola v případě, že první znak parametru *ratio* je „0“

S_P je počet kroků určený druhým parametrem příkazu

R je hodnota druhého znaku třetího parametru příkazu (*ratio*)

Pokud prvním znakem parametru *ratio* bude jiný znak než „0“ proběhne ještě další transformace hodnot ***S_L*** a ***S_R***. Pokud prvním znakem je:

„1“ – hodnoty obou proměnných se prohodí

„2“ – hodnoty obou proměnných se odečtou od 0

„3“ – provedou se obě předchozí transformace

Záporným počtem kroků enkodéru je míněn pohyb v opačném směru než je standardní.

Nyní, když je určená doba provádění příkazu a počet otáček jednotlivých kol (resp. počet kroků jejich enkodérů), můžeme jednoduše určit rychlost jejich otáčení.

$$V_L = \frac{S_L \cdot s}{T} \tag{6.4}$$

$$V_R = \frac{S_R \cdot s}{T} \tag{6.5}$$

V_L (resp. ***V_R***) jsou rychlosti levého (resp. pravého) kola v mm/s

S_L (resp. ***S_R***) je počet kroků enkodéru levého (resp. pravého) kola

T je čas vykonávání příkazu

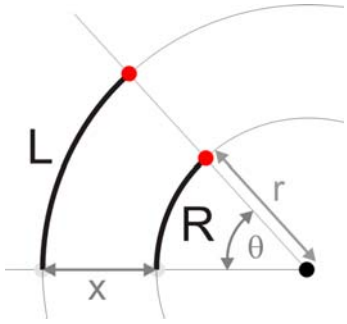
s je ujetá vzdálenost odpovídající jednomu kroku enkodéru (tedy obvod kola / 44)

Tyto hodnoty jsou v rámci modelu robota konečným výstupem zpracování příkazu PM. Do modelu pohybu je však třeba zahrnout i výpočet nové polohy robota. Ta je určena délkou dráhy uražené jednotlivými koly. Pokud se tato dráha neliší, vypočítá se nová poloha jako translace stávající polohy ve směru natočení robota. Pokud se dráhy liší, je nová poloha určena rotací původní polohy kolem středu otáčení, který lze v obecném případě vypočítat podle vztahů (6.6) a (6.7). Pokud je dráha jednoho kola nulová, bude střed rotace ležet na souřadnicích tohoto kola, pokud jsou dráhy opačné (každé kolo se točí opačným směrem) leží střed na polovině spojnice kol.

$$r = \frac{R \cdot x}{L - R} \tag{6.6}$$

$$\theta = \frac{R \cdot 360}{2\pi \cdot r} \text{ (ve stupních)} \tag{6.7}$$

Význam veličin je patrný z obrázku 6.2.



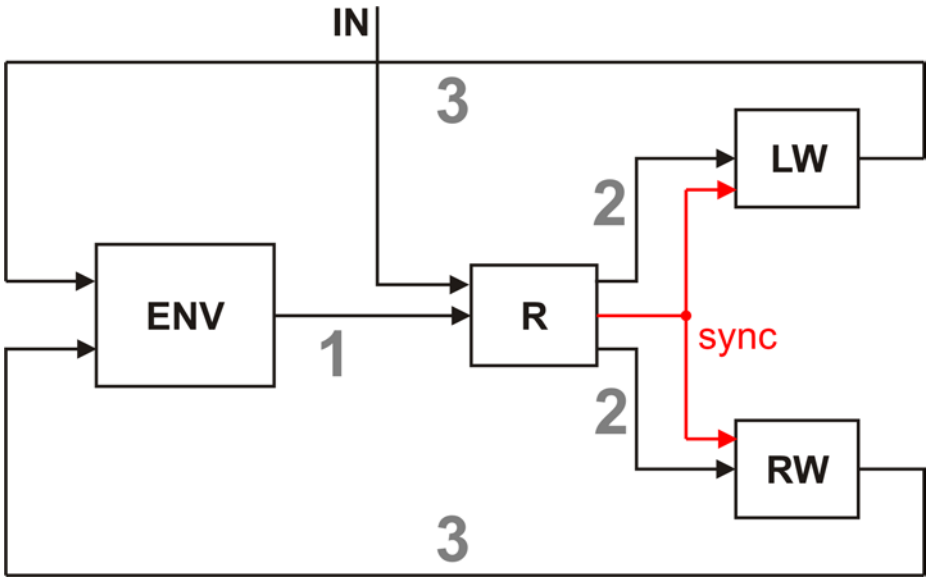
Obrázek 6.2: *Znázornění pohybu diferenciálního podvozku po kruhové dráze v případě rozdílného pohybu obou kol stejným směrem.*
L a R je dráha levého a pravého kola, x je rozchod, θ a r je úhel a poloměr rotace.

