

**Univerzita Jana Evangelisty Purkyně**  
v  
**Ústí nad Labem**  
**Pedagogická fakulta**  
**Katedra informatiky**



Diplomová práce  
**Adaptivní funkční řízení  
aplikované pomocí neuronové sítě**

Vypracoval: Dominik Jelínek

Vedoucí práce: Ing. Petr Haberzettl

Studijní program: Učitelství pro střední školy

Studijní obor: Matematika, Výpočetní technika

Ústí nad Labem 2012

zadání

## **Prohlášení**

Prohlašuji, že jsem předloženou diplomovou práci s názvem  
*Adaptivní funkční řízení aplikované pomocí neuronové sítě*  
vypracoval samostatně s použitím úplného výčtu citací informačních pramenů  
uvedených v seznamu, který je součástí této práce.

V Ústí nad Labem dne 15. listopadu 2012

Dominik Jelínek

## **Poděkování**

Děkuji Petru Habrzettlovi za odborné vedení, časovou flexibilitu a obětavost. Své partnerce Elišce Finsterlové děkuji za doprovodné ilustrace, kterými doplnila tuto práci, a za cenné rady v oblasti jazyka, které mi byly nápomocny.

Dominik Jelínek

## Anotace

Diplomová práce se zabývá obecně robotikou. Seznamuje s částečnou historií a poohlahuje trendy a směry současného vývoje tohoto oboru. Probírá postupně stavbu mobilního robota přes výběr součástek, návrhu hardware a jeho programování. Součástí práce jsou kapitoly věnovány firmware robota a nechybí ani popis aplikací naprogramovaných v programovacím jazyce Java a přístupných na přiloženém paměťovém médiu.

Dále se práce dotýká otázky neuronových sítí, snaží se krátce a výstižně shrnout nejdůležitější aspekty tohoto oboru a naznačit aplikaci společně s postaveným robotem.

## **Abstract**

This thesis is concerned with in general robotics. Introduces a partial history and reveals trends and current events in the field. Discusses gradually building a mobile robot through selection of components, hardware design, programming in Java and C. Part of this work are chapters devoted to the robot firmware and computer software. There is a description of the applications programmed in Java and C and all programs are accessible on CD with this thesis.

The work addresses the question of neural networks try to briefly and concisely summarise the most important aspects of this field and outline application with built robot.

## **Klíčová slova**

Robotika, umělá inteligence, hardware, mikroprocesor, jazyk C, PIC16F877A, Java, C#, Bluetooth OEMSPA310, SONAR SRF05, neuronové sítě, perceptron, ADALINE, aktivační funkce, back-propagation, McCulloch-Pittsův neuron, EAGLE, elektrická schémata, zdrojové kódy, generování bludiště.

## **Key words**

Robotics, artificial intelligence, hardware, microchip, C language , PIC16F877A, Java, C#, Bluetooth OEMSPA310, SONAR SRF05, neural networks, perceptron, ADALINE, activation function, back-propagation, McCulloch-Pitt neuron, EAGLE, electric schematics, source codes, maze generator.

---

# OBSAH

<b>Úvod</b>	<b>10</b>
<b>1 Roboti a robotika</b>	<b>12</b>
1.1 Vymezení pojmu robot . . . . .	13
1.2 Dějiny a současnost robotiky . . . . .	16
<b>2 Stavba robota</b>	<b>20</b>
2.1 Hardware . . . . .	21
2.1.1 Podvozek . . . . .	22
2.1.2 Pohon . . . . .	23
2.1.3 Senzory . . . . .	24
2.1.4 Bluetooth modul . . . . .	25
2.1.5 Napájení . . . . .	27
2.2 Řídící elektronika . . . . .	28
2.2.1 Napájecí obvod . . . . .	29
2.2.2 Procesor . . . . .	29

2.2.3	Senzory . . . . .	32
2.2.4	Deska pro řízení motorů . . . . .	32
2.2.5	Komunikace procesoru s bluetooth . . . . .	33
<b>3</b>	<b>Řídící aplikace</b>	<b>35</b>
3.1	Firmware . . . . .	36
3.1.1	Diagram řízení a popis algoritmu . . . . .	39
3.1.2	Makra . . . . .	41
3.1.3	Modul pro měření napětí . . . . .	41
3.1.4	Komunikační modul . . . . .	42
3.1.5	Modul pro ovládání motorů . . . . .	44
3.1.6	Ovládání ultrazvukového dálkoměru . . . . .	47
3.1.7	Rutiny metody main . . . . .	49
3.1.8	Prerušení . . . . .	51
3.2	Řídící software pro PC . . . . .	52
3.2.1	Komunikace s RS232 . . . . .	54
3.2.2	Knihovna RobotComm . . . . .	56
3.2.3	Ukázková aplikace . . . . .	59
<b>4</b>	<b>Umělá inteligence</b>	<b>60</b>
4.1	Vymezení pojmu umělá inteligence . . . . .	61
4.2	Historie umělé inteligence . . . . .	66
4.3	Neuron . . . . .	69
4.3.1	Centrální nervová soustava . . . . .	69
4.3.2	Biologický neuron . . . . .	70

4.4 Matematický model neuronu . . . . .	71
4.4.1 McCulloch-Pittsův model neuronu . . . . .	71
4.4.2 Perceptron . . . . .	72
4.4.3 ADALINE . . . . .	74
4.5 Neuronová síť . . . . .	75
4.5.1 Organizační dynamika . . . . .	76
4.5.2 Aktivní dynamika . . . . .	76
4.5.3 Adaptivní dynamika . . . . .	78
4.6 Metody učení neuronových sítí . . . . .	79
4.6.1 Zatřesení . . . . .	79
4.6.2 Simulované žíhání . . . . .	79
4.6.3 Metoda genetických algoritmů . . . . .	80
4.6.4 Hebbovo pravidlo . . . . .	81
4.6.5 Back-propagation . . . . .	84
4.7 Aplikace metod . . . . .	86
4.7.1 Bludiště . . . . .	86
4.7.2 MazeSolverApi . . . . .	88
4.7.3 Základní algoritmy řešící útěk z bludiště . . . . .	89
4.7.4 Matematika . . . . .	90
<b>Závěr</b>	<b>94</b>
<b>Makra</b>	<b>96</b>
<b>Příloha CD</b>	<b>98</b>
<b>Seznam obrázků</b>	<b>99</b>

<b>Seznam tabulek</b>	<b>102</b>
-----------------------	------------

<b>Použitá literatura a jiné zdroje</b>	<b>103</b>
---	------------

---

# ÚVOD

Robotika je v současném světě na předních místech žebříčků zájmů mnoha lidí. V dávné i blízké historii lidé snili o něčem, co bychom my dnes nazvali robotem, samotný pojem však starý není. První kapitola mé práce je věnována právě historii robotů, pojmu robot a robotice samotné. Historii robotiky se venuji jen okrajově, uvádím časovou osu, jež poslouží jako ucelený náhled na historické pozadí. K hlubšímu poznání doporučuji [21] nebo [1]. V první kapitole také zmiňuji informace ze současnosti robotiky, z prostředí vojenských robotů nebo expertních systémů <sup>1</sup>.

Stavba robota, který je ústředním tématem této práce, je rozebrána ve druhé kapitole. Postupně je popisován hardware a jednotlivé komponenty, ze kterých je robot sestaven. Kapitola následně navazuje na návrh elektronických obvodů. Metodikou návrhů plošných spojů jsem se konkrétně nezabýval.

Název kapitoly „Řídící aplikace“ je souhrnem kódů jazyku C a Java. Tyto aplikace jsem naprogramoval k ovládání robota z počítače pomocí sériového portu.

Robot je naprogramován v jazyku C a to v prostředí MPLAB IDE v8.88. K dispozici je již nová verze a zdrojové kódy jsou po nesložité úpravě funkční.

S výběrem vyššího programovacího jazyku jsem se dlouho rozhodoval. C# je mým oblíbeným jazykem, ale bohužel není tak platformově nezávislý, jak bych si

---

<sup>1</sup>Kapitola se zmiňuje o nedávném vítězství počítače ve Jeopardy.

přál a představoval. Java je na tom o poznání lépe, proto jsem ji nakonec vybral. Závěrem druhé kapitoly představuji program pro kontrolu ovládání robota, který ovládá periférie robota <sup>2</sup>.

Moji nejoblíbenější kapitolou je část věnovaná umělé inteligenci. Podstatou nebylo dohnat a doplnit výborná skripta a publikace na toto téma jako jsou například [20], [21] nebo [22]. Nerozvádím do detailů matematický aparát neuronových sítí, ale snažím se vše jasně vysvětlit. Vycházel jsem z vlastních zkušeností s učením se neuronovým sítím. Za nejsložitější z metod učení umělých neuronových sítí považuji metodu Back-propagation[4.6.5], ale domnívám se, že se mi i tuto metodu podařilo vyložit snazším způsobem.

V závěrečné, čtvrté kapitole seznamuji čtenáře s možnostmi použití probraných metod a maticovým zápisem váhového prostoru sítě.

Přál bych si, aby se moje práce neztratila v knihovních regálech, ale aby posloužila jako motivace pro ty, kteří kouzla adaptivních algoritmů teprve objevují. Ke své práci přikládám CD nosič, který obsahuje veškeré volné zdroje, jež mi byly při práci užitečné, navržená elektronická schémata a programy v jazyce Java i C. Na CD jsem také umístil velmi inspirativní audio seriál „Roboti a robotika“[1].

---

<sup>2</sup>SS motory, sonar, voltmetr,...

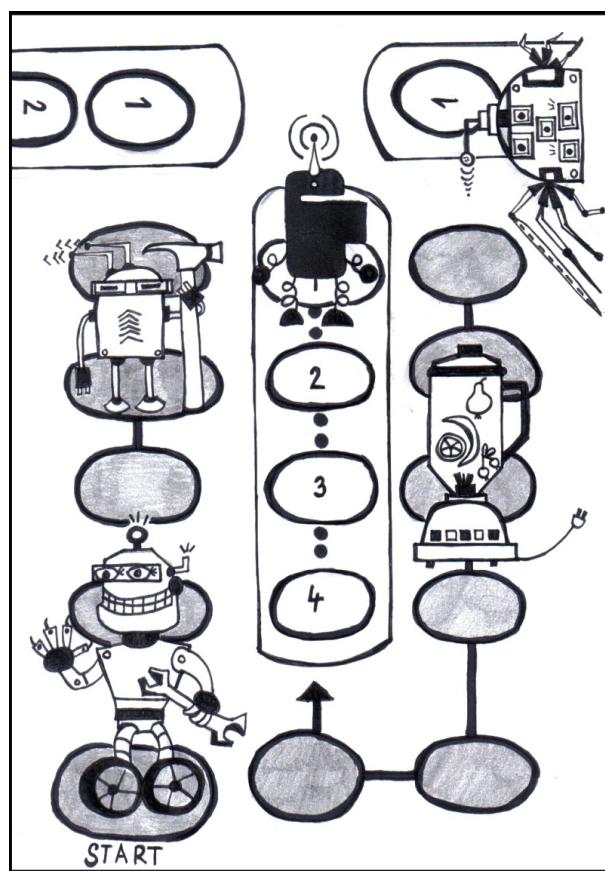
---

---

# KAPITOLA 1

---

## ROBOTI A ROBOTIKA



## 1.1 Vymezení pojmu robot

Dnes velmi rozšířený termín **robot** představuje stroj, který dokáže libovolným způsobem interagovat s prostředím. Slovo robot bylo poprvé v dějinách použito v roce 1920 ve hře R.U.R - Rossum's Universal Robots Karla Čapka. Jak uvádí zdroj [41], tak slovo robot přesto vymyslel bratr Josef Čapek.

Odborná literatura uvádí následující definice pojmu robot.

*Robotem rozumíme počítačem řízený integrovaný systém, schopný autonomní a cílově orientované interakce s reálným prostředím v souladu s instrukcemi člověka.*

Havel I. M. [12]

*Robots are physical agent that perform tasks by manipulating the physical world.*

Russell S. a Norvig P. [30]

*Roboti jsou fyzické agenti, kteří plní úkoly manipulací fyzickým světem.*

Russell S. a Norvig P. [30]

Robot je tedy nějaké fyzické zařízení, na kterém běží algoritmus řídící robota. Tento algoritmus je napsaný v programovacím jazyce, kterému hardware<sup>1</sup> robota rozumí. A takovému algoritmu se říká agent.

*An agent is just something that acts (agent comes from the Latin agere, to do).*

*Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals.*

Russell S. a Norvig P. [30]

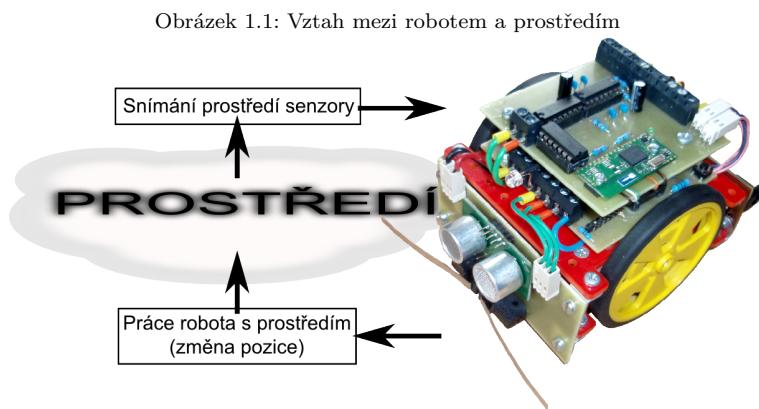
*Agent je jen něco co působí (agent pochází z latinského agere, dělat).*

*Samozřejmě, všechny počítačové programy něco dělají, ale od počítačových agentů se očekává víc: operovat autonomně, vnímat své prostředí, vytrvat delší dobu, adaptovat se změnám a vytvářet a dosahovat cíle.*

Russell S. a Norvig P. [30]

---

<sup>1</sup>Hlavním hardwarem je procesor, ten pak ovládá periferie, neboli okrajové části elektroniky robota.



V souvislosti agenta s počítačovým programem je vodné také přenést algoritmy <sup>2</sup> na silnější výpočetní systém a to počítač. Robot jako stroj bereme v této práci jako výkonově slabý výpočetní systém, který pouze ovládá aktuátory robota. Aktuátorem je například pohybový subsystém robota, může jím být také umělá paže nebo závora, kterou vpouští auta na parkoviště <sup>3</sup>. Aktuátor je tedy něco, čím agent interahuje s prostředím, ve kterém se nachází, nebo jej mění.

Pro práci robota <sup>4</sup> jsou důležité také vjemy z prostředí. Robot poznává své prostředí pomocí senzorů, kterými je vybaven. Senzorem může být kamera <sup>5</sup>, ultrazvukový, infračervený nebo laserový dálkoměr, různé druhy taktilních <sup>6</sup> čidel, teplotní čidlo, barometr, výškoměr, GPS navigace, kompas nebo elektrody pro měření EEG <sup>7</sup> signálu. Vztah robota a prostředí charakterizuje obrázek [1.1].

Roboty dělíme například podle aktuátorů, autonomie a podle účelu. Rozlišujeme tak roboty stacionární, nebo mobilní kolové a kráčející, řízený, ovládaný nebo inteligentní a nakonec můžeme odlišit robota kuchyňského, lékařského nebo

<sup>2</sup>Můžeme také napsat agenta nebo mysl robota.

<sup>3</sup>Tento pohled naznačuje agenta jako program, který detekuje obsazené parkovací pozice a na základě těchto vstupů vytváří svá rozhodnutí.

<sup>4</sup>Slovem robot tedy budeme chápát agenta a hardware dohromady jako dva neoddělitelné celky.

<sup>5</sup>Teorie zabývající se zpracováním signálů kamery se jmenej „Počítačové vidění“ a jedná se o podobor umělé inteligence.

<sup>6</sup>taktilní = dotykový, hmatový

<sup>7</sup>Elektroencefalograf je přístroj skládající se ze snímajících elektrod a procesoru. Pracuje na principu snímání elektrické aktivity mozku pomocí elektrod připevněných na povrchu hlavy. Elektroencefalograf získané signály zesílí a pak je vypíše na papír nebo zobrazí na obrazovce. Vzniklé EEG-křivky mají charakteristický průběh a typickou frekvenci vln. Jinou křivku zobrazí přístroj ve spánku, odlišnou při denní aktivitě [11].

průzkumného. Tento výčet je jen zlomkem z mnoha tvarů a účelů, kterých roboti nabývají.

Robotika, věda zabývající se designem, výrobou a aplikací robotů, je souhrnem mnoha oborů lidské činnosti. Zahrnuje například elektroniku a návrh elektronických obvodů, mechaniku, která se soustředí na mechanické vlastnosti konstrukce robotů a tvorbu softwaru, programů a algoritmických postupů.

*Robotics is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices. Examples of successful robotic systems include mobile platforms for planetary exploration [], robotics arms in assembly lines [], cars that travel autonomously on highways [], actuated arms that assist surgeons [].*

*Thrun S. [37]*

*Robotika je věda poznávání a manipulací fyzického světa počítačem kontrolovaným mechanickým zařízením. Příklady úspěšných robotických systémů zahrnují mobilní platformy pro planetární průzkum, robotické paže ve výrobních linkách, auta, která autonomně cestují na dálnici, paže asistující chirurgům.*

*Thrun S. [37]*

Podle (McKerrow, 1986)<sup>8</sup> je robotika disciplína zahrnující:

- návrh, výrobu, řízení a programování robotů,
- použití robotů pro řešení úloh,
- zkoumání řídicích procesů, senzorů, akčních členů a algoritmů u lidí, zvířat a strojů,
- použití výše uvedeného pro návrh a použití robotů.

---

<sup>8</sup>Použitý text je reprodukcií textu z prezentace přednášek Václav Hlaváče z katedry kybernetiky FEL ČVUT <http://cmp.felk.cvut.cz/~hlavac/>. Phillip McKerrow působil jako profesor ve škole informačních technologií a počítačových věd na univerzitě Wollongong v Austrálii [works.bepress.com/pmc kerrow](http://works.bepress.com/pmc kerrow).

## 1.2 Dějiny a současnost robotiky

**H**istorie robotiky začíná už v období starověkého Řecka, kdy už Homér ve svém eposu Ilias píše o ženě vyrobené z čistého zlata, která pomáhala Hefaistovi, bohu ohně, v jeho kovářské dílně.

Mezi první funkční roboty patří mechanický pták od Archytáse z Tarentu. Tohoto ptáka poháněla voda nebo pára a mohl létat z jedné větve stromu na druhou.

Dalším příkladem je robot navržený kolem roku 1206 muslimským učencem jménem Al-Jazari. Postavil loď, na které byli umístěni mechaniciť muzikanti pro pobavení královských hostů.

V roce 1738 sestrojil Francouz Jacques de Vaucanson svoji proslulou kachnu, která napodobovala chování živé kachny. Jedla, vybírala zrna a napodobovala trávicí systém<sup>9</sup>[1].

Dlouhou dobu šlo spíše o hračky než o seriózní zařízení a o prostředky pro ulehčení práce. Nejstarším příkladem které ulehčovalo lidem práci a zvyšovalo produktivitu byl tkalcovský stav. Z oblasti zemědělství můžeme vybrat pluh, mlátičku nebo trakař. K průmyslové revoluci patří parní stroj. Skutečný raketový pokrok v tomto odvětví nastal až elektrifikací a rozvojem výpočetní techniky. Bylo nutné vyřešit celý komplex dílčích částí jako je výroba elektřiny, její distribuce a využívání. A za zmínku jistě stojí technologie jako benzínový a vznětový motor, železnice nebo automobil. Dílčí úspěchy a práce tvůrčích myslí dali postupem času vzniknout prvním skutečným robotům tak, jak si je představoval už Homér, Leonardo da Vinci nebo Karel Čapek.

Od počátku šedesátých let se do prvních robotických experimentů spustila špičková pracoviště jako MIT, Standford Research Institute (SRI), University of Edinburgh, výzkumné laboratoře firmy Hitachi a další. Startovní projekty se zabývali řešením jednoduchých manipulačních a navigačních úloh [22].

Dnešním trendem vývoje robotiky je kooperace, miniaturizace<sup>10</sup> a větší autonomie robotů. Zajímavé jsou projekty agentury amerického ministerstva obrany DARPA (Defense Advanced Research Projects Agency). Humanoidní robot na-

<sup>9</sup>Odpadním produktem pak byly perly.

<sup>10</sup>Rozměry robotů se dělí na mikrorobot menší než 1mm, milirobot menší než 1cm, minirobot menší než 10cm a malý robot menší než 100cm [16].

Obrázek 1.2: Recon Scout Throwbot (robot Špulka)[19]



podobující lidskou chůzi do schodů Petman[3] nebo pneumatický „želé“ robot pohybující se změnou tvaru částí svého těla[13]. Společnost také vyhlašuje jednu z nejznámějších robotických soutěží DARPA Grand Challenge<sup>11</sup> a prvním autobotem<sup>12</sup>, který tento závod vyhrál se jmenuje Stanley a je produktem Stanford University<sup>13</sup>. Uvedený závod vyhrál v roce 2005 a trasu měřící 132 mil projel za 6 hodin a 53 minut. Další soutěž, kterou společnost DARPA vyhlásila, požaduje zkonstruovat humanoidního dvojnohého robota, schopného pohybovat se v lidském prostředí a používat lidské nástroje a mechanismy.

Nerad bych zapomněl na známého robota BigDog společnosti Boston Dynamics. BigDog je první variantou přepravní muly, jež dokázala unést až 110 kg. Velikostně se blížila menšímu oslu nebo hodně velkému psu. Později unesla až 150 kg na vzdálenost 20 km a zvládala stoupání až 35 stupňů. Díky řídícímu počítači bylo nemožné jej poválit na zem[2]. Další verze a vylepšení tohoto robota je možné najít na stránkách společnosti Boston Dynamics.

Dalším vojenským robotem je SandFlea v roli průzkumného robota se čtyřmi koly překonávajícího výšku až 9 metrů[4], špiónažní robot „Recon Scout Throwbot“[16][1.2] a odminovací robot Božena[15].

A mezi poslední zmíněné roboty ve vojenství patří inteligentní miny[1], které se snaží pomocí evolučních algoritmů „poskakováním“ pokrýt co největší plochu nebo zaútočit na projíždějícího nepřítele vysláním jednoho ze skupiny.

<sup>11</sup>Soutěž spočívá autonomně přejet stovky mil z Los Angeles do Las Vegas. Po cestě se musí každý robot vyrovnat s překážkami jako poušt, kopce, kamenité cesty nebo podjezd.

<sup>12</sup>robot auto

<sup>13</sup>The Google Driverless Car je projekt společnosti Google, jehož cílem je vyvinout autonomní auto pohybující se v běžném provozu. Projekt je veden inženýrem Sebastianem Thrunem, vedoucím Stanford Artificial Intelligence Laboratory. Projekt Stanley byl předchůdce Google Cars.

### Časová osa[18]

*400 př. n. l. - Holubice Archytáše z Tarentu*

Za prvního konstruktéra mechanické hračky „robota“ je považován Archytás z Tarentu.

*1206 - Robotičtí hudebníci*

Al-Jazariho Pro pobavení královských hostů navrhl Al-Jazari čtyři mechanické hudebníky, kteří seděli na lodi plující na jezeře a hráli na hudební nástroje.

*1495 - Mechanický rytíř Leonarda da Vinciho*

Mark Rosheim vytvořil pro televizi BBC pohyblivého rytíře podle návrhů univerzálního génia Leonarda da Vinciho.

*1738 - Mechanická kachna*

Mistrovským dílem Jacquesa de Vaucanova je měděná kachna, která byla schopna třepat křídly, štěbetat, pít, jíst a také napodobovat trávení.

*1921 - Karel Čapek použil slovo „robot“*

V divadelní hře „R.U.R.“ (zkratka z Rossumovi univerzální roboti) použil Karel Čapek poprvé slovo robot. Poradil mu ho však nejspíše jeho bratr Josef.

*1927 - Televox*

Prvním „užitečným“ robotem s lidskou podobou byl „Televox“. Dokázal kontrolovat hladinu vody, aktivovat čerpadla, otevírat a zavírat okna a také ovládat světla.

*1948 - Elmer & Elsie*

V roce 1948 sestrojil William Grey Walter robotické želvy (o něco větší než bota). Měly dva elektronkové neurony, které řídily jejich rychlosť směr pohybu. Navíc měly i fotočidlo a reagovaly na podněty z okolí.

*1956 - Průmyslový robot*

George Devol získal patent na jednorukého pracovního robota, který byl pojmenován „Unimate“. Jeho hlavním účelem bylo přenášení objektů.

*1970 - Shakey*

Na Stanford Research Institute (S.R.I.) byl sestrojen mobilní robot Shakey. Pomocí několika senzorů mohl sledovat svoje okolí a podle toho plánovat akce.

*1973 - Kuka Famulus*

Německá společnost Kuka představila průmyslového robota se šesti elektromechanickými řízenými osami. Mohl nahradit člověka na místech s těžkým přístupem.

*1996 - Sojourner*

Robot (vozítko), který přistál na Marsu pomocí sondy Pathfinder a pohyboval se v jejím okolí rychlostí 1 cm/s. Za úkol měl především fotografovat a odebírat vzorky půdy a analyzovat je.

#### *1999 - Sony Aibo*

Malý robotický pes byl vyvinut firmou Sony jako hračka, ale používal se i pro výzkumy v oblasti umělé inteligence.

#### *2000 - Honda Asimo*

Pomocí dvou nohou se mohl pohybovat jako člověk, dokázal používat schody, otáčet se a rozpoznávat předměty ve svém okolí. Dokázal rozeznat lidské jednání a řeč a reagovat na ně.

#### *2003 - Lady robot*

„Repliee Q1“ je první robot (ženského pohlaví), který vypadá jako člověk, jako člověk mluví, pohybuje víčky a reaguje na lidský dotek. Místo umělé hmoty tvoří jeho kůži poddajný silikon.

#### *2007 - Ping-pong*

Díky své umělé inteligenci je robot „Topio“ schopen vylepšovat svoje umění v hraní stolního tenisu.

#### *2010 - NASA Robonaut 2*

Své síly při stavbě „humanoidního robota pro automobilový a vesmírný průmysl“ spojily společnosti GM a NASA. Robot měl vykonávat obtížné a nebezpečné úkoly místo kosmonautů.

