STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Autonomní robotické vozidlo Saladin

Tomáš Báča & Antonín Novák

Praha 2010

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 10. elektrotechnika, elektronika a telekomunikace

Autonomní robotické vozidlo Saladin

(Autonomous robotic vehicle Saladin)

Autoři: Tomáš Báča
Antonín Novák

Škola: Gymnázium Arabská
Arabská 14
Praha 6
160 00
Česká republika
http://www.gyarab.cz

Konzultant práce: Ing. Božena Mannová, Ph.D.

Praha 2010

Prohlášení

Prohlašujeme, že jsme naší naši práci vypracovali samostatně, použili jsme pouze podklady (literaturu, SW atd.) citované v práci a uvedené v přiloženém seznamu a postup při zpracování práce je v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Praze dne ……. *podpis: ……*

podpis: ……

Poděkování

Poděkování náleží paní Ing. Boženě Mannová, Ph.D., za ochotu a pomoc při tvorbě naší práce.

Dále pak také panu Janu Báčovi za technickou podporu a pomoc a za poskytnutí dílenského zázemí.

Anotace

Cílem této práce bylo vytvořit koncept univerzálního robota a umělou inteligenci, jejíž funkčnost na něm budeme testovat. Na základě návrhu byl sestrojen mechanický prototyp robota, který je plně programovatelný.

Kladli jsme důraz na robustnost řešení - univerzálnost robota. Chceme na něm dokázat použitelnost biologicky inspirovaných algoritmů v oboru mobilní robotiky. Tímto se na světové úrovni zabývá nemnoho laboratoří.

Klíčová slova: robot, umělá inteligence, neuronové sítě, genetické programování

Anotation

The goal of our effort was to create a concept of a universal robot and an artificial intelligence, whose functionality we are going to test on it. Based on the plan, a mechanical prototype of the robot has been created, that is fully programmable.

We emphasized the integrity of the solution, versatility of the robot. We would like to prove the usability of biology inspired algorithms in the field of mobile robotics. Only a handful of laboratories research this on a worldwide scale.

Key words: robot, artificial intelligence, neural network, genetic programming

Obsah

Úvod

Robotika a kybernetika je činnost a vědní obor, který není z daleka tak rozšířen, jako programování a jiné počítačové vědy. Jedná se o užívání programovacích technik k ovládání hardwarových komponent (které ve spojení tvoří stroje - roboty) a to takovým způsobem, aby vykonávaly příkazy počítače. V případě robotů, které jsou tvořeny z mnoha takových komponent, od nich často chceme komplikované úlohy a jejich vyvíjení zabere mnoho času, úsilí a testování.

Programovací metody užívané pro práci s roboty jsou v mnoha případech rozdílné od klasického programování a vyvíjení softwaru. To proto, že výsledná práce s fyzickým světem zahrnuje nestandardní vstupní a výstupní data, časové uvažování dle fungování reálného světa a jiný koncept aplikační logiky, než u klasického software. Programátor a tvůrce musí znát kus od každého – elektrotechniky, mechaniky, programování a vývojářství. Právě zde jsou používané, jinak nestandardní, funkcionální jazyky – např. LISP.

Pokud máme dále mluvit o aplikační logice robota, nazvěme jí umělou inteligencí (AI). Poté je třeba si jí definovat. Již v roce 1950 navrhnul Alan Turing test, (Turingův test) kdy by posadil účastníky experimentu do jedné místnosti a ti by měli za úkol komunikovat s člověkem a strojem na straně druhé. Pokud by nedokázal člověk určit, kde sedí člověk a kde pouze stroj, byl by stroj považován za inteligentní.

Tímto testem prošlo již několik umělých inteligencí. Např. známý virtuální psycholog jménem ALICE (Artificial Linguistic Internet Computer Entity) dokázal obelstít a zmást spoustu lidí. Přesto se dnes neopovažujeme tvrdit, že bychom uměli vyrobit perfektní umělou inteligenci.

Způsobů implementace umělé inteligence je několik. První cestou jsou takzvané expertní systémy. Ty se skládají z „uvažovacího“ mechanismu a báze znalostí, která je vytvářena odborníky v daném oboru. Používají se s úspěchem v diagnostice a jiných odvětvích lidské činnosti. Dalším směrem, jak vytvořit umělou inteligenci, je inspirovat se přírodou. Na teorii přirozeného výběru a evoluci stojí genetické algoritmy

a genetické programování. Pokud nedokážeme naprogramovat dostatečně výkonné řešení problému, či ho nedokážeme ani navrhnout, je možnost použít tyto metody.

Další a poslední, námi uvažovanou, možností, jak vytvořit AI, jsou neuronové sítě. Tyto sítě jsou simulací neuronů a jejich propojení v mozku. Každý umělý neuron, stejně tak jako ten skutečný, dostává vstup (vstupy), na který aplikuje přenosovou a později aktivační funkci. Pokud výsledný signál překročí určitou mez (tzv. threshold), tak je poslán na další neurony vynásobený koeficientem synapse. Tyto neuronové sítě mají zajímavé schopnosti – učit se a pamatovat si.

Zadání

Sestrojit univerzálního robota, schopného pohybu, který ponese počítač na platformě x86 s běžným operačním systémem. Bude osazen množstvím senzorů pro zjištění stavu okolí. Těmi bude digitální kompas a otočný IR radar, laserový radar, kamera, podlahové IR senzory a senzory nárazu.

Dále si klademe za cíl vytvořit umělou inteligenci, která by na reálné fyzické implementaci tohoto robota dokázala stroj řídit a navigovat jej bezpečně uvnitř budovy či plnila rozličné úlohy. Další práce spočívá ve vytvoření obslužného softwaru pro analýzu a tvorbu knihoven, potřebných k vytvoření této umělé inteligence.

Toto robotické vozidlo nám bude díky své univerzálnosti sloužit k dalšímu výzkumu a vývoji umělé inteligence a aplikaci různých algoritmů robotiky. Bude brána v úvahu i budoucí, snadná rozšiřitelnost stroje o další moduly.

Analýza

Přestože jsme psali, že robotika je obor s menším počtem zájemců, než programování a vývojářství, i zde jsou oblasti, kterým se dnes věnuje širší počítačová veřejnost. Velice populárními jsou např. souboje robotů „sumo“, které jsou dnes již snadno dostupným koníčkem pro počítačové nadšence a to formou stavebnic.

Oblíbené jsou i jiné robotické disciplíny, které spočívají v přimění malého robota dělat elementární úlohy jako např. vytlačování míčků z arény (stolu), projíždění prostorem, aniž by naboural aj. Tyto všechny však pracují s roboty, kteří obsahují jednoduché jednočipové procesory. Roboti s nimi mají nízkou variabilitu a jejich možnosti jsou takové, jaké jim vývojáři definují jejich konstrukcí a programem pro procesor. Konstrukce takových strojů je dnes možná a cenově nenáročná za pomoci stavebnice Lego.

Naše myšlenka je však o krok dále. S příchodem malých počítačů platformy x86 se otevírá možnost nasadit takovýto počítač na robota a ovládat ho přímo z vyššího operačního systému. To by přinášelo mnoho výhod. Od modernějších vývojových nástrojů, přes možnosti bezdrátové komunikace, až k výpočetnímu výkonu a možnosti zpracovávat mnoho dat rovnou na počítači robota. Přímo se otevírá cesta k nasazení složitější umělé inteligence a zpracovávání dat pomocí složitějších algoritmů – např. neuronových sítí.

Velmi zajímavé bude použít spolu s různými konvenčními senzory i kameru. Analýza obrazu z kamery je dnes samostatná a velmi perspektivní vědní disciplína. Při kvalitní implementaci a dostatečném výpočetním výkonu může být kamera velmi praktickým senzorem.

Globálním cílem naší práce je tedy vytvoření univerzální robotické platformy, jenž dovolí další výzkum a vývoj umělé inteligence v reálném prostředí. Vzhledem k požadavku na univerzálnost celé platformy se zdá být nejvýhodnější právě počítačová platforma x86 (namísto platformy postavené výhradně na jednočipech - PICAXE, Javelin). Platforma x86 je levná, dosti variabilní a poskytuje již bohaté zázemí jak pro její konstrukci, tak i pro vývoj softwaru nad ní.

Návrh

Základní podoba robota

Nejméně náročné je vytvořit pásový podvozek tvaru „pramice“. Uvnitř mohou být uloženy komponenty a nad nimi se dají snadno vystavět patra s dalšími.

Pro snadnou konstrukci a jízdní vlastnosti bude lepší užití pásového podvozku, než podvozku s koly, nebo nohami. Situace si vyžádá vlastní konstrukci podvozku. Komerčně vyráběné budou pro náš koncept nevyhovující.

Modul K8055 – ovládání periferií

Abychom mohli z počítače ovládat periferie – spínat výstupy a kontrolovat vstupy, je nutné na to mít samostatné zařízení. Operační systém našeho počítače není tzv. real-time. To znamená, že není možné kontrolovat jeho činnost a s tím související ovládání vstupů/výstupu s přesností v řádu milisekund.

Jednou z možností je užití stavebnicové desky od firmy Velleman s názvem K8055. Jelikož vlastnění tohoto zařízení předcházelo návrhu (a bylo původcem nápadu naší práce) nebudeme se zabývat výběrem tohoto modulu.

Velleman, společně s touto kartou, dodává i DLL knihovnu, která umožňuje ovládat ji z různých programovacích jazyků. Jazyk Java neumožňuje volat přímo jakoukoliv metodu z DLL knihovny, je zde nutnost používat tzv. JNI. Až pomocí tohoto rozhraní můžeme ovládat desku. Velice důležité je ovládání dvou takovýchto desek v jediném programu. To také bude zajišťovat tento modul.

Funkce tohoto modulu budou volány vyšší vrstvou, která bude zajišťovat samotné ovládání hardwaru robota.

Modul pro jízdu

Tento modul bude obsahovat funkce pro jízdu a otáčení vozidla. Důležitou součástí je spolupráce s digitálním kompasem a ostatními senzory pro přesné otáčení a rovnou jízdu. Bude obsahovat funkce pro přepočet hodnot z úhlových jednotek do jednotek digitálního kompasu a zpět. Vhodné zapojení elektromotorů bude umožňovat volit libovolné rychlost otáčení každého pásu robota zvláště.

Modul radar

Modul radar bude mít na starosti ovládání hlavních senzorových zařízení robota. Bude zajišťovat otáčení IR radaru a snímání informací ze senzorů na něm umístěných. Bude integrovat funkce pro počítání časových prodlev v závislosti na úkonech (otáčení o malé a velké úhly), přepočítávací funkce pro vzdálenosti a úhly, se kterými každá jeho komponenta pracuje.

