Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Fuzzy regulátor pro roboty Khepera

2015

Lukáš Oščádal

Vedoucí práce: doc. RNDr. Michal Krupka, Ph.D.

Studijní obor: Informatika, prezenční forma

Bibliografické údaje

Autor:	Lukáš Oščádal
Název práce:	Fuzzy regulátor pro roboty Khepera
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Michal Krupka, Ph.D.
Počet stran:	36
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Lukáš Oščádal
Title:	Fuzzy controller for Khepera robots
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Computer Science, full-time form
Supervisor:	doc. RNDr. Michal Krupka, Ph.D.
Page count:	36
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Tato práce navazuje na bakalářskou práci Martina Kauera, který vytvořil systém snadného programování robotů Khepera III. Cílem práce bylo na základě již existující komunikace s robotem sestavit fuzzy regulátor pro programování těchto robotů pomocí jednoduchých pravidel.

Synopsis

This thesis continues on Bachelor thesis of Martin Kauer, who created the system of easy programming of robots Khepera III. Purpose of this thesis was compile fuzzy regulator for programming these robots using simple rules on principle existing communication with the robot.

Klíčová slova: Khepera III; fuzzy regulátor

Keywords: Khepera III; fuzzy controller

Děkuji doc. RNDr. Michalu Krupkovi Ph.D. za vedení této bakalářské práce a poskytnuté konzultace.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

1 Úvod

Roboti Khepera III jsou víceúčeloví roboti, jejichž programování je velice nízkoúrovňové. Tato práce má za úkol vytvořit systém, pomocí něhož by mohli roboti Khepera III řešit úlohy, které spočívají v reakci na danou situaci kolem sebe, kterou zjišťují pomocí svých senzorů.

Práce navazuje na bakalářskou práci Martina Kauera z roku 2012, který vytvořil kompletní systém pro jednoduché programování těchto robotů. Nyní bylo snahou vytvořit systém, který by řídil roboty pomocí fuzzy regulace a to s pomocí komunikační vrstvy z práce Martina Kauera, která zprostředkovává komunikaci mezi počítačem a robotem.

Vytvořený systém umožňuje uživateli snadno a rychle definovat situace a způsob, jakým na ně má robot reagovat. Využívá k tomu fuzzy regulátor, který podle definovaných pravidel chování robota vždy vypočítá nejvhodnější reakci na danou situaci. Celý systém běží na počítači a robot pouze přijímá příkazy a odesílá hodnoty na svých senzorech.

V tomto textu je popsána teorie nutná k pochopení práce se systémem, způsob jak pracovat s programem, jazyk jakým jsou definována pravidla, krerými se robot řídí a implementace nejdůležitějších částí programu. Dále jsou zde také uvedeny ukázkové příklady řešení některých úloh.

Program byl vyvíjen v prostředí *LispWorks 6.1.1 Personal Edition* pro platformu *Windows* a testován na operačním systému *Windows 7 Home Premium, Service pack 1, 64–bit.*

2 Khepera III

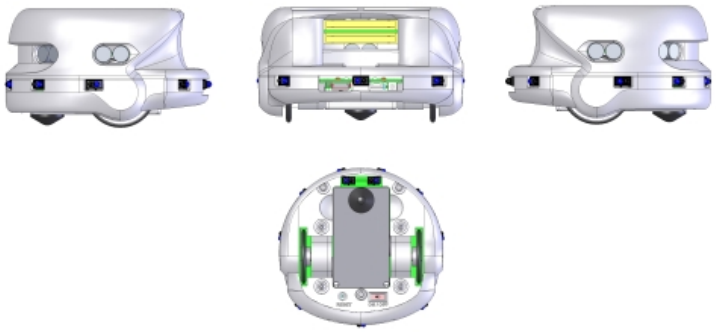
Robot Khepera III, který je vidět na obrázku č.1 je vyrobený společností K-Team Corporation a veškeré informace o něm jsou čerpány z manuálu od výrobce [**2**] nebo z práce Martina Kauera [**1**].

2.1 Základní informace

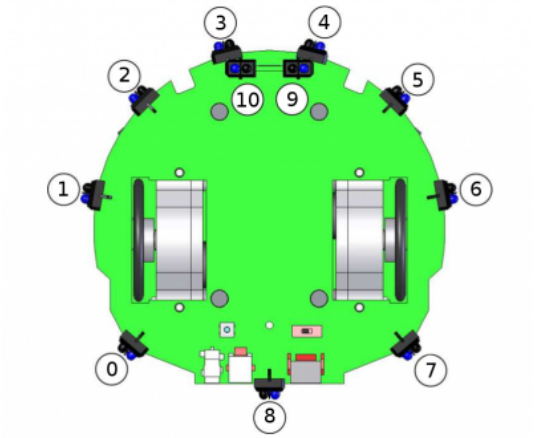
Khepera III je multifunkční robot vybavený infračervenými a ultrazvukovými senzory po obvodu konstrukce pro zjisťování informací o svém okolí. Jeho pohyb v prostoru zajišťují dvě kolečka otáčející se oběma směry nastavitelnou rychlostí, což zajišťuje velice dobrou motoriku robota. Komunikace s počítačem je možná několika způsoby a to pomocí sériové linky, USB kabelem nebo bezdrátově pomocí Bluetooth.

2.2 Infračervené senzory

Robot je vybaven celkem jedenácti infračervenými senzory, z nichž dva jsou na spodní straně a devět po obvodu robota. Jejich přesné rozmístění popisuje obrázek č.2.



Obrázek 1: Robot Khepera III



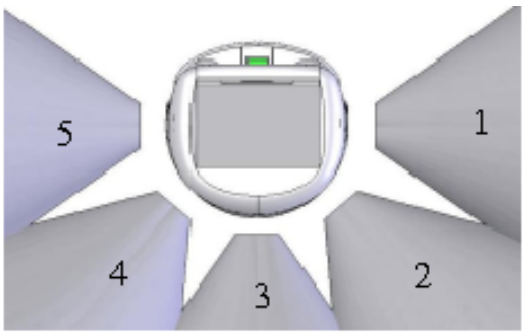
Obrázek 2: Rozmístění infračervených senzorů

Tyto senzory měří vzdálenost od překážek maximálně do třiceti centimetrů. Při měření infračerveným senzorem robot vrací číselnou hodnotu odraženého světla z intervalu od 0 do 4096, kde číslo 4096 znamená nulovou vzdálenost od překážky a 0 znamená vzdálenost větší než je měřitelná vzdálenost tímto senzorem. Naměřené hodnoty také závisí na barvě a povrchu dané překážky. Pokud je povrch překážky světlý a lesklý, pak dobře odráží světlo a naměřené hodnoty budou vyšší.

2.3 Ultrazvukové senzory

Po obvodu robota je rozmístěno pět ultrazvukových senzorů pro měření větších vzdáleností. Jako výsledek měření pak robot vrací hodnotu vzdálenosti v centimetrech. Ve většině případů jsou však tyto senzory velice nepřesné a je velmi obtížné s nimi rozumně pracovat. Obrázek č.3 ukazuje rozmístění ultrazvukových

senzorů na robotovi.



Obrázek 3: Rozmístění ultrazvukových senzorů

2.4 Kolečka

Na spodní straně jsou dvě kolečka poháněná elektromotory, která zajišťují pohyb robota. Otáčejí se oběma směry a tím zajišťují dobrou motoriku robota, jako třeba možnost otočení se na místě.

2.5 LED diody

Na zadní straně robota jsou umístěny dvě LED diody, které může uživatel ovládat podle potřeby.

3 Teoretický úvod do fuzzy regulace

Informace v této kapitole jsem čerpal z velkého množství nejrůznějších zdrojů, kde většina z nich jsou elektronické publikace dostupné z internetu [**5**].

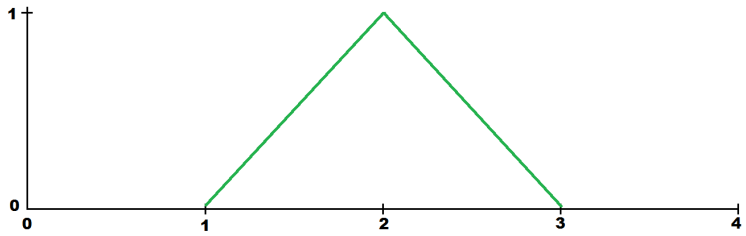
Pod pojmem regulace si můžeme představit řízení běhu nějakého procesu. Jednoduchým příkladem je třeba řízení otáček ventilátoru na základě teploty. Tento proces řízení se pak řídí jednoduchými pravidly například : „jestliže je teplota 20 až 25, pak nastav otáčky na 250 ot/min“.

Fuzzy regulace potom při vyhodnocování daných pravidel využívá principů fuzzy logiky. Pro pochopení principu fuzzy regulace je nutné znát pojem *fuzzy množina*, který hraje klíčovou roli při definování a vyhodnocování jednotlivých pravidel.

3.1 Fuzzy množiny

Fuzzy množina je soubor prvků podobný běžné množině. Pro prvky dané fuzzy množiny platí, že nemusí být dáno jednoznačně, zda do této množiny patří nebo nepatří, ale prvek do ní může patřit třeba jen částečně. Každému prvku ve fuzzy

množině je přiřazen tzv. *stupeň příslušnosti*, což je číslo z intervalu $[0, 1]$ vyjadřující pravdivostní hodnotu jak moc prvek do dané množiny patří. Stupeň 1 pak chápeme, že prvek do množiny patří úplně a stupeň 0, že do množiny nepatří vůbec. Fuzzy množiny mnohem lépe vyjadřují jazykové výrazy používané v běžné řeči než množiny klasické. Příkladem může být třeba „fuzzy množina vysokých lidí“, kdy nejsme schopni určit jednoznačně hranici, od které je člověk vysoký. Další příklad může být třeba „fuzzy množina reálných čísel, které jsou zhruba 2“, kdy není uvedeno, která čísla ještě jsou blízká číslu dva. Grafické znázornění pak můžeme vidět na obrázku č. 4.

