Koncept Hayekova stroje pro řízení robotů Khepera IV

Lukáš Mamula

Slezská univerzita v Opavě, Filozoficko-přírodovědecká fakulta v Opavě
Bezručovo náměstí 13, 74601 Opava
mamula.lukas@gmail.com

Abstrakt

V tomto článku si ukážeme koncept disertační práce, na které v současnosti pracujeme. Článek vychází z předchozích studií, které byly zasvěceny fuzzy logice v multiagentových systémech. Díky doposud zjištěným informacím a použitých programových knihoven jsme nyní schopni náš vyvíjený simulátor pohybu robota upravit tak, aby bylo možno s naší vytvořenou aplikací řídit nejen robota v abstraktním prostředí, ale hlavně fyzického robota. S fyzickým robotem jsme schopni postoupit ve vývoji aplikace do mnohem robustnějších rozměrů. Ukážeme si v článku jak budeme pokračovat v dosažení toho, abychom vytvořili umělou inteligenci, která bude schopna ovládat robota na dálku. Předpokládáme vznik nových situací, do kterých se robot může dostávat a které nemohly vzniknout v abstraktním světě s omezenou množinou stavů světa. Pro náš výzkum jsme zvolili robota: *Khepera IV*. Tito roboti jsou vybaveni ultrazvukovým a také infračervenými senzory pomoci nichž budeme schopni vybudovat bázi znalostí a reprezentaci prostředí.

1 Úvod

Umělá inteligence a přístupy k této problematice spadají do vědní disciplíny, která si zasluhuje obzvlášť velkou pozornost. I v tomto článku se podíváme na jeden z možných způsobů, jak dosáhnout takového jevu. Jevu, který se pozorovateli fyzického zařízení (robota) nebo softwarové aplikace, může jevit jako samostatně fungující entita.

V našem článku se zaměříme na vývoj umělé inteligence pomocí konceptu Hayekova stroje. Tento stroj je značně odlišný od toho, co popisuje ve své knize A. Kubík (2004). Náš koncept vychází z diplomové práce, která je také uvedena v literatuře. Již v první kapitole si ukážeme, jakým způsobem jsme vytvořili koncept mechanismu. Díky vytvořeného mechanismu jsme mohli ve virtuálním prostředí vidět a udělat si představu, jak by mohla naše aplikace ovládat robota.

Během vývoje jsme se setkali s několika problémy, které v tomto článku také zmíníme a samozřejmě zmíníme i možné řešení, ke kterému jsme došli. Podíváme se na využití fuzzy logiky ve spolupráci s externími knihovnami, které používáme v našem projektu a na přínosy těchto knihoven. Čtenáře zasvětíme do parametrů robota *Khepera IV* a okrajově nastíníme náš záměr. Záměrem myslíme ovládání výše zmíněného robota pomocí naší vyvíjené aplikace.

Nyní se posuneme k následující kapitole a podrobně si vysvětlíme naši pozici ve vývoji a kam chceme dále směřovat.