Robot bude také obsahovat laserový radar. Ten bude využívat linku světla, která bude promítána před robota. Scenérie tvořená tímto paprskem, která se bude zakřivovat na objektech před ním, bude snímána kamerou. Vlivem zakřivení laserové linie v obrazu bude možné určit vzdálenosti od objektů v blízkém okolí.

Modul jazyka SLISP

Tento modul zprostředkovává interpretaci zdrojového kódu programu jazyka SLISP nad JVM. Využívá k tomu námi upravený open-source LISP interpretu Jatha. Jeho výhodou je taktéž snadnější interoperabilita se třídami a metodami napsanými v jazyce Java.

Modul genetického programování

Tento modul slouží ke generování programů ve formě stromu jazyka SLISP a jejich šlechtění (mutace, křížení atd.). Modul bude obsahovat funkce na selekci jedinců z populace typu *fitness proportionate*. Pro jejich pěstování bude obsahovat *full method* a *grow method.*

Modul komunikace s hardwarem

Tento modul obsahuje interface komunikačních prostředků na ovládání periferií robota. Tento modul obsahuje také fasádovou třídu, jež ještě zaobaluje konkrétní implementaci komunikace s hardwarem. Tato třída např. zjišťuje hardwarovu adresu komunikačního prostředku.

Modul komunikace v režimu s2c

Modul má za úkol zprostředkovávat komunikaci mezi klientem a serverem. Bude zapojen, pokud robot bude v manuálním režimu ovládání. Využívá se komunikace RMI. Modul obsahuje serverové rozhraní, generování „stub“ objektů a komunikaci s RMI registry. Alternativní komunikace probíhá přímo přes sockety. V tomto režimu poběží celá softwarová architektura na klientu a přenášet se budou pouze informace o ovládání periferií. S výhodou se zde dá použít socketové implementace modulu ovládání přípravku K8055.

Modul neuronových sítí

Modul má zejména 2 úkoly: natrénovat síť na určitou úlohu a již naučené sítě používat. K obojímu je potřeba modulu zpracování vstupních dat a signálů. Z něho proudí již relevantně zpracovaná data pro konkrétní neuronovou síť. Neuronové sítě jsou různého typu, pro každou úlohu je navržena jak architektura, tak její rozměr a další jiné parametry.

Pokud je síť naučena, je uložena do lokálního filesystému na flash paměť. Tím je zaručena její znovu-použitelnost. Když je síť zrekonstruována v paměti, tak může začít nejdůležitější fáze – vyhodnocování signálů z prostředí. Jelikož robot bude disponovat solidním výpočetním výkonem, tak počítáme s analýzou téměř v reálném čase.

Modul plánování cesty

Tento modul slouží k naplánování trasy jízdy z aktuální pozice do cílové. Cíl bude určen pomocí motivačního modulu (to, co chce robot vlastně dělat) a analýzou prostředí pomocí neuronových sítí či statické analýzy. Cesta se určí pomocí jednoduchých „Bug“ algoritmů, např. VisBug či automatickým upravováním směru jízdy v reálném čase.

Hardware

Podvozek a pohon

Výběr typu podvozku

Výběr podvozku je klíčová otázka, kterou si každý konstruktér robota položí na samotném začátku práce. Na správném výběru totiž závisí budoucí možnosti robota, množství příslušenství, které robot ponese a finální uplatnění. Pokud budeme mluvit o robotech pohybujících se po zemi, rozlišujeme 3 základní typy. Robot s koly, pásy a kráčející robot. Každý typ je hojně používán, každý má své nejlepší uplatnění, výhody a úskalí. My jsme od robota požadovali pohyb všemi směry, možnost otáčení na místě a snadnou konstrukci. Tomuto nejlépe vyhovuje pásový model.

Při pozdějším sestrojování nám bylo dáno za pravdu. Kráčející podvozek by byl příliš náročný na konstrukci a robot s koly by nesplňoval všechny podmínky. Otáčení na místě je daleko lepší u pásového podvozku. Mimo to, aby měl podvozek s koly alespoň nějakou prostupnost (myšleno v rámci prostředí v budově, případně venku) bylo by vyžadováno samostatné odpružení všech kol. A to je taktéž konstrukčně náročné.

Velikou předností současné verze podvozku je značně veliká plocha, na kterou lze umístit komponenty a moduly robota. Mimo to pásové verzi nedělá problém uvést i 11,5 kg hmoty, kterou robot nabyl. Předností je i veliká variabilita, s jakou se dají na podvozek moduly umístit.

Realizace podvozku

Vzhledem ke konceptu celého projektu je potřeba podvozek značných rozměrů. Všechny standardní komerčně prodávané kusy jsou nevyhovující a současně enormně drahé. Návrh vlastního řešení tedy byl na místě. První krok se musel učinit s výběrem pásů. Na trhu je opět jen omezený výběr plastových a gumových pásů, které jsou ke všemu ještě dosti krátké. Proto jsme zvolili vlastní variantu, a to rozvodový řetěz z automobilu značky Škoda (Favorit, Felicia). Spojením dvou těchto dvouřadých řetězů

z kalené oceli jsme dostali pás, který vyhovoval pro vytvoření podvozku o délce minimálně 30 cm. S výhodou jsme využili součást rozvodové soustavy vozidla – zubaté kolo, které na robotu posloužilo pro přenos hnací síly na pásy.

Podvozek samotný je konstrukce 1,5 mm tlustého, hliníkového plechu, ohnutého do tvaru pramice. Dlouhý je 41,5 cm a s pásy široký 28,5 cm. V něm jsou usazené 2 stejnosměrné motory s planetovou převodovkou, které přes jeden převod (do pomala) pohánějí zubatá kola řetězů. Každý pás je poháněn samostatným motorem. To umožňuje, aby se každý pás otáčel na opačnou stranu, a tím se podvozek otáčí kolem svislé osy na místě. Vypínáním (změnou rychlosti) jednoho, nebo druhého pásu je umožněno zatáčení. Pohonná soustava je umístěna v zadní části stroje. Přední velké kolo je tedy pouze opěrné. Je vyrobeno obráběním z jednoho kusu hliníku. Samotný robot stojí na každé straně na šesti malých, pojezdových kolech, která jsou vysoustružena ze silonu. Ta jsou vždy po dvojicích (vedle sebe) spojena plechovým lichoběžníkovým vahadlem. To je za střed volně zavěšeno na boku podvozku. Takto jsme docílili jistého stupně volnosti nosných kol, která se přizpůsobí terénu podobně, jako by každé z nich bylo zvlášť zavěšeno na odpruženém rameni.

Druhou polovinu podvozku zabírají baterie k napájení počítače a pohonná baterie pro motory. V jejich okolí je umístěná pomocná elektronika ovládání motorů a měniče k napájení počítače. Prostor motorů je překryt železným plechem, to z důvodů stínění elektromagnetického pole. Na něm je umístěn měnič z 24V na 5V a USB přípravek K8055 (viz kapitola o K8055). Nad prostorem s bateriemi je umístěn hliníkový plech nesoucí základní desku počítače. Nad vším tímto je vystavěno druhé patro, opět z hliníkového plechu, které skýtá prostor pro další komponenty. Jsou jimi další deska K8055, otočný IR radar, kompas a pomocná elektronika. Zadní svislá část podvozku obsahuje spínače a konektor pro nabíjení baterií.

Ovládání elektromotorů

Jsou zde dvě základní metody, jakými můžeme motory ovládat. První spočívá v prostém vypínání a zapínání přívodu plného napájecího napětí do motorů. Tato možnost je konstrukčně jednodušší, a proto jsme ji pro začátek použili. K tomuto účelu pohodlně posloužili digitální výstupy přípravku K8055. Tato varianta se později ukázala, jako nevyhovující, pro nemožnost regulace rychlosti otáčení motorů. Pro citlivé ovládání

celého stroje je regulace otáček nezbytná. Otočení robota o určitý úhel není prakticky možné bez zpomalení otáčení ve fázi přibližování ke konkrétnímu úhlu. Je nutné také kompenzovat nestejnou rychlost, s jakou se motory otáčejí při stejném napájecím napětí (toto nelze jinak konstrukčně ovlivnit).

Druhá varianta spočívá v regulaci napájecího napětí motorů. Tím se docílí regulace otáček každého motoru. Realizace spočívala v obohacení předchozího zapojení spouštění motorů o prvek, který zajistí proměnlivost napětí.

Zapojení elektromotorů

Každý motor je k napájecímu akumulátoru připojen přes tzv. "reléový H můstek". Je to dvojice elektromagnetických relé, které ovládáme z počítače přes přípravek K8055. Toto zapojení umožňuje dvěma digitálními výstupy spouštět motor do obou směrů otáčení. Pro dva motory jsme tedy použili čtveřici relé a 4 digitální výstupy z K8055.

K regulaci otáček motorů využíváme pulzně-šířkovou modulaci (PWM) analogových výstupů na přípravku K8055. Těmito vysokofrekvenčními pulzy (23kHz) modulujeme proud do motorů a tím ovlivňujeme jejich otáčky.

Požadavky na počítač

Koncept našeho robota vyžaduje značný výpočetní výkon. Do dnešní doby byl právě on hlavní brzdou umělé inteligence implementované do samotného robota. S novými, malými procesory a velkými kapacitami operačních pamětí se nám otevírají dveře pro pokročilé autonomní systémy. Požadovali jsme základní desku malých rozměrů mini ITX s alespoň dvoujádrovým procesorem a operační pamětí s kapacitou v řádech GB. Zásadní pro nás byla spotřeba celého systému. U stolních počítačů není problém překročit hranici spotřeby 100W. To je však pro napájení z baterií neúnosné. Problém by nastal i při konstrukci měničů jednotlivých napájecích větví.

Parametry počítače

Základní deska: Intel D945GCLF2 Little Falls 2
CPU: Dvoujádrový procesor Intel Atom 330 na frekvenci 1,6 GHz

Paměť RAM: 2GB DDR2

Paměťové médium: Paměťová CF karta s kapacitou 2GB, rychlost 300x
(připojeno přes redukci do sběrnice IDE)

Požadavky počítače na napájení

Napájení počítačové základní desky je zprostředkováno (dle standardu ATX) pěti různými napěťovými větvemi. Konkrétně +12V, +5V, +3,3V, -5V a -12V. Podstatné je, aby byly stabilizované (podávaly stále toto napětí) a aby byly připravené na značné proudové odběry. Ve větvi +5V jednotka vyžaduje proud několika ampér. Bylo nutné vyřešit přívod s označením PG (Power Good), na kterém standardní zdroj hlásí do základní desky, že je funkční a v pořádku (počítačové zdroje mají přepěťové a podpěťové ochrany). Z důvodu jednoduchosti našeho zdroje je nutné vždy po zapnutí ručně přivést na vodič PG napětí +5V.