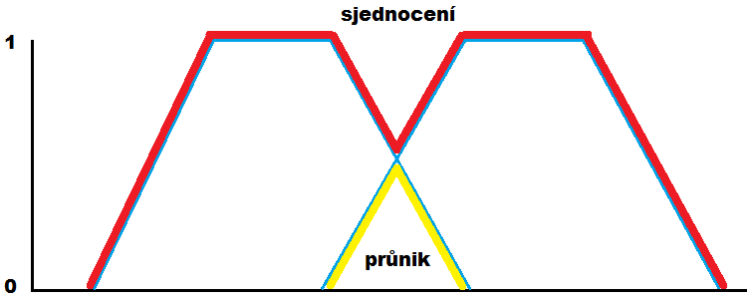


Obrázek 4: Fuzzy množina reálných čísel, které jsou „zhruba 2“

Podobně jako v klasické teorii množin lze nad fuzzy množinami dělat základní množinové operace jako jsou *sjednocení* a *průnik*. Do *průniku* dvou fuzzy množin A a B náleží všechny prvky obsažené v obou množinách se stupněm příslušnosti, který se určí jako minimum z obou množin. Pro operaci sjednocení se naopak použije maximum. Konkrétně tyto operace popisuje obrázek č. 5

Operace průniku i sjednocení dvou fuzzy množin lze zobecnit na operace libovolného počtu fuzzy množin.

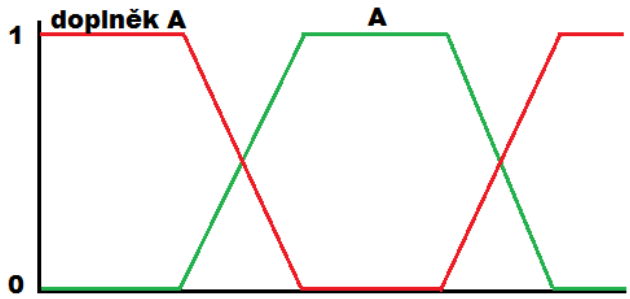
Kromě těchto operací je také definovaná operace *doplňku*. Doplněk fuzzy množiny vznikne odečtením st. příslušnosti od jedničky viz obrázek č. 6



Obrázek 5: Sjednocení a průnik dvou fuzzy množin

3.2 Fuzzy regulátor

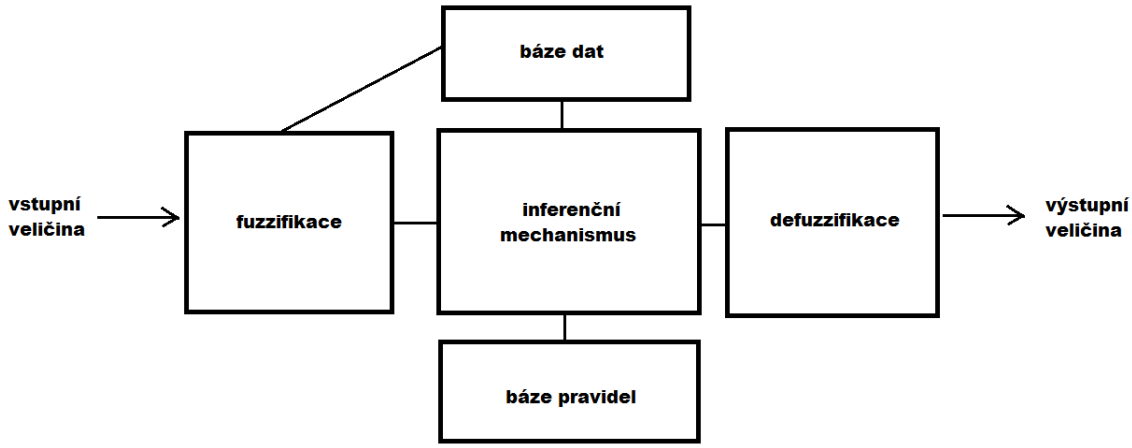
Obecně fuzzy regulátor je logický blok, na jehož vstupech jsou vstupní veličiny potřebné pro vyhodnocení pravidel a na výstupech jsou veličiny, které chceme



Obrázek 6: Doplněk fuzzy množiny

regulovat. Počet vstupů a počet výstupů může být obecně libovolný.

Celý regulátor se dá rozdělit do několika menších funkčních bloků. Jsou to *fuzzifikace*, *báze dat*, *báze pravidel*, *inferenční mechanismus* a *defuzzifikace*. Logické schéme fuzzy regulátoru popisuje obrázek č. 7.

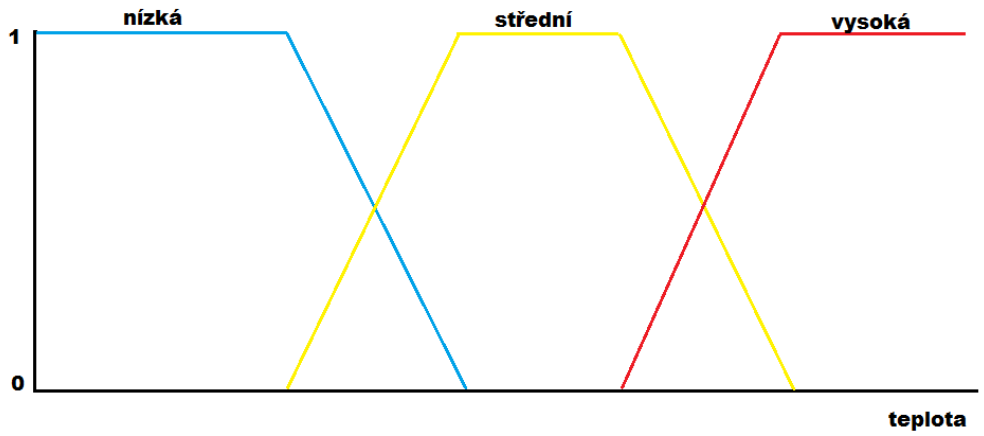


Obrázek 7: Vnitřní logické schéma fuzzy regulátoru

3.2.1 Báze dat

Báze dat fuzzy regulátoru obsahuje důležitá data potřebná pro vykonávání činnosti regulátoru. Především jsou to definované hodnoty vstupních a výstupních veličin, kterých mohou v průběhu tyto veličiny nabývat.

Tyto hodnoty jsou definované pomocí fuzzy množin a jsou to tedy jazykové výrazy typu *velká teplota, malá teplota, střední teplota* a k nim jsou přiřazeny příslušné fuzzy množiny, které vyjadřují jejich reálné hodnoty. Dobře je to vidět na obrázku č. 8



Obrázek 8: Fuzzy množiny reprezentující teplotu

3.2.2 Fuzzifikace

Blok fuzzifikace je první část výpočetního procesu fuzzy regulátoru. Vstupují do něj vstupní veličiny a tento blok je převádí na fuzzy množiny, aby s nimi bylo možné dále pracovat. Konkrétně se zjišťují stupně příslušnosti ve fuzzy množinách z báze dat pro aktuální hodnoty vstupních veličin. Výstupem jsou tedy fuzzy množiny, do kterých spadá aktuální hodnota vstupní veličiny a k nim odpovídající stupně příslušnosti, s jakými do nich patří.

3.2.3 Báze pravidel

Báze pravidel obsahuje všechny pravidla, které regulátor v každém cyklu svého běhu vyhodnocuje a na jejich základě pak stanovuje hodnoty výstupních veličin. Všechna pravidla jsou ve tvaru

$$JESTLIŽE \ \langle antecedent \rangle \ \ PAK \ \ \langle konsekvent \rangle$$

a tedy ve tvaru implikace. V antecedentu je vždy vstupní veličina a její hodnota, v konsekventu je pak výstupní veličina a hodnota, která se jí má přiřadit, jestliže platí antecedent. Konkrétní pravidlo pak může být například

$$JESTLIŽE \ \langle TEPLOTA \, je \, VYSOKÁ \rangle \ \ PAK \ \ \langle OTÁČKY \, jsou \, VYSOKÉ \rangle$$

Antecedent může být tvořen více členy, kteří jsou pak spojeni logickou spokou AND nebo OR a díky tomu můžeme tvořit složitější pravidla závislá na více vstupních veličinách.

3.2.4 Inferenční mechanismus

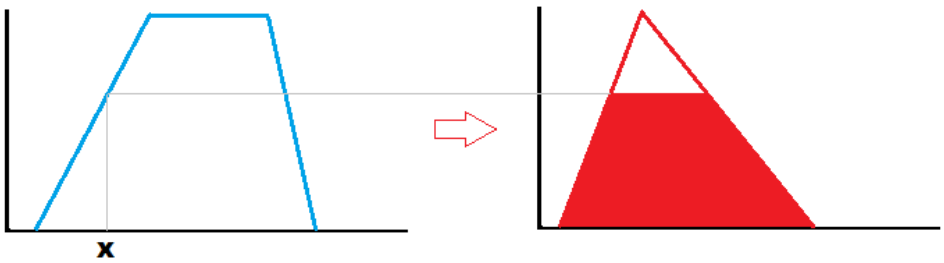
V bloku inferenčního mechanismu jsou vyhodnocována všechna pravidla z báze pravidel vzhledem k bázi dat a k aktuálním hodnotám vstupních veličin. Do

bloku vstupují fuzzifikované hodnoty vstupních veličin a vystupují z něj fuzzy množiny výstupních veličin, ze kterých se dále v bloku defuzzifikace určuje konkrétní hodnota výstupních veličin.