14. února 2011 superpočítáč Watson společnosti IBM, vyhrál americkou soutěž Jeopardy proti dvěma lidským soupeřům. Jeopardy je hra jejíž alternativa s názvem Riskuj se vysílala i v české televizi. Watson nebyl v průběhu hry připojen k internetu. Jeho databáze obsahovala však data z celé Wikipedie a jiných internetových encyklopedií. Ke slovu u lidského moderátora se hlásil fyzickým stiskem tlačítka tak, jako lidští soupeři. Watson musel komunikovat pouze hlasem a pouze lidský hlas mohl poslouchat. Ten musel dekódovat, vyhledat ve své databázi odpověď na položenou otázku a odpovědět. Mimoto využíval nejrůznější optimalizační strategie a občasným vtipným výsledkem této optimalizačních algoritmů byla sázka 9987 amerických dolarů. Jeho další využití se nyní aplikuje v lékařství, kde pomáhá lékařům diagnostikovat onemocnění pacientů.

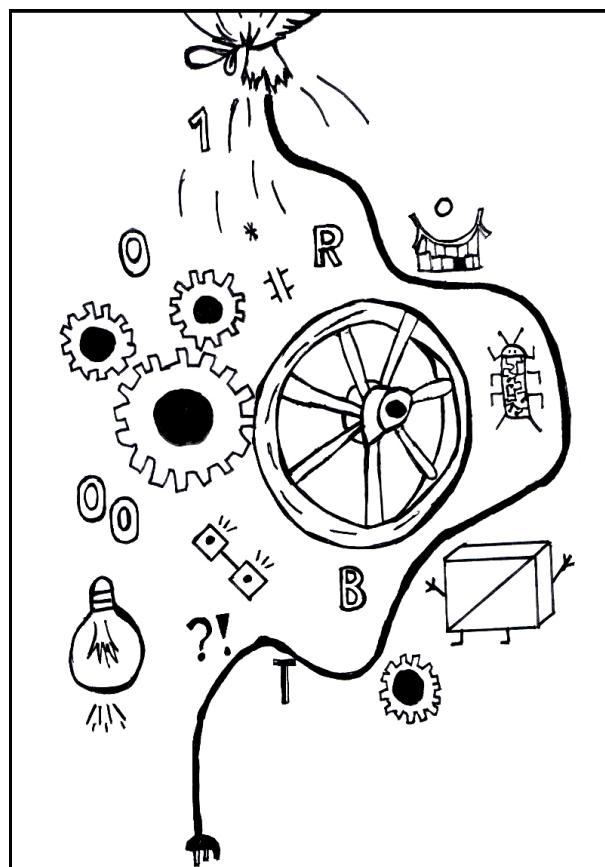
---

---

## KAPITOLA 2

---

### STAVBA ROBOTA



## 2.1 Hardware

Mezi hardware robota patří všechny jeho fyzické součástky, které jsou potřeba pro stavbu. Tato podkapitola se věnuje fyzické stavbě robota, použitým součástem a elektronice.

Výsledkem práce je kolový robot navržený pro jízdu v bludišti. Je vybaven taktilními senzory pro určení překážky nulové vzdálenosti a ultrazvukovým dálkoměrem, který je schopen měřit vzdálenost od jednoho centimetru do čtyř metrů. Na čelní straně robota je umístěna kontrolní LED dioda a komunikace s počítáčem<sup>1</sup> probíhá prostřednictvím bluetooth modulu. Jako napájení byly zvoleny dvě devíti voltové baterie s kapacitou 280 mAh. Nabíjení baterií je řešeno externí nabíječkou, která je k bateriím připojena pomocí kablíku se souosým konektorem. Stejným konektorem je opatřen i robot. Aby nedošlo při nabíjení ke zničení elektroniky, je robot opatřen kolébkovým třístavovým spínačem, který propojí baterie buď s elektronikou robota, nebo s nabíječkou.

Koncepce výsledného vzhledu a použitých senzorů robota nebyla od začátku optimální. Hlavně použité baterie nemají příliš velkou kapacitu a vyplatilo by se použít například modelářský akupack což je balíček několika baterií<sup>2</sup> s vysokou kapacitou, nebo napětím. Jistě by se vyplatilo použít dvě napájecí soustavy. Jedna by napájela pouze elektroniku a druhá by byla určená pro pohon robota.

Také představa bludiště a snímání prostředí se postupně formovala. Levnějším řešením by jistě byla stavba robota sledujícího černou pásku. Tento přístup je velmi přenositelný, praktický a úsporný.

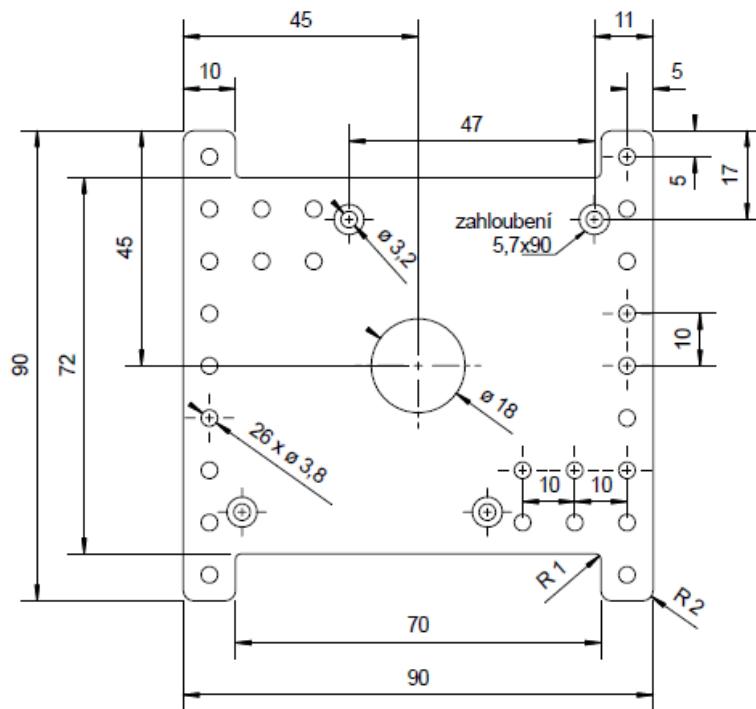
Internetové portály věnující se robotům<sup>3</sup> a jejich stavbě obsahují mnoho nápadů, ze kterých je možné se inspirovat. Bohužel na mnoho inspirativních článku jsem narazil až v průběhu stavby.

<sup>1</sup>Komunikace je možná také například s mobilním telefonem.

<sup>2</sup>V internetových obchodech jsou v nabídce balíčky mikrotužkových, tužkových ale i knoflíkových baterií.

<sup>3</sup> Mezi takové patří například [www.robodoupe.cz](http://www.robodoupe.cz), [www.letsmakerobots.com](http://www.letsmakerobots.com), [www.robotrevue.cz](http://www.robotrevue.cz), [www.robotika.cz](http://www.robotika.cz) nebo [www.hobbyrobot.cz](http://www.hobbyrobot.cz).

Obrázek 2.1: UMU-01 [33]



### 2.1.1 Podvozek

Podvozek robota tvoří díl koupený u společnosti Snail Instrument <sup>4</sup> nazvaný UMU-01. Jednoduché červené díly [2.1] o tloušťce 3 milimetry navržené pro použití společně se stavebnicí Merkur jsou spojeny distančními šroubkami o délce 22 milimetrů. Šroubky nám tak vytvářejí prostor, ve kterém jsou umístěny stejnosměrné motory. Uprostřed spojených dílů se nachází otvor o průměru 18 milimetrů, který je vhodný použít pro propojení řídící elektroniky, motorků a baterie.

Dodaná kola <sup>5</sup> o průměru 68 milimetrů mají gumové lemování tvořící pneumatiku. Kolečka jsou k motorům přišroubována samořezným šroubkem, takže je možné je kdykoliv odmontovat. K podvozku se dodává také všeobecné ocasní kolečko, které nám vytváří spolu s koly trojúhelníkový charakter. Díky tomu a vhodnému umístění těžiště získáme stabilní konstrukci robota.

<sup>4</sup>Internetová stránka prodejce: [shop.snailinstruments.com](http://shop.snailinstruments.com).

<sup>5</sup>Toto plastové kolečko lze najít na internetu pod označením SW65.

Protože je podvozek plastový<sup>6</sup>, nebyly překážkou ani drobné mechanické úpravy. Díky kompatibilitě se stavebnicí Merkur bylo snadné použití různých propojovacích částí, šroubků, matiček a ramen. Dokonce nebyl problém ani se stavebnicemi jiných výrobců<sup>7</sup>. Pro tyto vlastnosti může být podvozek použit nejen ke stavbě robota, ale lze jej také použít jako pohybový aparát pro jinou konstrukci. Místo kolečka připojeného k motoru se může například použít ozubené kolečko, které bude pohánět další části větší konstrukce.

Velmi příjemná vlastnost se ukázala při řešení napájení. Baterie jsou uchycené ve spodní části a tuto část není snadné demontovat. Napájení tvoří váhově nejtěžší část robota a pokud bychom postupovali bez rozumu, těžiště by se nacházelo příliš vpředu a robot by se v lepším případě mírně naklonil. Naštěstí jsou v takovém případě díly středově souměrné, a tak se spodní část podvozku pouze otočí o 180°.

### 2.1.2 Pohon

Podvozek je vybaven dvěma stejnosměrnými motory<sup>8</sup>. Maximální napájení jednoho motoru je šest voltů a jeho odběr bez zátěže činí 58 miliampér. Při maximální zátěži motor odebírá 670 miliampér. Motor obsahuje převodovku s převodem 143:1 a při napájení pěti volty se kolo otočí 70 krát za minutu.

Tyto údaje uvádí prodejce na svých internetových stránkách<sup>4</sup>. K motorkům bohužel nelze najít dokumentaci. Toto se ukázalo být překážkou ve chvíli, kdy bylo potřeba zjistit počet otáček v závislosti na napětí. Ovšem podle zdroje [14] je tato závislost lineární, takže lze tento údaj snadno vypočítat.

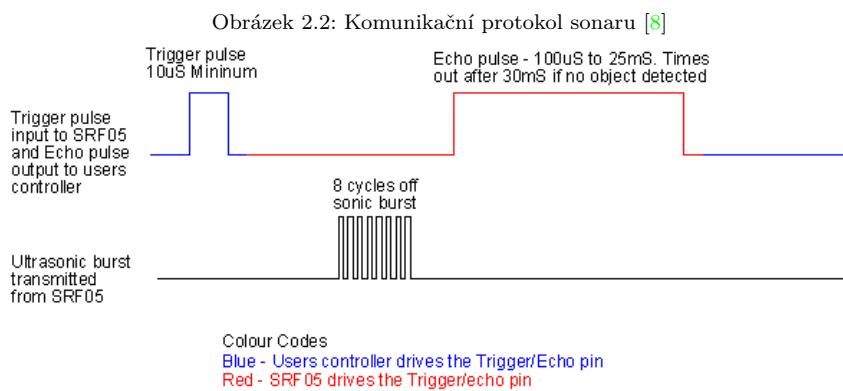
Motory jsou vybaveny odrušovacími kondenzátory, které zajistí, že při rychlé změně potenciálu na vývodech nedojde ke zničení řídící elektroniky<sup>9</sup> vlivem elektromotorického napětí. Dále jsou kondenzátory připájeny ke kostře elektromotoru a zabraňují tak elektromagnetickému rušení. K řídící elektronice se motory připojují pomocí vodičů s konektory, které jsou součástí podvozku.

<sup>6</sup>Výrobce uvádí jako materiál plexisklo.

<sup>7</sup>U nás běžně dostupné stavebnice výrobců Meccano, Eitech a Combined toys.

<sup>8</sup>Motory jsou značeny jako GM8PW.

<sup>9</sup>Podle [39] na straně 75.



### 2.1.3 Senzory

Robot je vybaven dvěma mikrospínači s dlouhými páčkami, které mají funkci analogickou s hmatovými vousky u živých organismů. Jsou určeny pro detekci nulové vzdálenosti robota od překážky.

Dalším senzorem je sonar. Výrobce Devantech Ltd <sup>10</sup> jej prodává pod označením SRF05. Tento ultrazvukový dálkoměr měří vzdálenost v rozsahu od jednoho centimetru do čtyř metrů. Jeho vyzařovací úhel je 55°, napájecí napětí pět voltů a proudový odběr modulu 30 miliampér.

Modul ovládáme podle schématu [2.2] z dokumentace [8] tak, že nejdříve vysleme na určený pin takzvaný Trigger puls. Ten musí trvat alespoň 10 mikrosekund, poté modul začne měřit vzdálenost vysláním zvukového signálu v osmi cyklech, to však nastane až po ukončení Trigger pulsu. Sonar nám po zpoždění 700 mikrosekund vrátí Echo puls o rozsahu 100 mikrosekund až 25 milisekund podle vraceného zvukového signálu. Pokud Echo signál trvá 30 milisekund, překážka není detekována. Dokumentace nám také dává možnost Trigger a Echo pin propojit a multiplexovat. Tuto možnost jsme také využili v našem zapojení.

Algoritmus, podle kterého počítáme vzdálenost překážky od robota je závislý na teplotě, neboť na té je závislá i rychlosť zvuku. Pokud například sonar vrátí délku Echo pulsu 5 milisekund v místnosti o teplotě 20°C, kdy zdroj [27] uvádí rychlosť zvuku na 343 metrů za sekundu, je robot vzdálen od překážky 1,715 metrů. Musíme si však uvědomit, že zvuk se šíří od sonaru k překážce a zpět, takže

<sup>10</sup>[www.robot-electronics.co.uk](http://www.robot-electronics.co.uk)

Tabulka 2.1: Parametry bluetooth modulu

Napájecí napětí	od 3 do 6 [V]
Odebíraný proud (min)	7,9 [mA]
Odebíraný proud (max)	70 [mA]
Odebíraný proud (průměr)	17 [mA]
Přenosová rychlosť	od 300 do 921,6k
Řízení toku	CTS/RTS nebo žádné
Rozměry	16 x 36 x 2,4 [mm]
Logická úroveň	3V3

musíme výslednou vzdálenost dodatečně vydělit dvěma. Správná vzdálenost je tedy 85,75 centimetrů. Podle dokumentace stačí sonarem vrácenou hodnotu v mikrosekundách vydělit číslem 58, pokud si přejeme převést hodnotu na vzdálenost v centimetrech. Podle tohoto postupu nám dává 5 milisekund vzdálenost 86,21 centimetrů. Tato nepřesnost je daná odvozením dělitele. 58 mikrosekund je hodnota, za kterou zvuk urazí vzdálenost od robota 1 centimetr k překážce a zpět. Rychlosť zvuku se v tomto případě dá vypočítat na 344,8 metrů za sekundu.

#### 2.1.4 Bluetooth modul

Pro bezdrátovou komunikaci mezi počítačem a robotem používáme modul zakoupený u společnosti SE Spezial-Electronic AG <sup>11</sup>. Avšak místo stolního počítače můžeme použít jakékoli jiné zařízení disponující bluetooth adaptérem, například mobilní telefon.

Internetové stránky prodejce <sup>11</sup> uvádí dosah antény na 75 metrů. Další důležité parametry, které nás zajímají už můžeme vyčíst z dostupné dokumentace [7] a [6] a jsou shrnutý v tabulce [2.1].

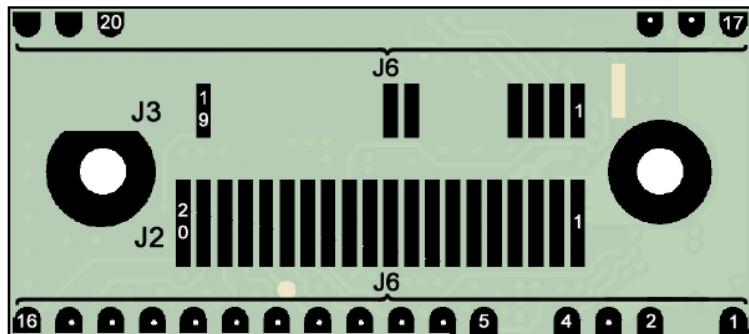
Rozložení pinů je ukázáno na obrázku [2.3]. Naše zapojení používá pouze napájecí pin (J2:3-4), GND (J2:1-2), signalizační pin <sup>12</sup> (J2:14) a komunikační piny UART-TxD (J2:16) a UART-RxD (J2:18). Protože nepoužíváme piny UART-CTS a UART-CRS, nemůžeme použít hardwarové řízení toku dat <sup>13</sup>.

<sup>11</sup>České zastoupení německé firmy je dostupné na internetových stránkách [www.spezial.cz](http://www.spezial.cz).

<sup>12</sup>Na tomto pinu je připojena LED dioda, která signalizuje probíhající komunikaci.

<sup>13</sup>Řízení toku dat představuje potvrzení příjmu dat či připravenost k přenosu a jeho zahájení na úrovni hardwarového nebo softwarového rozhraní [40].

Obrázek 2.3: Spodní pohled na bluetooth modul [6]



V základní konfiguraci probíhá komunikace rychlostí 9600 baudů<sup>14</sup>, o šířce dat 8 bitů<sup>15</sup>, bez paritního bitu<sup>16</sup>, s jedním stop bitem<sup>17</sup> a bez zmíněného řízení toku dat. Nastavení může být libovolné podle požadavků zákazníka. Pokud potřebujeme nastavení změnit osobně, použijeme program Serial Port Adapter Toolbox od společnosti connectBlue<sup>18</sup>.

*Baud je jednotka používaná pro měření rychlosti přenosu dat. Přenosová rychlosť definuje rychlosť přenosu dat z datového média na jiné datové médium. Baud rate udává počet změn signálu za sekundu. Počet změn se pak vyjadřuje v baudech. Jako základní jednotka informace v moderních počítačových systémech se bere jeden bit (nabývá hodnoty 0 nebo 1). Do jedné signálové změny lze zakódovat i více než jeden bit. A proto nelze slučovat pojmem bps (bits per second = byty za sekundu) s pojmem baud. Jednotka baud je pojmenována po Jean-Maurice-Émile Baudotovi (\*1845 - †1903).*

Vít Olmr [40]

Při prvním spuštění se nás program zeptá na produkt, ke kterému si přejeme se připojit. Naším produktem je *Bluetooth SPA*<sup>19</sup>. Bluetooth modul nabízí dvě

<sup>14</sup>Naše zapojení využívá pouze sériovou komunikaci a změna signálu nepřenáší více informací. V tomto případě můžeme považovat 1 [Bd] za 1 [bit/s].

<sup>15</sup>Každá informace odeslaná na sériový port bude obsahovat 8 cifer z množiny {0, 1} (hodnota 0 - 255).

<sup>16</sup>Bit přidaný k datovému slovu, který obsahuje informaci o počtu jedničkových bitů ve slově. Paritní bit je určen k jednoduché detekci chyby ve slově pomocí logické funkce XOR.

<sup>17</sup>Stop bit definuje ukončení rámce. Zároveň zajišťuje určitou prodlevu pro přijímač. Právě v době příjmu stop bitu většina zařízení zpracovává přijatý byte [40].

<sup>18</sup>[www.connectblue.se](http://www.connectblue.se)

<sup>19</sup>OEMSPA310

možnosti pro komunikaci. Budě můžeme posílat a přijímat data, nebo takzvané AT příkazy <sup>20</sup>. Pokud měníme nastavení bluetooth modulu, tak k tomu využíváme AT režim, do kterého přepneme modul odesláním řetězce "\\" <sup>21</sup>. Po připojení se zpřístupní ovládací prvky programu, ale i tak budou textová pole prázdná. Nastavení musíme nejdříve přečíst a to tlačítkem „Read“. Pokud do textových polí napíšeme nějaké hodnoty a stiskneme tlačítko „Write“, odešle se do modulu nové nastavení. Tímto způsobem můžeme změnit základní konfiguraci, heslo pro komunikaci nebo nastavit bluetooth modul jako server pro připojení jiných modulů <sup>22</sup>. Lze takto i vypnout bezdrátovou komunikaci. Museli bychom pak připojit modul přímo na sériový port počítače, abychom jej moli přenastavit.

### 2.1.5 Napájení

K napájení robota jsme použili dvě nabíjecí 9V NiMH baterie s kapacitou 280 mAh. Ty jsme zapojili paralelně, abychom dostali dvojnásobnou výdrž 560 mAh. Baterie jsou uchycené ve spodní části konstrukce pomocí držáku na baterie SN9V1. Protože při vytahování baterií hrozí poškození konstrukce jejich uchycení, dostal robot „vychytávku“ v podobě souosého konektoru pro připojení nabíječky. Pokud potřebujeme nabít baterie, připojíme robota k nabíječce pomocí kabelu opatřeného na jedné straně souosým konektorem a na straně druhé klipsem pro 9V baterii. Klips pak zapojíme rovnou do nabíječky <sup>23</sup>. Lepší volbou bylo použít nějaký modelářský akupack, díky kterému bychom dostali alespoň trojnásobnou výdrž při minimálním navýšení rozměrů a menší váhu.

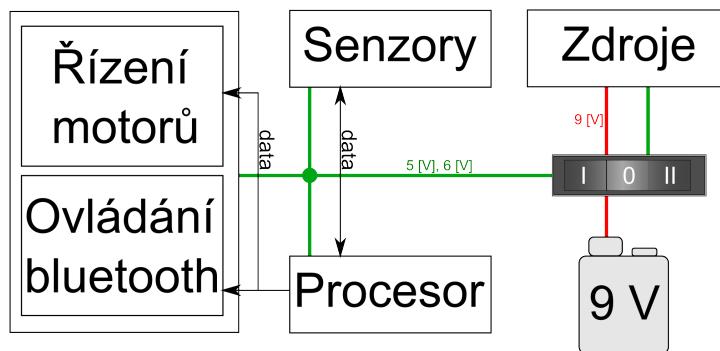
<sup>20</sup>Jedná se o soubor příkazů začínajících prefixem AT, které umožňují například nastavovat parametry komunikace pomocí mobilního zařízení či modemu.

<sup>21</sup>I tento řetězec lze v AT režimu změnit.

<sup>22</sup>Toto nastavení jsme nezkoušeli, ale prodejce uvádí, že je možné u modulu nastavit Point-To-Multipoint mód, nebo Repeater mód.

<sup>23</sup>Musíme si dát pozor na polaritu. Ta se nám v tomto místě otáčí. Napájecí kabel bereme jako jakési prodloužení konektoru baterie.

Obrázek 2.4: Blokové schéma řídící elektroniky



## 2.2 Řídící elektronika

Řídící elektronika je soustava plošných spojů, která pracuje s dostupným hardwarem. Každá elektronická destička obsahuje řadu obvodů a elektronických prvků <sup>24</sup>, které zpřístupňují použitý hardware pro elektronické zpracování. Vše ovládá jeden procesor. K dispozici má obvod pro měření vzdálenosti, zjištění stavu baterie, práci s motory nebo pro komunikaci s počítačem. Každý tento obvod i s procesorem je napájen napájecím obvodem. Diagram znázorňující popis závislosti elektronických obvodů je vyobrazen na obrázku [2.4].

Při práci s procesorem a také při jeho programování jsem hlavně čerpal z velmi praktické literatury [23]. Autor v ní vyčerpávajícím způsobem popisuje funkce, moduly a vnitřní stavbu procesoru PIC18F842, který je s námy použitým procesorem kompatibilní. V knize jsou popsány časovače a čítače, přerušení <sup>25</sup>, komunikace, práce s analog-digitálním převodníkem <sup>26</sup> a také mód PWM <sup>27</sup>, který využíváme pro řízení pohonu robota.

<sup>24</sup>Kondenzátory, rezistory, diody, ...

<sup>25</sup>Přerušení je dočasné ukončení hlavního programu a spuštění programu přerušení. V naší aplikaci se spouští speciálně definovaná funkce, ve které kontrolujeme stavy registrů (paměťových oblastí) a využíváme proč přerušení nastalo. Na základě naprogramovaných podmínek program vykoná algoritmus a po jeho ukončení se vrátí zpět do hlavního programu na opuštěnou pozici.

<sup>26</sup>Převodník je určen pro převod analogového signálu na signál digitální. Vstupem je pin procesoru, na který je přivedeno napětí v rozsahu 0 - 5 voltů. Převodník podle vstupního napětí bude generovat hodnotu v rozsahu 0 - 1023 (10 bitů), kdy 5 voltů odpovídá hodnotě 1023.

<sup>27</sup>PWM signál je diskrétně přerušovaná dodávka napětí na výstupu procesoru. Pokud bude přerušovací algoritmus nastaven na 1/2 časového impulsu, bude PWM signál dodávat 2,5 voltů ( $U_N/2$ ).

Každá část řídící elektroniky má vlastní elektronické schéma, které je k dispozici v příloze na konci práce a také vlastní návrh plošného spoje pro vyleptání obvodu na cuprexit <sup>28</sup>. Návrh plošného spoje je dostupný na přiloženém CD spolu s celým projektem vytvořeným v programu Eagle.

### 2.2.1 Napájecí obvod

Základním prvkem elektroniky robota je napájecí obvod, jehož funkce je převádět napětí dodávané bateriemi na šesti a pěti voltový okruh. K tomuto účelu jsme použili lineární regulátory napětí 7805T a 7806T <sup>29</sup>. Pěti voltový okruh je určen pro napájení obvodů a je označen červeným vodičem. Šesti voltový okruh slouží pouze pro napájení pohonu robota a je označen modrým vodičem. Speciálním případem je ještě okruh, který je připojen přímo na baterie. Ten poznáme podle žluté barvy a je určen pro zjítování aktuálního stavu napětí na bateriích pomocí analogového vstupu na procesoru. Podle dokumentace [35] spotřebovává použitý regulátor napětí dva volty a může jim procházet až 1,2 ampér <sup>30</sup>. Proto bylo potřeba použít alespoň osm voltů pro napájení robota. A maximálním vstupním napětím je 35 voltů. Obvod je zapojen podle dokumentace a pro kontrolu funkce jsme na plošný spoj umístili kontrolní led diody, které signalizují přítomné požadované napětí.

### 2.2.2 Procesor

Srdcem hardwarového návrhu je vysoce výkonný RISCový procesor od společnosti Microchip Technology Inc. s označením PIC16F877A a podpůrné obvody,

<sup>28</sup>Cuprexit je destička potažení malou vrstvou mědi. Tato destička se nastříká fotocitlivým postříkem a obvod se na ni nanese nasvícením UV lampou. Nebo se obvod nakreslí lihovým fixem. Poté se cuprexit ponoří do roztoku chloridu železitého a měď, která není součástí obvodu se vyleptá.