Napájení

Použité baterie

K provozu stroje jsme použili celkem 3 akumulátory, tzv. gelové, které vyhovují našemu účelu. K napájení motorů pro jízdu byl použit menší akumulátor o napětí 6V a kapacitě 5Ah. Použité stejnosměrné motory mají dohromady v zátěži odběr cca. 1A, což pro tuto baterii není problém. Do budoucna plánujeme použít větší model. Tento zatím však stačí. Pro napájení počítače jsou použité do série zapojené 2 akumulátory o celkovém napětí 24V a kapacitě 2,8Ah. Při současném odběru počítače a všech komponent stroje vystačí k 1,5hod provozu (s rezervou).

Napěťové měniče

Jak již bylo řečeno, počítač vyžaduje přísun elektrické energie na 5ti napěťových větvích. Jsou jimi +12V, +5V, +3,3V, -5V, -12V. K nim bylo nutno tedy vytvořit měniče z 24V. Každý z nich je tzv. pulzní stabilizovaný měnič napětí. Tato relativně mladá technologie nám umožňuje udělat perfektně stabilizovaný měnič s velikou účinností, který je schopen vytvořit požadované napětí z jiného, většího, napětí. K vytvoření zdrojů jsme použili integrovaných obvodů v jejich doporučeném katalogovém zapojení. Jeden ze zdrojů jsme získali už jako hotový kus. Jedná se o zdroj na napětí 5V, který je schopen dodávat proud 5A. Základní deska počítače vyžaduje cca. 4A. Pro správný chod počítače je nutné před spuštěním zapnout 5V zdroj. Po stisknutí tlačítka si základní deska přes relé zapne zbývající napěťové zdroje. Toto řešení není příliš elegantní, je však použito z důvodu jednoduchosti. Samotná konstrukce tohoto systému není hlavním předmětem tohoto projektu, proto zde nebudeme tuto problematiku více rozvádět. Mimo to při finálním profesionálním nasazení by situace vyžadovala elegantnější řešení - nejlépe elektronicky spínané, včetně vodiče PG (viz kapitola o počítači).

Návrh a sestrojení plošného spoje pro napěťové měniče provedl pan Jan Báča. Jelikož nevlastníme autorství, neuvádíme schéma v technických výkresech.

USB přípravek K8055

Obecné

Propojení počítače a periferií byla jedna z nejzávažnějších otázek při konstrukci našeho robota. Každá periferii vyžaduje jiné komunikační prostředky - digitální, nebo analogový signál. Hotových řešení k tomuto účelu je mnoho. My jsme zvolili stavebnici od firmy Velleman s názvem USB interface K8055. Vzhledem k tomu, že díky tomuto zařízení vznikl samotný nápad postavit toho robota, není snad nutné uvádět jiné argumenty, proč jsme takto volili. Deska umožňuje obsluhovat kontaktní vstupy a výstupy z počítače. Zároveň umožňuje obsluhovat analogové vstupy a výstupy z počítače, což znamená, že do počítače můžeme zavádět proměnné napětí, a to zpracovat v počítači. Analogové výstupy poskytují napětí 0V - 5V a nebo můžeme využít módu PWM (Pulse Width Modulation).

Specifikace přípravku

Přípravek se koupí jako stavebnice za cca 850,- Kč v potřebách s elektronikou. Balení obsahuje vše potřebné ke kompletaci - desku tištěných spojů, součástky, návod a CD s knihovnami pro Delphi, C++ a Visual Basic. Již na krabičce se dočtete, že zařízení je pro hobby použití. Pracuje totiž se značným zpožděním. Jeden cyklus aktivace, deaktivace a kontroly vstupů a výstupů se provede přibližně za 20ms. To je pro mnohé aplikace nepoužitelné. Například ovládání servomotorů jsme byly nuceni zajistit dalším zařízením, které je schopné generovat pulzy s přesností na mikrosekundy. Avšak pro naši potřebu toto zatím stačilo. Jsme si vědomi toho, že při profesionální výrobě robota by situace vyžadovala vytvoření podobného zařízení na míru našemu projektu. Tyto přípravky jsme na robotu použili dva, z důvodu nedostatku analogových výstupů (použité k ovládání rychlosti jízdy a natočení IR radaru).

Technické informace

Spojení s PC: USB 2.0

Počet digitálních výstupů: 8

Počet analogových / PWM výstupů: 2

Počet digitálních vstupů: 5

Počet analogových vstupů: 2 (lze přepnout do módu 16bit čítačů)

Maximální výstupní napětí analogových výstupů: 4.8V (odpor 1,5k ohmu)

Frekvence PWM: 23,43kHz (šířka impulzu 0 - 100%)

Odběr zařízení (přes USB 5V): max. 70mA

Kompas

Obecně

Jednou ze základních podmínek pro funkčnost našeho robota je, aby se uměl otáčet na místě. Pokud ovšem chce tuto vlastnost dokonale využít, je třeba znát úhel, o který se robot otočil. A poté samozřejmě možnost otáčení se o přesný úhel a s tím spojené podobné vlastnosti - udržování rovného směru jízdy. Toto jde špatně provést bez zařízení, které by kontrolovalo stav natočení stroje. Takových zařízení existuje více druhů. V letadle populární gyroskopický snímač natočení by byl zřejmě nejvhodnější, ale také technicky složitý. Nechali jsme si ho proto jen v záloze do příštích let. Místo toho jsme zvolili variantu digitálního kompasu. Toto zařízení nám elektronicky sdělí úhel svého natočení vůči severu.

Popis zařízení

Digitální kompas se koupí jako samostatné zařízení ve formě malé desky plošných spojů s mikroprocesorem. V ČR je k dostání např. v internetovém obchodě HobbyRobot. Cena se pohybuje kolem 1 500,- Kč. Informace lze z modulu kompasu získat dvěma způsoby. Ten přesnější a náročnější na zpracování je digitální cestou přes rozhraní I2C. Pokud do modulu zavedeme nosnou frekvenci (cca 40kHz), můžeme poté komunikovat přímo s mikroprocesorem v modulu. Ten má v sobě registry, kterými se kompas dá kalibrovat a také se z nich dají odečítat konkrétní hodnoty (převedené do binární podoby, samozřejmě). Tento způsob je však náročný pro použití dalšího externího počítače (jednočipu) který by s kompasem komunikoval a poté dával informace řídícímu počítači. Připojit kompas přímo do počítače nelze. Druhá varianta je speciální výstup kompasu, na kterém se objevuje PWM (šířkově modulované pulzy) signál. Délka impulzů je zde závislá na natočení kompasu vůči severu. Pohybuje se v rozmezí 10 - 37 ms. Tuto variantu jsme zvolili pro jednodušší zpracování dat. Šířku impulzů převádíme na napětí v rozmezí 0 - 4,8V, které zpracovává deska K8055.

K převodu signálu na signál analogový používáme dalšího zařízení, které navrhnul a sestrojil pan Jan Báča. Jelikož nám nepřísluší autorství, schéma není uvedeno v technických výkresech.

IR Radar

Obecně

K měření vzdálenosti od okolních objektů se používají převážně dva typy senzorů. Odrazové ultrazvukové a odrazové infračervení (dále jen IR). Ultrazvukové senzory mají značný rozptyl (v řádu desítek stupňů) a jsou proto velmi nepřesné. Infračervené jsou pro přesné měření vzdálenosti v interiéru budov lepší. Jejich rozptyl je jen okolo 2 úhlových stupňů. V exteriéru mohou být rušené přímým slunečním svitem, zatím jsme se rozhodli pracovat na verzi pro interiér. V dnešní době vykazují IR senzory velmi dobré vlastnosti s různými odrazovými plochami - lesklé, matné, světlé a tmavé zvládají prakticky shodnými výsledky. Vydali jsme se proto touto cestou.

Pro pokrytí nejbližšího okolí robota jsme zakoupili dva senzory, jejichž kombinací jsme dosáhli optimálního pokrytí celého okolí a velmi dobré přesnosti měření. Na trhu není velký výběr senzorů. Všechny jsou od jednoho výrobce a mají mnoho shodných vlastností. Jedna z nich je i délka jednoho měření. Ta může trvat až 50 ms. Toto je největší překážka, která brání navržení rychlejšího a přesnějšího zařízení. Pro změření okruhu kolem robota je nutné se vždy s celým zařízením zastavit na dobu potřebnou k měření. To dělá celé měření dosti dlouhým (v závislosti na počtu měření / požadované úhlové přesnosti). Standardně dostupné senzory nám však lepší podmínky nenabízejí.

Samotný radar je tvořen hliníkovým rámem ve tvaru obráceného U, do kterého je zasazený modelářský servomotor, který přes ozubené kolo otáčí ramenem s čidly. Rameno je taktéž zasazené do hliníkového rámu radaru a je tvořeno nosnou hřídelí (s ozubeným kolem), plochou k uchycení senzorů (opět z hliníkového plechu) a čepem, ve kterém se celé rameno volně protáčí. Celý radar je umístěn na vrchu robota. Servomotor dovoluje otočit ramenem v rozsahu 180 stupňů (s rezervou) v dopředném směru.