Způsobů, jak vyhodnocovat jednotlivá fuzzy pravidla je více, z nichž nejpoužívanější jsou tzv. *Mamdaniho implikace* a *Larsenova implikace*.

3.2.4.1 Mamdaniho implikace

Tento typ vyhodnocení spočívá v tom, že konsekvent pravidla může mít maximální st. příslušnosti, jaký má st. příslušnosti hodnota vstupní veličiny vzhledem k antecedentu. To znamená, že konsekvent je „oříznut“ na úroveň, v jaké je splněn antecedent. Výsledná množina tedy vzniká jako minimum z st. příslušnosti prvku a stupně, ve kterém je antecedent splněn. Obrázek č. 9 znázorňuje vyhodnocení pravidla pomocí Mamdaniho implikace za předpokladu, že vstupní veličina nabývá hodnoty x.



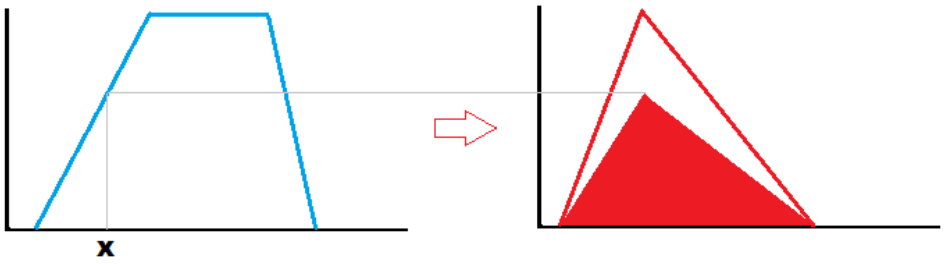
Obrázek 9: Vyhodnocení pravidla pomocí Mamdaniho implikace

3.2.4.2 Larsenova implikace

Při vyhodnocování pomocí Larsenovy implikace také platí, že konsekvent pravidla může mít maximální st. příslušnosti, jaký má st. příslušnosti hodnota vstupní veličiny vzhledem k antecedentu. Výsledek ale není určován jako minimum z st. příslušnosti prvku a stupně, ve kterém je antecedent splněn, ale jako násobek těchto dvou hodnot. Na obrázku č.10 můžeme vidět vyhodnocení pravidla pomocí Larsenovy implikace za předpokladu, že vstupní veličina nabývá hodnoty x.

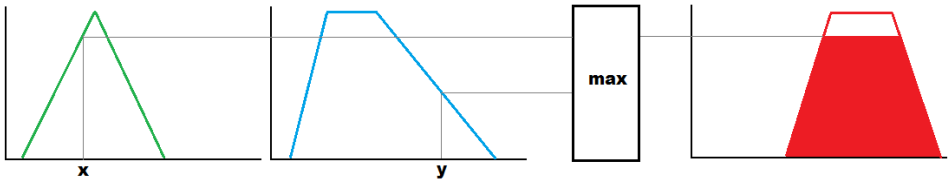
3.2.4.3 Vícerozměrně závislá pravidla

Vícerozměrná pravidla jsou pravidla, jejichž antecedent není tvořen jedním členem, ale více členy spojenými logickou spojkou AND nebo OR. Při vyhodnocování takových pravidel se určí stupně pravdivosti jednotlivých členů antecedentu a potom v případě logické spojky AND se vezme jejich minimum jako výsledná hodnota splnění antecedentu. Pro logickou spojku OR se bere jejich maximum. Grafické znázornění vyhodnocení vícerozměrného pravidla s logickou spojkou

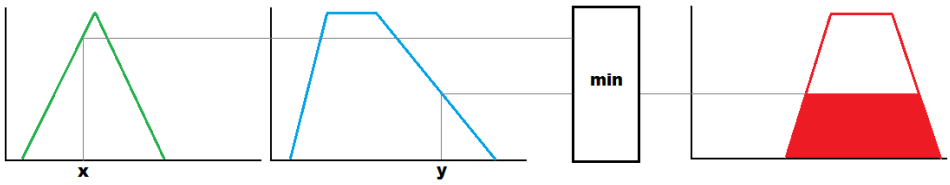


Obrázek 10: Vyhodnocení pravidla pomocí Larsenovy implikace

OR a AND můžeme vidět na obrázku č. 11 a obrázku č. 12, za předpokladu, že vstupní veličiny nabývají hodnot x a y.



Obrázek 11: Vyhodnocení vícerozměrného pravidla s logickou spojkou OR



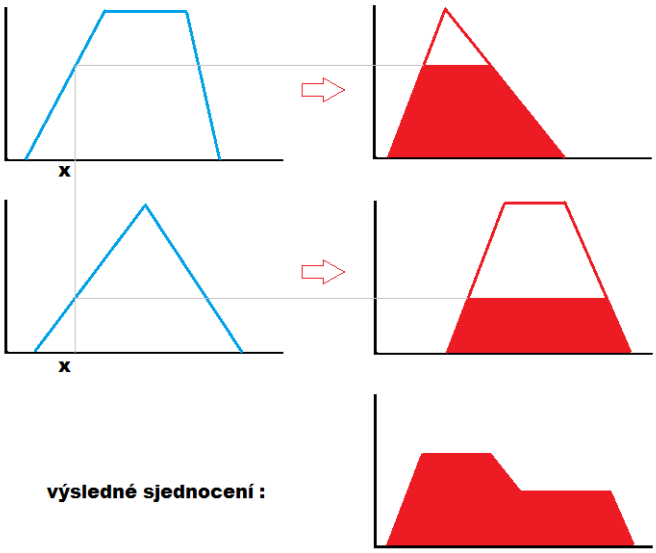
Obrázek 12: Vyhodnocení vícerozměrného pravidla s logickou spojkou AND

3.2.4.4 Výsledek vyhodnocení všech pravidel

Vyhodnocením každého z pravidel získáváme upravenou fuzzy množinu konsekventu pravidla. Výsledný výstup bloku inferenčního mechanismu je kombinace všech výsledků jednotlivých pravidel. Kombinování těchto výsledků se provádí pomocí sjednocení fuzzy množin a tento proces je zachycen na obrázku č.13.

3.2.5 Defuzzifikace

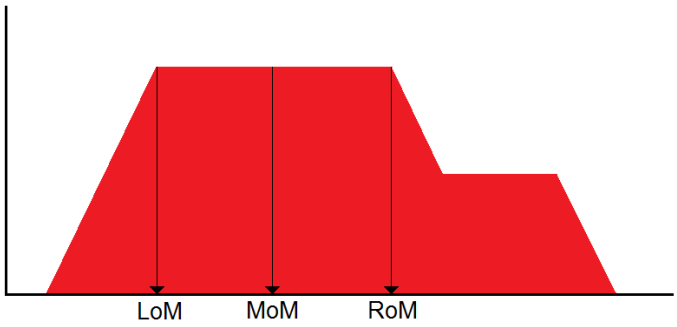
Defuzzifikace je finální část výpočtu fuzzy regulátoru. Na vstupu tohoto bloku je fuzzy množina, která je výsledkem vyhodnocení všech pravidel inferenčním mechanismem, a na jehož výstupu očekáváme již reálné hodnoty výstupních veličin fuzzy regulátoru. Metod pro stanovení hodnot ze vstupní fuzzy množiny



Obrázek 13: Výsledné sjednocení všech pravidel

je hned několik a v podstatě se tyto metody dělí do dvou skupin a to *metody nejvýznamějšího maxima* a *metody těžiště*.

Metody nejvýznamějšího maxima jsou *Left of Maximum (LoM), Mean of maximum (MoM)* a *Right of Maximum (RoM)*. Tyto metody najdou hodnoty, ve kterých množina dosahuje maxima a vyberou z nich hodnotu nejvíce vlevo (LoM), uprostřed (MoM) nebo nejvíce vpravo (RoM). Graficky tyto metody znázorňuje obrázek č. 14



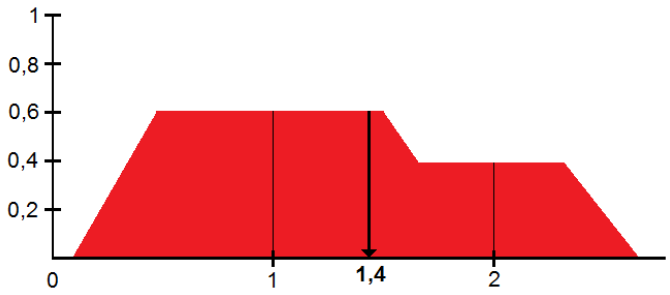
Obrázek 14: Metody nejvýznamnějšího maxima

Druhou skupinou jsou metody těžiště. První z nich je metoda *Center of Maximum (CoM) – těžiště singletonů*. Ta pro každou dílčí část množiny určí ostrou hodnotu a její stupeň příslušnosti pomocí některé z dříve uvedených metod nejvýznamnějšího maxima a z těchto hodnot vypočítá výslednou hodnotu pomocí

vzorce:

$$CoM = \frac{\sum_{i=1}^{n} a_i * u_i}{\sum_{i=1}^{n} a_i}$$

kde a_i je hodnota stupně příslušnosti a u_i je souřadnice i-té hodnoty. Na obrázku č. 15 můžeme vidět tento výpočet s použitím metody MoM.