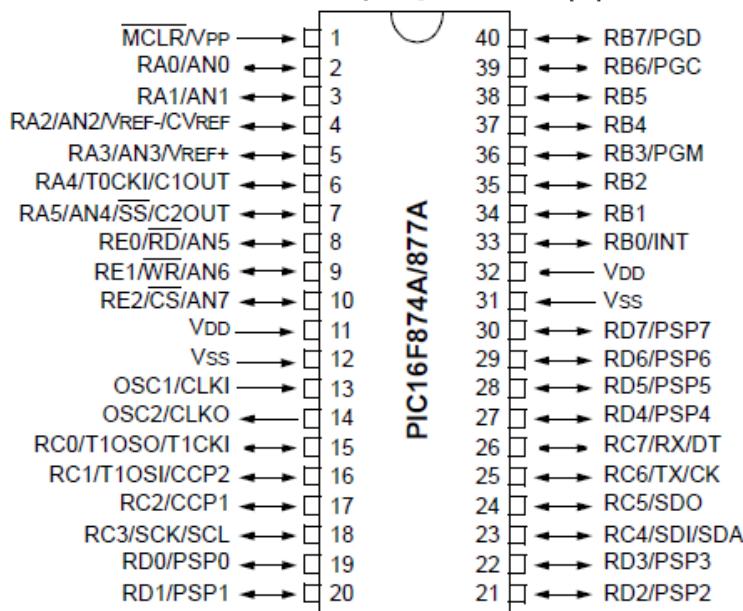
<sup>29</sup>Elektronická součástka, která slouží převodu napětí. K tomuto účelu využívá středně výkonné tranzistory.

<sup>30</sup>Tato hodnota může být limitující v případě velkého odběru obou motorů najednou. Oba motory při největší zátěži 0 otáček za minutu odebírají 1340 miliampér. Když k tomu přičteme ještě odběr dalších elektronických součástí robota (sonar, bluetooth), tak se jedná o slabý článek v realizaci robota, který by bylo vhodné nahradit.

Tabulka 2.2: PIC16F877A

Napájecí napětí $V_{DD}$	5,5 [V]
Rozsah napětí na pinech	od -0,3 do ( $V_{DD} + 0,3$ ) [V]
Maximální výstupní proud na pinech	25 [mA]
Pouzdro	PDIP 40 pinů
Programová paměť	8 [KB]
Paměť dat	368 [B]
EEPROM	256 [B]

Obrázek 2.5: Pin diagram procesoru PIC [26]



ke kterým patří krystal <sup>31</sup>, resetovací obvod, ICSP pro nahrání programu a napěťový dělič <sup>32</sup>. Zapojení jednotlivých podpůrných obvodů je vidět na schématu v příloze.

Základní vlastnosti procesoru jsou shrnuty v tabulce [2.2]. A mezi jeho přednosti, které využíváme patří Timer <sup>33</sup>, PWM mód <sup>27</sup> pro řízení pohonu, 10 bitový analog-digitální převodník <sup>26</sup>, který nám slouží pro měření stavu baterie, USART modul pro sériovou komunikaci a 15 druhů přerušení <sup>25</sup>.

Pro komunikaci s perifériemi (sonar, senzorová tlačítka, ovládání směru otáčení jednotlivých motorů a ovládání LED diody) používáme pouze PORTD <sup>34</sup>. Pin

<sup>31</sup>Určující pracovní frekvenci procesoru.

<sup>32</sup>Soustava dvou odporů, pomocí které rozdělujeme napětí na dvě hodnoty. Součtem těchto dvou hodnot dostaneme opět rozdělené napětí.

<sup>33</sup>Vnitřní modul procesoru, který vydává signál po předem nastavené době, nebo může pracovat jako čítač vnějších impulsů.

<sup>34</sup>Slovem „PORTx“ kde „x“ je písmeno v rozsahu od A do D označujeme výstupy procesoru, které

diagram je znázorněn na obrázku [2.5]. Dále máme zapojený také PORTA a to v konfiguraci analogových vstupů AN0, AN1 a AN2<sup>35</sup>. AN0 využíváme pro měření napětí na baterii přes odporový dělič. Ten jsme museli použít, neboť na vstupní pin nesmíme podle [26] přivést napětí větší než 5,3 voltů. Baterie nám dodává maximálně 9 voltů  $U_N = 9$ . Pro oba odpory děliče jsme použili stejnou hodnotu  $R = 4k7[\Omega]$ . A napětí  $U_{AN0}$  na vstupu AN0 vypočítáme podle vzorce  $U_{AN0} = 1/2 \cdot U_N$ , který je odvozen z Ohmova zákona. Obvodem prochází proud zhruba 0,9574 miliampér. Pin AN0 je nastaven jako vstup a vzhledem k vnějšímu zapojení má velmi velký odpor. „Průsakový“ proud, který tu pak naměříme může dosáhnout maximálně 19 mikroampér. Další analogové vstupy nepoužíváme. Na plošném spoji je máme vyvedeny pro možné rozšíření.

Piny TX a RX jsou určeny pro komunikaci s počítačem a jsou připojeny k bluetooth modulu.

In-circuit serial programming (ICSP) využíváme pro programování kontroléru. Na schématu je vidět implementace tohoto rozhraní. Oproti [25] jsme si implementaci trošku zjednodušili. Výhoda programování pomocí ICSP tkví v možnosti programovat procesor přímo na desce bez nutnosti vyjmutí.

Procesor může pracovat na maximální frekvenci 20 MHz. Krystal, kterým jsme osadili plošný spoj, má hodnotu 9,8304 MHz<sup>36</sup>. Krystal je zapojen podle datasheetu procesoru [26] na piny OSC1 a OSC2.

K programování procesoru používáme jazyk C a standard ANSI C. Zdrojové kódy jsou přeloženy do souboru s koncovkou hex pomocí kompilátoru HI-TECH PIC od společnosti HI-TECH Software a tento soubor se nahrává do procesoru programem UP společnosti ASIX s. r. o.

---

jsou označeny písmenem a číselnou hodnotou. Například pin D5 je obsažen v datovém registru D a jeho číslo je 5. PORTD označuje všechny porty registru D a analogicky to platí také pro PORTA, PORTB a PORTC.

<sup>35</sup>Každý výstup procesoru lze využít různým definovaným způsobem. Jednou jej můžeme použít jako digitální přepínač, nebo přepnout do stavu kdy bude synchronizovat vnitřní hodiny časovače. Pokud bychom tuto vlastnost využívali za běhu programu, říkáme této skutečnosti „multiplexování“.

<sup>36</sup>Tato hodnota krystalu nebyla v původním návrhu. Původní krystal byl při oživování desky nefunkční, a tak jsme byli donuceni vyměnit jej za krystal s hodnotou *co dům dal*.

Tabulka 2.3: 4 kanálový můstek L293D

Napájecí napětí $V_{CC1}$	od 4,5 do 7 [V]
Napájecí napětí $V_{CC2}$	od $V_{CC1}$ do 36 [V]
Špičkový výstupní proud ( $t \leq 100\mu S$ )	1,2 [A]
Kontinuální výstupní proud	600 [mA]

### 2.2.3 Senzory

Robot potřebuje pro svoje operace také vstupy z vnějšího prostředí, ve kterém se nachází. V prvotním návrhu jsme počítali s ultrazvukovým dálkoměrem, infračerveným dálkoměrem a dotykovými mikrospínacími. Platí, že čím více senzorů robot může použít, tím lépe. Ovšem z prostorového omezení konstrukce jsme se rozhodli pouze pro sonar a dvě dotyková čidla. Toto omezení není nijak zásadní. Robot je však omezen na sledování překážky pouze vpředu. Sonar detektuje vzdálenosti větší a dotyková čidla vzdálenost bezprostřední<sup>37</sup>.

Schéma plošného spoje je velmi jednoduché a obvodová deska je navržena podle radlice přiložené k podvozku<sup>38</sup>.

### 2.2.4 Deska pro řízení motorů

Pohonný substituční systém robota ovládáme pomocí 4 kanálového můstku L293D. Součástka je navržena pro dvojitý směr proudu o velikosti 600 miliampér a vysokém rozsahem napájecího napětí<sup>39</sup>. Lze ji použít pro ovládání relé, stejnosměrných motorů a dvoupólových krokových motorů. Všechny vstupy jsou kompatibilní s pěti voltovou TTL logikou. Vlastnosti můstku jsou shrnuty v tabulce [2.3].

Na obrázku [2.6] je ukázáno rozložení pinů na součástce. Pomocí pinů 1A, 2A, 3A a 4A měníme směr otáčení motoru přivedením logické jedničky, nebo nuly<sup>40</sup>. Pin  $V_{CC1}$  je napájecí pin integrovaného obvodu a  $V_{CC2}$  je pin, na který přivádíme napětí určené pro stejnosměrné motory. Rychlosť otáčení motoru ovládáme po-

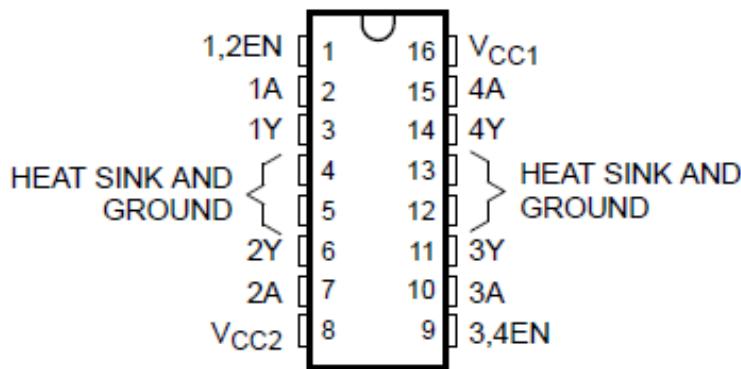
<sup>37</sup>Právě kvůli omezení sonaru.

<sup>38</sup>O radlici jsem se zatím nezmínil, protože není v realizaci práce nijak důležitá

<sup>39</sup>Takto měníme směr otáčení motoru, tj. změnou potenciálu napětí na vývodech 1Y a 2Y nebo na 3Y a 4Y.

<sup>40</sup>Vstupy 1A a 2A jsou doplňkové, takže na ně posíláme opačné signály. Pokud pošleme například na 1A logickou jedničku, musíme poslat na 2A logickou nulu.

Obrázek 2.6: Pin diagram obvodu L293D



Tabulka 2.4: Hradlo AND 74HT08

Napájecí napětí $V_{CC}$	od 2 do 6 [V]
Vstupní napětí $V_{IH}$ při $V_{CC} = 4,5$ (min)	minimum 3,15 [V]
Vstupní napětí $V_{IL}$ při $V_{CC} = 4,5$ (max)	maximum 1,35 [V]
Výstupní proud na pinech (max)	maximum 25 [mA]

mocí pulzní šířkové modulace PWM <sup>27</sup> přivedené na piny označené x,xEN <sup>41</sup>. Mezi poslední patří piny 4, 5, 12 a 13 <sup>42</sup>, které slouží pro odvod tepla a uzemnění.

## 2.2.5 Komunikace procesoru s bluetooth

Bluetooth modul komunikuje pouze 3,3 voltovou logikou. Propojení procesoru a bluetooth jsme realizovali přes hradlo AND<sup>43</sup>. Tato realizace je doporučena výrobcem a znázorněna v dokumentaci [6]. Tabulka [2.4] znázorňuje elektronickou charakteristiku použitého hradla.

Obvod 74HC08 <sup>44</sup> mezi procesorem a bluetooth modulem nám tedy slouží jako rozhraní mezi napěťovou logikou 3V3 a 5V. Signál, který do tohoto rozhraní vstupuje je přímo z procesoru a signál, který vystupuje vede na komunikační piny modulu <sup>45</sup>. Zapojení hradel a odporového děliče <sup>46</sup> je ukázáno obrázku [2.7].

Napájecí napětí jsme připojili na pěti voltový okruh a podle uvedených informací v tabulce [2.1] je tato volba zcela v pořádku.

<sup>41</sup>Symboly x tu zastupují hodnoty, které jsou vidět na pin diagramu.

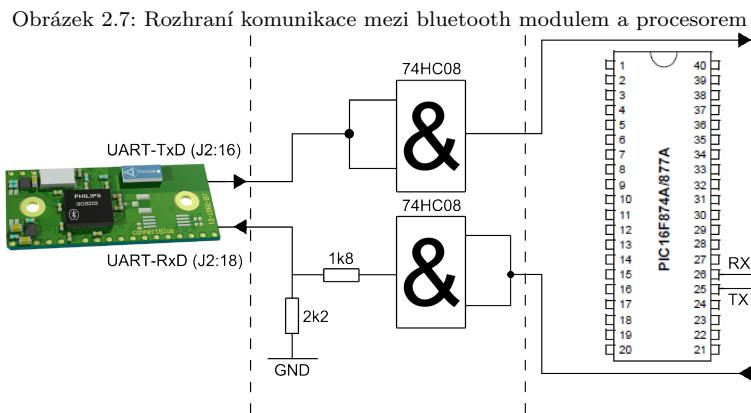
<sup>42</sup>HEAT SINK AND GROUND

<sup>43</sup>Neboli 74HC08 Jak je ukázáno na schématu

<sup>44</sup>Integrovaný obvod integruje 4x 2-vstupový AND.

<sup>45</sup>UART-TxD (J2:16) a UART-RxD (J2:18)

<sup>46</sup>Výstupem hradla je opět pět voltů a je nutné toto napětí snížit.



Na 14. pin modulu jsme připojili zelenou LED diodu pro určení stavu komunikace. Pokud je robot v provozu, dioda slabě svítí. Jestli započneme přenos dat mezi počítačem a robotem, tak tato dioda signalizuje přenos zvýšením jasu.

Pro přidání modulu OEMSPA není k dispozici schématická značka pro používaný návrhářský editor Eagle. Potřebnou knihovnu nevlastní ani výrobce, prodejce a ani není dostupná na internetu ke stažení. Proto bylo nutné si takovou značku vyrobit.

Bluetooth modul obsahuje pájecí plošky, které jsou určené pro umístění na patici. Tato patice není běžně dostupná v prodejně sítí v České republice, ale prodejce je schopen ji dodat. Podle dokumentace k patici [31] jsem vytvořil schématickou značku s využitím internetového zdroje [17].

Součástí práce je knihovna pro program Eagle s názvem ROBOT, která obsahuje schématickou značku znázorňující bluetooth modul se všemi piny a symbol pro umístění na desku plošného spoje. V knihovně je dostupná verze pro jednovrstvou a také pro dvouvrstvou desku <sup>47</sup>.

<sup>47</sup>Protože program Eagle neobsahuje ani mikrospínač s dlouhou páčkou nebo jiný mikrospínač se stejným rozložením vývodů, je součástí knihovny ROBOT také tato součástka.

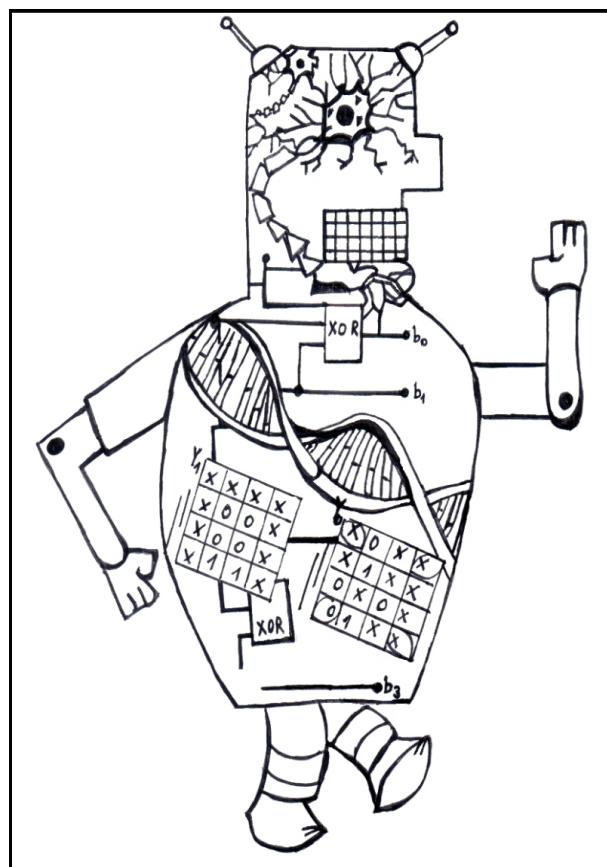
---

---

# KAPITOLA 3

---

## ŘÍDÍCÍ APLIKACE



### 3.1 Firmware

Firmware je řídící program robota pracujícího přímo s jeho hardwarovými prostředky. Je to něco stálého, většinou poměrně malého, program, který vnitřně ovládá různá elektronická zařízení. Takový malý program můžeme najít například v automatické pračce, set-top-boxu nebo v DVD přehrávači.

Program, který využívám v robotovi je naprogramován v programovacím jazyku C.

*C is a compact general-purpose programming language that was originally developed by Dennis Ritchie for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972.*

McGrath M. [24]

*C je kompaktní univerzální programovací jazyk který byl původně vyvinut Dennisem Ritchiem pro operační systém Unix. Poprvé byl implementován na počítači PDP-11<sup>1</sup> společností Digital Equipment Corporation v roce 1972.*

McGrath M. [24]

Základní programovací struktury jsou shrnutý v [9] a [10]. Tato literatura je velmi užitečná jako učební text.

Nejužitečnější se ukázalo použití ukazatelů. Pomocí této techniky jsem zapouzdřil data, která byla předávána návratem volaných funkcí. Nejvíce mě však zaujala možnost zapouzdřit (ukrýt) a předat ukazatel na funkci jiné funkci jako argument, která pak pomocí ukazatele k předané metodě<sup>2</sup> přistupuje.

Nelehká byla při programování práce s polem. U programování procesorů nelze pole inicializovat za běhu. Při překladu programu do binárního souboru musí už být velikosti polí známé. Dalším omezením, které programování procesorů obnáší,

---

<sup>1</sup>16 bitový minipočítač

<sup>2</sup>V této práci zaměňuji pojem funkce a metoda. Funkce je část programu, kterou můžeme volat opakováně z různých míst kódu. Funkce může mít argumenty (parametry), údaje, které jí jsou předávány při volání a návratovou hodnotu, kterou vrací. Metoda je také funkce, ale tento pojem se používá u objektově orientovaného programování, kdy metoda je funkce z vnějšku programu neviditelná, tj. je ukrytá (zapouzdřená) v objektu a metody, které jsou vidět se jmennují rozhraní objektu.

je nemožnost sdílení kódu mezi hlavním programem a algoritmem volaným při přerušení. Vyjmenovaná omezení mě vedli při programování k chybám, které se nesnadno odhalovaly vzhledem k tomu, že hardware robota je vybaven jedinou vizuální kontrolou, LED diodou.

Velmi užitečné je programovat metody v prostředí počítače například ve vývojovém prostředí Dev-C++ a po jejich vyladění je přenést do hardwarového prostředí.

Celý firmware je rozdelen do několika programových modulů, které pracují s jednotlivými hardwarovými obvody. Hlavní program **main**<sup>3</sup> pak zvlášť pracuje s komunikačním modulem, s modulem který ovládá pohon robota, s modulem měřícím vzdálenost pomocí sonaru a s modulem pro měření napětí.

„Kdyby každý modul jako samostatný program vyhodnocoval vstupy a vytvářel na základě toho výstup, tak je možné jednotlivé moduly považovat za agenty a v celku by se pak jednalo o multiagentový systém.“

Robot je stavovým automatem<sup>4</sup>. Tento pohled je nevyhnutelný vzhledem k možnosti nesdílení kódu mezi přerušením a hlavním programem. Pokud robotovi pošleme data, tak nastane přerušení, které odvede zpracování algoritmu z hlavního programu. Robot v přerušení přijme data, zpracuje je (tím se nastaví do určitého stavu) a po zpracování se vrátí pokračovat do hlavního programu. Robot pak podle tohoto nastavení pracuje.

Celkem je k dispozici devět stavů (S, V, L, M1S, M1D, M2S, M2D, T1, T2).

- S - sonar (on/off)
- V - voltmeter (on/off)
- L - LED dioda (on/off)
- M1S - rychlosť levého motoru (32 stupňů)
- M1D - směr levého motoru (dopředu/dozadu)

---

<sup>3</sup>Main je funkce, která spouští jako první při spuštění programu. Tedy po připojení procesoru k napájení.

<sup>4</sup>Základní informace o stavových automatech jsou k dispozici v [36] nebo v [38]

- M2S - rychlosť pravého motoru (32 stupňů)
- M2D - směr pravého motoru (dopředu/dozadu)
- T1 - levé tlačítko (sepnuto/rozepnuto)
- T2 - pravé tlačítko (sepnuto/rozepnuto)

V tomto stavovém prostoru se nachází  $2^7 \cdot 32 \cdot 32 = 131072$  stavů.

Z počítače nemůžeme nastavovat stav tlačítka T1 a T2. O jejich stavech nás informuje robot. Pokud nastavíme proměnnou V do stavu on, tak robot změří a pošle jedenkrát hodnotu stavu baterií naslouchajícímu programu a poté nastaví V zpět na off. Ostatní hodnoty nastavuje naslouchající aplikace a robot na ně reaguje podle programu.

Prakticky se tedy můžeme dotádat robota na stav baterie, zapnout a vypnout LED diodu, vypnout, nebo zapnout sonar<sup>5</sup> a ovládat pohon robota. Vypnutý sonar šetří energii. Robot kontroluje stavy tlačítka zhruba každých 37,150 milisekund<sup>6</sup> tj. 26,92 krát za sekundu. Výpočet je součtem doby, kterou potřebuje sonar pro změření maximální vzdálenosti<sup>7</sup>, doby, kterou potřebuje procesor pro změření stavu baterie<sup>8</sup> a čas potřebný pro odeslání dat<sup>9</sup>. Ale protože je programový cyklus příliš rychlý a při zpracování dat v počítači docházelo ke zkreslení, tak bylo do cyklu vloženo zpoždění 21973  $\mu S$ . Po přičtení k odvozené době potřebné pro měření a odeslání dat dostaneme 59123  $\mu S$ , tj. 60  $mS$ . Stav tlačítka a odeslání dat do PC proběhne každých 60 milisekund a to je 16,6667 krát za sekundu.

---

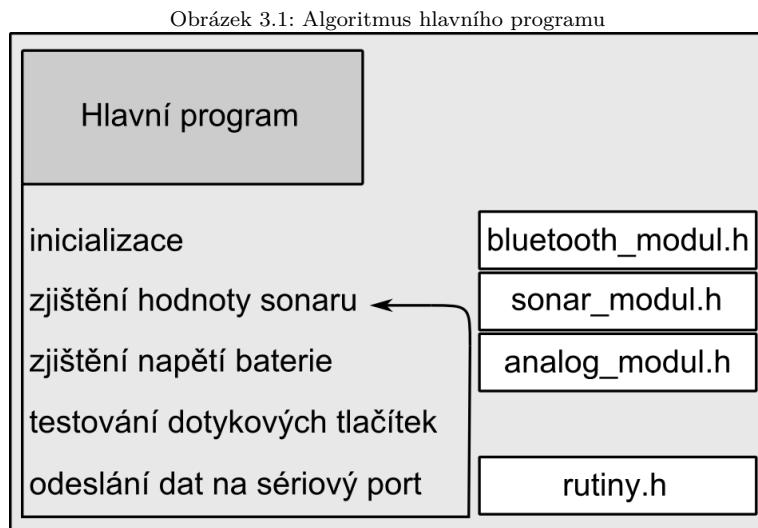
<sup>5</sup>To se projeví tak, že nám bude robot posílat souvislé informace o vzdálenosti.

<sup>6</sup>Hrubý horní odhad při zapnutém sonaru a při aktivovaném měření napětí baterie.

<sup>7</sup>Pokud sonar vrátí 30 milisekund, tak překážka nebyla detekována.

<sup>8</sup>Simulace v programu MPLAB měří tento čas na 150  $\mu S$ .

<sup>9</sup>Simulace v programu MPLAB měří tento čas na 7ms.



### 3.1.1 Diagram řízení a popis algoritmu

Program nahraný v procesoru, který řídí základní operace a komunikaci robota s počítačem využívá modulární přístup. Tyto moduly jsou naprogramovány tak, aby byly co nejvíce nezávislé na hlavním programu. Toho nelze sice docílit na 100%<sup>10</sup>, ale bylo dosaženo uspokojivé úrovně nezávislosti.

Obrázek [3.1] ukazuje jak je naprogramován algoritmus hlavního řízení robota. Návrh robota byl zjednodušen tak, aby pouze odesílal po sériovém portu data a přizpůsoboval se datům přijatým. Hardware tak nedělá zbytečně náročné operace, ale jsou převedeny na silnější výpočetní systém.

V části inicializace definujeme vstupy a výstupy pinů procesoru<sup>11</sup> a nastavujeme USART modul<sup>12</sup> pomocí metod naprogramovaných v modulu **bluetooth\_modul.h**.

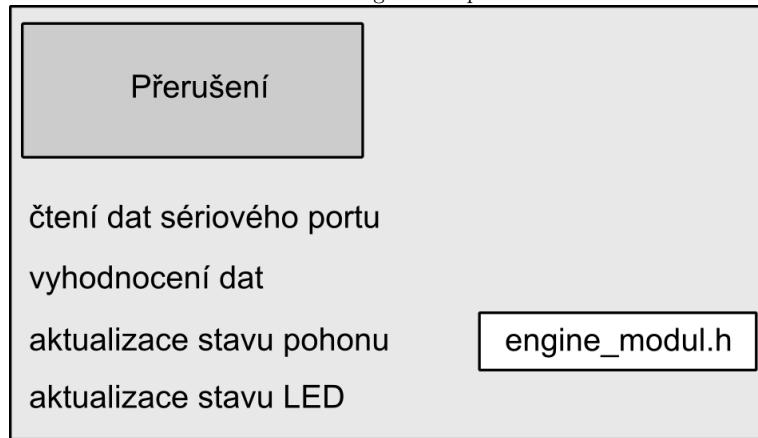
Ultrazvukový dálkoměr je řízen algoritmem z modulu **sonar\_modul.h**. Vracená hodnota je typu ukazatel na definovanou strukturu dat. Tato struktura nese

<sup>10</sup>Když bude v jednom modulu použit například Timer2, tak musíme dát pozor na použití v jiném modulu. Tento problém jsme však řešit nemuseli, neboť procesor disponuje třemi časovači a my využíváme pouze dva.

<sup>11</sup>Pouze piny, na kterých jsou připojena dotyková tlačítka a LED dioda. Tyto piny nejsou použity v žádném modulu.

<sup>12</sup>Tento modul je hardwarovou součástí procesoru a využíváme ho pro komunikaci s počítačem prostřednictvím bluetooth modulu

Obrázek 3.2: Algoritmus přerušení



informace o vzdálenosti od překážky a obsahuje dvě proměnné typu `unsigned char`.

Napětí baterie zjišťujeme stejným principem pomocí modulu `analog_modul.h`. Ten nese algoritmus pro zjištění hodnoty analogového vstupu, na kterém je připojena baterie. Metoda pro určení hodnoty napětí nám vrací ukazatel na strukturu.