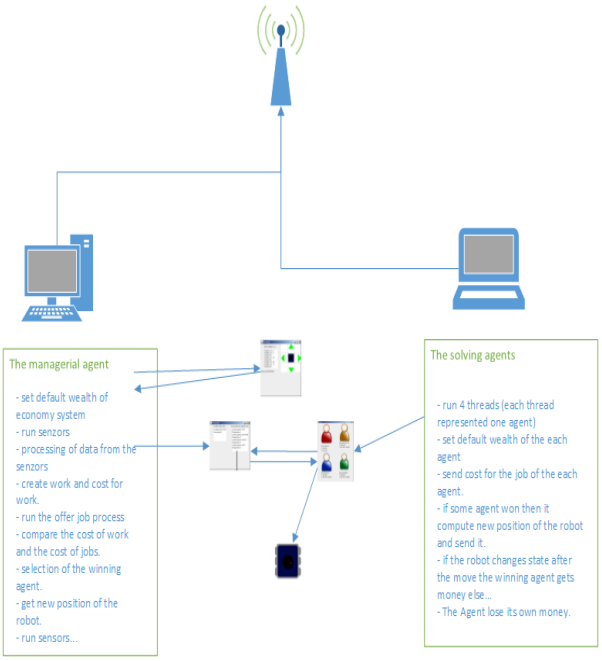
2 Aktuální stav

Na začátku této kapitoly je potřeba čtenáře upozornit, že v době, kdy byl tento článek psán (březen 2015), jsme pracovali na velice důležité části tohoto projektu, kterou jak doufáme využijeme při práci s fyzickým robotem. Tato část je vyvíjena paralelně spolu s klíčovou aplikací a bude použita jako softwarový balík pro další studium. Balík, který vytvoříme bude inferenčním mechanismem pro transparentní intenzionální logiku, o které budeme diskutovat v některém z dalších článků a která je velice dobře zpracována v knize *M. Duží (2012)*. Ale proč tuto informaci zmiňujeme? V následujících měsících byly veškeré údaje, které zde v tomto článku uvedeme aplikovány do fyzického robota. Takže ačkoliv článek vychází z teoretických poznatků, tyto poznatky jsou v době psaní tohoto článku již ověřovány v praktickém měřítku.

Nyní se přesuneme k první části této kapitoly, kde si velice jednoduše popíšeme kooperační mechanismus, pomocí něhož zajišťujeme výsledný pohyb robota ať už v reálném nebo virtuálním prostředí. Podrobněji je tento mechanismus popsán *L. Mamula (2014)*.

2.1 Popis kooperačního mechanismu

Pro popis kooperačního mechanismu použijeme Obr. 1. Na tomto obrázku můžeme vidět jednoduchou topologii, ve které máme dva počítače. Druh počítačů pro naše účely není až tak důležitý. Operační systém používáme od společnosti Microsoft, ale protože je aplikace psaná v Java SE, je možné použít i jiné platformy. Celá topologie by v případě simulace pohybu ve virtuálním prostředí mohla být navržena i jako model *peer to peer*.



Obr. 1: Popis kooperačního mechanismu

Přejdeme k vysvětlení kooperačního mechanismu. V našem stroji nepoužíváme genetické algoritmy, tak jako bylo použito v knize A. Kubíka. Zaměřili jsme se v našem případě na jeden z mnoha ekonomických mechanismů s přihlédnutím na myšlenky F. von Hayeka.

Jako kooperační mechanismus jsme zvolili nabídku a poptávku na trhu práce. Mechanismus jsme modifikovali pro vlastní účel. Tímto mechanismem jsme vytvořili vnitřní procesy, které nám umožnili libovolně pohybovat s robotem (v rámci 2D prostoru).

Na Obr. 1 máme celý mechanismus popsán a protože jsme se o tomto mechanismu zmiňovali již v dřívějších pracích, popíšeme si tento mechanismus velice zkráceně. V následující kapitole budeme vidět stavový diagram, ve kterém si znázorníme celý mechanismus.

Kooperační mechanismus je vytvořen ve smyslu odměňování a trestání. Na začátku má manažerský agent, který se stará o systém nabídek práce a přiřazování práce, nastaveno defaultní množství bohatství. Po nastavení bohatství jsou spuštěny senzory, z kterých se vytvoří řetězec reprezentující aktuální polohu robota. Ze získaného řetězce se vytvoří nabídka práce a cena za práci. Mechanismus vytváření ceny za práci je trochu složitější a záleží na více podmínkách. Nejdůležitější podmínkou je, zda agent svou prací splní stav robota. Stavy robota jsou tři: *nese, sbírá a potrava.*

Spolu s výše popsanými procesy je paralelně na druhém PC spuštěna aplikace, jenž reprezentuje řešitelské agenty. I v této aplikaci je jednotlivým agentům přiřazeno defaultní bohatství.