Obrázek 15: Metoda Center of Maximum

Druhou metodou defuzzifikace v této skupině je metoda *Center of Gravity (CoG)*. Ta stanovuje přesnou hodnotu výstupu jako souřadnici těžiště plochy, kterou ohraničuje vstupní fuzzy množina. Výpočet tohoto těžiště je podle vzorce:

$$CoG = \frac{\int m(u) * udu}{\int m(u)du}$$

kde m je funkce příslušnosti fuzzy množiny, která pro dané u vrací jeho stupeň příslušnosti.

Existují ještě některé další metody defuzzifikace, ale v praxi se nepoužívají a není potřeba se jimi dále zabývat.

4 Vytvořený jazyk pravidel

Tento jazyk má kvůli jednoduchosti implementace běžnou syntaxi jazyku Common Lisp. Dělí se na dvě části a to na část pro definici *hodnot* a na část pro definici *pravidel*. Pro tyto účely jazyk obsahuje tři základní funkce: *define-value* a *define-values* pro definici hodnot a funkci *define-rule* pro definici pravidel.

4.1 Funkce define-value

Pomocí této funkce se definují tzv. *hodnoty*, pro vstupní a výstupní veličiny. Tyto hodnoty se později používají při definici pravidel fuzzy regulátoru. Hodnota jako taková je ve skutečnosti fuzzy množina a od toho se také odvíjí její definování. Syntaxe této funkce je ve tvaru:

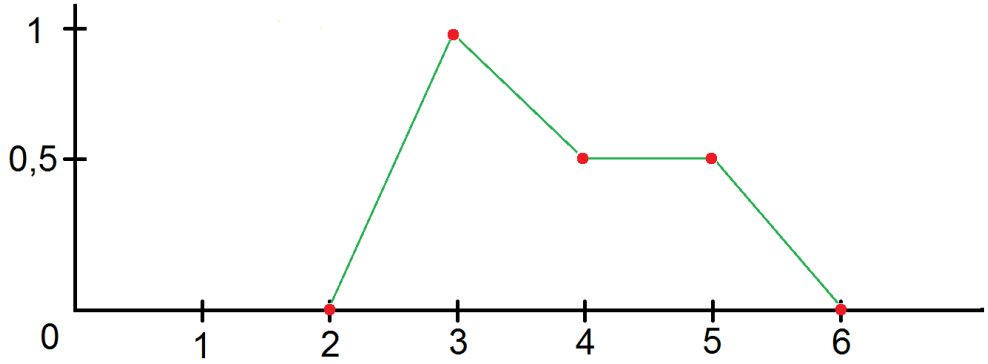
```
(define-value jméno-hodnoty tvar-fuzzy-množiny pozice)
```

kde:

- **jméno-hodnoty** je libovolný název začínající dvojtečkou („:big“, „:small“).
- **tvar-fuzzy-množiny** je kvótovaný seznam reálných čísel z intervalu $[0, 1]$, který obsahuje y-souřadnice bodů fuzzy množiny zleva doprava.
- **pozice** je kvótovaný seznam reálných čísel, který obsahuje x-souřadnice bodů fuzzy množiny. Tzn jsou to hodnoty vstupních a výstupních veličin patřící definované hodnotě.

Argumenty **tvar-fuzzy-množiny** a **pozice** dohromady tvoří souřadnice bodů definujících fuzzy množinu právě definované hodnoty. Proto oba argumenty musí mít vždy stejnou délku. Na obrázku č.16 je znázorněn ukázkový příklad definice hodnoty příkazem:

```
(define-value :hodnota1 '(0 1 0,5 0,5 0) '(2 3 4 5 6))
```



Obrázek 16: Hodnota definovaná pomocí define-value

4.2 Funkce define-values

Při definici hodnot potřebujeme často pokrýt celý interval reálných hodnot vstupů a výstupů. To však může být velice zdlouhavé a nepřehledné pokud máme mnoho takto na sebe navazujících hodnot. K usnadnění slouží funkce **define-values**, která vytvoří podle zadání pilovité pokrytí intervalu. Syntaxe:

```
(define-values jména rozložení levá-mez pravá-mez)
```

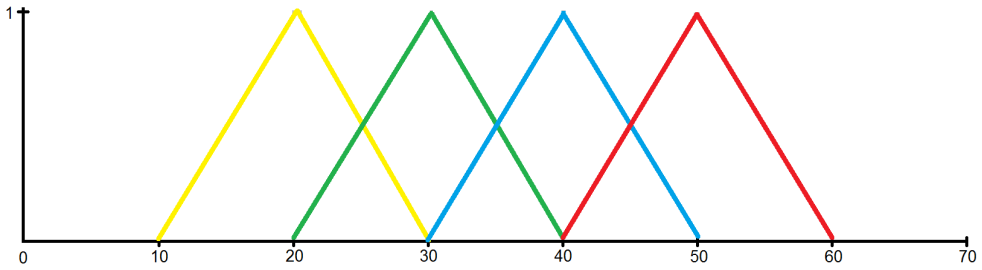
kde:

- **jména** je kvótovaný seznam jmen definovaných hodnot (každé jméno začíná dvojtečkou) zleva doprava v pořadí, v jakém následují za sebou

- **rozložení** je kvótovaný seznam reálných hodnot vstupů nebo výstupů, ve kterých každá z právě definovaných hodnot v tomto pořadí dosahuje stupně příslušnosti 1.
- **levá-mez** je reálná hodnota vstupů nebo výstupů, kde začíná nejlevější právě definovaná hodnota.
- **pravá-mez** je reálná hodnota vstupů nebo výstupů, kde končí nejpravější definovaná hodnota.

Například na obrázku č.17 je vidět, jak budou vypadat hodnoty definované příkazem:

```
(define-values
    '(:maly :stredni :velky :nejvetsi) '(20 30 40 50) 10 60)
```



Obrázek 17: Hodnoty definované pomocí define-values

Argumenty **levá-mez** a **pravá-mez** tedy vymezují interval, který funkce **define-values** pokrývá. Definované hodnoty nemusí být v pořadí od nejmenší po největší, ale můžou být seřazeny i od největší po nejmenší. Například následující dvě definice hodnot jsou ekvivalentní :

```
(define-values
    '(:maly :stredni :velky :nejvetsi) '(20 30 40 50) 10 60)

(define-values
    '(:nejvetsi :velky :stredni :maly) '(50 40 30 20) 60 10)
```

4.3 Funkce define-rule

Tato funkce slouží pro definici konkrétního pravidla. Každé pravidlo regulátoru je ve tvaru implikace a skládá se z předpokladu tedy *antecedentu* a z důsledku tedy *konsekventu*. Antecedent i konsekvent se dále dělí na tzv. *stavy*, což jsou takové jsou dvouprvkové seznamy, kde první je některá ze vstupních nebo výstupních veličin a druhé je jméno některé z definovaných hodnot. Funkce **define-rule** je dána následující syntaxí:

```
(define-rule antecedent :=> konsekvent)
```

Kde:

- **antecedent** je kvótovaný seznam stavů
- **konsekvent** je kvótovaný seznam stavů

Při definování stavů je možné použít tyto jména vstupů a výstupů :

1. infračervené senzory: **ps-1**, **ps-2**, **ps-3**, . . . , **ps-11**
2. ultrazvukové senzory: **us-1**, **us-2**, **us-3**, **us-4**, **us-5**
3. rychlost-koleček: **left-speed**, **right-speed**
4. LED diody: **led1**, **led2**

V antecedentu jsou všechny stavy implicitně spojeny logickou spojkou AND. V případě, že bychom chtěli stavy spojit logickou spojkou OR, stačí na první místo v antecedentu umístit symbol „ **:or** “. Antecedent může být také prázdný, to znamená že, předpoklad pravidla je vždy splněný a konsekvent pravidla se vždy projeví v práci robota. Také lze v antecedentu použít negaci některé z definovaných hodnot a to tím způsobem, že do stavu namísto jména dané hodnoty napíšeme seznam, kde první prvek je symbol „ **:not** “ a druhý je negovaná hodnota.

Zde můžete vidět definice některých pravidel:

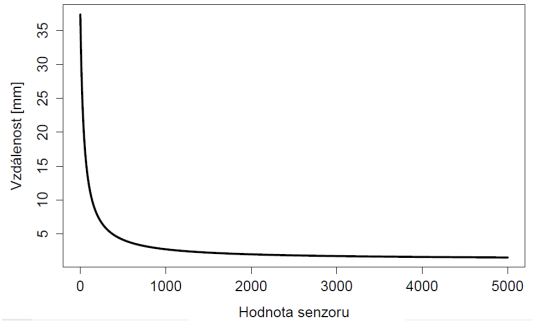
```
(define-rule
    '((ps-1 :big) (ps-3 :small)) :=> '((left-speed :slow)))

(define-rule
    '((us-4 :small)) :=> '((right-speed :fast) (led1 :on)))

(define-rule
    '(:or (ps-5 :big) (ps-5 :medium)) :=> '((led2 :off)))

(define-rule
    '() :=> '((left-speed :max) (right-speed :max)))

(define-rule
    '((ps-9 (:not :big))) :=> '((right-speed :fast)))
```

4.4 Jednotky vstupních a výstupních veličin

Při definici hodnot používaných v pravidlech regulátoru je nutné znát jednotky veličin, pro které tyto hodnoty definujeme a také rozsah ve kterém se reálně

naměřené hodnoty těchto veličin můžou pohybovat. V systému jsou zachovány původní jednotky, se kterými pracuje robot.