6.2. Návrh simulačního modelu

Vzhledem k záměru implementace modelu dle formalismu DEVS je potřeba abstraktní model rozdělit na podsystémy které budou tvořit samostatné funkční části modelu (atomické DEVS modely). Základní entity jsou v tomto případě pouze vlastní robot a prostředí v kterém se pohybuje. Model robota se dále dělí na kola (resp. pohonné jednotky) a řídící část robota. Ostatní systémy není třeba pro potřeby **simulace pohybu** robota uvažovat.

Vnější prostředí je v modelu chápáno jako prvek, který „obaluje“ robota a poskytuje mu informace pro jeho senzory. Z pohledu modelované funkčnosti je to tedy nadřazená entita, která také uchovává informace o skutečné poloze a natočení robota (reálný stav). Robot samotný „zná“ pouze informace, které může získat ze svých odometrů.

Toto rozdělení na modely, které jsou definovatelné dle formalismu DEVS je znázorněno na obrázku 6.2.



Obrázek 6.2: *Schématické znázornění modelu systému. Funkční bloky představují atomické DEVS modely.*

Popis schématu:

ENV – vnější prostředí s kterým je robot v interakci

R – robot

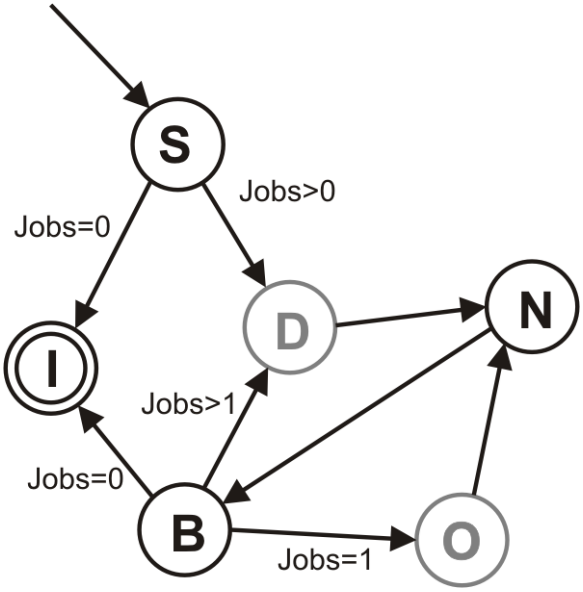
LW – levé podvozkové kolo Trilobota

RW – pravé podvozkové kolo Trilobota

6.2.1. Popis jednotlivých komponent systému na bázi DEVS

Robot

Jeho hlavní stavové veličiny jsou určeny frontou příkazů (typu PM) které má vykonat, aktuálně vykonávaným příkazem a vyhodnocenou rychlostí kol (dle aktuálního příkazu). V rámci vnitřní přechodové funkce robot nejdříve vyhodnotí první příkaz na vstupní frontě a výsledky (rychlosti kol a doba vykonávání příkazu) uloží do stavových proměnných. Zároveň se uloží příznak, který určuje, že byl právě vyhodnocen nový příkaz. Podle tohoto příznaku je tedy možné naplánovat další aktivaci přechodové funkce na nulový čas a tím okamžitě výsledky odeslat na výstupní porty. Další aktivace vnitřní přechodové funkce se naplánuje na dobu určenou délkou zpracování příkazu.



Obrázek 6.3: *Stavový stroj reprezentující fungování zpracovávání příkazů v rámci modelu robota. Přechody mezi stavy se uskutečňují jen při aktivaci vnitřní přechodové funkce, nebo pokud se systém dostal do stavu označeného šedě.*

Popis stavového stroje:

S (start) – počáteční stav

D (decode) – zpracování příkazu

N (newJob) – stav po zpracování nového příkazu

B (busy) – vykonávání příkazu

O (out) – zpracování imaginárního příkazu zastavení

I (idle) – koncový stav systému

Podvozková kola

V rámci těchto podsystémů probíhá simulace převodu řídících signálů robota na fyzickou práci. Vstupem jsou tedy signály od řídící části robota určující rychlost otáčení kol. Výstupem jsou důsledky „skutečného“ fyzického působení hnaných kol. Těmito důsledky je míněna skutečná rychlost otáčení kola, která ovlivňuje pohyb robota. Pokud by tedy tento systém simuloval podklouznutí kola, bude výstupní rychlost nižší (popř. nulová) než udává vstupní signál. Tuto skutečnost (tedy že došlo například k prokluzu kola) však není možné zjistit žádnou zpětnou vazbou, protože ani u skutečného Trilobota není možné tento stav detekovat. Podobným způsobem je možné generovat výstupy odometrie, pouze s tím rozdílem, že vliv na jejich „nepřesnost“ mají jiné události.