Pokud máme vytvořenou cenu za práci, manažerský agent kontroluje, zda jsou přihlášeny řešitelské agenty, které budou chtít získat nabízenou práci. Řešitelské agenty z paralelně běžící aplikace se přihlásí manažerskému agentu. Spolu s přihlášením (jméno agenta) poskytnou manažerskému agentu také cenu, za kterou svou práci budou vykonávat. Tato cena je vytvářena na základě více informací.

Cena, za kterou budou řešitelské agenty vykonávat svou práci se odvíjí od aktuálního bohatství. Jak už jsme naznačili výše, řešitelské agenty stejně jako manažerský agent dostanou na začátku defaultní množství bohatství. V případě řešitelských agentů se toto bohatství mění více dynamičtěji něž u manažerského agenta. V případě řešitelských agentů se změna bohatství projevuje získáním práce a následným vyhodnocením jeho práce. Pokud některý z řešitelských agentů získá práci a vykoná svou práci, je kontrolováno, zda po vykonání práce se změnil stav robota, ve kterém se robot nacházel. Když se zjistí, že stav robota je stále stejný, pak řešitelský agent, který provedl práci, ztratí část svého bohatství. V opačném případě se k jeho bohatství přičte odměna. Tato odměna se odečítá z bohatství manažerského agenta (bohatství systému).

Pokud se některý z řešitelských agentů dostane do stavu, že není schopný pracovat z důvodu malého bohatství, může požádat o pomoc manažerského agenta. Ten mu zašle z jeho bohatství částku, aby se řešitelský agent mohl účastnit dalších řízení.

Tento proces částečně simuluje palivo resp. stav baterie ve fyzickém robotu. Pokud manažerskému agentu dojde bohatství, pak se to projeví tím, že robot přestane pracovat z nedostatku energie. A s tímto procesem také souvisí jiný proces a to přechod do stavu *potrava*. Robot se dostane do stavu potrava pouze v té chvíli, pokud bohatství manažerského agenta dosáhne hraniční úrovně. Za tohoto předpokladu robot začne vyhledávat v prostoru zdroj potravy (energie).

Zatím jsme v tomto projektu neřešili problém reprezentace prostředí. V našem simulátoru jsme měli umístěný zdroj potravy a místo, kde robot nosil odpad, na jednom neměnném místě. Jeho úkolem bylo prohledávat prostor bez potřeby zaznamenávat nebo si vytvářet mapu. Na tento problém se podíváme v některé z dalších kapitol.

Nyní jsme si připomněli, jak pracuje náš kooperační mechanismus. Teď se přesuneme do další kapitoly, kde si popíšeme, jak pracuje proces *výběrového řízení.* Výběr vítězného agenta, který provede práci a následně je za ní ohodnocen, tak jak jsme popsali výše.

2.2 Báze znalostí s použitím fuzzy logiky

V této kapitole se zaměříme na fuzzy logiku, kterou jsme také řešili v článku uvedeném jako *L. Mamula (2014).* V tomto článku se nebudeme příliš podrobně zabývat touto oblastí, pouze si předvedeme fakta, na které jsme v průběhu přišli popřípadě řešení, které jsme zjistili.

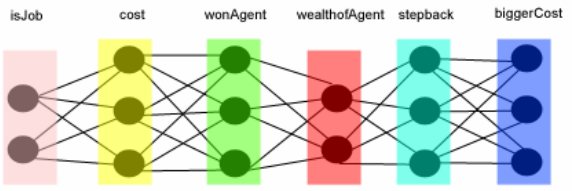
Fuzzy logika v našem systému představuje velmi důležitou část. Tuto součást systému používáme v procesu výběrového řízení, kde se snažíme co

nejefektivnějším způsobem vybrat agenta, který získá práci. Výběr vítězného agenta je závislý právě na výsledku z procesu fuzzifikace a defuzzifikace.

Pro práci s fuzzy logikou jsme použili Java knihovnu *jFuzzyLogic*. Podrobný popis této knihovny nalezneme na webových stránkách, kde také můžeme tuto knihovnu stáhnout [1]. Díky této knihovně jsme vytvořili pravidla pro náš systém. Nedílnou součástí tvorby báze znalostí a samotná práce s fuzzy logikou bylo navrhnout fuzzy množiny tak, abychom mohli pracovat se systémem, jak potřebujeme.