Infračervené senzory robota měří hodnoty odraženého světla a tuto hodnotu vracejí. Naměřené hodnoty se pak pohybují v intervalu od 0 do 4096, kde 0 znamená vzdálenost větší než vzdálenost měřitelná senzorem a naopak 4096 znamená vzdálenost nulovou. Naměřené hodnoty s přibližující se překážkou však nerostou lineárně. Přibližnou funkci popisující závislost mezi vzdáleností překážky a hodnotou na senzoru popisuje obrázek č.18.



Obrázek 18: Závislost mezi vzdáleností překážky a hodnotou na senzoru

Ultrazvukové senzory robota měří přímo vzdálenost robota od překážky a robot tuto hodnotu vrací v centimetrech.

Rychlost koleček je ve speciálních jednotkách robota vypočítávána z jeho technických parametrů. Reálně se tyto hodnoty pohybují v intervalu od -70000 do 70000 jednotek.

4.5 Základní databáze hodnot

Systém obsahuje základní sadu předdefinovaných hodnot pro rychlé psaní pravidel a uživatel si může sám zvolit, zda použije předdefinované hodnoty, nebo použije hodnoty vlastní.

4.5.1 Předdefinované hodnoty infračervených senzorů:

Hodnoty pro infračervené senzory pokrývají celý rozsah reálných hodnot, které můžou být na senzorech naměřeny a jsou to vzestupně za sebou tyto hodnoty : **:prZ**, **:prVS**, **:prS**, **:prM**, **:prL**, **:prVL**. Tyto hodnoty vyjadřují vzdálenost od překážky a ne reálnou hodnotu senzorů (tzn. při malé zvdálenosti od překážky je na senzoru vysoká reálná hodnota) a jsou definovány těmito příkazy:

```
(define-value :prZ '(1 1 0) '(4096 3200 950))
(define-values '(:prVS :prS :prM :prL )
     '( 950 400 300 260 ) 3200 230)
(define-value :prVL '(0 1 1) '(260 230 0))
```

4.5.2 Předdefinované hodnoty ultrazvukových senzorů

Hodnoty ultrazvukových senzorů jsou definovány na intervalu jednoho metru a vzestupně za sebou jsou to: **:usZ**, **:usVS**, **:usS**, **:usM**, **:usL**, **:usVL**, **:usVVL**, **:usVVVL**, **:extremeVL**. Tyto hodnoty vyjadřují vzdálenost od překážky a jsou definovány těmito příkazy:

```
(define-value :usZ '(1 1 0) '(0 3 10))
(define-values '(:usVS :usS :usM :usL :usVL :usVVL :usVVVL)
     '( 3 10 20 35 50 65 80) 0 100)
(define-value :extremeVL '(0 1 1) '(80 100 120))
```

4.5.3 Předdefinované hodnoty rychlostí koleček

Hodnoty rychlostí koleček nejsou definovány až do maximálních hodnot, ale pouze na nejpoužívanějším intervalu rychlostí. Jsou to tyto hodnoty vzestupně od negativních po kladné: **:NL**, **:NM**, **:NS**, **:NZ**, **:Z**, **:PZ**, **:PS**, **:PM**, **:PL**. Tyto hodnoty jsou definovány následujícími příkazy:

```
(define-value :NL '(1 1 0) '(-25000 -17500 -10000))
(define-values '( :NM :NS :NZ  :Z :PZ :PS :PM )
    '( -10000 -5000 -2500 0 2500 5000 10000 ) -17500 17500)
(define-value :PL '(0 1 1) '(10000 17500 25000))
```

4.5.4 Předdefinované hodnoty LED diod

Pro ovládání LED diod jsou určeny pouze předdefinované hodnoty a ne hodnoty definované uživatelem. Pro rozsvícení LED diody slouží hodnota **:on** a pro zhasnutí slouží hodnota **:off**. Poslední předdefinovaná hodnota pro LED diody je hodnota **:change**, která slouží ke změně stavu diody. Pokud tedy LED dioda svítí, pak zhasne a obráceně.

5 Uživatelská příručka

Vytvořený program realizuje obecný fuzzy regulátor pro roboty Khepera III od společnosti K-team Corporation. Program je spuštěn na počítači, který komunikuje s robotem pomocí sériového portu přes bluetooth. Veškerý výpočet probíhá na počítači a robot pouze přijímá instrukce pro nastavování svých parametrů jako například rychlost koleček a odesílá hodnoty naměřené na infračervených a ultrazvukových senzorech. Program využívá komunikační vrstvu mezi počítačem a robotem, kterou implementoval Martin Kauer ve své bakalářské práci, na kterou tato práce navazuje.

5.1 Popis programu

Program očekává na vstupu pravidla, podle kterých se bude řídít vyhodnocovací proces, tak jak bylo popsáno dříve. Tímto způsobem lze tedy robota naprogramovat pro řešení specifických úloh, které spočívají v tom, že robot reaguje na změny svého okolí, které detekuje pomocí svých senzorů.

Pro srovnání jsou implementovány dva typy vyhodnocování pravidel inferenčního mechanismu a to Mamdaniho a Larsenova implikace. Celý inferenční mechanismus je naprogramovaný tak, aby byl použitelný pro jakýkoliv fuzzy regulátor s libovolným počtem vstupů a libovolným počtem výstupů.

Program má grafické uživatelské rozhraní a je určeno pro platformu Windows.

Protože není volně k dispozici profesionální edice LispWorks, nelze vytvořit spustitelný soubor programu. Program se tedy spouští v prostředí LispWorks kompilací zdrojového souboru *system.lisp*. Pro zajištění správnéno běhu programu by měl být program spouštěn v prostření *Lispworks Personal Edition 6.1.1*. Po kompilaci se program spustí a na obrazovce se zobrazí uživatelské rozhraní programu.

5.2 Prerekvizity

Aby bylo možné robota připojit pomocí *bluetooth* musí být plně nabitý a spárováný s počítačem. Obecně se může postup na různých verzích Windows lišit. Obecný postup párování je:

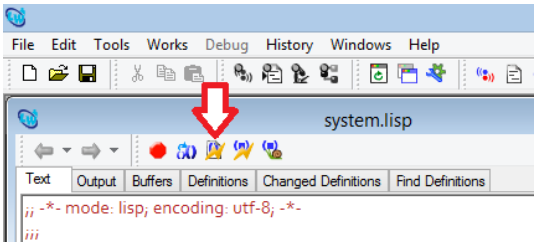
1. Zapnout robota.
2. Zapnout adaptér *bluetooth* na počítači.
3. Na počítači vyhledat nová zařízení *bluetooth*. Robot bývá často označován jako kapesní počítač s názvem „KHIII <číslo robota>“.
4. Na počítači spustit párování s robotem. Při výzvě o zadání bezpečnostního kódu zadat „0000“.
5. Po dokončení párování systém vyhradí sériové porty pro komunikaci s robotem, kterým je třeba nastavit následující parametry
   - Bity za sekundu: 115200
   - Datové bity: 8
   - Parita: žádná
   - Stop-bity: 1
   - Řízení toku: žádné

Přidělené sériové porty se poté používají pro komunikaci s robotem. Platforma Windows většinou vyhradí dva sériové porty, ale pro komunikaci je používaný jen jeden z nich.

Častým problémem po spárování je, že platforma Windows někdy po pár vteřinách přerušuje nepoužívané spojení s možností, že v případě potřeby toto spojení znovu naváže. To se ale prakticky nikdy neděje a spojení je nutné obnovit ručně. Řešení problémů s připojením robota je popsáno v přiloženém soubotu *readme.txt*

5.3 Spuštění programu

Pro spuštění programu je nutné mít stažené a nainstalované vývojové prostředí *Lispworks Personal Edition 6.1.1*. Toto prostředí lze stáhnout na adrese http://www.lispworks.com/downloads. Po nainstalování a spuštění LispWorks v levém horním rohu vyberete v menu *File* položku *Open*, najdete zdrojový soubor *system.lisp* a kliknete na tlačítko *O*tevřít. Nyní bude zdrojový soubor načtený a připravený ke kompilaci. Kompilaci kódu spustíte tlačítkem *Compile Buffer* zobrazeném na obrázku č.19. Ihned po kompilaci se program spustí a zobrazí se uživatelské rozhraní programu.