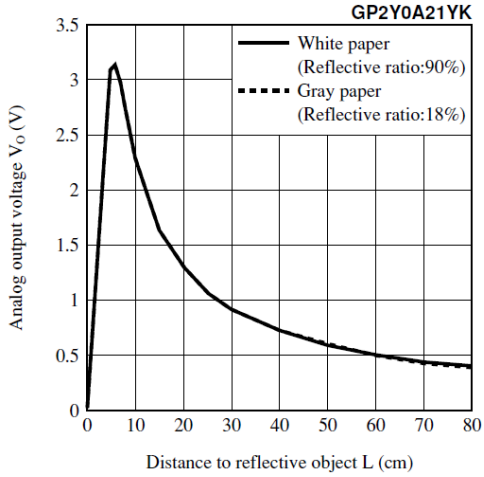
Senzor radaru č. 1

Typ: Odrazový infrasenzor SHARP GP2Y0A21
Měřící rozsah: 10 - 80 cm

Napájecí napětí: 5V

Výstupní napětí: 0-5V

Výstupem senzoru je napětí úměrné vzdálenosti od objektu



(Obr. 1.1 - Graf závislosti napětí na vzdálenosti objektu od senzoru)

Senzor radaru č. 2

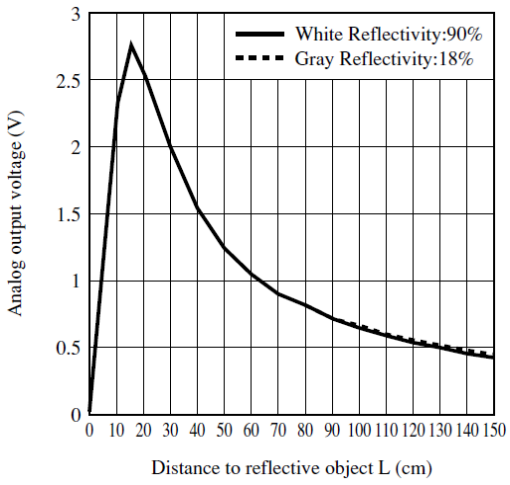
Typ: Odrazový infrasenzor SHARP GP2Y0A700

Měřící rozsah: 20 - 150 cm

Napájecí napětí: 5V

Výstupní napětí: 0-5V

Výstupem senzoru je napětí úměrné vzdálenosti od objektu



(Obr. 2.1 - Graf závislosti napětí na vzdálenosti objektu od senzoru)

Ovládání servomotoru

Pro ovládání modelářských servomotorů existuje již mnoho let ustálený standard. Pracuje na principu šířkově modulovaných pulzů (PWM). Do servomotoru se přivedou pulzy o frekvenci 50 Hz. Jejich délka se pohybuje v rozmezí 1-2 ms. Této délce je přímo úměrné natočení servomotoru. Při délce pulzů 1,5 ms je tedy hřídel servomotoru otočena do své středové polohy. Tento princip je používán z historického hlediska. Není složité sestrojit zařízení pro generování těchto šířkově modulovaných pulzů o frekvenci 50 Hz i bez použití mikroprocesorů, které zde v počátcích modelářství nebyly.

Pokud bychom chtěli vytvářet tento signál pomocí počítače, byla by zde možnost využít sériového portu počítače. Ten však není přizpůsoben pracovat s přesností jednotek mikrosekund (kterou potřebujeme pro stabilitu) a samotné generování pulzů spotřebuje značnou část procesorového výkonu počítače. Velmi vhodné jsou k tomuto účelu jednočipové mikroprocesory (PICAXE, Atmel, ...), některé z nich mají již integrované funkce pro ovládání servomotorů a krokových motorů. My jsme se však rozhodli využít analogového výstupu desky K8055, který zavádíme do zařízení na generování pulzů.

Zařízení na generování pulzů navrhl a sestrojil pan Jan Báča. Jelikož nám nepřísluší autorství, schéma není uvedeno v technických výkresech.

Při návrhu softwaru je nutné dbát na fyzické prodlevy jednotlivých senzorů, ale i prodlevu, za jakou se servomotor otočí do své konečné polohy. Tyto časy se samozřejmě liší v závislosti na použitém servomotoru a na vzdálenosti, o kterou se má otočit. Pohybují se v řádu desetin sekundy.

Laserový dálkoměr (ve vývoji)

Na našem stroji chceme použít stejný senzorový systém, jaký byl na prvním robotu na Marsu. Ten vytvářel několik rozmítaných paprsků laseru, které snímal kamerou a dle zakřivení těchto čar analyzoval prostředí okolo. My chceme použít jednodušší variantu této myšlenky. Robot bude před sebe svítit jedním laserovým paprskem, který bude z předku stroje vodorovně vycházet jen několik cm nad podlahou. Do budoucna bude nutné zakoupit kvalitní kameru, která svými vlastnostmi bude vyhovovat snímání prostředí za šera, ve kterém bude oblast s přesvíceným laserovým paprskem. Takovým požadavkům vyhovuje např. kamera Fire-i od firmy Unibrain, která se připojuje přes port FireWire.

Vytvoření rozmítaného paprsku

Z bodového zdroje světla se rozmítaný paprsek vytvoří nejlépe svícením skrz skleněnou tyčinku, nebo trubičku, která je ve svislé poloze. Jako zdroj světla poslouží klasické laserové ukazovátko. Komerčně prodávané lasery s nástavci vytvořenými přímo pro tyto účely stojí v řádu tisíců korun. Proto bude opět na místě domácí výroba tohoto zařízení. K plné funkčnosti je třeba rozsvěcet laser na signál z počítače, připojení opět vyřešíme přes desku K8055.

Měření

Měření začneme zapnutím laseru signálem z počítače. Ten vykreslí před robotem linii, která bude zakřivena okolním prostorem. Obraz obsahující tuto linii snímáme kamerou, která bude umístěna na co nejvyšším místě stroje. Pokud se vyskytne nějaký bližší či vzdálenější předmět před robotem, linie bude patřičně zakřivena a my můžeme přesně spočítat jak je daleko.

Abychom získali z obrazu tuto linii charakterizující prostor, je vhodné snímat dvojici obrázků – jednou se zapnutým laserem a podruhé bez něj. Odečtením těchto obrázků od sebe získáme zakřivenou linku, kterou analyzujeme. Pro lepší kontrast můžeme předsadit před objektiv kamery vhodný filtr.

Software

Operační systém

Prvním a velmi důležitým požadavkem na operační systém, který poběží na hardwaru robota je, aby byl kompatibilní s x86 platformou. Tato platforma se vyskytuje převážně v počítačích typu PC. Po operačním systému jsme také požadovali stabilitu a schopnost spouštět nad sebou JVM, který je potřeba pro běh programů v jazyce Java. Těmto požadavkům vyhovuje celá řada moderních operačních systémů. Operační systém také musí obsahovat či podporovat ovladače různých hardwarových periférií.

Na trhu se převážně vyskytují operační systémy společnosti Microsoft (ty jsou komerční, rodina Windows) a systémy s jádrem typu Unix (Solaris, OpenSolaris, GNU/Linux apod.). Unixové operační systémy bývají povětšinou zdarma a jsou schopné běhu na různých platformách, kdežto Windows běží prakticky pouze na PC. Zjistili jsme, že GNU/Linux a jiné operační systémy z rodiny Unix nemají podporu pro jeden z našich stěžejních modulů – ovládací desku K8055.

Po zvážení všech faktů jsme se rozhodli pro operační systém Windows Server 2003. Obsahuje všechny potřebné ovladače či je dodávají výrobci třetích stran. Také společnost Sun Microsystems dodává svou JVM právě pro tento systém. Jediným zádrhelem by mohla být cena celého systému.

Tento problém jsme vyřešili díky programu Microsoft DreamSpark[1], který umožňuje studentům středních a vysokých škol zdarma užívat vybrané softwarové produkty firmy Microsoft zdarma. Mezi takové produkty patří například Visual Studio či právě Windows Server 2003.

Operační systém Windows Server 2003, vzhledem na hardwarovou platformu robota, se instaloval na poměrně malou, 2GB velikou flash paměť, proto standardní instalace tohoto OS nepřicházela v úvahu . Museli jsme z instalace vypustit různé pro

[1] http://dreamspark.com/

nás nedůležité moduly, jako např. IIS, nápovědu, zvuky aj.. Toho jsme dosáhli za použití nástroje WinLite, který velikost instalačního obrazu zmenšil na cca. 70% původního objemu. Toto volné místo jsme upotřebili pro instalaci podpůrných programů a důležitých ovladačů.

Softwarová platforma

Softwarovou platformou rozumíme komplexní celek nástrojů, knihoven, programů a frameworků které dohromady vytvoří prostředí pro vývoj a produkční nasazení námi vytvořeného kódu. Tyto platformy se zejména liší svým zaměřením, složitostí nebo třeba tím, které firmy (či komunity) za nimi stojí.

Od softwarové platformy, která by byla prostředním pro ovládání robota a jeho samotnou umělou inteligenci, jsme požadovali, aby byla dostatečně robustní pro naše účely. Musí mít velké zázemí již v hotových knihovnách, které bychom mohli případně použít a musí být kvalitně zdokumentována. Přihlíželi jsme také k našim předchozím znalostem, abychom využili co nejvíce toho, co už známe.

Po technické stránce musí být platforma schopna běhu na operačním systému Windows Server 2003. Musí podporovat více vláknové (multi-thread) prostředí a být dostatečně výkonná. Preferovali jsme technologie používající VM (virtual machine)

Analýza

Dva vážnější kandidáti byli jasní hned. Jedná se o Javu a .NET. První jmenovaná platforma (a také stejnojmenný jazyk) je vyvíjena společností Sun Microsystems a je uvolněna jako open-source. Druhá jmenovaná je výrobkem firmy Microsoft a je uzavřená. Našim výše zmíněným požadavkům vyhovují obě.

Přesto volba pro nás nebyla příliš obtížná. Vzhledem k naší zkušenosti s programovacím jazykem Java jsme zvolili právě jej. Java má širokou podporu komunity a tak pro ní existují mnohé různé knihovny a frameworky. Zejména jsme s výhodou využili neurálního frameworku Encog[2] jehož Javová verze je o generaci dále

[2] http://www.heatonresearch.com/encog

než .NETová implementace. Jazyk Java a jeho JVM jsou pro naše účely taktéž dostatečně výkonné.

Implementace

Z hlediska softwarové platformy je nejníže umístěn operační systém a JVM – Java Virtual Machine. To je program, který dokáže interpretovat bytekód produkovaný javac kompilátorem. JVM běží jako program na operačním systému (v našem případě je jím Windows Server 2003). Platforma Java obsahuje také základní balík knihoven, který je poměrně bohatý. Nazývá se JRE – Java Runtime Enviroment.

Používáme aktuální (v době psaní dokumentace) verzi Javy tj. 1.6.0_17. V tomto kontejneru běhají již naše programy. Zejména je tvoří třídy jazyka Java, které jsou podle logických celků rozděleny mezi balíčky (package). Jejich jednotlivý výčet a popis je uveden níže.

Nepoužili jsme čistě pouze jazyka Java. Nízkoúrovňové věci, jako např. ovládání portů USB a s tím související komunikace s přípravkem K8055, nejsou možné. Proto jsme museli přistoupit k napsání některých nativních kódů. Jedná se o JNI fasádu nad DLL knihovnou k K8055 a serverovou část socketové implementace komunikace s ovládací deskou. První jmenovaný je napsán v C++ a zkompilován pomocí kompilátoru společnosti Microsoft. Druhý je naprogramován v produktu Borland Delphi 7.

Ovládání periferií

Ke každé důležité periferii jsme museli vytvořit třídu, která jí reprezentuje v softwarové části robota. Tyto třídy obsahují vlastní aplikační logiku a komunikaci s k nim určeným hardwarovým prostředkem.

Modul K8055

Na tento přípravek jsou napojeny všechny používané periferie. K tomuto modulu výrobce dodal již zkompilovanou DLL knihovnu pro 32bitový operační systém Windows a také hlavičkové soubory jazyka C.