Hlavní algoritmus je také vybaven podmínkami, které nastavuje naslouchající program na druhé straně sériové komunikace. V závislosti na nastavení těchto podmínek může robot vracet pouze hodnoty sonaru, voltmetru <sup>13</sup> nebo stavy dotykových tlačítek.

Další důležitou součástí programového řešení robota je přerušení. Obrázek [3.2] ukazuje algoritmus, který se provede po vyvolání přerušení z důvodů příchozích dat do jednotky USART <sup>14</sup>. Poté vyhodnotíme přijatá data a podle jejich obsahu rozsvítíme, nebo zhasneme LED diodu, nebo upravíme nastavení pohybového aparátu pomocí modulu `engine_modul.h`.

<sup>13</sup>Obvod a algoritmus, který používáme pro měření napětí nazýváme voltmetrem.

<sup>14</sup>Jiné přerušení nepoužíváme, neboť jako optimální řešení se ukázalo pouze přijmout data a nastavit robota do daty požadovaného stavu. Bylo možné používat přerušení také pro dotyková tlačítka, ale v tomto případě by se zkomplikoval datový protokol a hardwarový návrh plošného spoje pro procesor a okolní obvody.

### 3.1.2 Makra

Při programování jsme si zjednodušili přístup k jednotlivým registrům procesoru pomocí takzvaných maker, neboli direktiv preprocesoru. Jedná se o instrukce pro překladač, který na jejich základě upravuje překlad programu do binárního souboru <sup>15</sup>.

Tímto přístupem jsou naprogramované moduly více přenositelné, neboť stačí pouze nahrát požadovaný soubor <sup>16</sup> do projektu a změnit nastavení maker <sup>17</sup>.

Pomocí maker si nastavujeme parametry, které používáme v našem programu a překladač je při překladu nahradí uvedenými hodnotami.

Makro zapisujeme `#define parametr hodnota_která_nahradí_parametr`.

Seznam použitých maker je uveden v příloze na konci práce.

Před překladem může spustit překladač algoritmus, jehož výsledkem nahradí jeho volání a této vlastnosti využíváme také následujícím makrem.

```
#define LENGTH_ARRAY(x) (sizeof(x) / sizeof(x[0]))
```

Při překladu musí být známá velikost všech polí. Pokud máme definované pole `unsigned char pole[]`, tak nám volání funkce `LENGTH_ARRAY(pole)` vrátí počet prvků v poli a tento výsledek dosadí překladač na určené místo.

### 3.1.3 Modul pro měření napětí

Hlavní metodou tohoto modulu je `struct anModul *ANALOG_READ(void)`. Metoda vrací ukazatel na strukturu `anModul` a ta obsahuje dvě proměnné `DATA_H` a `DATA_L` typu `unsigned char`. Napětí měříme pomocí analog-digitálního 10 bitového převodníku. Proto musíme hodnotu uložit do dvou bytu.

Metoda `ANALOG_READ`, pokud je spuštěna poprvé, zavolá metodu `void init_con_0_1(void)`. Tato metoda nastaví registry `ADCON0` a `ADCON1`, pomocí kterých převodník nastavujeme.

<sup>15</sup>Tento soubor má příponu „hex“ a nahráváme jej do programové paměti procesoru.

<sup>16</sup>Máme na mysli soubor `*.modul.h`, `*.modul.c` a `macros.h`

<sup>17</sup>V případě použití jiného procesoru už makra přenositelná nejsou. Každý procesor je specifický a liší se v nastavení jeho zabudovaných funkcích.

V naší aplikaci je použito následující:

`ADCON0 = 0x80`

`ASCON1 = 0xce`

Díky tomuto nastavení je převodník vypnut, používáme hodinový signál  $f_{osc}/64$  a AN0 jako analogový kanál.

Dále spouštíme metodu `void read_analog_pin(void)`.

Tato metoda jako první nastaví bit ADON registru ADCON0 na jedničku. Tím zapneme A/D převodník. Samotný převod spustíme nastavením bitu ADGO registru ADCON0 do vysokého stavu. Následuje cyklus, který čeká až se bit ADGO vynuluje. Po vynulování je převod dokončen a jeho výsledek je uložen v registrech ADRESH a ADRESL.

Nakonec vypneme A/D převodník `ADON = 0`, uložíme výsledek do struktury a vrátíme její adresu.

### 3.1.4 Komunikační modul

**bluetooth\_modul** umožňuje pracovat s komunikačním rozhraním procesoru, jednotkou USART. V tomto smyslu je název modulu trošku zavádějící. Bluetooth funguje jako samostatná součástka, na kterou jsou připojeny piny RX a TX. Ovšem programátor, který **bluetooth\_modul** používá nemusí vědět nic o jeho vnitřní funkčnosti a hardwarovém zapojení<sup>18</sup>.

Používat můžeme metodu `void BLUETOOTH_INIT(void)`, která inicializuje sériovou komunikaci a metodu `void BLUETOOTH_RESTART(void)`, pomocí které restartujeme USART jednotku v případě chyby<sup>19</sup>.

---

<sup>18</sup>Tady jsme si „vypůjčily“ myšlenku objektově orientovaného programování.

<sup>19</sup>To nastane pokud dojde k potlačení přenosu dat z registru RSR do RCREG. Tato chyba nastává pouze při příchozí komunikaci.

Obrázek 3.3: Registr TSXTA							
R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Obrázek 3.4: Registr RCXTA							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

## Inicializace USART jednotky

Pro nastavení komunikace slouží registry TXSTA a RCSTA. Jejich jednotlivé bity jsou zobrazeny na obrázcích [3.3] a [3.4]. Nám postačilo nastavení, které shrnuje tabulka [3.1]. Pak je ještě důležité nastavit TX.TRIS jako výstup a RX.TRIS jako vstup.

Registr SPBRG kontroluje periodu volně běžícího 8 bitového časovače. Nastavením tohoto registru dosáhneme požadované přenosové rychlosti. Podle nastavení bitu BRG, SYNC a požadované rychlosti vypočítáme hodnotu SPBRG. Podle tabulky [3.2] vybereme potřebný vzorec.

Pro přenosovou rychlosť 9600 bitů za sekundu a asynchronní režim jsme vypočítali [3.1] [3.2] [3.3] na hodnotu SPBRG na 15.

Po kompletním nastavení jednotky USART nastavíme přerušení <sup>20</sup>, abychom mohli reagovat na příchozí data. V tomto místě je myšlenka modulárního přístupu mírně narušena. Metoda `void interrupt isr()` <sup>21</sup> může být umístěna pouze v souboru s metodou `main`.

$$Baud\_rate = \frac{F_{OSC}}{64 \cdot (X + 1)} \quad (3.1)$$

<sup>20</sup>Přerušení se práce věnuje v [3.1.8].

<sup>21</sup>Jakmile nastane přerušení, spustí se kód, který je umístěn právě v metodě `interrupt isr`.

Tabulka 3.1: Nastavení registrů TXSTA a RCSTA

TX9 =	0	Nastaví 8 bitový přenos dat
TXEN =	1	Povolí vysílání
SYNC =	0	Výběr asynchronního režimu
BRGH =	0	Výběr nižší přenosové rychlosti
SPEN =	1	Povolení funkce sériového portu
RX9 =	0	Nastaví 8 bitový přenos dat
CREN =	1	Povolení kontinuálního příjmu dat

Tabulka 3.2: Přehled vzorců pro výpočet přenosové rychlosti

SYNC	<b>BRGH = 0</b>	<b>BRGH = 1</b>
0	$Baud\_rate = \frac{F_{OSC}}{64 \cdot (X+1)}$	$Baud\_rate = \frac{F_{OSC}}{4 \cdot (X+1)}$
1	$Baud\_rate = \frac{F_{OSC}}{16 \cdot (X+1)}$	N/A

$$X = \frac{F_{OSC}}{64 \cdot Baud\_rate} - 1 \quad (3.2)$$

$$X = \frac{9830400}{64 \cdot 9600} - 1 = 15 \quad (3.3)$$

### Restart USART jednotky

Tuto záležitost řešíme bity SPEN a CREN z registru RCSTA. Bitem SPEN povolujeme, nebo zakazujeme funkci sériového portu a bitem CREN povolujeme a zakazujeme přijímač.

Metoda `void BLUETOOTH_RESTART(void)` restartuje USART jednotku vynutováním výše zmíněných bitů a jejich znova nastavením.

### 3.1.5 Modul pro ovládání motorů

Tento modul pracuje nezávisle na hlavním programu. To nám umožňuje měnit nastavení pohonu při přerušení a dále si jej nevšímat.

Rozhraní modulu nabízí pro práci s ním tři funkce.

```
void LEFT_M_State(unsigned char speed, unsigned char dir)
void RIGHT_M_State(unsigned char speed, unsigned char dir)
void ENGINE_UPDATE(void)
```

Funkcemi `LEFT_M_State` a `RIGHT_M_State` nastavujeme chování motorů, jejich směr a rychlosť. Hodnoty předané těmto funkcím se pouze uloží do vnitřních proměnných modulu. Zavolání funkce `ENGINE_UPDATE` obnovíme registry vnitřního nastavení PWM novými hodnotami. Tím se aplikují hodnoty předané funkci `LEFT_M_State` a `RIGHT_M_State`.

## Režim PWM

Přenosový signál, který nese informaci o přenášené hodnotě může nabývat hodnot 0, nebo 1 (zapnuto/vypnuto). V praxi se na pinu procesoru objeví při logické jedničce pět voltů a při logické nule nula voltů.

Motory ovládáme přivedením napětí na výstup. Tím se motor bude točit, nebo nebude podle toho, zdali je na jeho vstup přivedeno napětí <sup>22</sup>.

Pokud na motor přivedeme méně než 6 voltů, bude se točit pomaleji. Protože ale procesor umí vyslat na svůj výstup pouze 5, nebo 0 voltů, pomůžeme si rychlým přepínáním logické nuly a jedničky. Změna napětí na výstupu procesoru v čase pak bude vypadat jak ukazuje obrázek [3.5]. Písmeno T značí periodu, definovanou dobu trvání vyslaného pulsu nebo jiného definovaného děje. Periodu nastavíme například na 20 sekund  $T = 20[S]$ . Převrácená hodnota periody se jmenuje frekvence  $f = 1/T$  <sup>23</sup> a určuje počet opakování děje za jednotku času (třikrát za minutu).

Podle obrázku bude na výstupu procesoru 5 voltů po dobu 10 sekund (horní část obrázku) a druhých 10 sekund bude na výstupu 0 voltů. Pokud uděláme aritmetický průměr obou hodnot, dostaneme hodnotu 2,5 voltů za 20 sekund. Ovšem tato perioda je příliš dlouhá. V této práci používáme periodu  $1,67mS$  [3.4].

Na spodní části obrázku je nastavena doba periody na 20%. Pětina 20 sekund jsou 4 sekundy. Takže na výstupu naměříme  $(4 \cdot 0 + 1 \cdot 5)/5 = 1[V]$  <sup>24</sup>.

$$T = (256) \cdot \frac{4}{9830400} \cdot 16 = 1,67 \cdot 10^{-3} \quad (3.4)$$

Hodnota přenášeného signálu je v přenosu „zakódována“ jako poměr mezi stavy zapnuto/vypnuto. Tomuto poměru se říká střída. Perioda je doba, kdy dojde k přenosu jedné střídy.

Perioda je vždy součtem doby zapnuto a vypnuto.

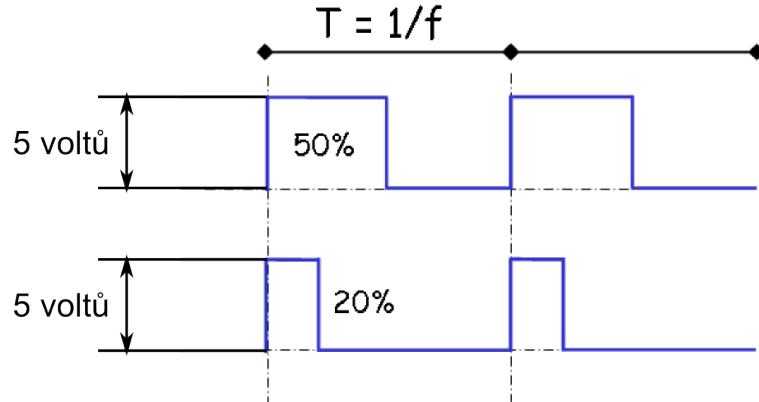
---

<sup>22</sup>Stejnosměrné motory ovládáme pomocí 4 kanálového můstku, ale pro zjednodušení budeme předpokládat, že jsou připojené přímo na výstup procesoru (i když bychom tím procesor ve skutečnosti zničili).

<sup>23</sup>Jednotkou frekvence je Herz [Hz].

<sup>24</sup>Samozřejmě ne doopravdy. Ve skutečnosti je perioda 20 sekund příliš dlouhý čas.

Obrázek 3.5: Časový průběh stavu napětí na výstupu procesoru



Obrázek 3.6: Registr CCP1CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

### Inicializace a spuštění režimu PWM

Pokud je metoda ENGINE\_UPDATE zavolána poprvé, spustí se podprogram `init()`, který inicializuje režim PWM a podprogram `PWM_OFF_ON(unsigned char value)`.

Metoda init inicializuje porty procesoru CCP [3.6] a datové porty RD0 a RD1. Datovými porty měníme směr přivedením logické nuly, nebo jedničky a na výstupech CCP (Capture/Compare/PWM) je generován PWM signál. Předděličku Timera2 nastavíme na hodnotu 16 nastavení bitu T2CKPS1 registru T2CON [3.7]. Důsledkem bude zvětšení periody na maximální možnou hodnotu. Toto tvrzení se opírá o vzoreček z dokumentace procesoru [3.5], který jsme už použili [3.4]. Nakonec funkce init nastaví délku periody PR2 na hodnotu 0xff<sup>25</sup>.

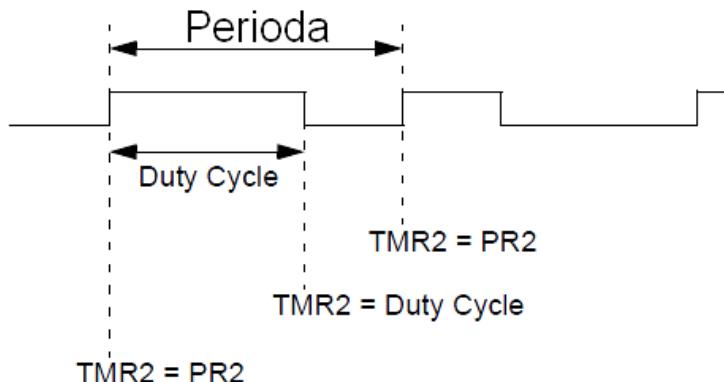
$$T = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (preddelickaTMR2) \quad (3.5)$$

<sup>25</sup>Význam registru PR2 je vidět na obrázku [3.8].

Obrázek 3.7: Registr T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Obrázek 3.8: Vztah mezi PWM periodou a PWM duty cykle



Metoda PWM\_OFF\_ON přijímá hodnotu, podle které se rozhodne o zapnutí, nebo vypnutí režimu PWM. Při vypnutém stavu se šetří energie <sup>26</sup>. PWM režim se aktivuje nastavením bitů CCPxM0 až CCPxM3 registru CCP1CON a CCP2CON. Procesor je vybaven dvěma CCP moduly pro režim PWM. Oba používají Timer2 a nastavení periody, ale rychlosti můžeme ovládat u každého modulu zvlášť. Nakonec musíme zapnout Timer2 nastavením bitu TMR2ON registru T2CON na logickou jedničku.

Rychlosť motorů (poměru střídy) měníme 10 bitovou hodnotou CCPR1L:CCP1CON<5 : 4><sup>27</sup>. Obrázek [3.8] vyznačuje vztah mezi PWM periodou a PWM duty cyklem. Pracovní cyklus, neboli duty cyklem, vypočítáme podle vzorce [3.6].  
Pokud uložíme do CCPR1L:CCP1CON<5 : 4> maximální hodnotu 1023, tak se měla hodnota pracovního cyklu rovnat periodě.

$$T_{duty} = (CCPR1L : CCP1CON <5 : 4>) \cdot T_{OSC} \cdot (preddelickaTMR2) \quad (3.6)$$

### 3.1.6 Ovládání ultrazvukového dálkoměru

Princip měření vzdálenosti je naznačen v oddíle [2.1.3]. Z popsaného postupu je zřejmé, že budeme potřebovat časovač. Použitý procesor umožňuje použít tři

<sup>26</sup>Díky vypnutému Timeru2 [26] a neaktivnímu pohonu.

<sup>27</sup>Celým registrem CCPR1L a 5. a 4. bitem registru CCP1CON.

druhy časovače. Timer2 používáme pro generování PWM signálu pro pohon robota, Timer1 pracuje jako 16 bitový čítač-časovač a Timer0 jako 8 bitový čítač-časovač. Všechny tři moduly umožňují nastavení předděličky časového signálu  $F_{OSC}$  a vyvolávají vnitřní přerušení při jejich přetečení<sup>28</sup>.

$$F_{+PULS} = \frac{9830400}{4} \cdot \frac{1}{8} = \frac{9830400}{32} = 307200 \quad (3.7)$$

$$t_{+PULS} = \frac{1}{F_{+PULS}} = 3,255208333 \cdot 10^{-6} \doteq 3,255\mu S \quad (3.8)$$

Ultrazvukový dálkoměr vrací hodnotu od  $100 \mu S$  do  $30 mS$ . Při volbě předděličky 1:8 se přičte jeden puls do registru každých  $3,255 \mu S$ . Této hodnoty dosáhneme výpočtem 3.7 a 3.8<sup>29</sup>. Čítač podle výpočtů načte  $30 mS$  za zhruba 10000 pulsů, takže volba výběru čítače padla jednoznačně na Timer1<sup>30</sup>. Timer1 přičítá pulsy do registrového páru TMR1H:TMR1L. Jakmile teto páru dosáhne hodnoty 0xffff a přičte se další puls, nastane vynulování hodnot a takzvané přetečení, které signalizuje přesun do vyššího číselného řádu. Další hodnotou by bylo číslo 0x10000, ale další registr pro záznam pulsů nemáme k dispozici. Pokud nastane přetečení, vyvolá se vnitřní přerušení a nastaví se bit TMR1IF registru PIR1. V naší aplikaci přerušení Timeru1 nepoužíváme. Omezili jsme se pouze na kontrolu bitu TMR1F.

Při každém volání metody `struct sonarModul *GET SONAR DATA(void)` se inicializuje Timer1, vyšleme TRIGGER signál [2.1.3], čekáme na signál ze sonaru a poté vytvoříme strukturu nesoucí hodnoty obou registrů TMR1H a TMR1L do kterých se časový údaj Timeru1 ukládá. Nakonec metoda vrátí ukazatel na místo v paměti, kde se vrácená struktura nachází. Tento přístup<sup>31</sup> umožňuje využít Timer1 i v jiném algoritmu ve stejné aplikaci.

---

<sup>28</sup>Při překročení maximální hodnoty čítacích registrů.

<sup>29</sup>Hodnota 9830400 je frekvence krystalu v Hz a za čtyřnásobek této doby se vykoná jeden instrukční cyklus [26].

<sup>30</sup>Timer1 obsahuje 16 bitový registrový páru a zvládne zaznamenat 65536 pulsů. Naproti tomu Timer0 zaznamená pouze 256 pulsů.

<sup>31</sup>Konkrétně mám na mysli inicializaci Timeru1 a ukládání výsledku do struktury při každém volání metody `struct sonarModul *GET SONAR DATA(void)`.

Obrázek 3.9: Registr T1CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7				bit 0			

Inicializaci časovače provádíme nastavení registru T1CON na hodnotu 0x30. Na obrázku [3.9] jsou vidět jednotlivé bity registru. Pomocí T1CKPS1:T1CKPS0 nastavujeme hodnotu předděličky, bitem T1OSCEN povolujeme oscilátor pro Timer1, bit T1SYNC synchronizuje vnější hodiny <sup>32</sup>, bitem TMR1CS vybíráme zdroj signálu a bit TMR1ON spouští, nebo zastavuje časovač <sup>33</sup>.

Trigger signál, který posíláme na port sonaru musí trvat alespoň  $10 \mu\text{s}$  [2.1.3]. To dosáhneme nastavením registrového páru TMR1H:TMR1L na hodnotu 0xffff. Po tomto nastavení zbývá do přetečení čtyři pulsy.

$$4 * 3,255[\mu S] = 13[\mu S] \quad (3.9)$$

Výpočet [3.9] teoreticky ukazuje dobu trvání do přetečení registrového páru. Simulace programem MPLAB IDE v8.76 ukazuje  $15.462240 \mu\text{s}$ .

Dále musíme registrový pář vynulovat, vynulujeme bit TMR1IF, spustíme Timer1 a nasloucháme sonaru na pinu. Pin se po určité době nastaví do vysokého stavu a vydrží v něm tak dlouho, jak daleko se nachází překážka před robotem <sup>34</sup>. Čekáme až Timer1 nastaví bit TMR1IF do vysokého stavu. Poté časovač zastavíme, uložíme hodnotu TMR1H:TMR1L do struktury a vrátíme její adresu.

### 3.1.7 Rutiny metody main

Metoda main využívá kromě rozhraní modulů také vlastní podprogramy, které jsou uloženy v souboru **rutiny.c**. Pro přehlednost jsme tak oddělili hlavní algoritmus od vedlejších funkcí. I když v současné verzi je v tomto souboru k dispozici pouze jediná funkce

---

<sup>32</sup>Pouze za předpokladu, že máme nastavený bit TMR1CS na jedničku a tedy používáme vnější zdroj hodinového signálu.

<sup>33</sup>Pokud je časovač vypnut (TMR1ON = 0), dosáhneme úspory energie. Proto je výhodné nenechávat časovač spuštěný pokud jej nepotřebujeme

<sup>34</sup>Jakmile spadne hodnota na „sonarovém“ pinu na nulu, znamená to, že tak dloouho co Timer1 počítal trvalo zvuku, než doputoval k překážce a zpět.

---

```
void SENDData(struct sonarModul *SONO, struct anModul *VOLT,
unsigned char BTL, unsigned char BTR) .
```

Tato funkce má za úkol poslat na sériový port informace o vzdálenosti překážky od robota, o stavu baterie a stavy dotykových tlačítek. Funkce přijímá ukazatele na požadované struktury dat a stavy tlačítek.

### Komunikační protokol

Algoritmus ukládá přijaté hodnoty do pole o pěti prvcích. Na adresu [0] je uložen stav tlačítka<sup>35</sup>, na adresu [1] a [2] data sonaru a na adresu [3] a [4] data voltmetu.

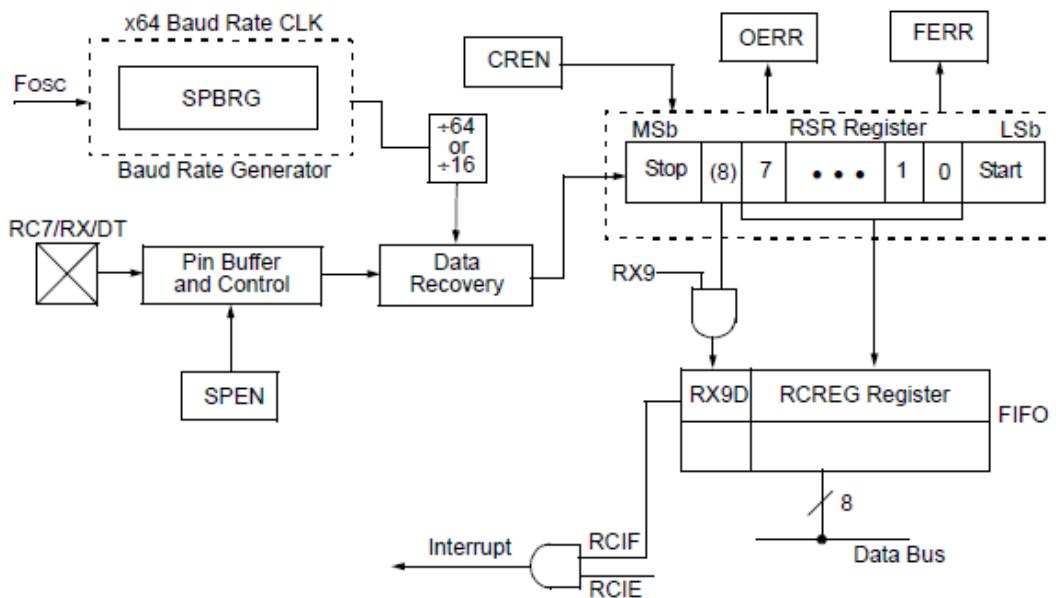
Následující funkce odesílá hodnoty uložené v poli na sériovou linku. Před odesláním jednotlivých bytů čekáme dokud se nevynuluje bit TRMT. Ten indikuje stav posuvného registru vysílače TSR, který se po naplnění odesílá na sériovou linku. Pokud je bit TRMT nastaven do vysokého stavu, indikuje prázdný registr TSR a je možné odeslat další data. Po odeslání dat z pole pošleme třikrát po sobě hodnotu 0xff. Ukončujeme tak komunikační protokol a naslouchající program pozná konec protokolu. Tím je zaručená konzistence dat i kdyby došlo k výpadku komunikace.

```
void odesliData(unsigned char pole [] ,
                 unsigned char delkaPole){
    int index;
    for (index = 0; index < delkaPole; index++){
        while (!TRMT);
        SERIAL_OUT = pole [index];
    }
    for (index = 0; index < 3; index++){
        while (!TRMT);
        SERIAL_OUT = 0xff;
    }
}
```

---

<sup>35</sup>Abychom ušetřili byte při odesílání, tak jsme informaci o tlačítkách uložili do jednoho bytu  $pose[0] = ((BTL \& 0b00000001) \ll 4) + (BTR \& 0b00000001)$ .

Obrázek 3.10: Blokové schéma přijímače



### 3.1.8 Prerušení

Chceme-li použít přerušení, musíme nastavit bit RCIE, který indikuje přijatá data. Pak také musí být povoleno globální přerušení bitem GIE a přerušení periferií bitem PEIE registru INTCON. Více o přerušení je popsáno v [23].

Jak už bylo zmíněno, přerušení používáme pouze pro příchozí komunikaci. Na obrázku znázorňující blokové schéma přijímače [3.10] je vidět vztah mezi posuvným registrém RSR a bity OERR a FEER registru RXSTA.

Pokud procesor přijal nějaká data, vyvolá přerušení a nastaví bit RCIF, který je umístěn v registru PIR1. Zdroj [26] uvádí, že pokud je OERR nastaven, tak došlo k potlačení přenosu dat z registru RSR do RCREG. To nastane pokud je při přenosu ztracen nějaký bit. Framing Error bit FERR je nastaven pokud nedošlo k detekování stop bitu. Tento bit nebrání nijak v činnosti přijímače, je důležitý jen pro uživatele. Bit FERR i případný devátý bit<sup>36</sup> musí být přečteny dříve než je přečten RCREG registr, jinak se přepíší novými hodnotami nově přijatého slova. Přijatá data můžeme přečíst v registru RCREG.