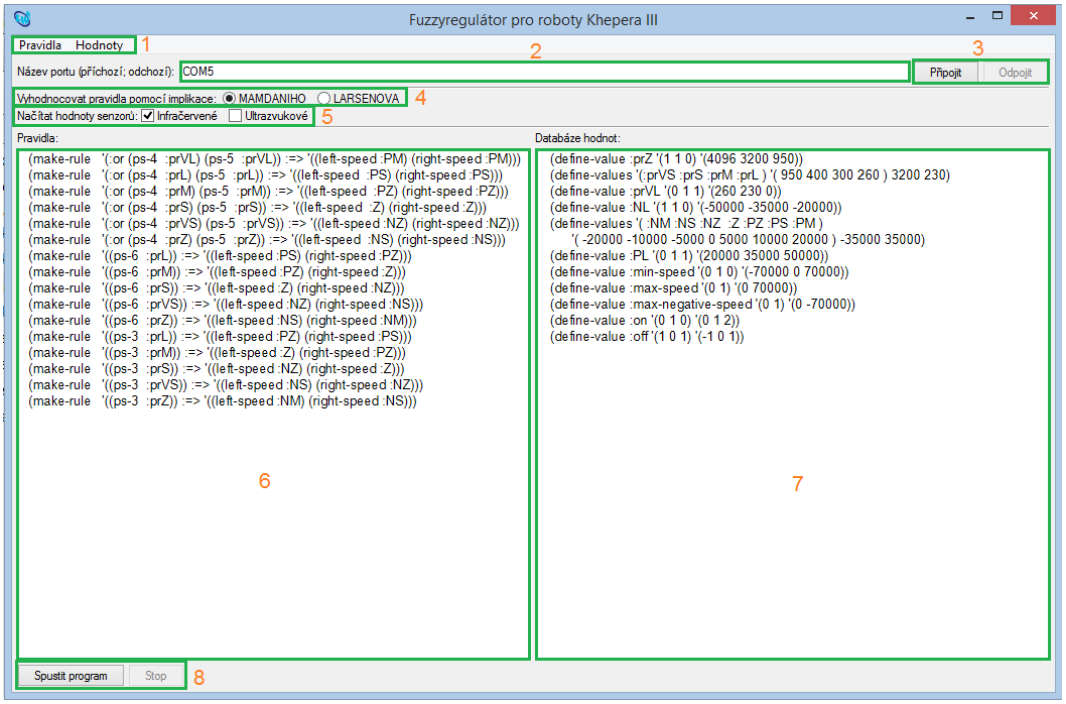
Obrázek 19: Tlačítko Compile Buffer

5.4 Uživatelské rozhraní

Grafické uživatelské rozhraní programu slouží k pohodlnému ovládání celého systému. Rozhraní je rozděleno do dvou základních částí, na část pro definici pravidel a část pro definici hodnot. Pravidla i hodnoty se ukládají do samostatných souborů a lze je libovolně načítat ze souborů nezávisle na sobě. To umožňuje například vytvoření jedné sady pravidel a k nim více sad různě definovaných hodnot se stejnými jmény a tím měnit chování programu. Stejně tak lze vytvořit jednu sadu hodnot, která se bude používat v různých sadách pravidel.

Na obrázku č.20 je znázorněno okno uživatelského rozhraní fuzzy regulátoru a jsou na něm zvýrazněny tyto důležité prvky:

1. Menu pro načítání a ukládání souborů hodnot a pravidel.
2. Textové pole pro zadávání portů, pomocí kterých bude připojen robot.
3. Tlačítka pro připojení a odpojení robota k programu.



Obrázek 20: Grafické uživatelské rozhraní

4. Menu pro výběr způsobu, kterým jsou vyhodnocována pravidla při výpočtu fuzzy regulátoru.

5. Menu pro nastavení regulátoru, které hodnoty senzorů se mají při běhu programu načítat.

6. Editor pro definici pravidel fuzzy regulátoru.

7. Editor pro definici hodnot fuzzy regulátoru.

8. Tlačítka pro spuštění a zastavení běhu regulátoru.

V levé horní části se nachází menu pro uložení a načtení zdrojových souborů pravidel a hodnot. O něco níže se nachází pole pro zadání sériových portů, pomocí kterých bude robot připojen. Porty se zadávají v pořadí *příchozí* a pak *odchozí* odělené středníkem tak, jak to znázorňuje popisek po levé straně pole. Po pravé straně tohoto pole jsou umístěny tlačítka pro připojení a odpojení robota, které se používají po spárování robota s spočítačem a zadání sériových portů. Pod připojovací částí leží položka výběru typu vyhodnocování pravidel fuzzy regulátoru. Uživatel si může zvolit ze dvou typů vyhodnocovacího procesu a to z *Mamdaniho implikace* a *Larsenovy implikace*. Těsně pod tímto výběrem je položka pro označení typu senzorů, jejichž hodnoty chce uživatel nechat načítat robotem a bude je používat ve svých pravidlech. To zajistí, že nebudou zbytečně měřeny a odesílány informace, které jsou pro běh regulátoru aktuálně zbytečné

a komunikace mezi robotem a počítačem nebude těmito informacemi zpomalována. Uprostřed po levé straně je umístěn editor pravidel, kam uživatel definuje všechny pravidla regulátoru. Naopak uprostřed po pravé straně se nachází editor hodnot. Celý regulátor se spouští tlačítkem „Spustit program“ umístěný v levém dolním rohu. Vedle něj je potom tlačítko „Stop“ pro zastavení regulátoru.

6 Logické schéma programu

Systém lze rozdělit do tří základních logických částí, které spolu komunikují. Celé schéma znázorňuje obrázek č.21 První část je Grafické uživatelské rozhraní, které dává možnost uživateli připojit robota k systému, definovat hodnoty a pravidla regulátoru, nastavovat typ vyhodnocování pravidel, spouštět a zastavovat regulátor a další.

Druhá část programu je samotný regulátor, zajišťující celkový výpočet programu a neustálé aktualizování výstupů robota podle reálně naměřených hodnot na senzorech.

Třetí část programu je komunikační vrstva navázaná na reálného robota, pomocí které je komunikace s robotem realizována. Tato vrstva tvoří abstrakci nad reálnou nízkoúrovňovou komunikací robota s počítačem a odstiňuje tak implementační detaily komunikace od zbytku programu.

7 Implementace

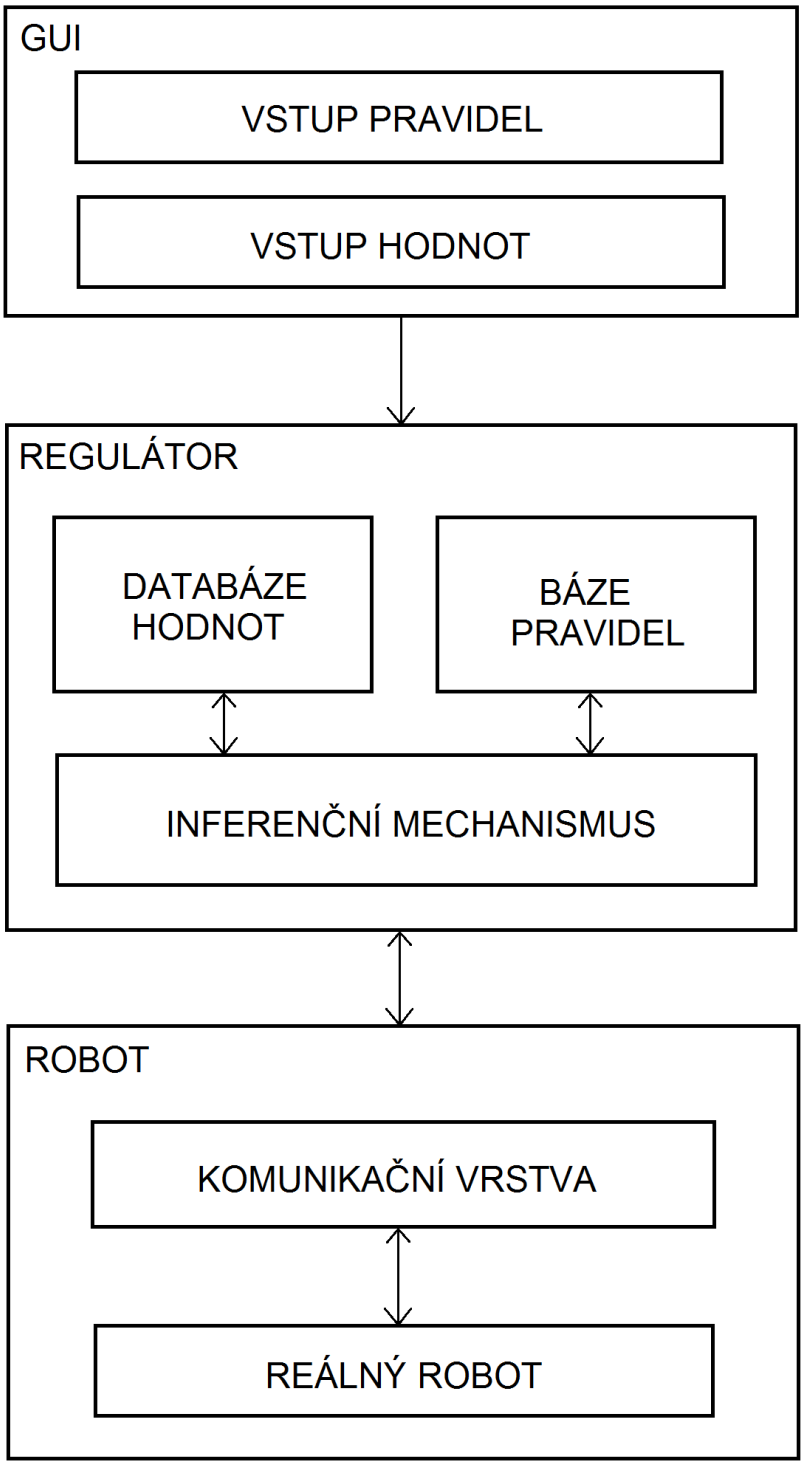
Tato kapitola popisuje nejdůležitější části programu. Popisované části nejsou rozebírány úplně dopodrobna a některé implementační detaily, které nejsou důležité pro pochopení principů, jsou vynechány.

7.1 Programovací jazyk

Programovací jazyk byl dán implementací komunikační vrstvy s robotem, kterou jsem měl použít. Ta je naprogramovaná v jazyce Common Lisp. Využívá funkcionální, procedurální i objektově orientované paradigma a dává tak programátorovi v tomto směru určitou volnost, která je často výhodou. Tento jazyk byl původně navrhnut jako jazyk pro matematické výpočty. Dnes je již spíše akademickým jazykem a nejčastěji se dnes používa pro vývoj umělé inteligence.