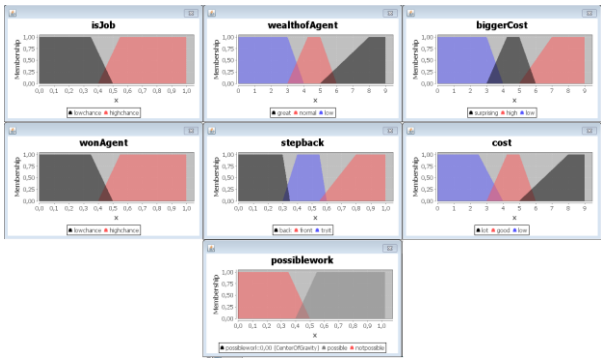
Na následujícím obrázku můžeme vidět, jak z již vytvořených fuzzy množin vytváříme pravidla. Můžeme si všimnout, že zde vzniká obrazec bipartitního grafu.

Pokud se podíváme na Obr. 2, je zřejmé, že zde kombinujeme všechny množiny. Tímto kombinováním získáme kolem 320 pravidel. Ne každé pravidlo je důležité, proto zde uplatníme na některé pravidla substituci, čímž snížíme množství pravidel.



Obr. 2: Znázornění tvorby pravidel.

Na Obr. 2 nám jednotlivé body znázorňují konkrétní fuzzy množinu. Fuzzy množiny jsou uzavřeny vždy barevným obdélníkem. Popisek nad barevným obdélníkem nám označuje název fuzzy množin. Na dalším obrázku lépe pochopíme jednotlivé názvy a propojení v našem grafu.



Obr. 3: Fuzzy množiny a jejich grafová reprezentace. Generováno programově.

Na Obr. 3 máme vytvořeny konkrétní fuzzy množiny zobrazeny v soustavě souřadnic. Z obrázku je patrné, že máme šest grafů, přičemž sedmý slouží k získání výsledného crisp-u. Díky využité programové knihovně jsme mohli upustit od plánované metody váženého průměru, kterou jsme chtěli použít pro proces defuzzifikace. Nyní můžeme použít metodu Centroid, která je mnohem přesnější než výše zmiňovaná metoda váženého průměru.

Po defuzzifikaci vstupních dat, získáme nové číslo z množiny reálných čísel. Toto číslo nám reprezentuje nebo spíše určí, který z agentů se nejvíce hodí pro vykonání aktuální práce.

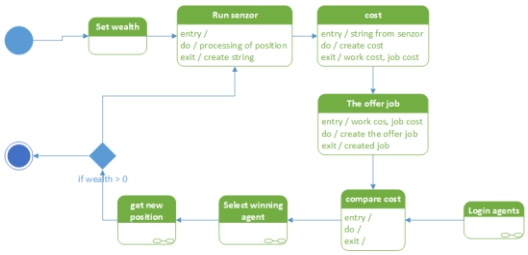
Používání fuzzy logiky, kterou jsme si v této kapitole popsali, je velice důležité v procesu *výběrového řízení*. Tato metoda nám umožňuje velice efektivně vybrat vítězného agenta, díky čemuž se pohyb robota zdá více přirozený.

V další kapitole se podíváme na fyzického robota, s kterým v tomto projektu budeme pracovat.

3 Hayekův stroj a Khepera IV

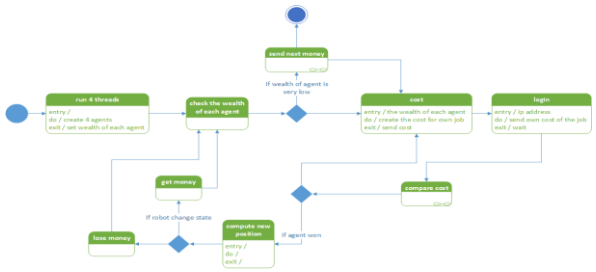
V této kapitole se podíváme na několik obrázků stavových diagramů a projdeme si, jaké úpravy musí být provedeny, abychom náš kooperační mechanismus mohli použít pro ovládání robotů typu Khepera IV.