<sup>36</sup>V této práci používáme slovo o šířce 8 bitů.

```
if (RCIF){  
    if (!OERR){  
        if (!FERR){  
            unsigned char prijata_data;  
            prijata_data = RCREG;  
  
            read_data (prijata_data);  
        }  
    }  
    else{  
        unsigned char rubbish;  
        rubbish = RCREG;  
    }  
}  
else{  
    BLUETOOTHRESTART();  
}  
}
```

## 3.2 Řídící software pro PC

Řídící software je program naprogramovaný pro počítač naslouchající výstupu robota. Protože robot komunikuje pomocí sériové linky, nezáleží na volbě programovacího jazyka. Proto pro zvolený programovací jazyk Java rozhodla jeho platformová nezávislost <sup>37</sup>.

Java je vysokoúrovňový programovací jazyk s rozsáhlou knihovnou funkcí a komunitou. Funguje jak na operačních systémech Windows, Unix a MAC OS, tak na mobilních zařízeních <sup>38</sup>.

---

<sup>37</sup>Ne pouze mezi operačními systémy, ale i mezi hardwarovými platformami.

<sup>38</sup>Rozhodně ne v celém svém měřítku. Přenos programu z jedné hardwarové platformy najinou má svá omezení.

Mezi její přednosti patří objektově orientované programování a všechny výhody s ním spojené <sup>39</sup>.

*Práce s objekty* - jednotlivé prvky modelované reality jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody).

*Zapouzdřenost* – zaručuje, že objekt nemůže přímo přistupovat ke skrytým metodám jiných objektů. Každý objekt navenek zpřístupňuje množinu metod, rozhraní, pomocí kterého se s objektem pracuje.

*Skládání* – objekt může obsahovat jiné objekty.

*Dědičnost* – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou dědit z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření.

*Polymorfismus* – odkazovaný objekt se chová podle toho, jaké třídy je instancí. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší podle implementace.

Java má i své nevýhody. Práce se sériovým rozhraním v jazyku Java není tak dobře zpracovaná jako v prostředí .Net. Přímá implementace tohoto rozhraní je v oficiální knihovně ukončena <sup>40</sup> a pokud programátor potřebuje s tímto rozhraním pracovat, musí si stáhnout jednu z mnoha knihoven <sup>41</sup>. My využíváme knihovnu **RXTX** ve verzi 2.1.7. Tato knihovna se podle nalezených příkladů a diskuzních skupin využívá nejčastěji. Její hlavní výhoda tkví v množství komunikačních protokolů. Pomocí této knihovny můžeme například komunikovat pomocí paralelního portu nebo I2C. Ovšem samotné čtení dat ze sériového portu je ve srovnání s jazykem C# a prostředím .Net práce úmorná. Zjistil jsem, že knihovna nepracuje s porty, které nemají název v rozsahu „COM0“ až „COM9“. S porty číslovanými od čísla 10 nejde pracovat. Knihovna generuje chybové zprávy, které nastanou v nativním kódu <sup>42</sup> knihovny RXTX. Dalším omezením je název portu, který musí být pouze velkými písmeny. A významným omezením, které se mi nepodařilo od-

<sup>39</sup>Mezi literaturu, která popisuje podrobněji zmíněné výhody patří například [32].

<sup>40</sup>API javax.comm dostupné v základních knihovnách JDK pouze popisuje takzvané „low-level interface“ a definuje požadované minimum pro práci se sériovým portem.

<sup>41</sup>K dispozici jsou například knihovny javacom20, JComm nebo JCommSerial

<sup>42</sup>RXTX je sice knihovna napsaná v jazyku Java, ale pro práci s konkrétním rozhraním a operačním systémem potřebuje binární soubory pracující s daným API (Application Programming Interface) operačního systému.

stranit je přespříliš dlouhé čekání<sup>43</sup> na bluetooth rozhraní počítače. Při každém použití počítač hledá nová zařízení. V C# se tato chyba nevyskytuje.

Pro naše potřeby jsem se rozhodl naprogramovat knihovnu, která zapouzdří práci s knihovnou RXTX a programování více přiblíží práci se sériovým portem v jazyku C#. Knihovnu jsem pojmenoval **COM\_port** a většinu zmíněných nedostatku se mi podařilo odstínit<sup>44</sup>.

Pro práci s robotem bylo potřeba naprogramovat vyšší abstrakci nad úrovní komunikace se sériovým portem. Vznikla tak knihovna **RobotComm**. Knihovna dává programátorovi k dispozici rozhraní metod, kterými ovládáme vstupy robota a součástí tohoto rozhraní jsou také metody pro práci s posluchači<sup>45</sup>. Po registraci posluchače bude robot všechny naslouchající objekty informovat o jakémkoliv změně stavu.

Dále bych chtěl v této podkapitole popsat práci s výše zmíněnými knihovnami a ukázat funkčnost na malém projektu. Pro ověření komunikace a funkčnosti robota byla navržena formulářová aplikace, pomocí které rozsvěcujeme LED diodu, čteme vzdálenost z ultrazvukového dálkoměru, data z voltmetru, ovládáme pohon robota a reagujeme na stisk tlačítka. Aplikace se jmenuje **TestRobot-CommGUI**.

Všechny projekty jsou dokumentovány a k dispozici na přiloženém CD.

### 3.2.1 Komunikace s RS232

Základní vizí při vytváření knihovny bylo konstruktorem vytvořit objekt sériového portu a předat tak objektu všechny důležité parametry komunikace. Mezi takové parametry patří označení sériového portu, rychlosť komunikace, počet bitů pro přenášená data, počet stop bitů a parita.

Navržené rozhraní třídy COM\_DOM je ukázáno na obrázku [3.11].

U objektu typu **COM\_DOM** registrujeme posluchače pro přijatá data. Posluchač musí implementovat rozhraní iSerialReaderListener. Takový posluchač pak

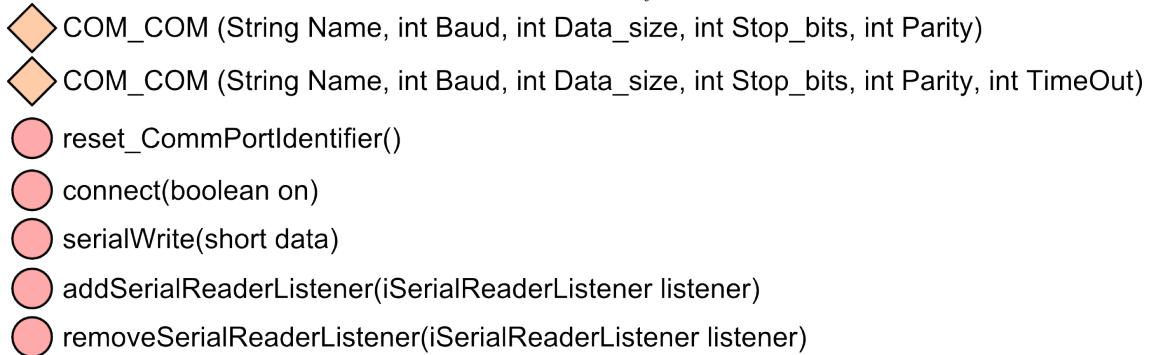
---

<sup>43</sup>Řádově 60 - 100 vteřin.

<sup>44</sup>Ne přímo odstranit. Uživatel knihovny COM\_port však bude vždy vědět jak daný problém vyřešit.

<sup>45</sup>Posluchačem myslím využití návrhového vzoru Listener.

Obrázek 3.11: Rozhraní třídy COM\_DOM



obsahuje metodu `void serial_readed(short [])`. Je vidět, že data, která objekt COM\_DOM posílá jsou typu `short[]`.

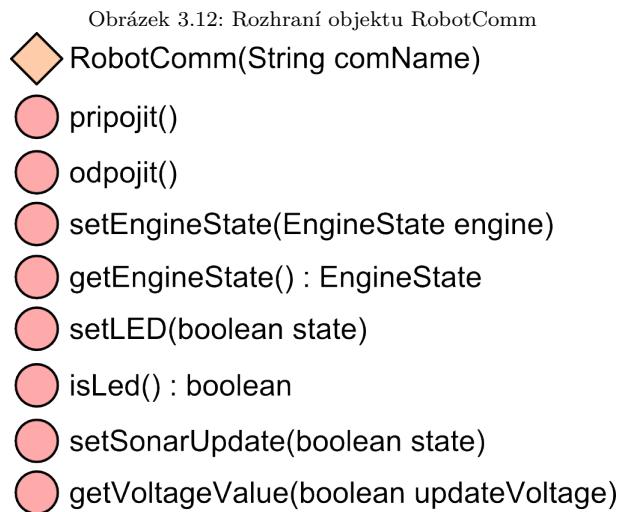
Pro čtení dat je ve třídě COM\_DOM vytvořena vnitřní třída **serialReaderThread**. Tato třída funguje jako samostatné vlákno, které neustále čte sériový port. Jakmile jsou nějaká data k dispozici, plní jimi seznam typu `ArrayList<Short>` a po jejich přečtení je předá nadřazené třídě ke zpracování v poli typu `short[]`. Nadřazená třída potom prochází seznam všech posluchačů a posílá jim přečtená data.

Po vytvoření objektu COM\_DOM a přihlášení se k odběru zpráv je potřeba připojit se na sériový port pomocí metody `void connect(boolean)`. Metoda přijímá logickou hodnotu a ta určí jestli se k portu připojíme, nebo se od něj odpojíme. Pro připojení musíme získat identifikátor sériového portu <sup>46</sup> „CommPortIdentifier“. Připojíme se k portu zavoláním metody `open` objektu `CommPortIdentifier`. Metoda `open` vrací objekt typu `CommPort`. `CommPort` je abstraktní třída a její odvozená třída v knihovně JDK je `SerialPort`. U tohoto objektu zavoláme metodu `setSerialPortParams` a nastavíme tím všechny pro komunikaci důležité parametry <sup>47</sup>. Pomocí metod `getInputStream` a `getOutputStream` získáme vstupní a výstupní proud dat.

Nakonec vytvoříme čtecí vlákno, kterému předáme vstupní proud dat. Z tohoto proudu bude vlákno po celou dobu své životnosti číst příchozí data. Vytvořené vlákno musíme nastavit jako Daemona. Pokud bychom tak neučinili, aplikace by

<sup>46</sup>V této chvíli dochází ke zdržení pokud použijeme bluetooth technologii. Pravděpodobně proto, že algoritmus zjišťuje všechny přístupné porty.

<sup>47</sup>Zde využijeme většinu informací předaných konstruktoru.



nešla korektně ukončit a po destrukci objektu COM\_DOM by stále běžela na pozadí.

Výstupní proud dat slouží pro zápis hodnot. Hodnoty zapisujeme pomocí metody `write` a předáváme ji argument typu `short` v rozsahu 0 až 255.

Pokud bude metoda `connect` zavolána s argumentem `false`, tak nejdříve zruší čtecí vlákno, zavře vstupní a výstupní proud a zavolá metodu `close` objektu `CommPort`. Metodu `CommPort.close()` je potřeba volat jako poslední. Pokud tomu tak není, zastaví se program právě na této metodě<sup>48</sup>.

### 3.2.2 Knihovna RobotComm

Navržené rozhraní objektu `RobotComm` [3.12] umožňuje nastavit stav LED diody, vyžádat informace sonaru nebo voltmetru a nastavit stav pohonu. K tomuto účelu naprogramované metody `setEngineState`, `setLED`, `setSonarUpdate` a `getVoltageValue`. Tyto metody vrací prázdný typ `void` a pouze nastavují robota do požadovaného stavu. Robot pak podle nastavení posílá požadovaná data zpět naslouchající aplikaci.

<sup>48</sup>Pravděpodobně čeká na ukončení nějakého jiného vlákna.

Obrázek 3.13: Rozhraní objektu RobotComm

- addChangeListener(RobotButtonsListener listener)
- addSonarDataListener(RobotSonarDataListener listener)
- addVoltageDataListener(RobotVoltageDataListener listener)
- removeChangeListener(RobotButtonsListener listener)
- removeSonarDataListener(RobotSonarDataListener listener)
- removeVoltageDataListener(RobotVoltageDataListener listener)

Metoda getEngineState vrací objekt EngineState, který slouží pouze jako přepravka pro výměnu dat mezi objekty. Objekt typu EngineState nese informace o rychlosti a směru pohybu jednotlivých motorů. Metoda getEngineState nečeťte tyto informace z robota, ale vrací poslední odeslaný požadavek do robota.

Stejně se chová i metoda isLed, která vrací logickou hodnotu podle toho jestli LED dioda svítí, nebo ne. Poslední nastavenou hodnotu si objekt RobotComm ukládá do vnitřní proměnné a obsah této proměnné se pak vrací metodou isLed.

Metody pripojit a odpojit připojí objekt RobotComm k robotovi.

Konstruktor třídy RobotComm přijímá jako argument řetězec, který nese název sériového portu, na kterém je robot připojen.

Metody add\*Listener a remove\*Listener [3.13] slouží pro přidání naslouchajícího objektu do vnitřního seznamu příjemců zpráv. Při události<sup>49</sup> objekt projde seznam příjemců a pošle jim požadované hodnoty<sup>50</sup>. Pokud by se chtěl nějaký objekt stát příjemcem, musí implementovat rozhraní požadované metodou. Po implementaci bude objekt schopen přjmout data prostřednictvím volání metody, kterou implementoval. Příjemcem se však stane až po přihlášení k odběru zpráv od objektu RobotComm.

RobotSonarDataListener a RobotVoltageDataListener obsahují pouze jednu metodu pro implementaci. Rozhraní RobotButtonsListener obsahuje předpis pro tři metody a to LeftButtonPressed, RightButtonPressed a ButtonStateChanged. K tomuto rozhraní jsem vytvořil abstraktní třídu podle návrhového vzoru Adaptér, který fitruje jednotlivé metody rozhraní falešnou implementací metod<sup>51</sup>.

<sup>49</sup>Příchozí data sonaru, voltmetru nebo změna stavu tlačítek.

<sup>50</sup>Mezi takovéto příjemce může patřit například prvek grafického uživatelského rozhraní (JLabel, JRadioButton,...).

<sup>51</sup>Tato třída se označuje jako obalová třída a často je označována anglickým slovem wrapper (obal).

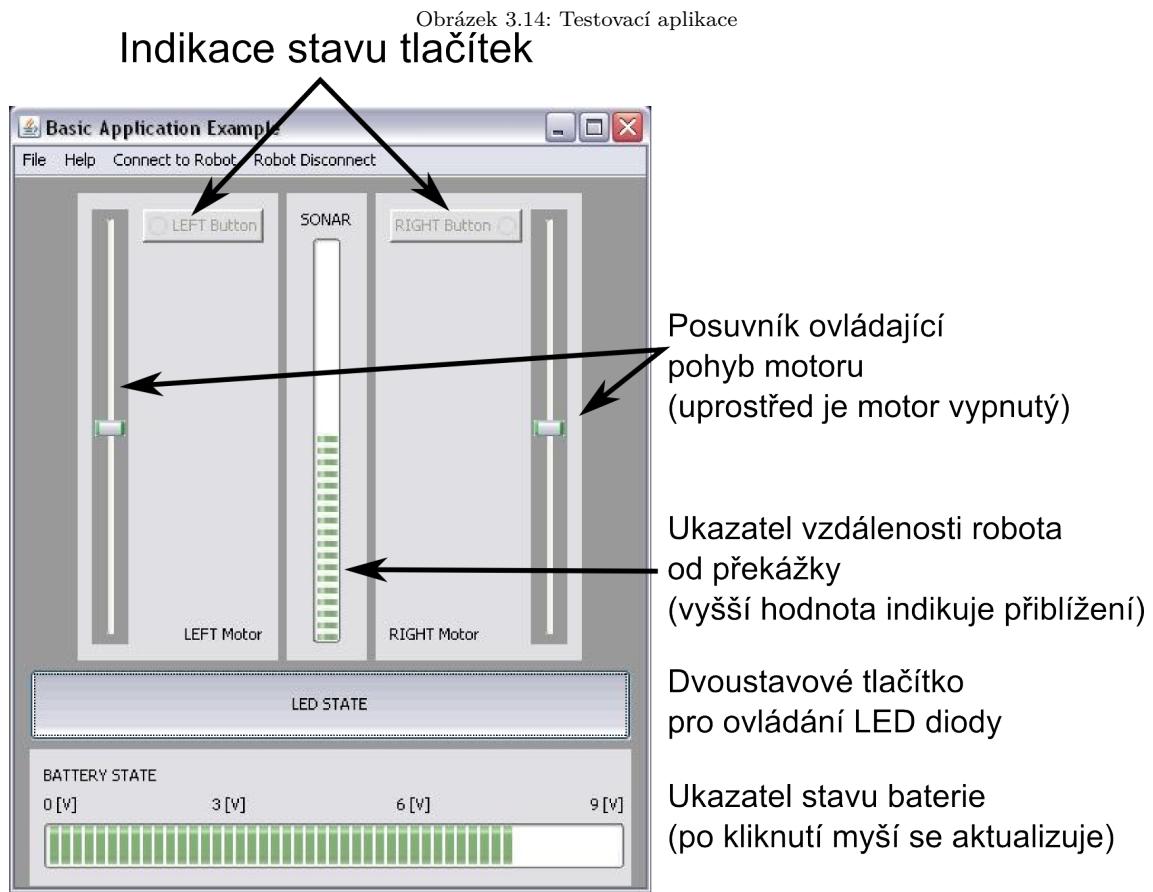
Objekt, který by chtěl reagovat například pouze na stav levého tlačítka může implementovat jen metodu k tomu určenou.

Použití této knihovny je velmi snadné.

1. vytvořit objekt typu RobotComm a předat mu název portu, na kterém je připojen i robot,
2. přidat naslouchající objekty,
3. spustit komunikaci mezi robotem a objekt zavolání metody pripojit.

Můžeme vytvořit i takzvaný anonymní objekt, který implementuje potřebné metody, například jen pro výpis do konzole.

```
robot . addVoltageDataListener (
    new RobotVoltageDataListener () {
        public void BatteryVoltageValue( float value ) {
            System . out . println ( value );
        }
    });
robot . addChangeButtonStateListener (
    new RobotButtonsAdapter () {
        public void ButtonStateChanged
            ( Buttons buttons , boolean value ) {
            if ( buttons == Buttons . left ) {
                RadioLeftButton . setSelected ( value );
            }
            if ( buttons == Buttons . right ) {
                RadioRightButton . setSelected ( value );
            }
        }
    });
}
```



### 3.2.3 Ukázková aplikace

Program je velmi jednoduchý [3.14]. Účelem je otestovat funkčnost robota jako celku, komunikace a navržených algoritmů.

V nabídkové liště jsou položky File <sup>52</sup>, Help <sup>53</sup>, Connect to Robot <sup>54</sup> a Robot Disconnect <sup>55</sup>.

Program zobrazuje všechny porty v daném rozsahu, takže je na uživateli, aby zvolil správný port. Pokud v průběhu programu nastane nějaká výjimka, vypíše se informace do konzole <sup>56</sup>.

<sup>52</sup>Obsahuje pouze podpoložku Exit, která ukončí program.

<sup>53</sup>Obsahuje odkaz na okno About (O programu).

<sup>54</sup>Zobrazí seznam sériových portů „COM0“ až „COM9“.

<sup>55</sup>Pouze indikuje, zdali je robot připojen. Pokud ne, je text zašednutý.

<sup>56</sup>Aby se výjimky zobrazily, musí se program spustit zadáním příkazu `java -jar TestRobotCommGUI.jar` do konzole.

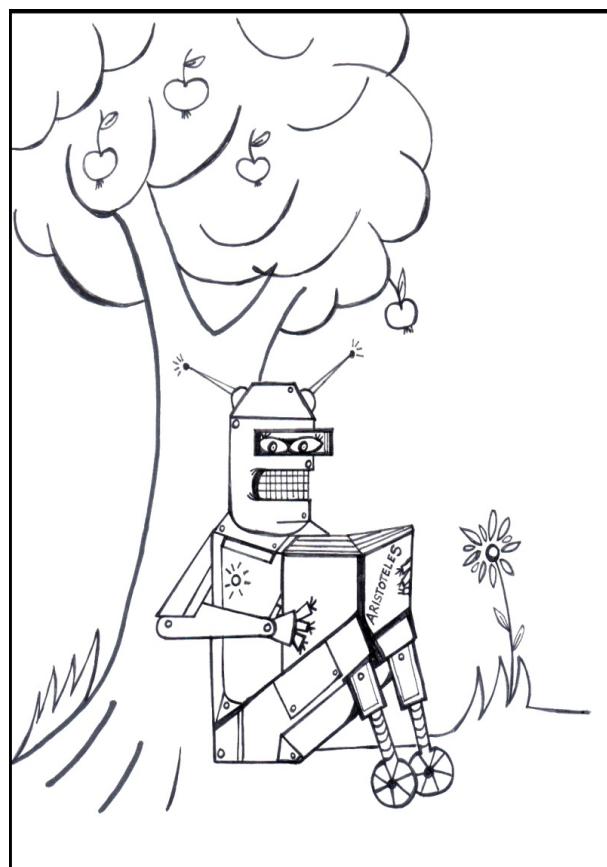
---

---

## KAPITOLA 4

---

### UMĚLÁ INTELIGENCE



## 4.1 Vymezení pojmu umělá inteligence

*Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který - kdyby ho dělal člověk - bychom považovali za projev jeho inteligence.*

*Minsky M. [21]*

*Umělá inteligence je vlastnost člověkem uměle vytvořených systémů vyznačujících se schopností rozpoznávat předměty, jevy a situace, analyzovat vztahy mezi nimi a tak vytvářet vnitřní modely světa, ve kterých tyto systémy existují, a na tomto základě pak přijímat účelná rozhodnutí, za pomoci schopnosti předvídat důsledky těchto rozhodnutí a objevovat nové zákonitosti mezi různými modely nebo jejich skupinami.*

*Kotek Z. a kol. [21]*

*Computational Intelligence is the study of the design of intelligence agents.*

*Poole D. [30]*

*Výpočetní inteligence je studium návrhu inteligenčních agentů.*

*Poole D. [30]*

*The study of mental faculties through the use of computational models.*

*Charniak & McDermott [30]*

*Studium mentálních vloh (duševních schopností) počítačovými modely.*

*Charniak & McDermott [30]*

*The art of creating machines that perform functions that require intelligence when performed by people.*

*Kurzweil R. [30]*

*Umění tvorby strojů, které vykonávají funkce, jež vyžadují inteligenci pokud jsou výkonávány člověkem.*

*Kurzweil R. [30]*

Z definic vyplývá, že hlavním měřítkem určení inteligence je lidská mysl.

Zdroj [34] publikuje zajímavý pokus. Výzkumníci společnosti SONY se zamýšleli na tím, jestli člověk pozná skutečně inteligentní stroj, nebo zdali není sama lidská mysl omezená. V pokusu se testovalo psí<sup>1</sup> chování na psího robota AIBO<sup>2</sup>. Byla testována různá psí plemena, dospělí psi i štěňata. V průběhu pokusu robot nebudil pozornost. Zvířata ho považovala za věc. Ovšem jakmile dostal maskování v podobě psích, nebo kočičích chlupů a pachu, byl dokonce napaden. Zvířata na robota reagovala stejně jako na živého tvora.

Hiroshi Ishiguro, japonský profesor který řídí Laboratoř intelligentních robotů univerzity v Ósace si postavil svého robotického dvojníka. Robot s umělou kůží a strnulým postojem prozrazoval svoji skutečnou podstatu. Hiroshi Ishiguro se nechal slyšet jak svého dvojníka vylepšil a naprogramoval algoritmus, který ovládal a generoval náhodné mikropohyby podobné lidskému tělu.

Robot má v současné verzi vypracovanou mimiku a je téměř k nerozeznání od skutečného lidského těla.

Umělá inteligence nemá přesně stanovenou definici. Ovšem i při počáteční neurčitosti a těžké definovatelnosti vzbuzují neuronové sítě a umělá inteligence velký zájem. Hlavním prvkem umělé inteligence je matematický model mozkové buňky, neuronu. Zájem o neuronové sítě pramení v poznání, že lidský mozek pracuje jiným způsobem než běžné číslicové počítače<sup>3</sup>. Počítače rychle a přesně provádějí posloupnost instrukcí, které pro ně byly formulovány. Lidský mozek je tvořen neurony, které pracují přibližně miliónkrát<sup>4</sup> pomaleji než obvody číslicového systému. Přesto člověk dokáže lépe řešit řadu výpočetně náročných úkolů (zpracování vizuální informace, porozumění řeči, hra šachů, . . . ). Neurovědci mají snahu napodobit chování biologické neuronové sítě živých organismů.

<sup>1</sup>Člověk jako vyspělejší druh, může lépe hodnotit chování jiných druhů. Tak jako by případná mimozemská inteligence mohla lépe hodnotit člověka než on sám sebe.

<sup>2</sup>V pokusu se objevilo i kočičí maskování

<sup>3</sup>Byly nalezeny neuronové buňky, které vysílají v mozků signál pravidelným neustálým způsobem což by se dalo chápat jako biologická obdoba synchronizačního signálu. O biologických neuronech víme, že jejich sítě pracují podobně jako statistický stroj.

<sup>4</sup>Postupem času se zpomalují narůstající rychlostí a výkonností výpočetní techniky.

Tento přístup vede k rozlišení umělé inteligence na takzvanou silnou a slabou.

*Slabé umělé inteligence dosáhneme namodelujeme-li slabé porozumění. Slabé porozumění (Turingovo) chápeme jako porozumění takové že systém na správné vstupní podněty vykáže korespondující reakce. Turingův test dokazuje zda-li jsme dosáhli implementace Slabého porozumění.*

Michal Pěchouček [29]

*Silné umělé inteligence dosáhneme namodelujeme-li silné porozumění. Silné porozumění (Brentanovo) chápeme jako porozumění takové že systém bude disponovat pocitem chápáním takovým jakým disponuje lidská mysl.*

Michal Pěchouček [29]

Slabá umělá inteligence shrnuje algoritmy a postupy, kterými se snažíme napodobit lidské myšlení a postupy. Pokud tímto způsobem navrhujeme stroj, tak se mluví o takzvaném návrhu shora dolů.

Silná umělá inteligence spočívá ve vytvoření umělé bytosti, která sama se naučí porozumět svému prostředí. Tomuto přístupu při návrhu stroje si říká zdola nahoru.