Při průzkumu Internetu jsme zjistili, že někdo rozhraní JNI pro desku K8055 již napsal. Bohužel, rozhraní že je určeno pouze pro třídy umístěné v defaultním package, což je pro nás nepřípustné. Také celá tato implementace nebyla příliš povedená a stabilní. Tyto důvody nás vedly k vytvoření vlastní implementace.

Vytvořili jsme nativní volání metod z této knihovny z Javy pomocí rozhraní JNI. K používání JNI je třeba, aby knihovna, jejíž metody voláme, měla specificky pojmenované názvy metody.

Java_Package_ClassName_MethodName(...) {

Kde Package je jméno balíčku, ve kterém je umístěna Javová třída, která volá metody přes JNI, `ClassName` je její jméno a `MethodName` je jméno metoty, která odpovídá jménu metody v Javovské třídě.

Takováto Javová třída musí mít ještě definovaný statický blok, ve kterém se pomocí volání `Systém.loadLibrary(„...“)` načte požadovaná knihovna a poté seznam statických, veřejných metod s modifikátorem public, které odpovídají metodám v binární knihovně (nativní kód). Všechny tyto náležitosti má naše třída `usb.JniBoard`.

V programovacím jazyce C++ jsme vytvořili fasádu, která má názvy metod požadované rozhraním JNI. V těchto metodách se volají metody z originálně dodávané knihovny.

Komunikace s periferiemi

Motory

Ovládání motorů pro pohon robota zajišťuje třída `robot.Engines`. Obsahuje metody pro otáčení vlevo a vpravo, couvání i jízdu dopředu. Jelikož jsou motory připojeny přes analogový výstup, lze tak ovládat i rychlost jízdy. V této třídě je taktéž možnost otáčení, které je možné o libovolný úhel. Tato třída si bere potřebné informace z třídy `robot.Compass`. Znalost aktuálního úhlu natočení a proces přibližování k cílovému úhlu se neobejde bez modulu kompasu. V dalším vývoji bude součástí této

třídy i spolupráce s modulem radaru. Robot si nejprve zjistí, zdali se v jeho okolí nenachází viditelná (pro radar) překážka, která by zapříčinila nedotočení. Tuto skutečnost se samozřejmě dozvíme již při zastavení otáčení za spuštěného pohonu, je však lepší jí předcházet.

Kompas

Tento modul je v softwaru realizován pomocí třídy `robot.Compass`. Obsahuje statickou metodu `getAngle()`, která vrací aktuální úhel natočení robota. Signál z modulu kompasu k nám přichází přes analogový vstup desky K8055. Hodnoty na vstupu (0-255) však nejsou přímo úměrné úhlu natočení. Bylo potřeba vytvořit si tabulku, do které jsme zanesli naměřené hodnoty pro konkrétní úhel natočení robota. Měření jsme prováděli každých 10 stupňů natočení. Mezi těmito hodnotami poté provádíme lineární aproximaci. Ještě před aproximací je však nutné provést softwarové vyhlazení signálu, jelikož hodnoty se neustále pohybují na vždy různém intervalu hodnot. Toto vždy provádíme průměrováním tří posledních naměřených hodnot. Vzhledem k tomuto vyhlazení počítáme natočení pouze s přesností 10 úhlových stupňů. Pro pohyb robota je to dostatečné. V budoucnu je zde možnost využít I2C výstupu kompasu a s použitím další externí elektroniky tak získat mnohem přesnější a nezkreslený signál.

Radar

Tuto periferii tvoří soustava servomotoru a dvou IR radarů. Třída `robot.sensors.Radars` dokáže natočit rameno radaru na určitý úhel a poté změřit vzdálenost objektu. Vzdálenost se projeví změnou napětí na čidlu, tuto hodnotu přivedeme do analogového vstupu, který nám do aplikace vrátí hodnotu z rozsahu přibližně 20-150. Vzdálenost objektu od senzoru není lineárně závislá na hodnotě na analogovém vstupu. Vybrali jsme několik vzdáleností, pro které jsme změřili hodnoty napětí a zbytek hodnot mezi nimi lineárně aproximujeme.

Balíček `robot.sensors` obsahuje objekt `Ground`, který je určen pro implementaci čidel na podlaze, které by měly zabránit např. pádu robota ze schodů. Hardwarově tato část není zatím realizována, počítá se s jejím zhotovením v delším časovém horizontu.

Kamera

Na těle robota je umístěna webkamera o rozlišení 1,3 Mpix. Obraz z ní je do počítače a do našich programů přenášen pomocí Java Media Framework. Tento framework umožňuje Javě, jakožto platformě hardwarově a systémově nezávislé, přistupovat na jisté periferie připojené k systému. Jednou takovou je naše webkamera.

Třída `MediaLocator` nám zprostředkuje tento multimediální prostředek na adrese `vfw:Microsoft WDM Image Capture (Win32):0`. Pokud chceme získat jednotlivý obrázek z kamery a pracovat s ním, musíme vytvořit instanci třídy `FrameGrabbingControl` která nám přes další dodatečné metody vrátí obrázek z kamery. Pokud chceme obraz po síti tzv. streamovat, tak nám JMF nabízí možnost použití RTP (Real-time media protocol) protokolu. Pro zpracování dat z kamery se používá tzv. procesorů, které jsme v současné době implementovali dva. `FrameProcessor` a `SelectionProcessor`.

První jmenovaný spočítá průměrnou barvu obrázku (aritmetický průměr dekadických hodnot jednotlivých barevných složek všech pixelů). Pixely tmavší než průměr zamění za černou barvu, světlejší za bílou. Takto nám vznikne dvoubarevný obrázek, který používáme k testování a učení neuronové sítě na rozpoznávání obrazu.

Druhý jmenovaný dokáže v obrazu zachytit pohybující se předmět. Funguje tak, že projde celý obrázek „po čtvercích“ stanovené velikosti. Pro každý čtverec metoda `rankSquare` ohodnotí jeho stav číslem. Dělá tak součtem dekadických hodnot jednotlivých barevných složek všech pixelů v daném čtverci. Pokud se hodnota takto ohodnoceného čtverce změní oproti následujícímu snímku o jistou mez, tak se tento čtverec označí červeným rámečkem. Tento způsob ohodnocování a vyhodnocování změny obrazu se ukázal v praxi jako velmi dobře fungující. Vedlejším efektem tohoto procesoru je, že pokud se robot pohybuje, tak dokáže detekovat některé významné objekty v místosti, jako např. rárubně, nohy stolu aj.

Implementaci těchto tříd je možno nalézt v balíčku `jsaladin` a jména tříd viz. výše.

Komunikace s robotem

Fyzická vrstva

Komunikace, tj. proces předávání informací z jednoho místa do druhého, s robotem probíhá pomocí standardu bezdrátové komunikace Wi-Fi, specifikace IEEE 802.11. Využíváme verze 802.11g, která pracuje na frekvenci 2,4 GHz s teoretickou propustností sítě až 54 Mbit/s.

Síťová vrstva

Využíváme standardního protokolu TCP/IP nad kterým stavíme další komunikační protokoly. Prvním z nich je vyvinut námi pro potřeby socketové implementace komunikace Javy a ovládacího prostředku K8055. Jeho popis následuje níže.

Síťový protokol pro komunikaci s K8055

Tuto komunikaci na straně Javy zajišťují celkem 3 třídy:

- Boardable
- SocketBoard
- UsbBoard

Boardable je interface, který obsahuje seznam metod, která musí jeho implementace bezpodmíněčně zvládat. Tento interface nám zajistí jednoduchou výměnu implementace komunikace s deskou napříč celou platformou.

SocketBoard je třída implementující rozhraní Boardable. Obsahuje socketovou komunikaci s naším protokolem, jenž bude popsán níže.

UsbBoard je třída, zaobalující konkrétní implementaci rozhraní Boardable. Má zjednodušený konstruktor, který zajistí správnou inicializaci třídy SocketBoard.

Prvním krokem je navázání TCP/IP spojení na adresu localhost (127.0.0.1) – v případě autonomního módu či na adresu odpovídající hostname „saladin“ (ip adresa

robota). První deska používá port 45054 a druhá 45055. Ke každé desce jsou připojeny jiné přístroje. Komunikace vypadá tak, že se pošle přes otevřené spojení řetězec zakončený standardním koncem řádky (\n). Na začátku je jméno metody z DLL knihovny ovládající přípravek K8055, následuje mezera a seznam argumentů této metody. Např.

```
Program: OpenDevice 1

Server: 1

(program požádal o zinicializování přípravku K8055)

...

Program: ReadAnalogChannel 1

Server: 125

(program požádal o načtení hodnoty z analogového vstupu číslo 1)
```

Na druhé straně komunikace, zde označené jako „`Server`“, je program vytvořený v produktu Borland Delphi 7. Zde pomocí komponenty IndyServer je vytvořen jednoduchý TCP/IP server, jenž je schopen volat metody z dodávané DLL k přípravku K8055 a tím ho ovládat.

RMI komunikace

Druhým způsobem komunikace, který používáme, je tzv. RMI – Remote Method Invocation. Tuto technologii vyvinula firma Sun Microsystem a platforma Java jí podporuje. Jedná se o možnost volání metod objektů instancovaných na úplně jiné JVM, na úplně jiném počítači. Díky tomu můžeme vzdáleně kontrolovat a ovládat chování i velice složitých programátorských struktur.

Je třeba vytvořit interface, který musí odpovídat třídě kterou chceme ovládat (respektive její konkrétní instanci). Jelikož tyto metody mohou vyvolat `RemoteException`, je třeba, aby kód, který je volá, byl obalen v bloku try-catch.

Vzdálená třída, kterou budeme ovládat, toto rozhraní implementuje. Poté se na nějakém počítači (typicky server) se pustí program `rmiregistry`. Java aplikace potom do tohoto registru uloží tzv. stub objektů, na němž chceme volat metody. Klientská část aplikace se připojí na stroj, kde běží program `rmiregistry` a volá metody vzdáleného objektu stejně tak, jako by byl lokální.

Moduly

Základní jednotky – stav robota v prostředí

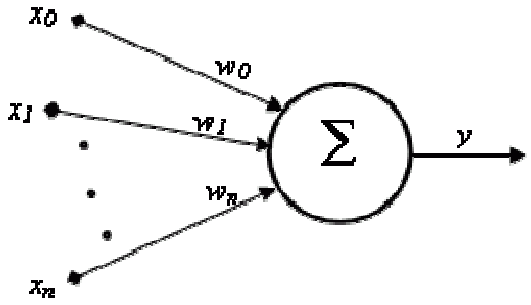
Robot musí vždy vědět, kde se v prostředí nachází. K tomu je potřeba stanovit nějaký souřadný systém a jednotky. Např. pro svět, který vypadá jako šachovnice by se stav dal charakterizovat jako dvousložkový vektor (x,y), kde čísla x, y náleží 1..8. Náš reálný svět ovšem vypadá jinak a nepoměrně složitěji, tudíž musíme zvolit sofistikovanější vyjádření stavu robota.