Na prvním obrázku v této kapitole přesněji na Obr. 4 máme znázorněn stavový diagram, který reprezentuje kooperační mechanismus tak, jak jsme si jej vysvětlili v předchozí kapitole. Nyní se zaměříme na některé ze stavů, které na obrázku můžeme vidět.



Obr. 4: Stavový diagram kooperačního mechanismu

První stav z Obr. 4, na který se zaměříme je stav login. Tento stav v našem diagramu tvoří *substroj*, který si můžeme představit tak, jak můžeme vidět na následujícím obrázku.

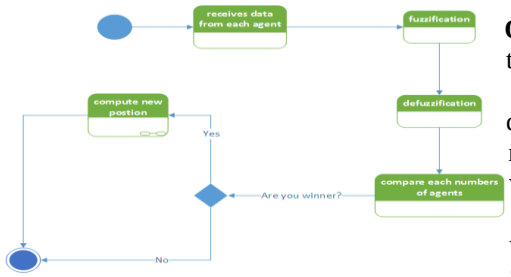


Obr. 5: Stavový diagram znázorňující životní cyklus řešitelského agenta.

Obr. 5 nám zobrazuje výše zmíněný *substroj* s názvem *login*. V tomto diagramu můžeme přesně vidět posloupnost kroků, od začátku životního cyklu agenta až po ukončení cyklu.

1 http://jfuzzylogic.sourceforge.net/html/index.html

Následující obrázek nám znázorňuje stavový diagram, kde můžeme vidět výběr vítězného agenta. Opět se jedná o jeden substroj z Obr. 4.



Obr. 6: Stavový diagram výběru vítězného agenta.

Na Obr. 6 máme zakreslen proces výběrového řízení, které jsme si popisovali v předchozích kapitolách. Pro naše účely jsme popis mechanismu vyčerpali do dostatečné úrovně a nyní se můžeme přesunout dále v našem článku.

3.1 Khepera IV

Tento typ robota je ideální pro náš výzkum. S ohledem na jeho hardwarové vybavení jsme schopni aplikovat náš koncept tak, aby robot byl schopný pracovat v prostoru bez větších chyb. Technické parametry tohoto robota najdeme na stránkách, které jsou uvedeny u popisku Obr. 7.

Obr. 7: Khepera IV. Dostupné na WWW: *http://www.k-team.com/mobile-robotics-products/khepera-iv*

Robot Khepera IV se pohybuje v 2D světě (pokud nepočítáme čas jako další dimenzi). Tento fakt využijeme, protože v současnosti také nepočítáme s pohybem v 3D prostoru (možná v budoucích studiích). Další zajímavostí může být, že robot má v sobě C/C++ překladač. Zde bychom chtěli upozornit na velkou výhodu naší práce. Protože naše aplikace pracuje s vlastními protokoly, které jsme navrhli pro komunikaci mezi agenty, vzniká zde nezávislost na jazyce, v jakém bude programován robot. Tuto informaci si více rozebereme.

Zprávy, které si mezi sebou agenty posílají, jsou zasílány skrze standardní TCP/IP. Toho jsme využili v náš prospěch, abychom zaručili *„mlutiplatformní programování“*. Robot může být programován v libovolném jazyce, musí být dodržen pouze formát přijímaných zpráv a formát odesílaných zpráv. Pokud se formát zpráv zachová, pak druhá strana bude vždy vědět, o jaký příkaz jde, ať už je robot programován v jakémkoli jazyce.

3.2 Reprezentace prostředí

Již v některé předchozí kapitole jsme zmínili absenci reprezentace prostředí v našem projektu. Absence reprezentace prostředí není z důvodu, že bychom nechtěli řešit tuto problematiku, právě naopak. Rozsah řešení této problematiky je značně složitý a proto z tohoto důvodu při rozdělení naší práce na dílčí subprojekty, jsme se museli i my zaměřit pouze na jeden problém, což byl kooperační mechanismus. Nutnost řešení reprezentace prostředí nebylo až tak důležité také z toho důvodu, že jsme pohyb robota simulovali v uměle vytvořeném prostředí našeho simulátoru.