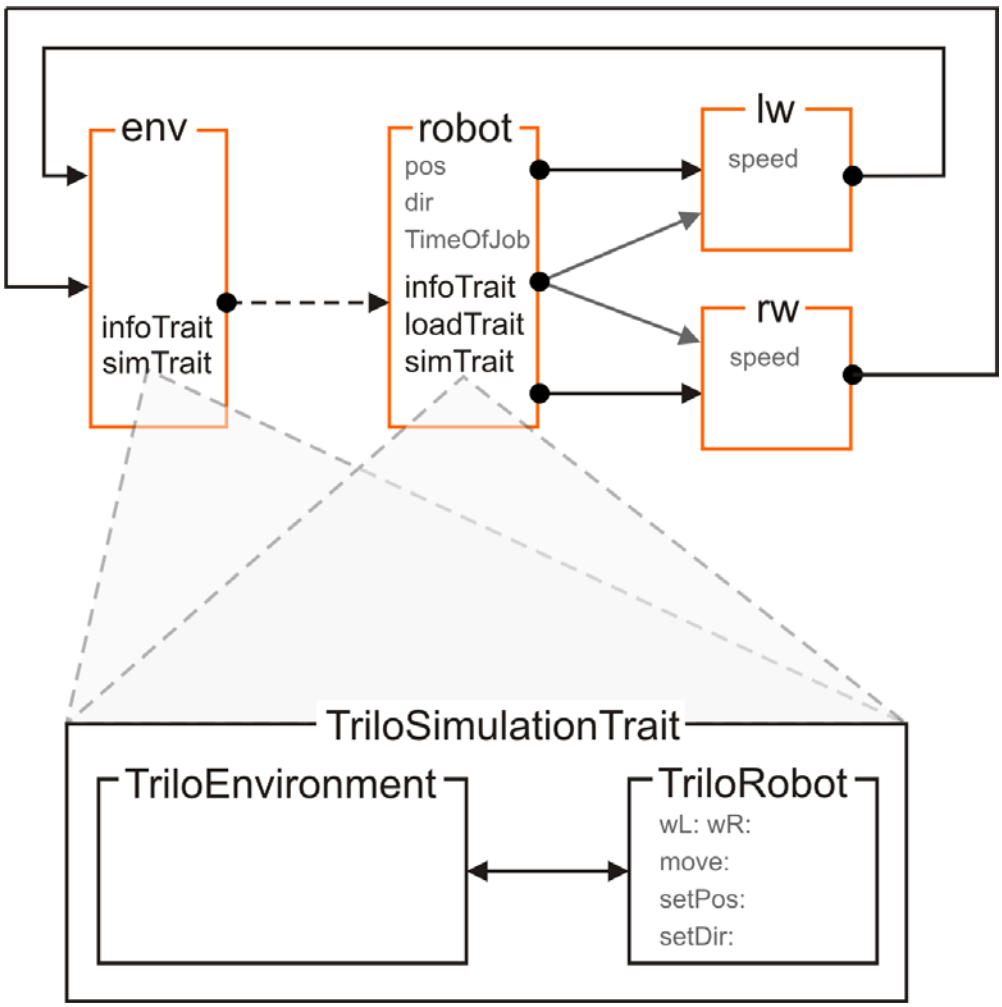
Vnější prostředí

Vnější prostředí přijímá signály o rychlosti podvozkových kol a podle nich určuje aktuální pozici a natočení robota. Protože tento objekt modeluje „chování“ prostředí, v kterém se robot pohybuje, poskytuje i základní informace o prostoru jako je možnost určit přítomnost překážky v určitém místě, což je základní funkční předpoklad pro implementaci detekce kolizí (zatím neimplementováno).