Systém, kdy by robot začínal na virtuálních souřadnicích [0;0] a kdy by se pomocí počítání doby jízdy v různých směrech počítaly aktuální souřadnice, jsme museli zavrhnout jako nefunkční. Ačkoliv máme velmi přesné přístroje a výpočetní výkon k počítání polohy tímto způsobem, tak tato metoda není vyhovující. Je to dáno tím, že žádné měření není a ani nemůže být absolutně přesné, takže postupem času by odchylka od skutečné polohy byla neakceptovatelná. Tento aspekt při tvorbě robotů je obecně poměrně znám[1]. Rozhodli jsme se tedy využít systému globální znalosti prostředí.

V tomto systému nám stav robota reprezentuje úhel natočení (snadno a absolutně zjistitelný z kompasu) a globální mapa prostředí. Tato mapa je tvořena pomocí otočného infračerveného radaru. Robot se tímto radarem „rozhlídne“ a po krocích určité velikosti (např. 10 úhlových stupňů) zjistí vzdálenost od nejbližšího objektu v tomto směru. Seznam těchto hodnot spolu s úhlem natočení robota vůči severu nám dává stavový vektor robota v prostředí. Díky tomuto způsobu je chyba měření jen jako konstanta oproti způsobu průběžné akumulace informací, kdy by odchylka nevyhnutelně vzrůstala.

Neuronové sítě

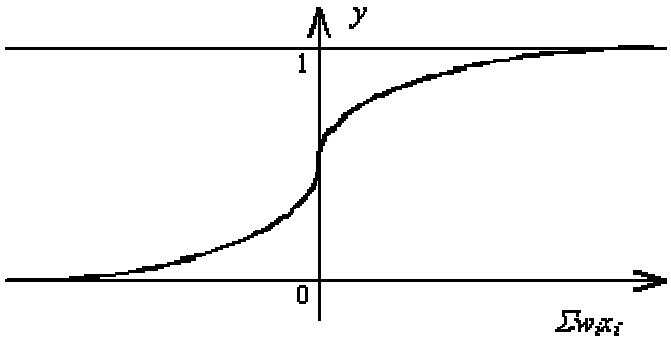
Dnes již víme, že lidský mozek obsahuje mnoho buněk zvaných neurony vzájemně propojených pomocí spojení zvané synapse. Umělé neuronové sítě, se kterými se setkáváme v počítači, se snaží modelováním těch živočišných dosáhnout stejných, ba dokonce lepších výsledků než jejich živočišní kolegové.



(Obr. 3.1 – Matematický model neuronu)

x0, x1, … xN jsou vstupní hodnoty nebo jinak řečeno vektor vstupních hodnot. w0, w1, … wN jsou koeficienty synapsí. Určují tím sílu a důležitost spojení. Každý neuron má také aktivační funkci f(Σ (xi*wi) - θ), kde θ je aktivační práh. Jednoduše řečeno, pokud signály nebyly dostatečně silné, tak se vlivem existence prahu nedostane žádný signál do neuronů spojených s tímto.

Aktivační (přenosová) funkce se aplikuje na sumu vstupních signálů vynásobených příslušnými koeficienty synapsí. Máme různé aktivační funkce např. signum, tangens hyperbolický nebo jak vidíme níže, často používanou funkci sigmoid.



(Obr. 3.2 – Aktivační funkce Sigmoid)

Neuronovou sítí se v zásadě mohou šířit logické hodnoty, bipolární či reálné. Za logické hodnoty považujeme true x false, 1.0 x 0.0. Bipolárními rozumíme hodnoty 1.0 a –1.0. Každá používá jiných hodnot.

Samotný neuron přirozeně neumí skoro nic. Proto je sdružujeme do struktur zvané sítě.

Architektury sítí

Propojením neuronů pomocí synapsí získáme strukturu zvanou síť. Tato struktura má opět jistou analogii v lidské, potažmo v živočišné, anatomii. Rozlišujeme několik typů sítí. Každá se hodí na nějaký jiný úkol. Popíšeme jich zde pouze pár, zejména ty, které používáme.

Sítě se obecně skládají z vrstev. Vstupní vrstva obsahuje takový počet neuronů jako je velikost vstupních dat. Pro obrázek to může být např. počet jeho pixelů. Síť má také vždy výstupní vrstvu. Ta má tolik neuronů, kolik je velikost výstupního vektoru. Pokud se například snažím klasifikovat obrázek a zařadit jej do jedné ze dvou množin, použiji jeden neuron na výstupní vrstvě. Neuronové sítě mohou a také často obsahují tzv. schované vrstvy (hidden layers). Tyto vrstvy se naházejí mezi vstupní a výstupní vrstvou a tvoří je množství neuronů, jejichž počet a propojení záleží na konkrétním typu sítě. Např. v případě feedforward network je při jedné schované sítě přibližně počet neuronů roven 2/3 * (počet neuronů ve vstupní + počet neuronů ve výstupní).

Učení sítě

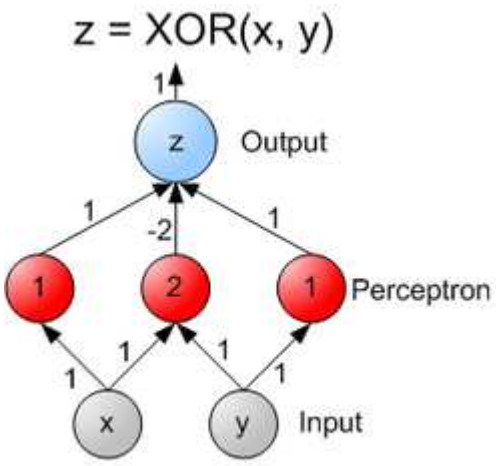
Když vytvoříme síť, tak přirozeně ještě neumí plnit úkol námi požadovaný. Stejně jako u člověka musí dojít k procesu zvanému učení. Proces učení probíhá zhruba tak, že síti předkládáme ukázková data a sledujeme výstup ze sítě. Podle chyby na výstupu upravujeme koeficienty v matici vah synapsí.

Rozlišujeme v zásadě dva typy učení – s učitelem a bez učitele. Při učení s učitelem předložíme síti data, porovnáme získaný výstup s očekávaným výsledkem a podle chyby sítě upravíme koeficienty synapsí. Tento proces opakujeme s různými i se stejnými daty, dokud síť nepodává výsledky s akceptovatelnou chybou.

Při učení bez učitele předem neznáme výsledek. Síť pouze dostává sadu vzorů, které si sama klasifikuje.

Feedforward network

Síť s dopředným šířením – signál v sítí se šíří pouze jedním směrem; dopředu. Síť toho typu je uvedena na obrázku níže a je např. schopna vykonávat logickou funkci XOR. K učení těchto sítí se často používá algoritmus Backpropagation.



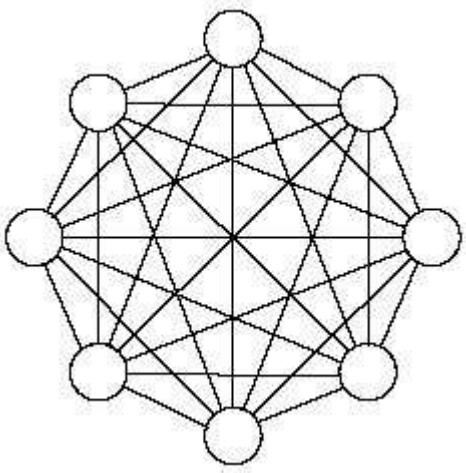
(Obr. 4.1 – Percepton network)

Obecně jsou takovéto sítě schopny klasifikace lineárně oddělitelných vzorů. Pokusili jsme se použít této sítě klasifikaci obrazu z kamery, který jsme převedli na matici logických hodnot reprezentující pixelový obraz[1]. Touto metodou jsme dosáhli pouze střídavých úspěchů. Naše síť obsahovala 3072 vstupních neuronů, skrytá vrstva měla 200 neuronů a na výstupní vrstvě 1 neuron. Chtěli jsme zkusit klasifikaci předmětu pouze na je – není. Výsledky nebyly příliš slibné. Detekce fungovala pouze se střídavými úspěchy.

Soudíme, že námi zvolený učební vzor, je chybný. Předpokládáme, že musíme objekt na kameře detekovat a přenést ho na nějakou matematickou funkci, čímž bychom dosáhli vyššího výkonu sítě či zvolit úplně jinou architekturu sítě, např. SOM nebo Hopfieldovu síť.

Hopfield network

Takováto síť obsahuje shodný počet neuronů na vstupní a výstupní vrstvě. Síť je schopna se naučit určitý vzor a posléze zreprodukovat poškozený vzor zpět do původního tvaru.



(Obr. 4.2 – Hopfield network)

Z obrázku vidíme, že vstupní neurony jsou zároveň výstupními. Počet vzorů, které si síť dokáže „vybavit“ závisí na počtu neuronů. Čím uložíme do sítě více vzorů,

tím je pravděpodobnější vznik tzv. fantomů – falešné vzory, které do sítě nebyly vloženy a které bude vracet jako výsledek.

Self-organizing map

Sítě SOM, neboli samoorganizační mapa, jsou určené ke klasifikaci vzorů. Této vlastnosti využíváme při analýze mapy prostředí. To, co robot vidí, se rozebere na význačné objekty (popřípadě se použije celá mapa) a vloží se do sítě. Síť klasifikuje vzor do určité skupiny (např. dveře, roh místnosti, volno atd.). Díky této informaci robot již rozumí, co se kolem něj nachází (seznam úhlů a k nim odpovídající hodnoty zná samozřejmě dávno, ale tady se jedná o faktické pochopení situace) a může dále s touto informací naložit. Používá se učení bez učitele.

Pokročilé metody práce s neuronovými sítěmi

Jednou takovou metodou je prořezávání neboli prunnig. Tento proces umožňuje optimalizovat již naučenou síť pomocí techniky, která dokáže lokalizovat nepoužívané neurony (s nulovou či velmi vahou synapse, tj. na výsledky se podílí nepatrným dílem) a ty odstranit bez vlivu na správnost výsledku. Důsledkem je vyšší rychlost zpracování vstupu. Tento mechanismus obsahuje nejnovější betaverze knihovny Encog o které budeme hovořit níže.