Dnes se používá střední cesta, která využívá silné stránky obou přístupů a kombinuje je. Mluví se pak o „střední umělé inteligenci“. Tento přístup je logickou cestou k využití robustnosti neuronových modelů a rychlosti a přesnosti číslicových systémů.

Neuronové sítě je výhodné používat u procesů a dějů, které nemají přesný algoritmický postup<sup>5</sup> nebo je příliš mnoho možností jak danou úlohu řešit.

Jako příklad uvedu pomyslný stroj<sup>6</sup>, který míchá nebezpečné chemikálie a jeho chod nelze zastavit. Při tomto míchání se odpařují nebezpečné výpary a přítomnost člověka je nežádoucí. Do stroje jsou přiváděny potrubím látky potřebné pro míchání. Toto potrubí je osazeno ventily a je potřeba přívod určitým způsobem regulovat na základě toho jak se míchání vyvíjí. Klasický výpočetní

<sup>5</sup>Pouze přibližný.

<sup>6</sup>Příklad slouží jen jako ilustrace.

algoritmus <sup>7</sup> by musel porovnávat různými měření stav nádrže, množství výparů, barevný vzhled kapaliny, . . . Mnoho vstupů by znemožnilo rychlé řešení úkolu a dílčí problémy jako je vzhled kapaliny jsou s použitím klasického přístupu téměř neřešitelné. Neuronová síť si s problémem poradí mnohem snadněji.

### Příklady použití neuronových sítí

#### *Klasifikátory*

Úkolem je zařadit vstupní data do skupin (tříd), podle vzájemné podobnosti.

#### *Aproximátory funkcí*

Z několika zadaných (naměřených) hodnot je třeba sestavit funkční závislosti.

#### *Navigační systémy*

Například automatické řízení auta. Vstupem je obraz silnice z kamery a laserový obraz vzdálenosti. Výstupem je směr auta, kterým je třeba jet, aby auto nevybočilo ze silnice.

#### *Paměti a rekonstruktory*

Na základě předloženého vstupního obrazu je síť schopna „vybavit si“ odpovídající vstupní obraz (asociační paměť), případně je schopna zrekonstruovat zašumělý vstupní obraz do původní podoby.

#### *Optimalizace*

Úkolem optimalizace je minimalizovat určitou ztrátovou funkci, která je obvykle definována uživatelem (rozvrhování činností, hledání optimální cesty a podobně).

#### *Shlukování a redukce příznaků*

Objevování shluků ve vstupních datech a redukce počtu příznaků. Základní charakteristika těchto sítí je takzvaná samoorganizace.

Robot plnící úkoly a tedy ovlivňující prostředí, ve kterém se nachází, je s tímto prostředí určitým způsobem propojen. Má senzory a na základě dat, které mu senzory dodávají si vytváří nějaký vnitřní model vnějšího prostředí. V tomto prostředí může robot vykonávat akce, které vedou k určitým změnám. Posloupnost akcí určitým způsobem mění prostředí. Pokud robot bude mít za úkol naplnit sklenici vody, bude vytvářet řadu akcí, které povedou ke splnění cíle. Předpokládáme robota, který je schopen se pohybovat v lidském světě, manipulovat s objekty, tento svět měřit a plánovat v něm.

<sup>7</sup>I neuronové sítě jsou výpočetními algoritmy, zde mám ale na mysli klasický přístup měření⇒reakce.

Jeden zajímavý pokus si může doma vyzkoušet každý sám na sobě. Pro člověka není problém splnit úkol daný robotovi (naplnit skleničku vodou). Lidé mají mnoho senzorů na svém těle a mozek zvládné automaticky plánovat vzhledem k prostředí, ve kterém se nachází. Pokud si ale člověk vyřadí zrakový smysl<sup>8</sup>, bude plnění úkolu o něco těžší.

V této chvíli je dobré se zamyslet na několika myšlenkami:

1. jak koresponduje náš vnitřní model světa s vnějším modelem a jak si tento model náš mozek aktualizuje,
2. jaké jsou povolené akce a jaké důsledky budeme muset řešit vzhledem k vykonaným akcím,
3. jaký je aktuální a cílový stav prostředí?

Fyzický robot (humanoid) musí překonávat stejné nástrahy jako člověk. Množina akcí, které robot v tomto prostředí vykonává je téměř nekonečna<sup>9</sup>.

Je spoustu aspektů, nad kterými je třeba se zamýšlet při rozhodování o tom, je-li daný systém inteligentní.

V průběhu formování oboru Umělá inteligence se postupně oddělovaly podoobory, které jsou dnes samostatnými obory a zabývají se výše naznačenými aspekty. Mezi takové patří:

Reprezentace znalostí a dolování znalostí z databází, Znalostní inženýrství, Strojové učení, Plánování, Počítačové vidění, Zpracování přirozeného jazyka, Evoluční výpočetní techniky, Expertní systémy, . . .

Samozřejmě všechny zmíněné obory se navzájem ovlivňují a prolínají. Jeden obohacuje ten druhý a zasahuje také do oborů, které s výpočetní technikou nemají zdánlivě nic společného. Mezi takové patří Psychologie člověka, právě proto, že lidský mozek v dějinách velmi ovlivňoval téma umělé inteligence, ale patří sem i Sociologie, Neurologie, Biologie a hlavně matematika, která dává zmíněným oborům formální aparát. V počítačích se modeluje chování davu nebo psychologie zvířat. Techniky evoluční biologie se aplikují na počítačové modely burzovních systémů nebo předpovědi počasí. Velmi zajímavé jsou pokusy aplikování citů do

<sup>8</sup>Například když večer zhasne světlo.

<sup>9</sup>Omezená konstrukcí robota.

rozhodovacích procesů robotů, podle zásady „cit je rychlejší než rozum“. Nebo simulace mravenčí kolonie, které prohledávají prostředí a hledají například extrémy vícerozměrných funkcí.

## 4.2 Historie umělé inteligence

Počátek oboru Umělá inteligence se spojuje s malou konferencí konanou v létě roku 1956 na Dartmouth College v New Hampshire, na které se objevili přední odborníci zajímající se o mentální schopnosti lidí i strojů. Své zástupce tu měli obory jako matematika, elektrotechnika, lingvistika, neurologie, psychologie a filozofie. Cílem bylo prodiskutovat domněnku, že každé hledisko lidského učení nebo jiný náznak inteligence může být prozkoumán a popsán tak, aby šel sestrojit stroj, který bude danou věc simuloval. Poprvé se objevilo pole společného zájmu a byl definován obsah nového vědního oboru, který se jmenuje díky J. McCarthymu Umělá inteligence.

### Imitační hra a argumenty proti myslícím strojům

K umělé inteligenci už dříve (1950) přispěli Alan Turing a John von Neumann. Alan Turing formuloval takzvaný Turingův test, který vychází z imitační hry. V imitační hře je cílem rozlišit dva lidé podle pohlaví. Pozorovatel, na jehož pohlaví nezáleží, má proti sobě například ženu a muže, který předstírá, že je žena. Trojice lidí sedí v oddelených místnostech a zprostředkovatel mezi nimi přenáší popsané lístečky. Pozoroval musí odhalit kdo je kdo. Turing si položil otázku, zdali může být místo muže počítač a jestli dokáže počítač imitovat ženu stejně dobře jako muž.

Proti této hře a rozlišení inteligence počítače se staví Argument čínského pokoje. Jde o myšlenkový pokus, kdy máme uzavřenou místnost, naplněnou velkým množstvím čínských textů, ve kterých se hypoteticky nalézá každá smysluplná věta tohoto jazyka. Do takového pokoje umístíme člověka, který čínštinu neovládá, ale k dispozici všechny možné slovníky a je schopen v konečném čase najít na základě předaného textu odpověď. Tomuto člověku budeme písemně předávat otázky a ten je schopen pomocí slovníků a čínských textů vytvořit smysluplnou odpověď, kterou předá ven. Vnější pozorovatel se může domnívat, že člověk uv-

nitř pokoje čínštině bez problému rozumí, ale ten ve skutečnosti pouze pracuje se symboly mechanickým způsobem <sup>10</sup>.

Turing nashromáždil řadu argumentů proti uměle myslícím strojům, ale postupně je vyvrátil. Například v původních Turingových poznámkách se uvádí, že počítač by mohl být v testu odhalen právě tím, že nedělá chyby, nebo počítá rychleji než člověk.

Turing také poznamenává, že k tomu, abychom mohli hodnotit, co počítač prožívá, bychom jím také museli sami být.

Fyzik Stephen Hawking přímo dodává, že má smysl posuzovat pouze jevy viditelné navenek. Tedy inteligenci a myšlení, nikoliv kvality, které lze uvidět pouze zevnitř (vědomí, prožívání, …). Podle Hawkinga prostě nevíme, zda počítač má „vědomí“, stejně jako nevíme, zda ho budou mít představitelé nějaké hypotetické mimozemské civilizace. Dokonce to jistě nevíme ani v případě druhých lidí (jistotu máme pouze u sebe).

Do umělé inteligence byly vkládány velké vyhlídky a byly formulovány různé předpovědi. Například, že v roce 1970 bude počítač velmistrem v šachu <sup>11</sup> nebo bude schopen komponovat hudbu na úrovni klasiků.

V roce 1951 stál Marvin Minsky u zrodu prvního neuropočítače Snark. Z technického hlediska byl úspěšný, dokázala automaticky adaptovat váhy <sup>12</sup>, ale nebyl nikdy použit k řešení nějakého praktického problému.

V roce 1957 byl F. Rosenblattem vyvinut matematický model neuronové buňky, perceptron. Tento model byl schopen simulovat nervová spojení živých organismů a položil základ výzkumu v oblasti klasifikace a rozpoznávání <sup>13</sup>. Výzkum perceptronů a modelů jednoduchých neuronových sítí vedly ke studiu prvotních algoritmů adaptace a učení.

Další neuropočítač Mark I Perceptron byl dílem F. Rosenblatta a Charlese Wightmana. Vznikl v letech 1957 až 1958. Mark I Perceptron byl navržen pro

<sup>10</sup>Tuto práci může zastat stroj.

<sup>11</sup>Stalo se tak až 10. února 1996 kdy Garry Kasparov změřil svoje síly s počítačem Deep Blue od společnosti IBM

<sup>12</sup>Šířka spoje mezi neurony. Čím vyšší váha, tím větší vliv má daný spoj.

<sup>13</sup>Automatické rozpoznávání objektů, jevů a situací do tříd

rozpoznávání znaků promítaných na světelnou tabuly. Jeho váhy byly realizovány potenciometry, které byly připevněny k samostatným motorům. Motory byly řízeny učícím algoritmem a ten upravoval váhy neuronové sítě. Při každé prezentaci bylo použito náhodných hodnot, aby byla demonstrována schopnost učícího algoritmu.

Výzkum neuronových sítí byl později v 70. letech silně utlumen vlivem kritiky M. Minského a S. Paperta. Jejich argumentace spočívala v neschopnosti neuronu řešit logickou funkci XOR. Řešením bylo použití několika neuronových buněk v neuronové síti, ale tehdy pro takovéto sítě nebyl znám žádný učící algoritmus.

Po deseti letech nastala vlna restaurace tohoto oboru. Díky J. J. Hopfieldovi, D. E. Rumelhartovi, T. Kohonenu a dalších byly postupně vyvráceny námitky M. Minskeho. Byly vyvinuty nové algoritmy a postupy a neuronové sítě započali téměř raketový rozvoj.

Rozsáhlejší historické poznámky jsou k dispozici v [21] a [20].

### Časová osa

1941 - Donald Hebb 1. zákon účení NS<sup>14</sup>.

1943 - McCulloch-Pittsův model neuronu.

50. léta - První neuropočítáče.

1957 - Frank Rosenblatt - perceptron.

Začátek 60. let - ADALINE<sup>15</sup> (Adaptive Linear Neuron), později rozšířená na MADALINE (Many ADALINE). Widrow-Hoffovo pravidlo učení (Bernard Widrow a Marcian Edward Hoff).

Polovina 60. let až polovina 80. let - Pokles zájmu o neuronové sítě (někteří nadšenci přesto pokračují ve výzkumu). Teuvo Kohonen - samoorganizující se sítě. Stephen Grossberg - grossbergovy zákony učení.

Od poloviny 80. let až dosud - Renesance neuronových sítí.

John Hopfield (1984) - rekurentní neuronové sítě.

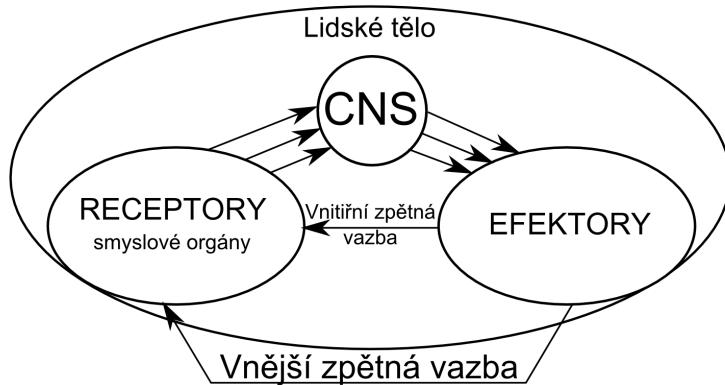
Další druhy vícevrstvých nelineárních sítí a nová pravidla jejich učení.

---

<sup>14</sup>NS - neuronové sítě

<sup>15</sup>Model neuronu, jehož aktivační funkce je lineární. Nadrovina rozdělující vstupní prostor na 2 podprostory. O aktivační funkci se práce zmiňuje v [4.4.1].

Obrázek 4.1: Struktura nervového systému - zpětnovazební systém řízení



## 4.3 Neuron

### 4.3.1 Centrální nervová soustava

Lidský mozek se skládá z přibližně  $10^{11}$  výpočetních elementů nazývaných neurony, které spolu komunikují prostřednictvím sítě vazeb mezi jejich vstupy a výstupy. Hlavním přenosovým médiem informace je elektrický signál v chemickém prostředí.

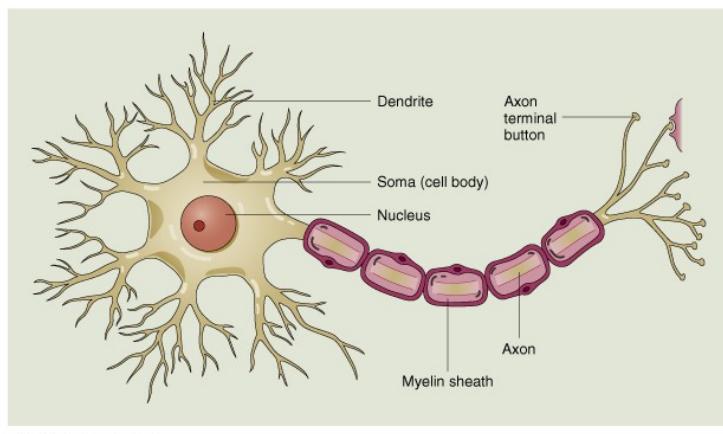
Vstup do sítě je umožněn smyslovými receptory, které získávají podněty jak z vnitřku těla, tak prostřednictvím smyslových orgánů (zrak, sluch, ...) z vnějšího prostředí.

Podnět je ve formě elektrických impulsů přenášen z receptorů do CNS<sup>16</sup>, kde je zpracován. Podle výsledků zpracování jsou řízeny efektory<sup>17</sup> a odezva člověka na daný podnět se projeví ve formě rozmanitých akcí [4.1]. Mozek lze chápat jako biologickou hustě propojenou elektrickou síť, jejíž činnost je ovlivněna biochemickými procesy.

<sup>16</sup>CNS - centrální nervová soustava

<sup>17</sup>Vykonavatel (například sval je vykonavatelem plánovaného pohybu), realizátor.

Obrázek 4.2: Neuronová buňka



© 2000 John Wiley & Sons, Inc.

### 4.3.2 Biologický neuron

Neuron jako základní jednotku nervové tkáně popsal roku 1835 J. E. Purkyně. Někdy je tento objev připisován Španělovi Cajalovi.

Biologický neuron je základní buňkou CNS.

Obrázek [4.2] schématický znázorňuje tělo neuronové buňky.

Dendity (Dendrite) jsou vstupy neuronové buňky, na které je přiveden signál z buňky jiné. Podle síly vazby mezi těmito neurony má vysílající neuron vliv na signál přijímajícího neuronu. Podle vlivů, které na přijímající neuron působí předá signál dalším neuronům, nebo jej utlumí <sup>18</sup>.

Do těla neuronu (Soma (cell body)) vstupují dendity, které jsou dlouhé asi 3 milimetry, a z jeho těla vystupuje až 60 centimetrů <sup>19</sup> dlouhý výstup axon. Tělo neuronu obsahuje jádro (Nucleus) buňky a jeho velikost se pohybuje od  $6 \mu m$  (malé buňky kůry mozečku) do  $100 \mu m$  (velké pyramidové neurony motorických oblastí mozkové kůry).

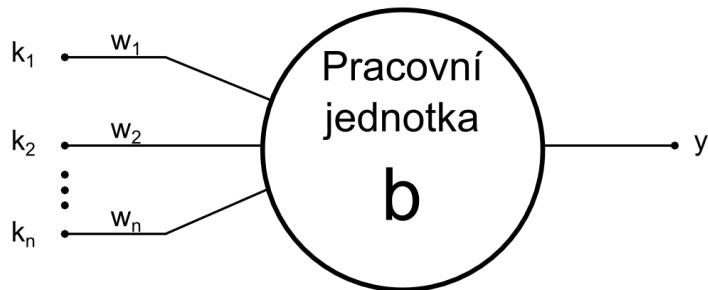
Terminály axonu jsou konektory tvořící rozhraní k připojení se na dendity jiných neuronů. Takovému propojení se říká synapse a je jich asi  $10^4$  na každý neuron. Rozhraní funguje jako jednosměrné brány ve směru axon $\Rightarrow$ dendrit.

Neuron vyšle signál dalším neuronům tehdy, překročí-li hodnota vstupního

<sup>18</sup>Rozlišujeme takzvaný inhibiční (potlačující) a excitační (vybuzující) charakter.

<sup>19</sup>Nejdelší neuron v lidské těle je 1 metr dlouhý a vede od páteře až po konečky prstů na nohou.

Obrázek 4.3: McCulloch-Pittsův model neuronu



signálu hodnotu tlumícího signálu o určitou velikost <sup>20</sup>.

Po vygenerování signálu se neuron na určitou dobu dostane do stavu, kdy nereaguje na žádný vstupní signál. Tento stav nastává u každého neuronu v jinou dobu a délka takzvaného období pauzy se u jednotlivých neuronů liší. Na rozdíl od číslicových počítačů pracují neurony asynchronně.

## 4.4 Matematický model neuronu

**M**atematická reprezentace neuronu vychází z podstaty šíření elektrického signálu u neuronových sítí. Excitační signály jsou nahrazeny kladnými čísly a inhibiční signály čísly zápornými. Aktivace neuronu nastává tehdy, překročí-li součet vstupních signálů určitou prahovou hodnotu.

Existuje celá řada modelů neuronů. Záleží jakých vlastností si přejeme dosáhnout. Existují například speciálně navržené neurony pro zpracování obrazu podle modelů neuronů v lidském oku.

Toky signálů jsou jednosměrné a jsou v modelech znázorněny šipkou.

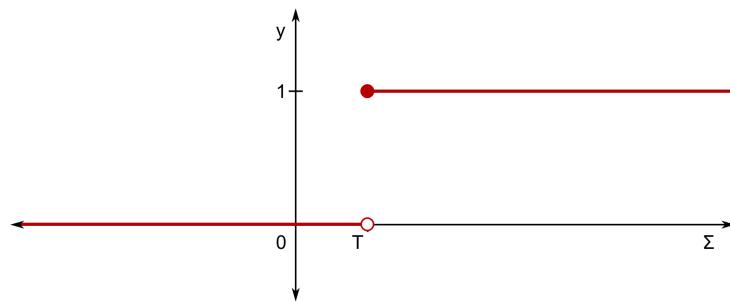
### 4.4.1 McCulloch-Pittsův model neuronu

$k_i; i = 1, 2, \dots, n$  jsou vstupy nabývající pouze hodnot 1 nebo 0 podle toho, zdali je nebo není přítomen vstupní signál.

$w_i; i = 1, 2, \dots, n$  jsou váhy, přičemž  $w_i = +1$  je budící (excitační) signál a

---

<sup>20</sup>Přibližně 40 milivoltů.

Obrázek 4.4: Závislost výstupu na vstupních hodnotách a prahu  $T$ 

$w_i = -1$  je tlumící (inhibiční) signál.

$n$  je počet vstupů.

$y$  je výstupní signál.

$T$  je hodnota prahu, kterou musí součet vstupních signálů překročit, aby byl neuron aktivován.

### Aktivační pravidlo

$$y(k+1) = \begin{cases} 1 & \Leftrightarrow \sum_{i=1}^n w_i \cdot k_i \geq T \\ 0 & \Leftrightarrow \sum_{i=1}^n w_i \cdot k_i < T \end{cases} \quad (4.1)$$

Aktivační pravidlo definuje kdy neuron bude vysílat signál a kdy jej bude tlumit [4.4]. V závislosti na nastavení prahové hodnoty se výstup neuronu rovná výsledku 0, nebo 1 (binární model). Prahová hodnota je pevně nastavena.

#### 4.4.2 Perceptron

$k_i; i = 1, 2, \dots, n$  jsou vstupy nabývající hodnot z množiny reálných čísel  $\mathbb{R}$ .

$w_i; i = 1, 2, \dots, n$  jsou váhy také z množiny  $\mathbb{R}$ .

$w_0$  je práh neuronu<sup>21</sup>, který se řadí mezi vstupy.  $x_0 = 1$  je nultý vstup neuronu.

$n$  je počet vstupů.

$y$  je výstupní signál.

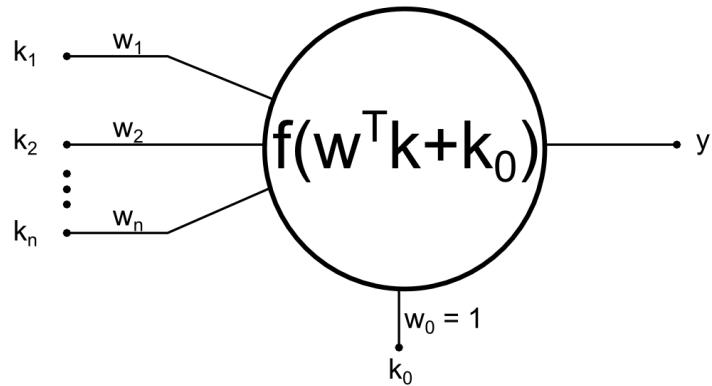
$f()$  je aktivační funkce neuronu [4.8][4.4][4.5][4.6][4.7].

$\xi = w^T \cdot k + k_0 = [\sum_{i=1}^n w_i \cdot k_i] + k_0$  je aktivační hodnota, potenciál.

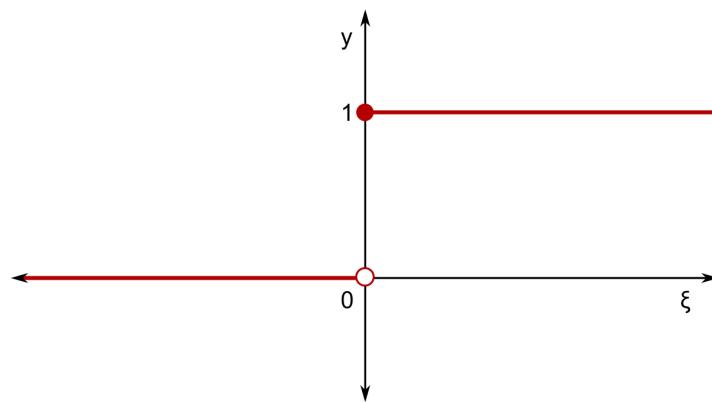
---

<sup>21</sup>bias

Obrázek 4.5: Model perceptronu



Obrázek 4.6: Ostrá nelinearity

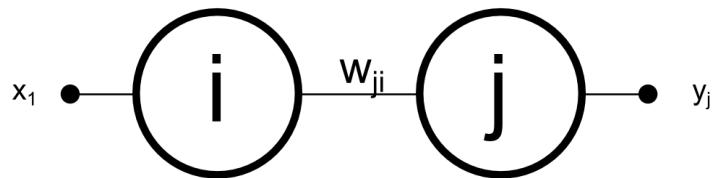


Tento model nám nabízí si zvolit libovolnou aktivační funkci. Výstup neuronu můžeme definovat podobně jako v případě McCulloch-Pittsova modelu.

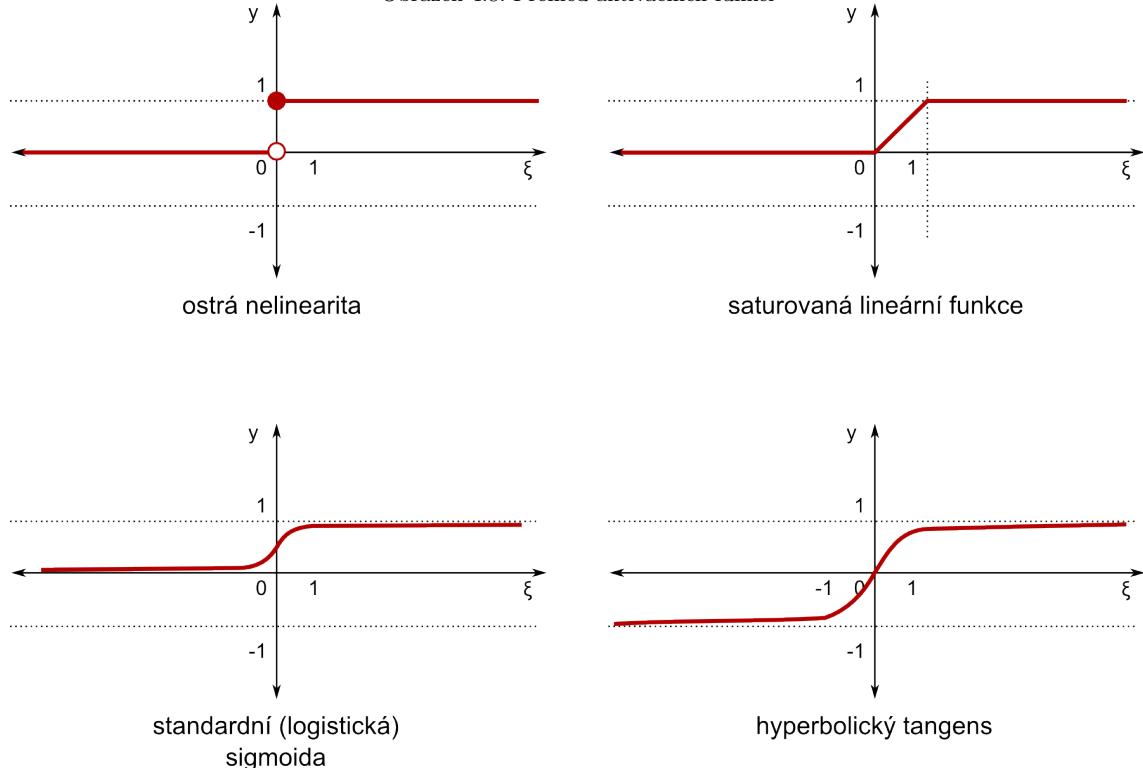
$$y(k+1) = \begin{cases} 1 & \Leftrightarrow \xi \geq 0 \\ 0 & \Leftrightarrow \xi < 0 \end{cases} \quad (4.2)$$

Takové aktivační funkci se říká ostrá nelinearity a má graf [4.6].