Vzhledem k tomu, že model je implementován s využitím formalismu definujícího systémy řízené diskrétními událostmi, je třeba spojité jevy nějakým způsobem diskretizovat. Aby se dal modelovaný systém považovat za diskrétní, mohli bychom například vykonání příkazu modelovat jako skokovou událost. Vizualizace takové simulace by ale nezdůrazňovala dynamiku pohybu. Proto je vhodnější použít diskretizace na takové úrovni, aby se skokové změny stavu systému daly zanedbat. Vhodnou hodnotou je tedy například 100ms trvající interval, v rámci kterého se vypočítají změny stavu systému a zobrazí se na výstupu.

7. Implementace modelu

Pro implementaci modelu jsem zvolil prostředí Squeak Smalltalk. Programování ve Smalltalku sestává v podstatě z vytváření nových tříd, resp. specializací těch, které už v systému jsou. Knihovna SmallDEVS k tomuto účelu poskytuje mj. třídy *AtomicDEVS* a *CoupledDEVS* na jejichž základě je možné zkonstruovat DEVS model. SmallDEVS ale nabízí flexibilnější přístup k modelování [11] – a to využitím prototypových objektů (*AtomicDEVSPrototype*, *CoupledDEVSPrototype*). Pro rozšíření funkčnosti atomických DEVS prototypů (např. vazba na GUI) jsem využil objektů třídy *AtomicDEVSTrait* z nichž prototypové objekty mohly dynamicky dědit. Schématické znázornění klíčových objektů výsledné aplikace (nazvané TriloSim) je na obrázku 7.1.



Obrázek 7.1: *Schéma propojení jednotlivých objektů systému. Oranžově vyznačené funkční bloky jsou objekty třídy AtomicDEVSPrototype*

Jádro systému tvoří spojovaný DEVS model, který se skládá z atomických DEVS objektů vytvořených dle výše uvedené specifikace simulačního modelu. Objekty, jež mají vazbu na vizualizaci simulace, dědí z trait objektu třídy *TriloSimulationTrait*, který zapouzdřuje instance dvou speciálních tříd. Jsou to třídy *TriloEnvironment* a *TriloRobot*, jejichž instance představují vizualizace simulovaného prostředí a robota. V nich jsou implementovány metody, které jsou zodpovědné za výpočet nové polohy robota na základě ujeté vzdálenosti kol a také vyhodnocování příkazů určených Trilobotovi.

7.1. Popis jednotlivých tříd kategorie TriloSim

7.1.1. Třída TriloEnvironment

Dědí z třídy *ImageMorph*, protože její základní úlohou je zobrazení modelu prostředí. Pro načtení požadované mapy prostředí slouží metoda *setFile:* jejímž parametrem je adresa umístění bitmapy prostředí. Další klíčové metody jsou *setScale:* a *setStart:*, pomocí nichž je možné definovat měřítko zobrazení (standardně 10, tzn. 1px = 10mm) a počátek souřadné soustavy prostředí vzhledem k počátku bitmapy. Metoda *isWall:* slouží k určení přítomnosti překážky na konkrétních souřadnicích prostředí.

7.1.2. Třída TriloRobot

Dědí z třídy *PolygonMorph* a využívá tak její základní metody pro zobrazování. Při inicializaci objektu je definována podoba polygonu, který bude reprezentovat robota (jeho základní rozměry jsou uvedeny v definici abstraktního modelu), nastaví se základní instanční proměnné – poloha a natočení robota. K tomu, aby metody tohoto objektu fungovaly korektně je však potřeba také určit vazbu na prostředí, v kterém se robot bude pohybovat. To se provádí metodou *setEnv:* jejímž parametrem je objekt třídy *TriloEnvironment*. Z objektu prostředí potom robot získává informace o měřítku v kterém se má zobrazovat, počátku souřadné soustavy atd.

Klíčová metoda této třídy je *wL: wR:*. Jejími parametry jsou vzdálenosti, které ujely jednotlivá kola. Tato metoda je zodpovědná za výpočet nové polohy robota v „jeho“ prostředí v závislosti na zmíněných vzdálenostech. K vlastnímu posunu robota na vypočtenou polohu se dají využít metody *setPos:* (pro nastavení polohy), *setDir:* (natočení robota) a *move:* (translace ve směru natočení robota).

7.1.3. Třídy TriloSimulationTrait, TriloInfoTrait a TriloLoadTrait

Tyto třídy dědí z *AtomicDEVSTrait*, zahrnují v sobě veškerou funkčnost, kterou využívají atomické DEVS modely, jež je nad rámec formalismu DEVS. Umožňují tedy přístup k objektům mimo spojovaný DEVS model. *TriloSimulationTrait* v sobě integruje instance třídy *TriloRobot* a *TriloEnvironment*. *TriloInfoTrait* slouží k výpisům základních údajů do *StringMorphu*, *TriloLoadTrait* zase zajišťuje načítání z textového pole a umožňuje tak DEVS modelu robota načíst příkazy do slotu (instanční proměnné).