7.2 Třída *fuzzyset*

Pro potřeby regulátoru se stačí omezit pouze na fuzzy množiny, které jsou ohraničeny libovolnou lomenou čárou. Toho lze využít při implementaci vnitřní reprezentace a zavést fuzzy množinu jako seznam bodů, které definují tuto lomenou čáru. Takto zavedená fuzzy množina umožňuje přesnou reprezentaci a veškeré



Obrázek 21: Logické schéma programu

operace nad fuzzy množinami jsou tedy prováděny pomocí výpočtů nad body jednotlivých fuzzy množin.

Třída *fuzzyset* je zavedená jako dva seznamy hodnot definující tyto body.

Sloty třídy *fuzzyset*:

- **degrees** – stupně příslušnosti v bodech definujících fuzzymnožinu (y souřadnice bodů)
- **members** – prvky množiny příslušné ke stupňům příslušnost ve slotu degrees (x souřadnice bodů)

Metody třídy *fuzzyset*:

- **make-inverse-fuzzyset** – pro danou fuzzy množinu vrátí její doplněk
- **get-degree-of-membership** – pro danou fuzzy množinu a prvek x vrátí stupeň příslušnosti prvku x k této fuzzy množině
- **mamdani-implication** – pro danou fuzzy množinu a stupeň příslušnosti vrátí stejnou fuzzy množinu upravenou na stupeň příslušnosti podle Mamdaniho implikace
- **larsen-implication** – pro danou fuzzy množinu a stupeň příslušnosti vrátí stejnou fuzzy množinu upravenou na stupeň příslušnosti podle Larsenovy implikace
- **union-fuzzysets** – pro dané fuzzy množiny vrací jejich sjednocení, implementuje algoritmus, který odstraňuje nepotřebné body pohlcené sjednocením a vypočítává body nově vzniklé jako průsečíky hran fuzzy množin
- **center-of-gravity** – pro danou fuzzy množinu vrací prvek který je jejím těžištěm podle metody defuzzifikace *Center of gravity*.

7.3 Třída *rule*

Objekty třídy rule reprezentují pravidla regulátoru. Regulátor při výpočtu pracuje s těmito objekty a na základě výsledků metody třídy *rule* pro vyhodnocení pravidla stanovuje hodnoty výstupů, vzhledem k aktuálním hodnotám vstupů.

Sloty třídy *rule*:

- **ant** – antecedent pravidla reprezentovaný seznamem párů, kde první je číslo vstupu a druhé je jméno hodnoty
- **con** – konsekvent pravidla reprezentovaný seznamem párů, kde první je číslo výstupu a druhé je jméno hodnoty
- **conjunction** – logická spojka která spojuje antecedent pravidla

Metody třídy *rule*:

- **value-for-cut** – zjistí v jakém stupni je splněn antecedent pravidla vůči aktuálním hodnotám vstupů a logické spojce antecedentu
- **eval-rule** – vrací konsekvent pravidla, ve kterém jsou upraveny fuzzy množiny podle zadané implikace a stupně ve kterém je splněn antecedent

7.4 Databáze hodnot

Databáze hodnot je struktura, kterou si fuzzy regulátor uchovává a nepřetržitě s ní pracuje. Jedná se o seznam, ve kterém jsou společně uloženy všechny jména definovaných hodnot a k nim přiřazené příslušné fuzzy množiny. Tyto dvojice jsou uloženy v tečkových párech, kde první z nich je jméno a druhá je fuzzy množina. Při vyhodnocování pravidel regulátor prochází touto strukturou a hledá podle jmen hodnot správnou fuzzy množinu.

Pokud je v pravidle použita negace hodnoty, regulátor vytvoří inverzní fuzzy množinu k fuzzy množině původní hodnoty, použije ji při vyhodnocení pravidla a tuto nově vzniklou hodnotu přidá do databáze. To znamená, že v dalším cyklu fuzzy regulátoru již bude tato negace hodnoty v databázi a může se rovnou použít bez dalších výpočtů.

7.5 Funkce *inference-mechanism*

Tato funkce realizuje celé tři bloky fuzzifikace, inferenčního mechanismu a defuzzifikace logického schématu fuzzy regulátoru. Funce je naprogramovaná tak, aby byla zcela obecná a při zachování struktury jejích argumentů lze použít pro fuzzy regulátory libovolného počtu vstupů a libovolného počtu výstupů. Jako výsledek je vráceno pole s reálnými hodnotami výstupů fuzzy regulátoru.

Argumenty funkce *inference-mechanism*:

- seznam reálných hodnot vstupů fuzzy regulátoru
- počet výstupů
- seznam pravidel
- databáze hodnot
- implikace která má být použita při vyhodnocování pravidel

7.6 Třída *regulator*

Třída regulator je hlavní částí programu. Ve slotech má uchován objekt robota, se kterým komunikuje a řídí ho, databázi všech základních i uživatelem definovaných hodnot a také všechna pravidla potřebné pro řízení běhu robota. Po

spuštění regulátoru je nastaven příznak běhu regulátoru a začne se periodicky provádět výpočet. Při ukončení běhu je vždy dokončen poslední cyklus regulátoru, nastaven příznak zastavení běhu a výpočet ukončen.

Metody třídy *regulator*:

- **connect-robot** – připojí robota k počítači a nastaví ho do slotu regulátoru
- **run** – cyklicky vykonává výpočet regulátoru, tzn. odešle dotaz robotovi na hodnoty senzorů, po obdržení hodnot provede výpočet inferenčního mechanismu regulátoru a nakonec nastaví nové hodnoty výstupů robota
- **start-regulator** – načte databázi hodnot a pravidla, nastaví načítání hodnot senzorů, nastaví příznak běhu regulátoru a spustí metodu *run*
- **stop-regulator** – nastaví příznak zastavení regulátoru a zastaví metodu *run*
- **set-regulator-implication** – nastaví typ implikace, která má být použita při výpočtu inferenčním mechanismem

7.7 Použitá komunikační vrstva

Komunikaci mezi počítačem a robotem zajišťuje objekt třídy *real-khepera*. Objekt je abstrakcí reálného robota a odstiňuje fyzickou komunikaci mezi počítačem a robotem od zbytku programu. Veškerá komunikace s robotem je potom tvořena pomocí zasílání zpráv tomuto objektu.

Metody třídy *real-khepera*:

- **start** – otevře komunikační kanály a provede inicializaci
- **close-channels** – zavře komunikační kanály a uvolní prostředky
- **do-command** – odešle příkaz na robota

Tato část je celá převzata z bakalářské práce Martina Kauera [**1**] a obsahuje tyto zdrojové soubory: *conditions.lisp, common.lisp, khepera-tech-params.lisp, environment.lisp, environment-pane.lisp, base-stream.lisp, serial-port-stream.lisp, serial-port-file-stream.lisp, communication.lisp, pending-commands.lisp, robot.lisp, robot-with-environment.lisp, khepera.lisp, virtual-khepera.lisp, real-khepera.lisp.*

8 Ukázkové řešení úloh

V této kapitole jsou uvedeny některé vzorové úlohy, které lze pomocí tohoto systému řešit a zdrojové kódy jejich řešení. Uvedená řešení úloh používají v pravidlech pouze základní předdefinované hodnoty vstupů a výstupů.

8.1 Zastavení před překážkou

Robot jede stálou rychlostí vpřed dokud nenalezne překážku, potom začne zpomalovat až před ní úplně zastaví.

Pravidla:

```
(make-rule   '(:or (ps-4  :prVL) (ps-5  :prVL)) :=> '((left-speed :PM) (right-speed :PM)))
(make-rule   '(:or (ps-4  :prL) (ps-5  :prL)) :=> '((left-speed  :PS) (right-speed :PS)))
(make-rule   '(:or (ps-4  :prM) (ps-5  :prM)) :=> '((left-speed  :PZ) (right-speed :PZ)))
(make-rule   '(:or (ps-4  :prS) (ps-5  :prS)) :=> '((left-speed  :Z) (right-speed :Z)))
(make-rule   '(:or (ps-4  :prVS) (ps-5  :prVS)) :=> '((left-speed :Z) (right-speed :Z)))
(make-rule   '(:or (ps-4  :prZ) (ps-5  :prZ)) :=> '((left-speed  :Z) (right-speed :Z)))
```

8.2 Rozsvícení LED diod

Robot kontroluje hodnotu na předním senzoru *ps-4*, pokud se nic nepřibližuje, pak jsou diody zhasnuté. Pokud robot na senzoru *ps-4* zaznamená nějakou překážku, pak rozsvítí zelenou diodu. Pokud se příblíží překážka velice blízko, pak rozsvítí i červenou diodu.

Pravidla:

```
(make-rule '((ps-4 (:not :prVL))) :=> '((led2 :on)))
(make-rule '((ps-4 :prVL)) :=> '((led2 :off)))
(make-rule '(:or (ps-4 :prZ) (ps-4 :prVS)) :=> '((led1 :on)))
(make-rule '(:or (ps-4 :prS) (ps-4 :prM) (ps-4 :prL) (ps-4 :prVL)) :=> '((led1 :off)))
```

8.3 Jízda podél zdi

Robot přistavený levým bokem ke zdi jede podél ní a udržuje si od ní stejnou vzdálenost.