Obrázek 4.7: Synaptická váha mezi neurony



Obrázek 4.8: Přehled aktivačních funkcí



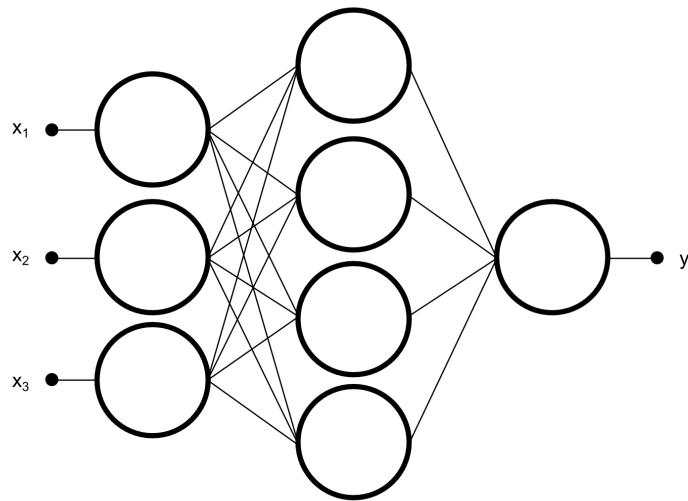
#### 4.4.3 ADALINE

Tento model neuronu je velmi podobný perceptronu. S tím rozdílem, že je odstraněna aktivační funkce a výstup neuronu  $j$  je dán vzorcem:

$$y_j = \sum_{i=0}^n w_{ji} \cdot x_i \quad j = 1, 2, \dots, m. \quad (4.3)$$

$w_{ji}$  je váha synapse mezi výstupním neuronem  $i$  a vstupním neuronem  $j$  [4.7].

Obrázek 4.9: Neuronová síť



$$f(\xi) = \begin{cases} 1 & \Leftrightarrow \xi \geq 0 \\ 0 & \Leftrightarrow \xi < 0 \end{cases} \text{ ostrá nelinearity} \quad (4.4)$$

$$f(\xi) = \begin{cases} 1 & \Leftrightarrow \xi > 0 \\ \xi & \Leftrightarrow 0 \leq \xi \leq 1 \\ 0 & \Leftrightarrow \xi < 0 \end{cases} \text{ saturovaná lineární funkce} \quad (4.5)$$

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \text{ standardní (logistická) sigmoida} \quad (4.6)$$

$$f(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}} \text{ hyperbolický tangens} \quad (4.7)$$

## 4.5 Neuronová síť

**N**euronová síť je zřetězení několika vrstev neuronových řad [4.9]. Mezi jednotlivými řadami (vrstvami) jsou neurony propojeny style „každý s každým“ a jednotlivým řadám se říká vrstvy. Většinou je známá pouze vstupní a výstupní

vrstva. Ostatní vrstvy jsou skryté <sup>22</sup>. Počet neuronů a jejich synaptické vazby určují „architekuru, topologii“ neuronové sítě.

Jak informace putuje postupně <sup>23</sup> sítí od vstupu k výstupu, tak se jednotlivé neurony dostávají do určitého stavu. Stavy všech tří neuronů definují stav neuronové sítě a synaptické váhy určují konfiguraci neuronové sítě.

V průběhu času se neuronová síť mění. Například se adaptují váhy, mění se počet neuronů a jejich propojení. Proto se zavádí pojem „dynamika“ neuronové sítě a rozděluje se do tří druhů.

#### 4.5.1 Organizační dynamika

Tato dynamika určitě architekturu sítě, uspořádání neuronů nebo jejich vzájemné propojení (do sítě může být nějaký neuron přidán). Při výpočtu výstupu sítě je organizační dynamika neměnná. Její změna se aplikuje při adaptaci.

Rozlišujeme architekturu cyklickou (rekurentní) a acyklickou (dopředná) [4.10].

#### 4.5.2 Aktivní dynamika

Aktivní dynamika sítě specifikuje počáteční stav sítě a způsob jeho změny v diskrétním čase <sup>24</sup> při pevné topologii a konfiguraci. V aktivním režimu se nastaví vstup sítě a poté se postupně vypočítávají hodnoty neuronů postupně v každé vrstvě a vypočtené hodnoty jsou poté předány jako vstupy následující vrstvy.

V každém časovém kroku je vybrán jeden neuron, nebo více neuronů, které aktualizují svůj stav a výstup na základě svým vstupů. Rozlišujeme pak sekvenční a paralelní výpočet.

Výpočet jednotlivých neuronů může být řízen centrálně, nebo nezávisle na sobě. Pak se jedná o synchronní, nebo asynchronní model neuronové sítě.

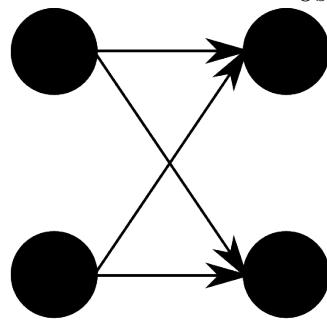
---

<sup>22</sup>Některé algoritmy učení přidávají do sítě neurony, takže počet neuronů ve vnitřní vrstvě není vždycky znám.

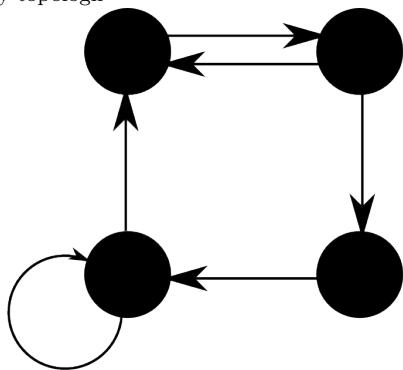
<sup>23</sup>V diskrétním čase.

<sup>24</sup>Kdy  $t = 0, 1, 2, \dots$

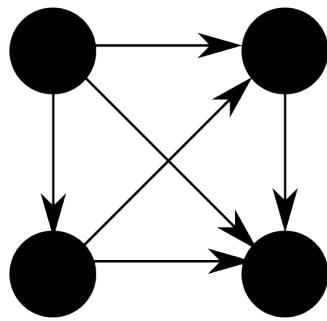
Obrázek 4.10: Příklady topologií



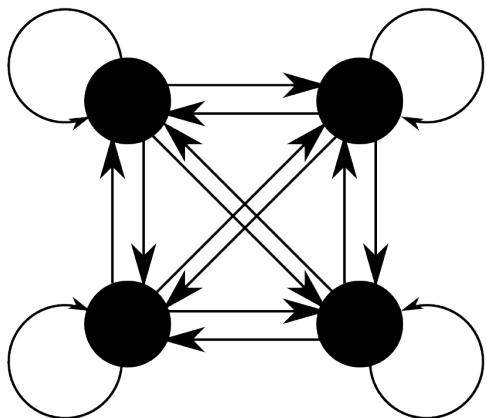
Acyklická (dopředná) topologie



Cyklická (rekurentní) topologie



Dopředná topologie



Úplná topologie

### 4.5.3 Adaptivní dynamika

Tato dynamika zahrnuje práci s váhovým prostorem<sup>25</sup>. Na váhovém prostoru je závislý výpočet neuronové sítě, v tomto prostoru je uložena paměť sítě (to co umí a co se naučila).

Na začátku adaptivního režimu, kdy se neuronová síť učí, jsou všechny váhy nastaveny blízko nule<sup>26</sup>. Učení neuronové sítě probíhá tak, že na vstup přivedeme vstupní vektor<sup>27</sup>, necháme sít na základě tohoto vstupu vypočítat výstup a porovnáme jej s požadovaných výstupem. Pokud se oba výstupy rovnají, přejdeme k dalšímu vstupu, pokud se nerovnají, upravíme učícím algoritmem váhový prostor tak, aby si byly výstupy rovny. Poté přejdeme k dalšímu vstupu.

Adaptivní dynamika potřebuje pro svoji aplikaci množinu vstupních a výstupních vektorů. Takové množině říkáme tréninková množina  $\tau$  a vypadá takto

$$\tau = \left\{ (x_k, d_k) \quad \begin{array}{l} x_k = (x_{k1}, \dots, x_{kn}) \in \mathbb{R}^n \\ d_k = (d_{k1}, \dots, d_{km}) \in \{0, 1\}^m \end{array} \quad k = 1, \dots, p \right\}. \quad (4.8)$$

Uvedená tréninková množina je určena pro perceptronový model neuronu.  $k$  ukrývá hodnotu tréninkového vzoru, na který se aktuálně síť adaptuje. Celkem jich je  $p$ .

Množina znázorňuje soubor hodnot v páru vstup-výsledek. Vstup je v množině  $\tau$  označen jako množina  $x_k$  a požadovaný výstup jako  $d_k$ .

Například máme neuronovou síť, kterou chceme adaptovat na provádění operace logického součinu AND [4.1]. Pro tento účel máme následující množinu  $\tau$  s  $p = 4$  vzory.

$$x = ((1, 1), (1, 0), (0, 0), (0, 1))$$

$$d = ((1), (0), (0), (0))$$

Takže  $\tau = \{((1, 1), (1)), ((1, 0), (0)), ((0, 0), (0)), ((0, 1), (0))\}$ .

---

<sup>25</sup>Takto označujeme množinu vah všech synaptických spojů.

<sup>26</sup>Většinou náhodně

<sup>27</sup> $(x_1, \dots, x_n)$

Tabulka 4.1: Logický součin AND

1	1	1
1	0	0
0	0	0
0	1	0

## 4.6 Metody učení neuronových sítí

### 4.6.1 Zatřesení

Zatřesení je metoda popisující stochastický přístup<sup>28</sup> k určování prostoru vah neuronové sítě. Princip spočívá v přičítání náhodných čísel k synaptickým vahám. Na toto číslo jsou kladený nějaké omezující podmínky. Náhodné číslo bývá velmi malé mezi nulou a jedničkou.

Mohli bychom tuto metodu přirovnat k ohrazené čtvercové podložce s důlky pro kuličky a stejný počet kuliček na této podložce umístěných. Každá kulička vlastní svůj důlek, do kterého se musí dostat. Pokud budou všechny kuličky ve správném důlku, je úloha vyřešena nejlepším možným způsobem.

Když touto podložkou zatřeseme, můžou kuličky spadnou do správného důlku.

Metoda nemá příliš dobré výsledky, ale často se kombinuje s jinými metodami v případě, že se neuronová síť nemůže zdokonalit, ale má k tomu ještě dostatek prostoru. Například když nějaká kulička vpade do důlku, který ji nepatří, musíme „zatřesat“ více, aby vylezla.

### 4.6.2 Simulované žíhání

Principem se metoda inspirovala myšlenkou uměle dodávaného tepla při chlazení. Pokud bychom chtěli získat z křemene sklo, musíme křemen roztavit a vhodně ochlazovat. Pokud ochlazujeme příliš rychle, vzniká v budoucím skle mnoho strukturálních nečistot. Pokud ochlazujeme příliš pomalu, trvá proces příliš dlouho. Proto se volí střední cesta, kdy roztavený křemen chladíme a pokud se začnou objevovat nečistoty, křemen zahřejeme a pokračujeme dál v ochlazování

---

<sup>28</sup>S prvkem náhody

[21].

V porovnání s kuličkami na podložce zatřeseme podložkou, zjistíme jaký je stav po zatřesení a pokud je lepší, v dalším kroku zatřeseme s nižší intenzitou. Pokud lepší není, vrátíme se k předchozímu stavu a intenzitu třesu nesnižujeme.

#### 4.6.3 Metoda genetických algoritmů

Metoda se opírá o model biologické evoluce. Zahrnuje pojmy jako *jedinec*, *potomek*, *populace*, *křížení* a další.

Pokud bychom chtěli tuto metodu použít, musíme definovat operace a prvky s kterými budeme operovat. Pro neuronové sítě platí, že jedince bude zastupovat váhový prostor  $w$  neuronové sítě. Váhový prostor bude genetickým kódem jedince.

Populace je pak množina těchto jedinců. Takže například nějaký seznam náhodně vygenerovaných váhových prostorů  $w_1, w_2, \dots$ .

Křížení mezi jedinci je definováno jako náhodné promíchání vah jedinců  $w_x$  a  $w_y$ .

Mutace je náhodná změna při křížení.

Fitness funkce je metoda podobná přirozenému výběru selekce. V užití evo lučních metod se například ze dvou jedinců vybere ten, který má menší chybu sítě.

Na začátku vytvoříme třeba 1 000 náhodných jedinců, kteří budou tvořit počáteční populaci. Aplikujeme na ně metody křížení a necháme populaci narůst například do 1 000 000 jedinců. Poté pomocí Fitness funkce vybereme 1 000 nejlepších a na tyto vybrané aplikujeme stejný postup jako na ty první.

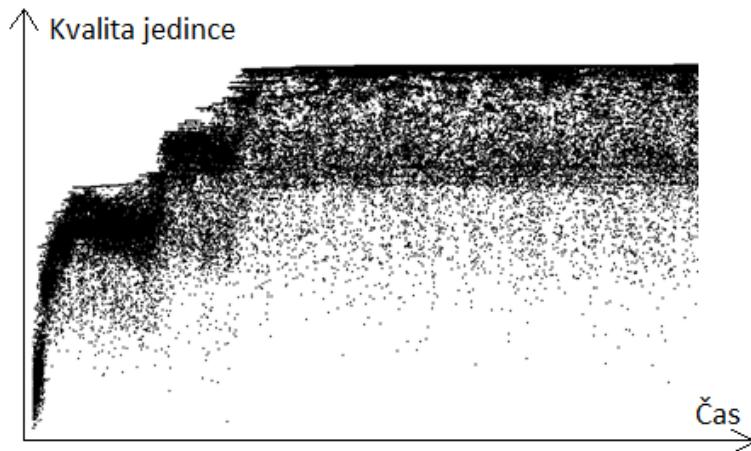
Graf [4.11] ukazuje zvyšování kvality jedinců v populaci v závislosti na čase. Počet jedinců v populaci je počet černých teček ve svislém řezu grafu. Je vidět že postupem času se kvalita populace zvyšuje. Většinou se celá populace vyvíjí v jakýchsi skocích<sup>29</sup>.

Pokud se populace dál nevyvíjí, neznamená to, že nemůže, ale v populaci nelze

---

<sup>29</sup>Na grafu jsou vidět 3 schodky.

Obrázek 4.11: Vztah kvality jedinců v populaci na čase [5]



nalézt lepší řešení. V takovém případě, můžeme na pár jedincích aplikovat metodu mutace v „mírně“ větším měřítku. Nebo můžeme použít simulované žíhání či jinou techniku a populaci tak stimulovat.

#### 4.6.4 Hebbovo pravidlo

Nejstarší metoda, která se dodnes používá v základních aplikacích. Je dokázáno [20], že pokud se síť může naučit data předložená tréninkovou množinou, tak Hebbovův vzorec vede po určitém čase k řešení.

Vzorec, který Donald Hebb navrhl je následující:

$$w_{ji}^t = w_{ji}^{t-1} + \varepsilon \cdot x_{ki} \cdot (d_{kj} - y_j(w^{t-1}, x_k^{t-1})).$$

Index  $t$  ve vzorci značí předchozí čas<sup>30</sup>.

$w_{ji}^{t-1}$  je tedy stav synaptické váhy propojení z neuronu  $i$  do neuronu  $j$  [4.2] v předchozím čase.

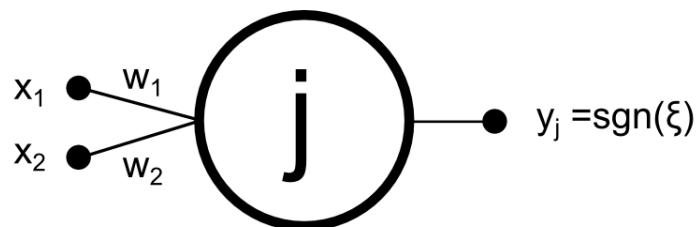
$w_{ji}^t$  je nová synaptická váha pro stejnou synapsi.

$0 < \varepsilon \leq 1$  je míra vlivu vzorů na adaptaci, neboli motivace (learning rate).

$x_{ki}$  je vstup  $i$  v  $k$  tréninkovém vzoru.  $d_{kj} - y_j(w^{t-1}, x_k^{t-1})$  je chyba  $j$  neuronu v předchozím čase vypočítaná podle [4.8], někdy se značí  $\Delta^{t-1}$  a dosazuje se za tuto hodnotu chyba celé sítě v čase  $t - 1$ .

<sup>30</sup>Neuronová síť se učí v diskrétních časových krocích.

Obrázek 4.12: Model perceptronu



Obrázek 4.13: Tabulka učení perceptronu Hebbovým pravidel

t	$x_1$	$x_2$	$o$	$w_1$	$w_2$	$w_0$	$\xi$	$f(\xi)$	$\Delta$
1	1	1	1	0,2	0,3	0,1	0,6	1	0
2	0	1	0	0,2	0,3	0,1	0,4	1	-1
3	1	0	0	0,2	-0,2	-0,4	-0,2	0	0
4	0	0	0	0,2	-0,2	-0,4	-0,4	0	0
5	1	1	1	0,2	-0,2	-0,4	-0,4	0	1
6	0	1	0	0,7	0,3	0,1	0,4	1	-1
7	1	0	0	0,7	-0,2	-0,4	0,3	1	-1
8	0	0	0	0,2	-0,2	-0,9	-0,9	0	0
9	1	1	1	0,2	-0,2	-0,9	-0,9	0	1
10	0	1	0	0,7	0,3	-0,4	-0,1	0	0
11	1	0	0	0,7	0,3	-0,4	0,3	1	-1
12	0	0	0	0,2	0,3	-0,9	-0,9	0	0
13	1	1	1	0,2	0,3	-0,9	-0,4	0	1
14	0	1	0	0,7	0,8	-0,4	0,4	1	-1
15	1	0	0	0,7	0,3	-0,9	-0,2	0	0
16	0	0	0	0,7	0,3	-0,9	-0,9	0	0
17	1	1	1	0,7	0,3	-0,9	0,1	1	0
18	0	1	0	0,7	0,3	-0,9	-0,6	0	0
19	1	0	0	0,7	0,3	-0,9	-0,2	0	0
20	0	0	0	0,7	0,3	-0,9	-0,9	0	0

Naučíme pomocí tohoto vzorce neuron počítat logickou funkci AND [4.1]. K tabulce je přiložen pro pomoc představivosti doprovodný obrázek [4.12]

Tabulka [4.13] ukazuje průběh učení v diskrétním čase. V čase  $t = 1$  nastavíme váhy neuronu  $j$  náhodně blízko nuly. Jako aktivační funkci jsme použili ostrou nelinearitu [4.4][4.8]. A požadovaný výstup je ve sloupci značeném jako  $o$ .

Vzorec uvedený dříve si napíšeme v čitelnější formě. Změna nastane u zápisu chyby v čase  $t - 1$ , použijeme kratší zápis a také nebudeme psát  $x_i k$  jako  $k$  tréninkový vstup  $i$ , ale pouze jako předchozí  $i$  vstup  $x_i^{t-1}$ . Neuron chceme naučit přesnou interpretaci logické funkce AND a protože při výpočtu vah v čase  $t$  požadujeme hodnoty dřívějšího času, není třeba rozlišovat tréninkový vzor, protože pokud dřívější hodnota vstupu byla 0<sup>31</sup>, nebude mít chyba sítě na nastavení váhy v aktuálním čase vliv.

$$w_i^t = w_i^{t-1} + \varepsilon \cdot x_i^{t-1} \cdot \Delta^{t-1}$$

Nejdříve vypočítáme podle náhodných vah výstup sítě v čase  $t = 1$  [4.6.4]. Nastavíme koeficient učení  $\varepsilon$  na pevnou hodnotu 0,5<sup>32</sup>.

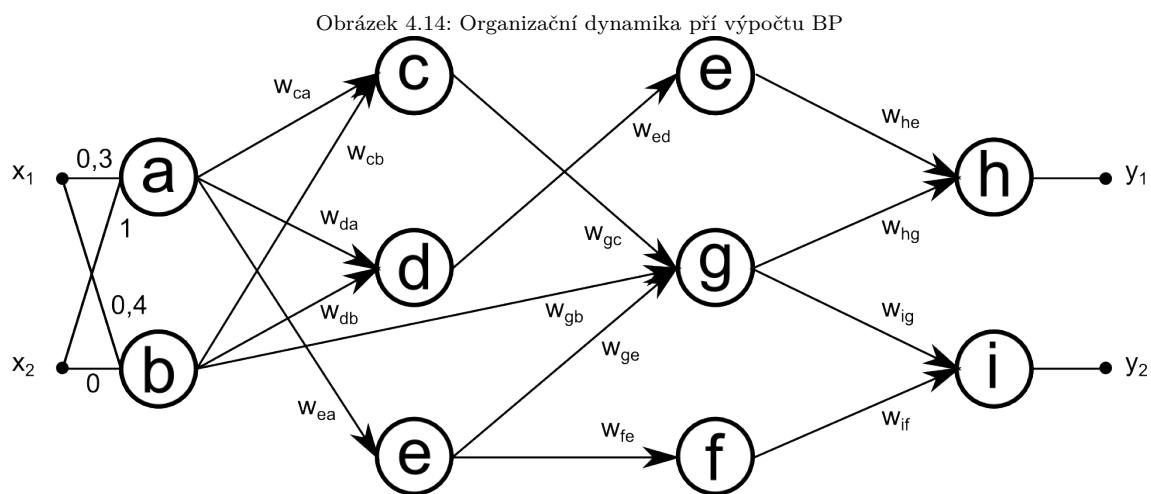
$$\begin{aligned} \xi &= \sum_{i=0}^2 x_i \cdot w_i = x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 = \\ &= 1 \cdot 0,1 + 1 \cdot 0,2 + 1 \cdot 0,3 = 0,6 \\ f(\xi) &= 1 \\ \Delta &= 0 \end{aligned}$$

Pro výpočet v druhém kroku ( $t = 2$ ) dosadíme do vzorce [4.6.4] známé údaje z diskrétního období  $t = 1$  [4.6.4].

$$\begin{aligned} w_i^2 &= w_i^{2-1} + \varepsilon \cdot x_i^{2-1} \cdot \Delta^{2-1} \\ w_i^2 &= w_i^1 + 0,5 \cdot x_i^1 \cdot 0^1 \\ w_i^2 &= w_i^1 \\ w_0^2 &= w_0^1 = 0,1 \\ w_1^2 &= w_1^1 = 0,2 \\ w_2^2 &= w_2^1 = 0,3 \end{aligned}$$

<sup>31</sup>Logická funkce má pouze vstup z množiny  $\{0, 1\}$ .

<sup>32</sup>Spíše se využívá koeficient učení podle vztahu  $\varepsilon^t = \frac{\kappa}{t}$ , kde  $\kappa$  je relativně malá konstantní hodnota.



#### 4.6.5 Back-propagation

Abych mohl ukázat použití metody BP<sup>33</sup>, definuji organizační dynamiku jako na obrázku [4.14], takže postavení neuronů v síti je 2-3-3-2 <sup>34</sup>.

Pro aktivní dynamiku (forward-propagation) bude platit, že všechny neurony budou mít stejnou aktivační funkci a to standardní logistickou sigmoidu [4.6]. Aktivační funkci budu značit  $f$ .

$$\begin{aligned}f(a) &= f(x_1 \cdot 0,3 + x_2 \cdot 1) \\f(b) &= f(x_1 \cdot 0,4 + x_2 \cdot 0) \\f(d) &= f(f(a) \cdot w_{da} + f(b) \cdot w_{db})\end{aligned}$$

Abychom mohli adaptovat síť, musíme nejdříve určit chybu sítě. Tu vypočítáme podle vzorce  $E = \sum_{k \in p} E^k = \frac{1}{2} \sum_{k \in p} (y^p - d^p)^2$  <sup>35</sup>, kde  $y^p$  je vypočítaný výstup a  $d^p$  je požadovaný výstup podle tréninkového vzoru  $p$ .

Cílem adaptace je minimalizace chyby sítě ve váhovém prostoru. Vzhledem k tomu, že chyba sítě přímo závisí na komplikované funkci, jedná se o komplikovaný optimalizační problém. Pro jeho řešení se v základním modelu používá nejjednodušší varianta gradientní metody. Gradientní metoda je založena na faktu, že po-

<sup>33</sup>back-propagation

<sup>34</sup>2 neurony ve vstupní vrstvě, 3 a 3 ve vrstvách skrytých a 2 neurony výstupní.

<sup>35</sup>Metoda nejmenších čtverců. Pro bližší seznámení s metodou doporučují navštívit internetové stránky [wood.mendelu.cz/math/maw-html](http://wood.mendelu.cz/math/maw-html)

mocí derivace funkce v jejím libovolném bodě zjistíme, jestli funkce roste, klesá, nebo je rovnoběžná s hlavní osou. Pokud budeme gradientní metodou adaptovat váhu  $w$  jednoho neuronu, který bude mít jeden vstup  $x = 1$  a výstup  $y$ , bude realizovat váhový prostor standardní sigmoidu [4.6] <sup>36</sup>.

V tomto váhovém prostoru je cílem najít takovou pozici, která generuje nejménší chybu. Dejme tomu, že  $w = 0,5$  a výstup neuronu má být pro  $x = 1$   $y = 0,75$ . Ale při  $w = 0,5$  je  $y \doteq 0,6225$ . Odečteme vypočítaný výstup od požadovaného a vypočítáme chybu sítě jako  $\delta_y = 0,6225 - 0,75 = -0,1275$ . Touto chybou jednoduše vynásobíme vektor funkce v bodě našeho aktuálního výskytu a tento výsledek ještě vynásobíme koeficientem učením  $\varepsilon$  a tento výsledek přičteme k aktuální váze. Tím se posuneme o kousek blíž k místu, kde je chyba menší a tak postupně se budeme posouvat až do místa, kde bude chyba minimální nebo nulová.