7.1.4. Třída TriloSim

Je to hlavní třída (dědící z *Object*), která se stará o inicializaci celého systému. Při spouštění aplikace se nejdříve vytvoří prototypové objekty atomických DEVS modelů, trait objekty a spojovaný DEVS model. Potom se vytvoří systémové okno (třída *SystemWindow*) a do něj se vloží ovládací prvky a části trait objektů, které mají zajišťovat výstupy simulace (vizualizace prostředí a Trilobota, stavový řádek informující o poloze robota). Na závěr se okno zobrazí.

Klíčové konstrukce vytvářející jednotlivé DEVS modely jsou tedy uvedeny v metodě *initialize* této třídy. Popis jednotlivých atomických DEVS modelů:

env – prostředí

Hlavním slotem je *discTime*, který určuje čas, podle kterého se bude děj diskretizovat, resp. po jak dlouhých krocích se budou vyhodnocovat aktuální pozice robota a výsledek zobrazovat. Funkce *timeAdvance* tedy vždy vrací hodnotu rovnou tomuto slotu. Funkce externího přechodu pouze „vybírá“ hodnoty ze vstupních portů do slotů. Interní přechodová funkce vyhodnocuje v daném intervalu získané hodnoty (rychlosti kol), resp. je předává trait objektu třídy *TriloSimulationTrait*, který na základě těchto hodnot vypočítává aktuální polohu robota.

rob – robot

Model robota je klíčovou komponentou celého modelu. Jeho vnitřní přechodová funkce pracuje podle konečného automatu na obrázku 6.3. Zdrojový kód této funkce a *timeAdvance* funkce je zde:

```
rob intTransition: '
  (self status) caseOf: {

     [#start] ->[ "počáteční stav"
       ((self loadTrait jobs size) > 0) ifTrue: [
```

```
      "pokud fronta příkazů není prázdná"
            self status: #newJob. "stav -> zpracování nového příkazu"
            "vlastní načtení příkazu a zobrazení odpovídající informace"
            self currentJob: ((self loadTrait jobs) removeFirst).
            self infoTrait infoLabel job: (self currentJob).
            self jobNo: ((self jobNo)+1).
            "dekódování příkazu"
            self simTrait decodeJob: (self currentJob).
            "rychlost kol a doba vykonávání příkazu se uloží"
            self leftSpeed: (self simTrait leftSpeedTST).
            self rightSpeed: (self simTrait rightSpeedTST).
            self timeOfJob: (self simTrait timeOfJobTST).
         ] ifFalse: [
            "pokud ve frontě nečekají žádné další příkazy, neděje se nic"
                  self status: #idle.
            ].
        ].
        [#busy] -> [    "právě se provádí příkaz"
          ((self loadTrait jobs size) > 0) ifTrue: [
      "fronta příkazů není prázdná"
            self status: #newJob. "stav pro odeslání rychlostí na výstup"
            self currentJob: ((self loadTrait jobs) removeFirst).
            self infoTrait infoLabel job: (self currentJob).
            self jobNo: ((self jobNo)+1).
            "dekódování příkazu a určení rychlostí kol"
            self simTrait decodeJob: (self currentJob).
            self leftSpeed: (self simTrait leftSpeedTST).
            self rightSpeed: (self simTrait rightSpeedTST).
            self timeOfJob: (self simTrait timeOfJobTST).
         ] ifFalse: [  "fronta příkazů je prázdná"
              self currentJob: nil.
              "zastavení motorů"
              self infoTrait infoLabel job: ''END''.
              self leftSpeed: 0.
              self rightSpeed: 0.
              self timeOfJob: (Float infinity). "zastavení aktivity"
              "přechod do stavu jako po zpracování příkazu"
              self status: #newJob.
                  ].
            ].
            [#newJob] -> [
            "pokud se systém dostal do tohoto stavu, příkaz už je
dekódovaný a výsledky jsou na výstupu, může se tedy vyčkávat"
                  self status: #busy.
            ].
      } otherwise: [].
   '.

   rob timeAdvance: '
   (self status) caseOf: {
      [#start] -> [^0.]. "počáteční stav"
"koncový stav, interní přechod už není nutné vyvolávat"
      [#idle] -> [^(Float infinity).].
"právě byl zpracován příkaz, vyvolá se okamžité vystavení hodnot na
výstupní porty"
      [#newJob] -> [^0.].
"vykonává se příkaz, bude se čekat dokud nebude jeho provádění u konce"
      [#busy] -> [^(self timeOfJob).].
   } otherwise: [^(Float infinity).].
   '.
```