Ovšem dostali jsme do fáze projektu, kdy musíme začít řešit i prostředí, do kterého robota umístíme. Reprezentaci prostředí můžeme řešit mnoha způsoby. My se zaměříme pouze na jeden, který se pokusíme v následujícím textu popsat.

Mapování prostoru je velice těžký problém. V našem případě budeme prostor mapovat způsobem, který je analogicky podobný způsobu mapování prostředí slepým člověkem, který je umístěný do neznámého prostředí.

Na začátku robot nebude mít žádnou představu o tom, kde se nachází nebo, co má kolem sebe. Postupným pohybem a získáváním dat ze senzorů si bude černá nezmapovaná místa zaznamenávat jednoduchými značkami. Protože v našem simulátoru robot nemohl za definované hranice, i v novém řešení bychom robota omezili pouze na ohraničenou oblast definovanou jednotkami vzdálenosti (cm, m). Pokud by robot danou oblast zmapoval celou, uložil by si danou oblast. Ve chvíli, kdy by naše aplikace zjistila, že má zpracovaný celý prostor například tím, že by robot prohledával stále stejný prostor, mohl by si systém vytvořit spojitý model prostředí (například půdorys bytu). Na základě tohoto modelu bychom upravili složitost některých procesů v systému v některých z etap životního cyklu manažerského agenta. Manažerského agenta z toho důvodu, protože se stará o spouštění senzorů a zpracovávání dat ze senzorů. Robot by nemusel tak často využívat senzorů pro skenování okolí. Zde bychom mohli opět použít analogicky podobný příklad se slepcem. Pokud jsme slepý člověk, který obývá nějaký prostor, většinou si tento prostor nastavíme tak, aby se rozložení jeho vybavení měnilo co nejméně. Ve své domácnosti se pak pohybujeme na základě paměti. Pamatujeme si rozložení nábytku a jiného vybavení v prostoru.

Toto řešení má své nevýhody. Jedna z největších nevýhod je i malá změna prostředí. Problém by se dal vyřešit tak, že bychom senzory nechali kontinuálně běžet na pozadí aplikace. Pokud bychom na senzorech zaznamenali kritickou hodnotu, tedy hodnotu, kdy by

mohlo dojit ke kolizi s jiným objektem, vypnul by se *„autopilot“*, který řídil robota podle zaznamenaného prostředí. Po vypnutí *autopilota,* bychom opět začali mapovat prostředí do té chvíle než bychom zaznamenali všechny změny.

Reprezentací prostředí se budeme jistě zabývat v některé další studii, kde bychom také rádi použili naši studii transparentní intezionální logiky. S ohledem na rozsáhlost tohoto problému, ale musíme postoupit dále a nezacházet do příliš velkých detailů.

3.3 Adaptace kooperačního mechanismu

O kooperačním mechanismu, který v našem projektu používáme, jsme si v tomto článku řekli spoustu užitečných informací. V této kapitole již nebudeme věnovat tolik času dalšímu vysvětlování, zaměříme se zde na konkrétní úpravy výše popsaného mechanismu. Úpravy poslouží pro adaptaci naší aplikace k ovládání fyzického robota nikoli robota v simulátoru.

Nejdůležitější úpravou našeho mechanismu je pohyb robota. V simulátoru se robot pohyboval v prostoru souřadnic. V reálném prostředí se robot po souřadnicích pohybovat nemůže. Zde musíme souřadnice nahradit jednotkami vzdálenosti. Takže první velká úprava je výpočet, který vykonávají řešitelské agenty. Nyní řešitelské agenty nebudou vypočítávat novou souřadnici robota, ale vzdálenost, o kterou se robot posune určitým směrem.