Pravidla:

```
(make-rule '((ps-2 :prM)) :=>'((left-speed :PS) (right-speed :PS)))
(make-rule '(:or (ps-2 :prL) (ps-2 :prVL)) :=>'((left-speed :PZ) (right-speed :PS)))
(make-rule '(:or (ps-2 :prS) (ps-2 :prVS)) :=>'((left-speed :PS) (right-speed :PZ)))
(make-rule '(:or (ps-3 :prM) (ps-3 :prS) (ps-3 :prVS)) :=>'((left-speed :PM) (right-speed :PZ)))
(make-rule '(:or (ps-4 :prM) (ps-4 :prS) (ps-4 :prVS)) :=>'((left-speed :PS) (right-speed :Z)))
(make-rule '(:or (ps-5 :prM) (ps-5 :prS) (ps-5 :prVS)) :=>'((left-speed :PZ) (right-speed :NZ)))
(make-rule '(:or (ps-6 :prM) (ps-6 :prS) (ps-6 :prVS)) :=>'((left-speed :PS) (right-speed :NS)))
```

8.4 Udržování vzdálenosti

Tato úloha spočívá v tom, že si robot udržuje stále stejnou vzdálenost od jakékoliv překážky, kterou nalezne. V případě že se překážka bude pohybovat směrem od robota, pak ji dojede a natočí se přímo proti ní. Naopak pokud se bude překážka přibližovat k robotovi, pak bude robot couvat.

Pravidla:

```
(make-rule   '(:or (ps-4  :prVL) (ps-5  :prVL)) :=> '((left-speed :PM) (right-speed :PM)))
(make-rule   '(:or (ps-4  :prL) (ps-5  :prL)) :=> '((left-speed  :PS) (right-speed :PS)))
(make-rule   '(:or (ps-4  :prM) (ps-5  :prM)) :=> '((left-speed  :PZ) (right-speed :PZ)))
(make-rule   '(:or (ps-4  :prS) (ps-5  :prS)) :=> '((left-speed  :Z) (right-speed :Z)))
(make-rule   '(:or (ps-4  :prVS) (ps-5  :prVS)) :=> '((left-speed :NZ) (right-speed :NZ)))
(make-rule   '(:or (ps-4  :prZ) (ps-5  :prZ)) :=> '((left-speed  :NS) (right-speed :NS)))

(make-rule   '((ps-7  :prL)) :=> '((left-speed :PS) (right-speed :NZ)))
```

```
 9  (make-rule    '((ps-7  :prM)) :=> '((left-speed :PZ) (right-speed :NS)))
10  (make-rule    '((ps-7  :prS)) :=> '((left-speed :Z) (right-speed :NM)))
11  (make-rule    '((ps-7  :prVS)) :=> '((left-speed :NZ) (right-speed :NL)))
12  (make-rule    '((ps-7  :prZ)) :=> '((left-speed :NS) (right-speed :NL)))
13
14  (make-rule    '((ps-2  :prL)) :=> '((left-speed :NZ) (right-speed :PS)))
15  (make-rule    '((ps-2  :prM)) :=> '((left-speed :NS) (right-speed :PZ)))
16  (make-rule    '((ps-2  :prS)) :=> '((left-speed :NM) (right-speed :Z)))
17  (make-rule    '((ps-2  :prVS)) :=> '((left-speed :NL) (right-speed :NZ)))
18  (make-rule    '((ps-2  :prZ)) :=> '((left-speed :NL) (right-speed :NS)))
19
20  (make-rule    '((ps-3  :prL)) :=> '((left-speed :Z) (right-speed :PS)))
21  (make-rule    '((ps-3  :prM)) :=> '((left-speed :NZ) (right-speed :PZ)))
22  (make-rule    '((ps-3  :prS)) :=> '((left-speed :NZ) (right-speed :Z)))
23  (make-rule    '((ps-3  :prVS)) :=> '((left-speed :NM) (right-speed :NZ)))
24  (make-rule    '((ps-3  :prZ)) :=> '((left-speed :NL) (right-speed :NS)))
25
26  (make-rule    '((ps-6  :prL)) :=> '((left-speed :PS) (right-speed :Z)))
27  (make-rule    '((ps-6  :prM)) :=> '((left-speed :PZ) (right-speed :NZ)))
28  (make-rule    '((ps-6  :prS)) :=> '((left-speed :Z) (right-speed :NS)))
29  (make-rule    '((ps-6  :prVS)) :=> '((left-speed :NZ) (right-speed :NM)))
30  (make-rule    '((ps-6  :prZ)) :=> '((left-speed :NS) (right-speed :NL)))
```

Závěr

Touto prací jsem navazoval na bakalářskou práci Martina Kauera z roku 2012 tím, že jsem použil část jeho práce pro komunikaci mezi počítačem a robotem pomocí sériové linky *Bluetooth*. To se ale ze začátku jevilo jako veliký problém, protože se mi prvních pár měsíců nedařilo jeho systém spojit s robotem. Nakonec se ale ukázalo, že problém není na straně používané komunikační vrstvy, ale na straně operačního systému.

Převzatá komunikační vrstva byla implementována v jazyce Common Lisp, což byl důvod proč jsem pro implementaci svého systému také zvolil tento jazyk. I přes to, že jsem se s tímto programovacím jazykem při studiu již setkal, ukázalo se, že práce v tomto jazyce byla pro mě vcelku obtížná kvůli mé slabé znalosti obrovských možností tohoto jazyka.

Při práci se systémem, se ukázalo, že volba komunikace počítače s robotem pomocí *Bluetooth* nebyla příliš vhodná kvůli její nízké rychlosti. Odesílání příkazů robotovi i zjišťování hodnot na jeho senzorech trvá řádově v destinách sekundy, což způsobuje, že regulátor stihne za vteřinu pouze 3 až 4 cykly svého vyhodnocovacího procesu. To je pro dobré výsledky řízení fuzzy regulátorem žalostně málo. V ideálním případě by měl regulátor vykonat minimálně 20 cyklů za vteřinu. Navíc kvůli takto pomalé komunikaci je reakční doba robota velice vysoká. Vzhledem k rychlostem, jakými je robot schopen se pohybovat a vzhledem k citlivosti senzorů to znamená, že v době, kdy systém obdrží nejnovější naměřené hodnoty senzorů a začne tyho hodnoty zpracovávat, tak se tyto právě zpracovávané hodnoty můžou kvůli dlouhotrvající komunikaci výrazně lišit od aktuálních hodnot senzorů. To má za následek skokové změny rychlostí koleček robota, což je právě u fuzzy regulátoru nežádnoucí, protože jeho úkolem je právě takovéto skokové změny vyhladit.

Řešením by mohlo být například použití komunikace pomocí USB kabelu, která je v případě robota Khepera III možná. To by sice zvýšilo rychlost předávání informací, nicméně tento způsob je omezující délkou kabelu a také může přinášet problémy při pohybu robota, kdy by mohl kabel často překážet. Další řešení, které se nabízí, je spouštět fuzzy regulátor přímo na robotovi, což ovšem vyžaduje rozšíření robota o modul *KoreBot II*, na kterém je možné spustit kód. Ovšem větší problém, který toto řešení přináší je, že uživatel přichází o možnost snadného programování robota a rychlou změnu pravidel. Každá změna pravidel, kterou by chtěl uživatel provést by totiž znamenala opětovné nahrávání programu do robota. Třetím a pravděpodobně nejnáročnějším řešením je vytvořit systém skládající se ze dvou částí, kdy jedna část bude spustitelný regulátor na robotovi a druhá část bude rozhraní na počítači, které zpracuje uživatelské vstupy a inicializuje jimi regulátor na robotovi pomocí komunikace přes sériovou linku *Bluetooth*. Po spuštění by pak regulátor běžel na robotovi bez další komunikace s počítačem, pouze by přijímal příkaz k zastavení běhu regulátoru.

V systému jsem vytvořil možnost výběru způsobu vyhodnocování pravidel pomocí Mamdaniho nebo Larsenovy implikace, aby bylo možné porovnávat vý-

sledky obou typů vyhodnocování. Při podmínkách v jakých regulátor pracuje, jako je například ona zmiňovaná rychlost komunikace, jsou ale rozdíly naprosto zanedbatelné a oba způsoby dosahují stejných výsledků.

Bibliografie

[1] KAUER Martin: *Řešení úloh s roboty Khepera*, Elektronická Publikace, 2012.

[2] LAMBERCI; THARIN: *Khepera III Manual ver 3.2*, Elektronická publikace, 2011. Dostupné z: http://ftp.k-team.com/KheperaIII/UserManual/Kh3.Robot.UserManual.pdf

[3] LISPWORKS ltd.: *Common Lisp Hyperspec*, Elektronická publikace, 2005. Dostupné z: http://www.lispworks.com/documentation/HyperSpec/Front/

[4] LISPWORKS ltd.: *CAPI Reference Manual ver 6.0*, Elektronická publikace, 2009. Dostupné z: http://www.lispworks.com/documentation/lw60/CAPRM/html/capiref.htm

[5] MODRLÁK, Osvald: *Fuzzy řízení a regulace*, Elektronická publikace, 2002. Dostupné z: https://www.kirp.chtf.stuba.sk/ bakosova/wwwRTP/tar2fuz.pdf

A Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémový chod systému. Spuštění systému se provádí kompilací souboru *system.lisp* v prostředí *LispWorks 6.1*.

readme.txt
: Instrukce pro spuštění systému včetně požadavků na jeho provoz.

Navíc CD/DVD obsahuje:

data/
: Soubory s ukázkovými příklady řešení úloh.

install/
: Instalátory aplikací potřebné nebo užitečné pro provoz systému.

literature/
: Vybrané položky bibliografie vztahující se k práci.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.