lw, rw – kola

Kola pouze načítají údaje ze vstupů (rychlost vypočítaná robotem) a částečně zkreslené – zde je místo pro model chyby přenosu momentu – je posílají na výstup. Externí přechodová funkce tedy při své aktivaci vloží hodnotu ze vstupního portu do slotu a převede systém do nového stavu, pro který poté *timeAdvance* odpovídá hodnotě 0. Výstupní funkce hodnotu rychlosti přečte ze slotu, zkreslí ji chybou a odešle na výstupní port. Interní převodová funkce poté převede systém do stavu vyčkávání (do stavu, pro který funkce *timeAdvance* vrátí hodnotu nekonečno), takže se bude čekat na novou vstupní hodnotu – změnu rychlosti otáčení kola.

7.2. Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je velice jednoduché. Jeho hlavním účelem je pouze vizualizace průběhu simulace a vstup základních řídících příkazů. Celé simulované prostředí včetně Trilobota je standardně zobrazováno v rozlišení 10mm na 1 pixel. Toto měřítko je tedy použitelné pro modelované prostředí o rozměrech maximálně cca 8 x 8m. Měřítko vizualizace je však možné jednoduchým způsobem upravit (viz. nápověda třídy *TriloSim*). Základní komponenty grafického uživatelského rozhraní již byly popsány v rámci trait objektů.

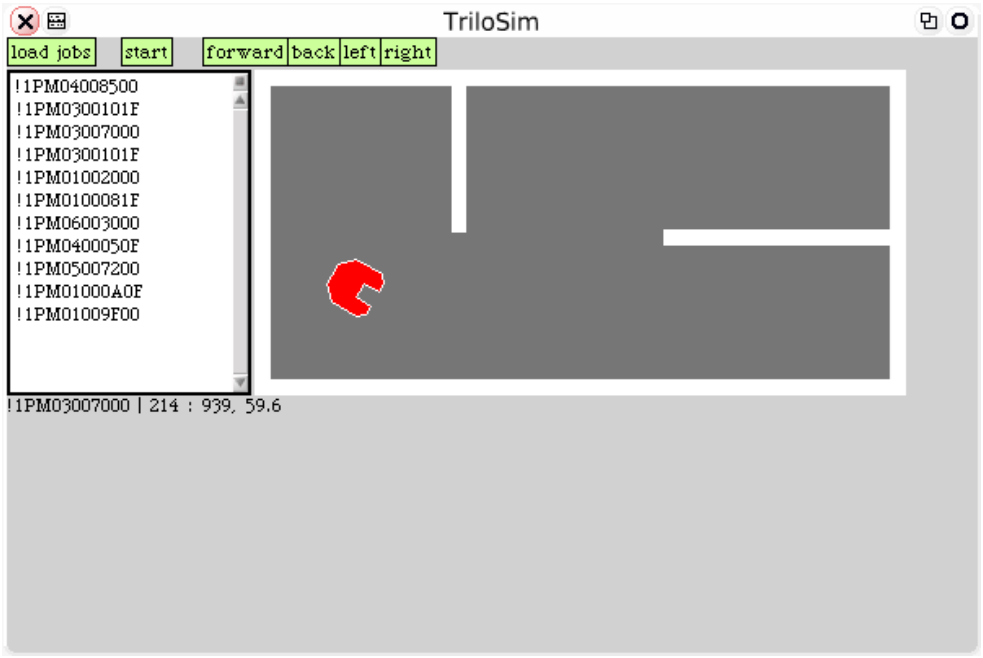
7.3. Ovládání programu

Systém lze spustit příkazem

```
TriloSim new jobs: 'jobs.txt'; env: 'mapa3.png'; openInWorld.
```

V parametru *jobs* je uvedena cesta k textovému souboru s příkazy jimiž bude řízen simulovaný Trilobot. Tento parametr je nepovinný, příkazy je možné napsat i přímo do textového pole v okně simulátoru. Parametr *env* udává cestu k bitmapovému souboru s mapou prostředí. Současná implementace detekce překážek předpokládá, že překážky jsou definovány bílou barvou – všechny ostatní barvy jsou považovány za volný prostor.