Další programovou úpravou bude vytvoření nového stavu. V předchozích kapitolách jsme si říkali, že robot může být ve třech stavech: *nese, sbírá a potrava*. Tyto stavy nadále zůstanou, ale nyní musíme počítat ještě s jedním faktem. Robot na začátku nebude mít představu o tom, kam má odpad odnést nebo kde nalezne potravu (dobíjecí zařízení). Proto nám přibude jeden nový stav a to *mapuje*.

Problém s nalezením napájení můžeme vyřešit velice jednoduše. Místo odkud robot vyjede, bude nastaveno jako defaultní a v jeho reprezentaci prostředí bude zaznačeno hned od začátku jako místo, kde nalezne zdroj napájení. Ve chvíli, kdy přejde na režim autopilot změní svůj stav na sbírá a jeho práce může začít.

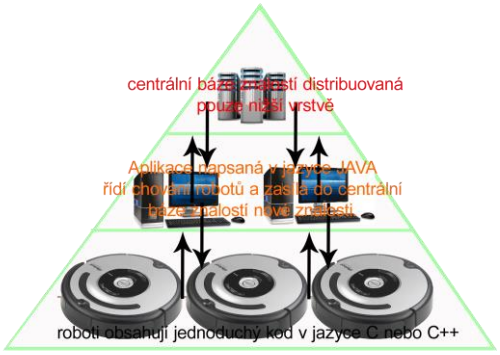
Důležitá programová úprava nastane v souvislosti se stavem *nese*. Doposud jsme se zde bavili pouze o pohybu robota. Co ale bude cílem robota? Kdy se změní stav *sbírá* na stav *nese*? Tyto otázky jsou důkazem složitosti našeho projektu. Součástí dalších studií, jak už čtenáři může být patrné, budou algoritmy rozpoznávání obrazů. Robot bude muset rozpoznat objekt, který mu nahrajeme do distribuované databáze objektů. Časová náročnost tohoto řešení, které budeme uvádět do praxe, bude značná a proto se nebudeme nyní pouštět do dalších teoretických spekulací.

Další změnou kooperačního mechanismu je práce s centrální bází znalostí. Tímto problémem se zde nebude zatím příliš zabývat, opět tuto část projektu necháváme pro budoucí studie a publikace. Pouze zde naznačíme možná řešení, která ovšem nejsou ověřena. Robot bude v čase mapování sbírat zkušenosti. Tyto zkušenosti budou zapsána v podobě pravidel, které se budou buď dávkově zasílat do centrální báze znalostí nebo kontinuálně (realtime).

4 Praktické využití

Zaměříme se na praktické využití práce, kterou jsme v předchozích kapitolách v tomto článku popisovali. Praktické využití tohoto projektu může být velice rozmanité. Podíváme se na jednu z možností, kterou si trochu popíšeme.

Na Obr. 8 máme znázorněnou základní myšlenku použití našeho konceptu. Můžeme zde vidět pyramidové znázornění, kde jsme se snažili zobrazit tři vrstvy resp. tři úrovňový pohled na naši aplikaci.



Obr. 8: Praktické využití konceptu Hayekova stroje pro řízení robotů.

Základna pyramidy je tvořena obrázky robotů. V této vrstvě poukazujeme na již zmiňovanou informaci, že robot může být programován v libovolném jazyce. Komunikace probíhá skrze předem navrhnutých postupů komunikace.

Další vrstvu obrázku znázorňují počítače. Zde jsme se snažili znázornit přímo naši řídící aplikaci, na které pracuje náš kooperační mechanismus. Význam šipek mezi spodními vrstvami by měl být čtenáři zřejmý z předchozích kapitol.

Poslední vrstva znázorňuje datové uložiště. Zde se dostáváme k hlavní myšlence. Jedná se o to, že bychom chtěli docílit, aby naše aplikace dokázala ovládat libovolný počet robotů. S tím samozřejmě souvisí i vytížení výkonu fyzického zařízení (stolního počítače, notebooku), na kterém naše řídící aplikace poběží.