Implementace sítí

Při implementaci neuronových sítí jsme využili knihovny Encog. Tato knihovna obsahuje implementaci několika typů sítí a učících algoritmů. Knihovna je poměrně jednoduchá na používání, přesto mocná. Obsahuje např. mechanismus na ukládání již naučených sítí na disk a jejich opětovné načítání do paměti.

Dalším příjemnou vlastností knihovny je, že obsahuje program na tvorbu a testování neuronových sítí. Tento program vám dokáže posléze vyexportovat funkční zdrojový kód v jazyce Java, který dokáže načíst natrénovanou síť a pracovat s ní.

Také jsme se zabývali neurálním frameworkem NeuralJ, který je taktéž pro Javu a obsahuje i více funkcí. Ovšem tato knihovna byla velmi složitá na používání, proto

jsme se jí pro začátek vyhnuli, ovšem nevylučujeme, že v budoucím vývoji ji použijeme.

Další vývoj

Při dalším vývoji robota se chceme zaměřit hlavně na lepší využití neuronových sítí. Hardware robota je mnohem výkonnější než u srovnatelné konkurence. To nám dává možnost využít plné síly umělých neuronových sítí v oblasti umělé inteligence. Jelikož je náš robot velmi variabilní a univerzální, můžeme si zkoušet různé sítě a úkoly pro ně.

Velmi se nám také líbí FPGA (Field programmable gate array) čipy. To jsou součástky, které obsahují soustavu velmi programovatelných hradel. Umožňují navrhnout takřka libovolný design čipu či obvodu a nahrát ho do FPGA. Ten pak začne vykonávat jeho funkci. Takto je možno např. nahrát do FPGA čipu PICAXE procesor. Čipy FPGA jsou poměrně levné, tudíž snadno pro nás dostupné.

Pohráváme si hlavně s myšlenkou implementovat neuronové sítě přímo hardwarově, na FPGA. Na toto téma se již objevilo pár vědeckých publikací. Myslíme si, že budoucí vývoj bude směřovat právě tímto směrem.

Genetické algoritmy

Stejně tak jako se umělé neuronové sítě inspirovaly u přírody, tak i genetické algoritmy, respektive genetické programování, našly vzor ve světě kolem nás. Genetickými algoritmy rozumíme postup, kdy se pracuje najednou s celou množinou individuí, kde každý jedinec reprezentuje jedno konkrétní řešení problému. To v informatice nazýváme genomem jedince. Na jeho zakódování používáme specifické typy kódování (dle úlohy), např. binární či Grayův kód. Tito jedinci se kříží, mutují a vytvářejí další potomstvo podobně jako v našem světě. Vzhledem k Darwinově evoluční teorii, přežije pouze nejsilnější jedinec, čili nejlepší řešení.

Při genetickém programování genom jedince tvoří zdrojový kód programu. Zde zjevně není možné použít triviální způsob kódování jedince, jako např. binární reprezentaci či jiné, protože by takoví jedinci byli z velké části nefunkčními – jejich

zdrojový kód by nebyl validní, nešel by provést. Abychom se vyhnuly těmto a jiným problémům, vyvinuli jsme náš scriptovací jazyk stromové struktury založen na LISPu.

Na našem robotovi používáme genetického programování k tvorbě umělé inteligence, která bude mít za úkol řešit konkrétní problém. Toho docílíme namodelováním úlohy v simulátoru a spuštěním genetického procesu (tvorba potomků, vliv mutací apod.) Výsledkem je validní zdrojový kód nejlepšího jedince, který řeší optimálně úlohu. Tento zdrojový kód může být komplikovanější než by vymyslel lidský mozek.

Reprezentace programu

To, že nemůže použít obyčejný scriptovací jazyk, který bychom křížili pouze přehazováním klíčových slov, jsme už vysvětlili. To, aby nám vždy po křížení či zárodečném vygenerování vznikly syntaktické programy, nám zajistí použití stromových struktur.

Simulátor

Pro genetické programování jsme stvořili simulátor. Reprezentuje dvourozměrný svět, robot má schopnosti se pohybovat pouze v 2D prostoru – nejezdí např. do schodů. Tento svět je rozdělen na čtvercovou síť potřebných rozměrů. Nyní stačí pouze namodelovat úlohu, kterou by pak měl řešit i robot ve skutečném prostředí.

Pro jednoduchost jsme reprezentovali jednorozměrného robota - mravence (umí se také otočit na místě jako náš skutečný) s velmi omezenými schopnostmi. Tento robot by měl v zadaném prostoru „vysbírat“ potravu, která je umístěna podle vzoru „Stezka Santa Fe“[2]. Umí se pouze otočit doprava, doleva, udělat krok o políčko vpřed a použít detekční senzor, který mu řekne, jestli je před ním „potrava“. Všechny tyto funkce se v reálném prostředí na funkce robota a vhodně implementují, aby byly co nejvíce shodné se simulací. Popis toho, jak tento proces šlechtění programů funguje, je uveden v následujících kapitolách. Implementaci naleznete v balíčku `ant`.

Množina terminálů vypadá takto:

$$\tau = \{\text{LEFT, RIGHT, MOVE}\}$$

Množina funkcí takto:

$$\Phi = \{\text{IF-VOLNO, PRG2, PRG3}\}$$

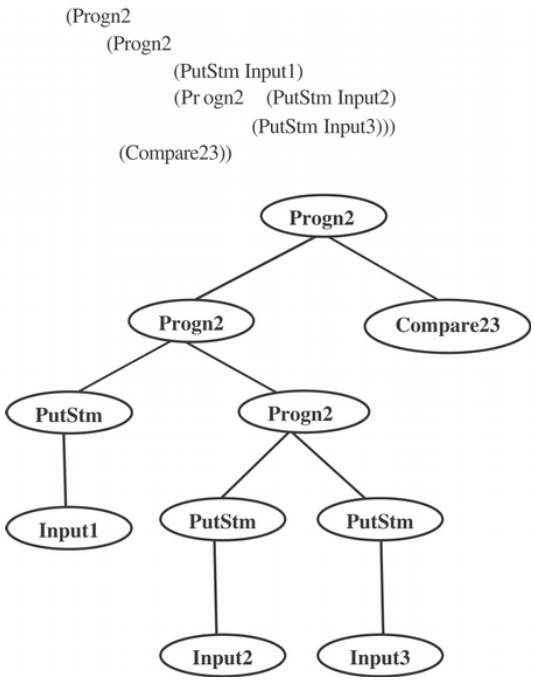
Množina terminálů je jasná a netřeba ji popisovat, pojďme se tedy podívat na množinu funkcí. Funkce IF-VOLNO reprezentuje schopnost robota (mravence) detekovat objekt před sebou. Funkce PRG2 pouze spustí právě 2 své podstromy v zadaném pořadí. PRG3 dělá totéž, avšak počet spuštěných podstromů je tři. Tyto funkce doporučuje Koza, bez nich by nebyl robot schopen kvalitně plnit zadaný úkol.

Program ve tvaru PRG2(LEFT, MOVE) otočí robotem vlevo o 90° a udělá krok vpřed.

Reprezentace programu – jazyk SLISP

To, že nemůže použít obyčejný scriptovací jazyk, který bychom křížili pouze přehazováním klíčových slov jsme už vysvětlili. To, aby nám vždy po křížení či zárodečném vygenerování vznikly syntaktické programy, nám zajistí použití stromových struktur.

Známý jazyk se stromovou strukturou je LISP. Rozhodli jsme se tedy znovu nevynalézat kolo a s výhodou použít jeho stromové struktury k našim účelům. Jeho struktura vypadá takto:



(Obr. 5.1 – LISP)

K implementaci našeho jazyka SLISP (Saladin LISP) jsme využili existujícího Java interpretu Common listu LISPU jménem Jatha[1]. Aby jednotlivé listy jazyka LISP byly schopné spouštět Javový kód, museli jsme implementovat vlastní funkce – upravit interpret tohoto jazyka. Udělali jsme to tak, že jsme napsali vlastní třídy, reprezentující nové funkce, které rozšiřují třídu LispPrimitive. Je potřeba taktéž překrýt metoda Execute a napsat správnou implementaci. Se stavovým automatem LISPu se komunikuje pomocí dvou zásobníků – tyto operace musí být psány pečlivě a opatrně, jakákoliv chyba s manipulací se zásobníkem LISPu způsobí chybný běh programu či dokonce jeho pád.

Naimplementované nové funkce je třeba zaregistrovat do standardního mechanismu inicializace tohoto interpretu. Implementovali jsme všechny potřebné funkce (viz. množina terminálů a funkcí). Jejich implementaci a další implementační detaily je možno nalézt v balíčku `slisp`.

V době pokročilého stádia vývoje jsme objevili alternativu k našemu postupu. Jedná se o použití nového jazyka Clojure[1], který běží nativně pod JVM a má taktéž stromovou strukturu programu (vychází dokonce z Common LISP). V takovémto

zdrojovém kódu, který umí nativně interpretovat JVM bychom mohli volat libovolné jakové metody ze tříd bez nutnosti implementace vlastních funkcí do existujícího interpretu. Nahradit naší implementaci za tuto by nebylo složité, a proto se k ní chystáme v blízké budoucnosti.

Selekce

Z každé generace jedinců se musí vybrat vyvolená skupina nejlepších, které se dále rozmnoží – zachovají potomstvo do další generace. Pro tento úkol využíváme sekečního mechanismu fitness-proportionate, popsaného podrobněji v [kniha 1.]. Kód každého jedince se začne provádět v simulátoru a je příhodně ohodnocen podle toho, jak velkého úspěchu dosáhl. V zadané úloze musíme být schopni vytvořit ohodnocovací, tzv. fitness funkci, která by neměla být složitá na spočítání. Implementaci celé selekční funkce lze možno nalézt v třídě `slisp.Selection`.

Křížení

Než začneme jisté zdrojové kódy mezi sebou šířit, musíme vytvořit nějakou výchozí populaci jedinců. Jelikož je struktura programů jedinců uložena ve formě stromu, máme na výběr zejména ze dvou hlavních metod tvorby výchozího jedince. První možností je tzv. *full method* – vygeneruje všechny podstromy a větve do zadané hloubky. Získáme tedy strom, který je plně zahuštěn. Druhou metodou je *grow method.* Při této metodě negenerujeme všechny podstromy do stanovené hloubky, syntaktický strom tím získá větší variabilitu.