Po otevření okna simulátoru je před spuštěním simulace potřeba načíst příkazy do modelu pomocí tlačítka „load jobs“. Směrovými tlačítky je možné robota přemístit do výchozí pozice simulace, popřípadě jimi upravovat polohu robota za běhu simulace - například pro navození specifického podklouznutí, kolize apod. Tlačítkem „start“/„stop“ je možné simulaci spustit nebo pozastavit. Klíčové údaje (aktuální poloha a natočení robota, aktuálně prováděný příkaz) se zobrazují v informačním řádku pod seznamem příkazů.



Obrázek 7.2: *Okno aplikace TriloSim. V levé části je seznam příkazů, které má robot v rámci simulace vykonat, ve spodní části je aktuálně prováděný příkaz, poloha a natočení robota. V horní části jsou ovládací tlačítka. Před spuštěním simulace je nejprve potřeba načíst příkazy (load jobs), simulace se dá spustit/pozastavit tlačítkem start/stop. Směrová tlačítka slouží pro „manuální“ posouvání robota – i když není simulace spuštěná.*

8. Závěr

Cílem této práce byla tvorba modelu robota Trilobot se zaměřením na chybu, které se dopouští při řízení svého pohybu. K úspěšnému dokončení bylo tedy potřeba nejdříve důkladně prozkoumat jeho chování, podle něhož jsem potom sestavit model. Výsledkem mé práce je implementace tohoto modelu nazvaná TriloSim.

Motivací byla snaha zjednodušit návrh navigačních algoritmů pro Trilobota díky možnosti simulovat jeho pohyb. Mojí osobní motivací však bylo navíc řešení projektu, který zasahuje do velice zajímavé oblasti - robotiky. Zajímavé mi přišlo i prostředí Squeak, kterému se i přes jeho unikátní vlastnosti za deset let existence ještě nedostalo zasloužené pozornosti.

Za hlavní přínosy mé práce na tomto projektu považuji spíše jeho dílčí výsledky. Užitečné je podle mého názoru osvětlení fungování některých systémů Trilobota, které nejsou popsané v manuálu, ale i demonstrace použitelnosti nástroje SmallDEVS v oblasti, která přímo nespadá mezi základní počítačové vědy.

V rámci pokračování této práce by se dala vylepšovat implementace modelu (detekce kolizí, rozšíření definičního oboru specifikace prostředí, doplnění modelu o další senzory). Podle mého názoru je ale zajímavá také jiná cesta řešení chyby Trilobota (například doplnění mechanického nebo jednoduchého optického senzoru pro detekci skutečné změny polohy), která však nemá přímou návaznost na řešení tohoto projektu.

9. Literatura

[1] Wikipedia, otevřená encyklopedie. Dostupná na URL http://www.wikipedia.org

[2] Stránky projektu Player/Stage. Dostupné na URL http://playerstage.sourceforge.net (10. ledna 2007)

[3] Seznam zařízení podporovaných v prostředí Player/Stage. Dostupné na URL http://playerstage.sourceforge.net/doc/Player-cvs/player/supported_hardware.html (10. ledna 2007)

[4] Orság F.: Robotika - Studijní opora. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/ROB/private/ROB.pdf (10. ledna 2007)

[5] Gates B.: A robot in every home, in: Scientific American issue 2007 / 1. Dokument dostupný na URL http://www.sciam.com/article.cfm?chanID=sa006&colID=1&articleID=9312A198-E7F2-99DF-31DA639D6C4BA567 (10. ledna 2007)

[6] Robotika.cz - Český server věnovaný robotice. Dostupný na URL www.robotika.cz (10. ledna 2007)

[7] Trilobot research robot – oficiální stránky robota Trilobot. Dostupné na URL http://www.arrickrobotics.com/trilobot/index.html (10. ledna 2007)

[8] Arrick Robotics: Trilobot user guide revision B7 9/24/98, 1998. Dokument dostupný na URL http://www.arrickrobotics.com/trilobot/guide.pdf (10. ledna 2007)

[9] Graham S.: Robot's Voice Helps It Find Its Way, in: Scientific American issue 2003 / 10. Dokument dostupný na URL http://www.sciam.com/article.cfm?articleID=00064CC7-43F8-1FD6-83F883414B7F0000&sc=I100322 (10. ledna 2007)

[10] Slavíček P.: Discrete event system specification – DEVS, in: Proceedings of the 9th Conference and Competition STUDENT EEICT 2003, VUT v Brně, 2003, s. 3, ISBN 80-214-2378-1