Řídících aplikací může být spuštěno také větší množství na různých zařízeních a každá řídící aplikace bude řídit jen určitou množinu robotů. Tohoto výsledku jsme schopni dosáhnout, avšak myšlenka jde dále. Pokud bychom vytvořili aplikaci, která by byla schopna řídít nějakou množinu robotů, chtěli bychom také mít distribuovanou bázi znalostí. Tato báze

znalostí by se dynamicky rozšiřovala, pomocí sběru dat každým robotem. Roboti by zasílaly nové situace nebo poznatky do řídící aplikace. Každá řídící aplikace by následně ukládala nové situace nebo poznatky do centrální báze znalostí. Tím bychom dosáhli obrovských rychlostí učení našeho systému jako celku. Každy robot s minimálním programovým vybavením by se mohl rozhodovat na základě informací, které by zjistili v pruběhu života jiní roboti.

Myšlenka by se dala použít i v komerčním sektoru. Problém, který v tomto řešení vzniká, je komunikace skrz veřejné a privátní sítě. Protože v našem řešení počítáme se vzdálenými přístupy k datovému uložišti a také je zde nutná komunikace mezi agenty v řídící aplikaci. Neposlední a důležitá komunikace je také mezi řídící aplikací a samotným robotem. Pokud bychom zajistili, že komunikace bude probíhat bez chyb a bez možných výpadků, pak by bylo možné naše praktické řešení aplikovat.

Tento problém by mohl vyřešit vzdálený přístup skrz VPN [2] technologie popřípadě jinou síťovou technologií. Ovšem hledání řešení výše zmíněného problému, budeme věnovat čas v některém z dalších článků.

5 Závěr

Závěrem této práce bychom si měli vyhodnotit informace, které jsme získali. V první části tohoto článku jsme si připomněli jaký kooperační mechanismus jsme vybrali v našem projektu. Tento mechanismus byl stručně popsán s ohledem na starší práce, kde jsme se na kooperační mechanismus přímo zaměřili. Zmínili jsme se zde o fuzzy logice, kterou v našem projektu používáme za koprodukce Java knihovny.

Další část této práce byla zaměřena na popis fyzického robota, s kterým budeme pracovat. Zaměřili jsme se na popis a možnost reprezentace prostředí. V poslední fázi této části článku jsme si vysvětlili, jak používáme kooperačního mechanismu při ovládání robota.

Poslední část tohoto článku jsme zaměřili na praktické využití našeho výzkumu. Jak bylo možno vidět, využití našeho systému není zatím mnoho. My ale věříme, že v průběhu vývoje získá náš systém mnohem komplexnější podobu, která již bude schopna konkurovat jiným nástrojům k řízení robotů, jenž jsou na trhu.

Projekt jak již bylo zmíněno na začátku je stále v pohybu. Proto při čtení tohoto článku, je nutno brát v úvahu také pokrok při vývoji, popřípadě samotnou změnu cesty ve vývoji.

2 Virtual private network

Poděkování

Tento článek vznikl za podpory projektu Biologicky motivované výpočetní modely, aplikace (2013-2014-2015) s kódem projektu SGS/24/2013.

Literatura

Mamula, L. (2013). Simulace hněvu v Hayekově multiagentovém systému. Diplomová práce, Opava.

Kubík, A. (2004). Inteligentní agenty. Computer Press, a.s. Brno. ISBN: 80-251-0323-4

Spell B. (2002). Java Programujeme profesionálně. Computer Press, a.s. Praha. ISBN: 80-7226-667-5

Thagard P. (2001). Úvod do kognitivní vědy. Mysl a myšlení. Portál, s.r.o. Praha. ISBN: 80-7178-445-1

Petrů M. (2007). Fyziologie mysli. Triton. Praha. ISBN 978-80-7254-969-6

Mamula L. (2014). Fuzzy logika v multiagentovém systému s umělou ekonomií. Kognitivní věda a umělý život. 2: 135-139.

Duží M. (2012). TIL jako procedurální logika. Aleph. Bratislava. ISBN 978-80-89491-08-7.