Stejně tak jako doporučuje Koza[3], při tvorbě počáteční populace jsme zvolili metodu půl na půl (ramped half and half). Např. pro populaci velikosti N = 500 a h_{max} = 6, se vygeneruje 100 jedinců s hloubkou 2, z toho 50 full method a 50 grow method, 100 jedinců s hloubkou 3 (opět 50 na 50) atd. Implementace v `slisp.TreeBuilder.`

[3] Koza J. R.: Genetic Programming. On the Programming of Computer by Means of Natural Selection, Cambridge, MA: MIT Press, 1992

Na samotné křížení jsme navrhli metodu, kdy se v každém ze dvou jedinců vyberou dva podstromy a ty se zamění. Implementace je uvedena v třídě `slisp.Operators.`

Mutace

Vliv mutace na řešení je při genetickém programování diskutabilní. Sám Koza[6] ve většině svých experimentů se spoléhá pouze na křížení. Přesto jsme mutaci implementovali alespoň způsobem, že dojdeme na některý z listů syntaktického stromu, a tento list nahradíme náhodně zvoleným. Implementace je uvedena v třídě `slisp.Operators` v metodě `mutation(Node[] nodes).`

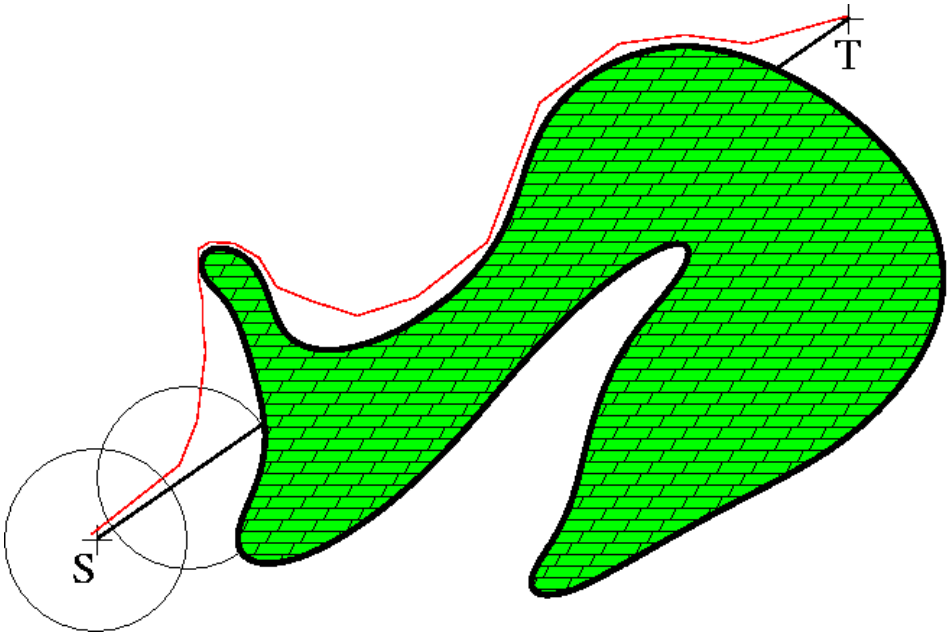
Konvenční algoritmy

Některé úlohy jsou poměrně dobře řešitelné i bez komplikovaných a komplexních nástrojů, jakými jsou neuronové sítě či genetické programování. Jednou takovou oblastí je plánování cesty.

VisBug

Při úloze, kdy neznáme okolní prostředí celé (můžeme si představit úlohu, kdy robot dostane úkol dojet na souřadnice, které nejsou přímo viditelné), je možné použít tento standardní algoritmus. Na rozdíl od ostatních Bug algoritmů, tento klade větší nároky na senzory robota – musí být schopen změřit vzdálenost překážky před ním. Tento požadavek náš robot bez výhrad splňuje.

Algoritmus pracuje tak, že vytvoří úsečku mezi bodem, kde robot stojí a cílem. Robot se poté snaží pohybovat po této úsečce (představuje nejkratší teoreticky možnou trasu z bodu A do bodu B). Pokud na cestě robot uvidí, že se blíží překážka, začne robota odklánět od úsečky a začne objíždět překážku. Při tomto objíždění radarem stále sleduje vzdálenost překážky a drží se jí. Pokud při tomto objíždění se setká opět s úsečkou ze startu do cíle, tak po ní pokračuje. Vše je patrné z obrázku:



(Obr. 6.1 – VisBug)

Pravděpodobnostní mřížka

Pravděpodobností mřížka slouží k tvorbě mapy okolí. Robot je virtuálně umístěn doprostřed rastrové mapy. Za pomoci svého otočného IR radaru získává vzdálenosti od okolních objektů. Mapu svého okolí doplňuje do rastrové mapy tak, že pro každou konkrétní naměřenou hodnotu v určitém úhlu (natočení IR radaru) zvýší v příslušné vzdálenosti od robota na mapě (od středu) pravděpodobnost výskytu překážky. V dlaždicích na rastru před tímto bodem naopak pravděpodobnost překážky sníží (před objektem, který radar zachytil, muselo být volno).

Závěr

Podařilo se nám sestrojit mechanického robota, který splňuje hardwarové požadavky, které jsme si stanovili. Robot má vlastní počítač, pásovou konstrukci a je schopen pohybu v budově a mírném terénu. Velkým úspěchem bylo uvedení počítače do provozu z baterií. Problémy nastaly při komunikaci mezi počítačem a periferiemi – ovládací přípravek K8055 nesplnil naše původní předpoklady a shledáváme jej pro ovládání servomotorů a komunikaci s dalšími zařízeními (např. digitální kompas) jako nepoužitelný. Důvodem jsou vysoké časové prodlevy při komunikaci. Tyto latence spolu s jistými aspekty operačního systému (např. non real-time prostředí) vedly k problémům, které jsme museli vyřešit.

Vyřešili jsme je, ale za cenu zkreslení signálů z kompasu a radarů. Do budoucna při stavbě jiného stroje či modernizaci tohoto počítáme s jiným řešením v podobě navržení vlastní ovládací desky založené na jednočipech. Celá architektura ovládání periferií bude distribuovaná a digitální. To nám umožní používat například i komunikační sběrnice I2C s periferiemi.

Problémy nastaly i se softwarem. Jedním z nich byla nefunkčnost zkompilované DLL knihovny, která zajišťuje fasádu k rozhraní JNI, na konkrétním počítači robota. Tento problém vyvrcholil až vytvořením alternativní implementace – socketovou komunikaci s nativním programem ovládací desku kontrolujícím. Další věcí, kterou jsme zjistili, bylo, že existuje propastný rozdíl mezi informatickou teorií a fyzickým světem. Krása matematického světa a exaktnost počítače má daleko k fyzickému světu a s tím souvisejícím chováním robota. Vlivem různých nepřesností, konverzí signálu a fyzikálních zákonů bylo třeba vytvořit mnoho různých konverzních tabulek a vztahů mezi teorií a praxí. Podařilo se nám naimplementovat ovládání pohybu a získání dat ze senzorů. Robot je v tomto ohledu připraven plnit své poslání – díky svým možnostem programování jeho činnosti je možno zrealizovat rozličné úlohy.

V oblasti umělé inteligence se nám podařilo vytvořit plně funkční modul genetického programování. Vlastníme implementaci našeho scriptovacího jazyka založeného na funkcionálním LISPu. Tento jazyk je schopen interakce s JVM a tudíž i fyzickým světem. Jsme schopni vytvářet stromy toho jazyka a aplikovat na ně evoluci

– tyto programy jsou v simulátoru (či na reálném robotovi) zkoušeny a křížením mezi sebou šlechtěny. Prakticky využít neuronových sítí se nám zatím nepodařilo, nicméně jsme při vývoji získali velké množství zkušeností a znalostí práce a teorie s nimi. Máme za to, že jsme zjistili, že neuronové sítě jsou opravdu uplatnitelné a možnosti, které nám nabízejí, jsou ohromné.

Projekt autonomního robotického vozidla Saladin považujeme za úspěšný v mnoha směrech. Prohloubili jsme si znalosti v širokém spektru vědních oborů – získali jsme mnoho znalostí z elektrotechniky, programování, umělé inteligence a biologie, ale hlavně jsme získali cenné praktické zkušenosti s nimi. Když se ohlédneme zpět, tak dnes bychom dělali mnoho věcí úplně jinak. A to je dobré znamení, jelikož učený z nebe ještě nikdy nespadl tak je toto známka toho, že jsme se něco naučili a profesně posunuli dále. Dalším kladem je, že nás tento projekt navedl na vědeckou dráhu a dal do kontaktu s vědeckými pracovníky. Na tomto projektu či projektu z toho vycházejícím hodláme pokračovat ve vědecké práci – výzkum a vývoj umělé inteligence v kybernetice a robotice.

Slovník pojmů

- AI – artificial inteligence (umělá inteligence)
- I2C - Inter-Integrated Circuit, sériová sběrnice
- PWM – pulse width modulation, pulzně-šířková modulace
- JNI – Java Native Interface, rozhraní pro volání nativního kódu
- RMI - Remote Method Invacation, framework pro volání metod vzdálených objektů
- Jednočip – jednočipový počítač, integrovaný obvod, který obsahuje kompletní mikropočítač
- LISP – List processing, funkcionální programovací jazyk se stromovou strukturou, prefixový zápis výrazů
- SLISP – Saladin LISP, námi obohacený jazyk LISP a jeho modifikovaný interpret
- Stub – kostru třídy, jejíž metody voláme přes RMI
- Socket – prostředek mezi procesorové komunikace přes síť
- JMF – Java Media Framework, rozšiřuje práci s multimédii v Javě
- FPGA – Field programmable gate array, programovatelné hradlové pole
- Script – programový kód, který se interpretuje
- JVM – Java virval machine,
- IR – Infra red, používající infračervené světlo

Seznam použité literatury a internetových zdrojů

- HYNEK, J. *Genetické algoritmy a genetické programování.* Praha: Grada Publishing a.s., 2008, ISBN 978-80-247-2695-3
- KUBÍK, A. *Inteligentní agenty – tvorba aplikačního software na bázi multiagentových systémů*. Brno: Computer Press, a.s., 2004, ISBN 80-251-0323-4
- WROBLEWSKI, P. *Algoritmy – Datové struktury a programovací techniky*, Brno: Computer Press a.s, 2004, ISBN 80-251-0343-9
- HEATON, J. Heaton Research – Introduction to Neural Networks with Java, http://heatonresearch.com/
- STERGIOU, CH., SIGANOS, D. *Neural Networks*, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html (stažení 07. 08. 2009)
- VESELOVSKÝ, M. http://avari.cz/uir/ (stažení 05. 08. 2009)
- Kolektiv autorů, sun.java.com, JNI, JMF, RMI specifikace (stažení 12. 04. 2009)
- Kolektiv autorů, *Časopis Robot Revue 01-2009*, Praha

Seznam použitých obrázků

Seznam použitého softwaru

- NetBeans 6.8 IDE
- Delphi 7
- Visual Studio 2008
- Encog Workbench
- AutoCAD 2010 LT
- DiaCze
- Delphi 7