[11] Janoušek V.: On the Prototype-Based Object Orientation Modeling and Simulation, In: Proceedings of Advanced Simulation of Systems 2006, Ostrava, MARQ, 2006, s. 241-246, ISBN 80-86840-26-3. Dokument dostupný na URL http://www.fit.vutbr.cz/~janousek/publications/2006-prototype-based-mas.pdf (10. ledna 2007)

[12] Janoušek V., Kironský E.: Exploratory Modeling with SmallDEVS, In: Proceedings of ESM 2006, Toulouse, France, 2006, s. 122-126, ISBN 90-77381-30-9. Dokument dostupný na URL http://www.fit.vutbr.cz/~janousek/publications/2006-esm-smalldevs.pdf (10. ledna 2007)

[13] Stránky projektu SmallDEVS. Dostupné na URL http://perchta.fit.vutbr.cz:8000/projekty/10 (10. ledna 2007)

[14] Stránky projektu Squeak smalltalk. Dostupné na URL http://www.squeak.org (10. ledna 2007)

[15] Stránky projektu Squeak smalltalk. Dostupné na URL http://www.squeakland.org (10. ledna 2007)

[16] České stránky věnované Squeak smalltalku. Dostupné na URL http://www.comtalk.net/ (10. ledna 2007)

[17] Křivánek P.: Squeak: návrat do budoucnosti. Dokument dostupný na URL http://www.root.cz/clanky/squeak-navrat-do-budoucnosti/ (10. ledna 2007)

[18] Sharp A.: Smalltalk by example: the Developer's Guide, McGraw Hill Text, 1997. Dokument dostupný na URL http://www.iam.unibe.ch/~ducasse/FreeBooks/ByExample/ (10. ledna 2007)

Příloha: Hodnoty naměřené při laboratorních experimentech

Měření rychlosti

příkaz	odo	vzdálenost [mm]	čas [s]	rychlost [mm/s]
1PM01003008	48	307	3,2	96
1PM01003008	48	307	3,1	99
1PM01003008	48	307	3,3	93
1PM01005008	80	511	5	102
1PM01005008	80	511	5,2	98
1PM01005008	80	511	5,2	98
1PM01005000	80	511	4	128
1PM01005000	80	511	4	128
1PM01005000	80	511	3,9	131
1PM0100500F	80	511	7,9	65
1PM0100500F	80	511	7,9	65
1PM0100500F	80	511	8	64
1PM01003000	48	307	2,9	106
1PM01003000	48	307	3	102
1PM01003000	48	307	3	102
1PM0100300F	48	307	5,4	57
1PM0100300F	48	307	5,1	60
1PM0100300F	48	307	5,2	59
1PM02003000	48	307	2,5	123
1PM02003000	48	307	2,6	118
1PM02003000	48	307	2,6	118
1PM03003000	48	307	2,6	118
1PM03003000	48	307	2,4	128
1PM03003000	48	307	2,5	123
1PM05003000	48	307	2,3	133
1PM05003000	48	307	2	153
1PM05003000	48	307	2,3	133
1PM04003000	48	307	2,3	133
1PM04003000	48	307	2,3	133
1PM04003000	48	307	2,3	133
1PM06008000	128	817	3,3	248
1PM06008000	128	817	3,2	255
1PM06008000	128	817	3,2	255
1PM07008000	128	817	2,9	282
1PM07008000	128	817	2,9	282
1PM07008000	128	817	3	272

Měření odchylky v ujeté, detekované a navigované vzdálenosti

Příkaz	Požadovaná vzdálenost		Skutečná vzdálenost [mm]	Detekovaná vzdálenost		Chyba odometrie		Chyba navigace [mm]
	hex	[mm]		hex	[mm]	[mm]	kroky odometru	
!1PM01008000	0080	818	835	0085	849	14	3	17
!1PM01008000	0080	818	850	0085	849	-1	0	32
!1PM01008000	0080	818	850	0085	849	-1	0	32
!1PM0100F000	00F0	1533	1554	00F3	1552	-2	0	21
!1PM0100F000	00F0	1533	1560	00F4	1558	-2	0	27
!1PM0100F000	00F0	1533	1558	00F4	1558	0	1	25
!1PM07010000	0100	1635	1753	0104	1661	-92	-14	118
!1PM07008000	0080	818	930	0085	849	-81	-12	112
!1PM05010000	0100	1635	1748	0105	1667	-81	-12	113
!1PM05010000	0100	1635	1745	0104	1661	-84	-13	110
!1PM02010000	0100	1635	1708	0102	1648	-60	-9	73