Protože podle vektoru a aktuální pozice můžeme určit, kterým směrem je minimum funkce a na jak strmém „kopec“ stojíme, pustíme se dolu ve směru vektoru. Tento vektor násobíme právě hodnotou  $\xi$  (learning rate), která nás posouvá o kousíček blíž k minimu funkce. Pokud tento problém řešíme pro rozsáhlejší neuronovou síť, je váhová krajina velmi různorodá a najít nejnižší minimum není jednoduché.

Proto pro zpětné šíření chyby (back-propagation) potřebujeme spočítat derivaci funkce  $f$ . Požadavkem takzvaných gradientních metod je diferencovatelnost funkce.

$$\begin{aligned} f' &= \frac{d}{d\xi} (1 + e^{-\xi})^{-1} = \frac{-(1 + e^{-\xi})'}{(1 + e^{-\xi})^2} = \frac{e^{-\xi}}{(1 + e^{-\xi})^2} = \\ &= \frac{1}{1 + e^{-\xi}} \cdot \frac{e^{-\xi}}{1 + e^{-\xi}} = \\ &= \frac{1}{1 + e^{-\xi}} \cdot \left[ 1 - \frac{1}{1 + e^{-\xi}} \right] = f \cdot [1 - f] \end{aligned}$$

---

<sup>36</sup>Obecně svoji aktivační funkci. Ale u metody BP jsme definovali jako aktivační funkci právě standardní logistickou sigmoidu.

$$\left| 1 - \frac{1}{1 + e^{-\xi}} = \frac{1 + e^{-\xi}}{1 + e^{-\xi}} - \frac{1}{1 + e^{-\xi}} = \frac{e^{-\xi}}{1 + e^{-\xi}} \right|$$

Při zpětném šíření chyby postupujeme stejně jako při počítání výstupu sítě, jen změníme směr a budeme postupovat od výstupu ke vstupu.

Pro výpočet chyby sítě na výstupním neuronu  $y_1$  použijeme vzoreček  
 $\delta_1 = (d_1 - y_1) \cdot f'(h)$  a pro  $y_2$  vzoreček  
 $\delta_2 = (d_2 - y_2) \cdot f'(i)$ .

Výpočet chyby vnitřního neuronu  $g$  vypočítáme podle  
 $\delta_g = [w_{hg} \cdot \delta_h + w_{ig} \cdot \delta_i] \cdot f'(g)$   
a například chybu neuronu  $a$ ,  
 $\delta_a = [w_{ca} \cdot \delta_c + w_{da} \cdot \delta_d + w_{ea} \cdot \delta_{ea}] \cdot f'(a)$ .

Váhu  $w_{gb}$  a  $w_{ax_2}$  vypočítáme takto:  
 $w_{gb}^t = w_{gb}^{t-1} - \varepsilon \cdot \delta_g^{t-1}$   
 $w_{ax_2}^t = w_{ax_2}^{t-1} - \varepsilon \cdot \delta_a^{t-1}$

## 4.7 Aplikace metod

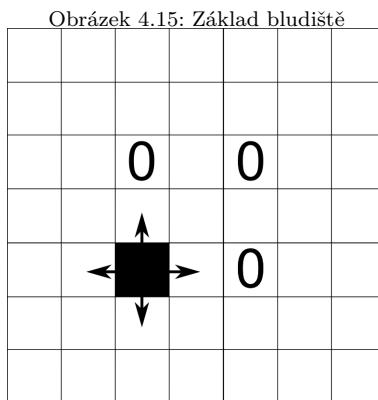
### 4.7.1 Bludiště

**B**ludiště budeme používat pro ukázku některých metod a algoritmů, které jsme vysvětlili dříve.

Algoritmus generující bludiště je velmi jednoduchý a díky tomu patří mezi nejznámější. Tyto algoritmy operují na dvourozměrnou mřížkou, na kterou si v jednotlivých cyklech ukládají značky, podle kterých výsledné bludiště vzniká.

Pro ukázku následujícího algoritmu budeme potřebovat tři značky. Jednu pro označení prázdného políčka, druhou pro zeď a třetí pro základ, ze kterého zeď „poroste“.

Jako první vygenerujeme základy pro stavbu zdí. To udělám následujícím algoritmem.



```

for (int i = start_y; i < vyska_pole - 2; i += 2) {
    for (int j = start_x; j < sirka_pole - 2; j += 2) {
        bludiste[i][j] = vycet_stavebni_prvky.zaklad;
    }
}

```

Pole má nyní značky mezi kterými je vždy jedna mezera. Tyto mezery jsou velice důležité, dokonce by neměla být ani žádná značka u kraje pole. Bludiště se pak negeneruje správně.

Ve své aplikaci používám startovní pozici na třetím řádku a třetím sloupci. Je to výsledek zpětné vazby a následných úprav.

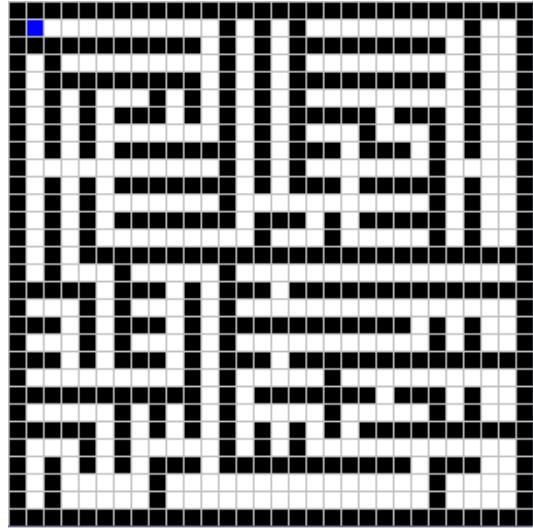
Dalším krokem je stavba obvodových zdí. Jednoduše olemujeme pole kolem dokola.

Nakonec necháme růst zdi ze základů, které jsme umístili v prvním kroku. Aby se nám vždy vygenerovalo jiné pole, vybereme náhodně jeden z umístěných základů. Označíme ho jako zeď. Jak napovídá obrázek [4.15], vybereme jeden z čtyř náhodných směrů políčka a tímto směrem stavíme zeď až do té doby, než narazíme na jinou zeď. Poté vybereme další základ a pokračujeme až do konce.

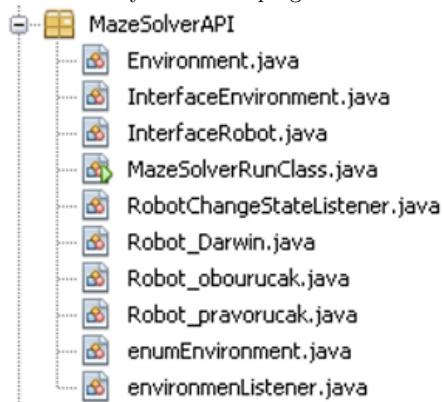
Výsledné bludiště může vypadat podobně jako na obrázku [4.16]

Knihovnu generující bludiště jsem pojmenoval „MazeGenerator“ a umístil na přiložené CD.

Obrázek 4.16: Hotové bludiště



Obrázek 4.17: Třídy a rozhraní programu MazeSolverApi



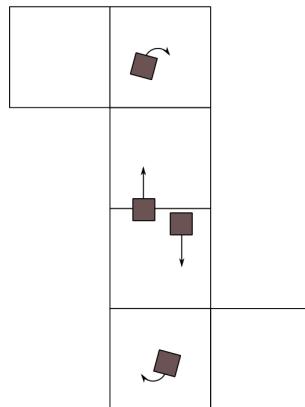
### 4.7.2 MazeSolverApi

Práce s fyzickým bludištěm není zdaleka tak flexibilní jako s bludištěm virtuálním. Při realizaci se objevilo mnoho problémů, které se obtížně řešili.

Přiklonil jsem se tedy k virtuálnímu bludišti, na který jsem napsal aplikaci v programovacím jazyku Java. Počítačové bludiště má velmi mnoho výhod. Pokud budeme aplikovat učícího se agenta, bude takové učení velmi rychlé. Můžeme zároveň učit i více agentů najednou. Bludiště je vždy náhodné a libovolně veliké. A hlavně se jedná o diskrétní prostředí nezatížené šumem nebo nedokonalým senzorem.

Spíše otevřené kódy než program se skládají z několika rozhraní a tříd [4.17].

Obrázek 4.18: Past na pravotočivého robota



Pro programátora, který by si chtěl vyzkoušet některé algoritmy pro bludiště je nejdůležitější rozhraní „InterfaceRobot“. Tato abstraktní třída má implementovanou většinu metod, které lze volat pouze v rámci dědičnosti. Nadřazená třída tak poskytuje například metodu `getLeftButtonState` nebo `getSonarDistanceData`.

Třída obsahující statickou metodu `main` se jmenuje `MazeSolverRunClass`. Poskytuje zobrazovací knihovnu <sup>[37](#)</sup> a propojuje algoritmus robota s prostředím.

Virtuální robot se vytvoří tak, že se implementuje zmíněná abstraktní třída <sup>[38](#)</sup> a tento robot se jako odkaz předá objektu `Environment`. Aby robot alespoň nedělal nic je nutné implementovat metodu `void go()`, která je volaná vždy, když má robot povolen.

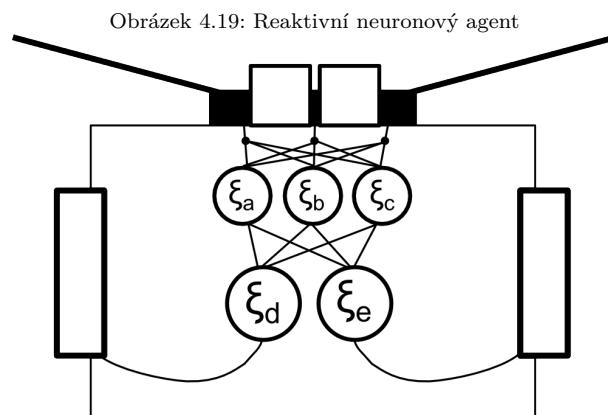
### 4.7.3 Základní algoritmy řešící útěk z bludiště

Mezi známé algoritmy řešící tento problém bych se chtěl jen krátce zmínit o metodě jedné ruky, kdy se zatáčí v bludišti stále na jednu stranu. Nevýhodou jsou cyklická prostředí nebo speciální topologie. Robot chodící stále doprava například snadno uvízne v zákrutu bludiště [4.18].

Tyto modely se vylepšují různým značkováním prostředí, či jednoduchou pamětí.

<sup>37</sup>Knihovna se jmenuje „vykresleniPole“

<sup>38</sup>Dál už záleží na programátorovi co bude s robotem dělat.



Zajímavějším modelem je reaktivní agent na bázi neuronů. Když například připojíme vstupy robota na vstupy neuronové sítě a výstup sítě použijeme pro řízení elektromotorů [4.19].

Takový robot sice neoplývá velkou pamětí, není schopen si pamatovat velké úseky mapy a vzory, které už potkal, ale je schopen se naučit komplexnější chování něž například pravotočivý robot.

Pro mě nejvíce zajímavým modelem jsou reaktivní neuronový agenti adaptování biologickými modely. Takovým je i Robot\_Darwin ve zmiňovaném projektu MazeSolverAPI.

Než je robot vpuštěn do bludiště, vytvoří si jiné virtuální bludiště s virtuálním prostředím, kde se „mentálně“ připravuje. Vytvoří si množinu neuronových sítí, které na své simulaci testuje. Postupně ohodnocuje jednotlivé neuronové sítě a ty nejlepší pak postupují do další generace. Jednotlivé etapy vývoje se vyznačují zrychlený růstem a delší stagnací. Simulace se vyvíjí ve skocích. Tento model je velmi výpočetně náročný.

#### 4.7.4 Matematika

Pro práci s neuronovými sítěmi se využívá takzvaná maticová algebra [28]. Matice je čtvercové či obdélníkové schéma nebo dvojrozměrná tabulka čísel se speciálními pravidly pro počítání. Uvedu zde pouze operace potřebné pro práci s neuronovými sítěmi.

Pro libovolnou neuronovou síť [4.21] se vypočítávají váhy postupným násobe-

Obrázek 4.20: Pravidlo pro násobení matic

$$m \left\{ \underbrace{\begin{pmatrix} \circ & \circ & \cdots & \circ \\ \circ & \circ & \cdots & \circ \\ \cdots & & & \\ \circ & \circ & \cdots & \circ \end{pmatrix}}_n \right\} \cdot n \left\{ \underbrace{\begin{pmatrix} \circ & \cdots & \circ \\ \circ & \cdots & \circ \\ \circ & \cdots & \circ \\ \cdots & & \\ \circ & \cdots & \circ \end{pmatrix}}_p \right\} = \underbrace{\begin{pmatrix} \circ & \cdots & \circ \\ \circ & \cdots & \circ \\ \cdots & & \\ \circ & \cdots & \circ \end{pmatrix}}_p \}_{m \times m}$$

ním vstupů a vah. Matice nám v neuronových sítích slouží jako datová struktura pro uchování vah sítě a také velmi zjednodušuje výpočet.

Váhy mezi vstupní vrstvou neuronů a první skrytou vyjádříme pomocí matice takto:

$$\begin{pmatrix} w_{14} & w_{15} \\ w_{24} & w_{25} \\ w_{34} & w_{35} \end{pmatrix}.$$

Analogický vyjádříme i váhy mezí výstupní vrstvou a vrstvou vnitřní:

$$\begin{pmatrix} w_{46} & w_{47} & w_{48} \\ w_{56} & w_{57} & w_{58} \end{pmatrix}.$$

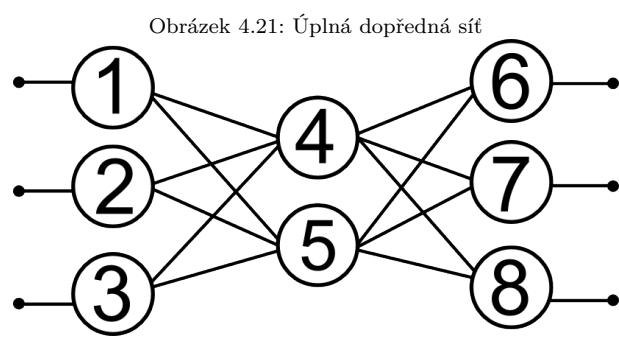
Pro výpočet výstupu sítě se matice mezi sebou vynásobí podle následujícího pravidla.

$$\begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix} \cdot \begin{pmatrix} g & i & k \\ h & j & l \end{pmatrix} = \begin{pmatrix} a \cdot d + g \cdot h & a \cdot d + i \cdot j & a \cdot d + k \cdot l \\ b \cdot e + g \cdot h & b \cdot e + i \cdot j & b \cdot e + k \cdot l \\ c \cdot f + g \cdot h & c \cdot f + i \cdot j & c \cdot f + k \cdot l \end{pmatrix}$$

Při násobení musí být dodrženo pravidlo [4.20].

Když dáme všechny tyto „věci“ dohromady, můžeme vyjádřit neuronovou síť jako

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 0 \\ 4 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 \\ 14 & 4 & 0 \\ 1 & 2 & 0 \end{pmatrix}$$



Konfiguraci sítě jsem zvolil náhodně. Další operace s maticemi, které jsem při práci potřeboval jsou zobrazeny na obrázku [4.22].

Obrázek 4.22: Maticové operace

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \xrightarrow{\text{jednotková matici}} \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} \xrightarrow{\text{transpozice}} \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} \xrightarrow{\quad} \begin{pmatrix} 4 & 2 \\ 3 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 3 \\ 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 15 & 12 \\ 13 & 10 \end{pmatrix} \quad \text{násobení matic}$$

$$\begin{pmatrix} 3 & 3 \\ 3 & 1 \end{pmatrix} + \begin{pmatrix} 4 & 3 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 7 & 6 \\ 4 & 2 \end{pmatrix} \quad \text{sčítání matic}$$

---

## ZÁVĚR

Neuronové sítě a umělá inteligence jsou v současné vědě velmi rozšířeným tématem. Bylo-li by možné vyjmenovat všechny obory a podobory, které mají s neuronovými sítěmi a umělou inteligencí něco společného, nedospěli bychom ke konci, ale k názoru, že není žádný obor či podobor, ale jen věda samotná. Mým největším cílem, takovým nadcílem, bylo inspirovat všechny zájemce o robotiku.

Díky zkušenostem získaným při tvorbě robota a zpracování práce v celkovém hledisku jsem dospěl k názoru, že podobný projekt je časově velice náročný a je nutné se mu věnovat velmi dlouho a s učením se robotice a umělé inteligenci není možné skončit ani následně, neboť je to věda neustále se progresivně rozvíjející. Během stavby robota jsem měl možnost mluvit a setkat se s lidmi, kteří se zabývají podobnými tématy jako já ve své práci. Byla to všechna velice inspirativní setkání, jež mě navedla na cesty dalších myšlenek. Inspirativní pro mě byla i ročníková práce mého studenta ze střední školy, na níž vyučuji. Rád bych některé jeho myšlenky šířil a poskytl k úvahám, na doprovodný nosič jsem umístil i dva jeho projekty. Program pro ovládání robota pomocí myši a program pro nahrání bootloaderu do procesoru <sup>39</sup>.

Při práci jsem narazil na mnoho dílčích problémů, které se zdály v určitém čase až neřešitelné. Nejsložitější bylo vyladit program pro komunikaci se sériovým

---

<sup>39</sup>Bootloader je zaváděcí program, který umožní programovat součástku bez programátora. Binární soubor si uloží do paměti procesor sám.

portem. Celé ladění probíhalo pomocí aplikace com0com (virtuální sériový port a propojení nulovým modem). Problém nastal při přechodu na ostrý provoz. Celý problém, jež jsem řešil, je rozebrán ve třetí kapitole.

Moje vize na začátku zadávání diplomové práce byla velmi naivní a znalosti neuronových sítí a hardwaru malé. Na začátku jsme s vedoucím práce používali platformu Arduino, kterou bych rozhodně nechválil. V podobném projektu je opravdu nevhodnější navrhnut vlastní elektroniku sám a vytvořit návrh na míru.

Jedním z cílů bylo seznámení se s platformou Arduino. Tento cíl byl zcela naplněn, ovšem absence přerušení mě od této platformy odradila, i když v novějších verzích by už tento problém měl být vyřešen.

Má původní vize byla robot jako samostatný stroj, který se bude sám řídit pomocí neuronové sítě v čipu. Dlouhou dobu jsem přemýšlel nad tím, jak programovat objektově pro takový hardware jako je například Arduino<sup>40</sup>. Postupný posun v myšlení byl znát ve chvíli, když jsem přemýšlel, jak dostat výpočty na trošku lepší hardware, ale pořád tak malý, aby mohl být součástí robota.

Chtěl jsem použít mobilní telefon, ale bohužel ne všemi modely lze komunikovat po sériové lince po připojení iniciované mobilním telefonem. Takový telefon musím mít Java standard JSR-82.

Ve své práci jsem se podrobně seznámil s nejčastějšími modely neuronových sítí a jejich optimalizačními metodami.

Práce začala u umělé inteligence a chtěl bych, aby u ní také skončila. Při návštěvách internetových stránek zaměřených na umělou inteligenci jsem narazil na souboje umělých inteligencí ve hrách. Nejvíce jsem si oblíbil souboje v piškvorkách nebo projekty nerogame.org a opennero. Společnost IBM nabízí volně ke stažení Java knihovnu Robocode. Ta obsahuje v podstatě jednoduchý herní engine a také se v ní konají soutěže umělých inteligencí.

---

<sup>40</sup>Jak jsem psal, má vize byla naivní

---

# MAKRA

- **SERIAL\_OUT TXREG**

TXREG je registr, který slouží pro posílání dat na sériovou linku

- **LED RD4**

D4 je pin, na kterém máme připojenou LED diodu

- **LED\_TRISD4 TRISD4**

TRISD4 je registr nastavující pin D4 jako vstup, nebo výstup

- **ON 0**

kdekoliv je napsáno slovo „ON“ dosadí překladač nulu

- **OFF 1**

kdekoliv je napsáno slovo „OFF“ dosadí překladač jedničku

- **INPUT 1**

kdekoliv je napsáno slovo „INPUT“ dosadí překladač jedničku

- **OUTPUT 0**

kdekoliv je napsáno slovo „OUTPUT“ dosadí překladač nulu

- **BUTTON\_L RD7**

na pinu D7 je připojeno levé tlačítko

- **BUTTON\_R RD6**

na pinu D6 je připojeno pravé tlačítko

- **BUTTON\_L\_TRIS TRISD7**

TRISD7 je registr nastavující pin D7 jako vstup, nebo výstup

- **BUTTON\_R\_TRIS TRISD6**

TRISD6 je registr nastavující pin D6 jako vstup, nebo výstup

- **SONAR\_PIN RD5**

na pinu D5 je připojen sonar

- **SONAR\_PIN\_TRIS TRISD5**

TRISD5 je registr nastavující pin D5 jako vstup, nebo výstup

- **TX\_TRIS TRISC6**

TRISC6 je registr nastavující pin C6 jako vstup, nebo výstup

- **RX\_TRIS TRISC7**

TRISC7 je registr nastavující pin C7 jako vstup, nebo výstup

- **CCP1\_TRIS TRISC2**

TRISC2 je registr nastavující pin C2 jako vstup, nebo výstup

- **CCP2\_TRIS TRISC1**

TRISC1 je registr nastavující pin C1 jako vstup, nebo výstup

- **M1\_TRIS TRISD0**

TRISD0 je registr nastavující pin D0 jako vstup, nebo výstup

- **M2\_TRIS TRISD1**

TRISD1 je registr nastavující pin D1 jako vstup, nebo výstup

- **M1\_DIR RD0**

TRISD0 je registr nastavující pin D0 jako vstup, nebo výstup

- **M2\_DIR RD1**

TRISD1 je registr nastavující pin D1 jako vstup, nebo výstup

---

## PŘÍLOHA CD

1. Projekty v programovacím jazyce C pro mikroprocesor
2. Fotografie dřívějších verzí robota
3. Java aplikace a knihovny jar
4. Jiné projekty (bootloader,...)
5. Volně dostupná použitá literatura
6. Schémata a seznamy součástek pro program EAGLE
7. com0com aplikace a ovládání RS232 pomocí C#
8. Software pro bluetooth modul OEMSPA310
9. Seriál Roboti a robotika Českého rozhlasu Leonardo

---

# SEZNAM OBRÁZKŮ

1.1	Vztah mezi robotem a prostředím . . . . .	14
1.2	Recon Scout Throwbot (robot Špulka)[19] . . . . .	17
2.1	UMU-01 [33] . . . . .	22
2.2	Komunikační protokol sonaru [8] . . . . .	24
2.3	Spodní pohled na bluetooth modul [6] . . . . .	26
2.4	Blokové schéma řídící elektroniky . . . . .	28
2.5	Pin diagram procesoru PIC [26] . . . . .	30
2.6	Pin diagram obvodu L293D . . . . .	33
2.7	Rozhraní komunikace mezi bluetooth modulem a procesorem . . . . .	34
3.1	Algoritmus hlavního programu . . . . .	39
3.2	Algoritmus přerušení . . . . .	40
3.3	Registr TSXTA . . . . .	43
3.4	Registr RCXTA . . . . .	43
3.5	Časový průběh stavu napětí na výstupu procesoru . . . . .	46

3.6 Registr CCP1CON . . . . .	46
3.7 Registr T2CON . . . . .	46
3.8 Vztah mezi PWM periodou a PWM duty cykle . . . . .	47
3.9 Registr T1CON . . . . .	49
3.10 Blokové schéma přijímače . . . . .	51
3.11 Rozhraní třídy COM_DOM . . . . .	55
3.12 Rozhraní objektu RobotComm . . . . .	56
3.13 Rozhraní objektu RobotComm . . . . .	57
3.14 Testovací aplikace . . . . .	59
4.1 Struktura nervového systému - zpětnovazební systém . . . . .	69
4.2 Neuronová buňka . . . . .	70
4.3 McCullochův-Pittsův model neuronu . . . . .	71
4.4 Závislost výstupu na vstupních hodnotách a prahu $T$ . . . . .	72
4.5 Model perceptronu . . . . .	73
4.6 Ostrá nelinearita . . . . .	73
4.7 Synaptická váha mezi neurony . . . . .	74
4.8 Přehled aktivačních funkcí . . . . .	74
4.9 Neuronová síť . . . . .	75
4.10 Příklady topologií . . . . .	77
4.11 Vztah kvality jedinců v populaci na čase [5] . . . . .	81
4.12 Model perceptronu . . . . .	82
4.13 Tabulka učení perceptronu Hebbovým pravidlem . . . . .	82
4.14 Organizační dynamika pří výpočtu BP . . . . .	84
4.15 Základ bludiště . . . . .	87

4.16 Hotové bludiště . . . . .	88
4.17 Třídy a rozhraní programu MazeSolverApi . . . . .	88
4.18 Past na pravotočivého robota . . . . .	89
4.19 Reaktivní neuronový agent . . . . .	90
4.20 Pravidlo pro násobení matic . . . . .	91
4.21 Úplná dopředná síť . . . . .	92
4.22 Maticové operace . . . . .	93

---

## SEZNAM TABULEK

2.1	Parametry bluetooth modulu	25
2.2	PIC16F877A	30
2.3	4 kanálový můstek L293D	32
2.4	Hradlo AND 74HT08	33
3.1	Nastavení registrů TXSTA a RCSTA	43
3.2	Přehled vzorců pro výpočet přenosové rychlosti	44
4.1	Logický součin AND	79

---

## LITERATURA

- [1] Bláha V. a Český rozhlas Leonardo: Roboti a robotika, 2010, 12 dílů audioslideshow povídání o robotech a robotice dostupné na [www.rozhlas.cz/leonardo/technologie](http://www.rozhlas.cz/leonardo/technologie).
- [2] Boston Dynamics: BigDog - The Most Advanced Rough-Terrain Robot on Earth, 2012, zpráva společnosti Boston Dynamics dostupná na [www.bostondynamics.com/robot\\_bigdog.html](http://www.bostondynamics.com/robot_bigdog.html).
- [3] Boston Dynamics: PETMAN - BigDog gets a Big Brother, 2012, zpráva společnosti Boston Dynamics dostupná na [www.bostondynamics.com/robot\\_petman.html](http://www.bostondynamics.com/robot_petman.html).
- [4] Boston Dynamics: SandFlea - Leaps Small Buildings in a Single Bound, 2012, dokumentace je dostupná na [www.bostondynamics.com/robot\\_sandflea.html](http://www.bostondynamics.com/robot_sandflea.html).
- [5] Chalupník V.: Biologické algoritmy (1) - Evoluční algoritmy. *Root*, 2012.
- [6] connectBlue AB: *OEM Serial Port Adapter<sup>TM</sup>- Electrical & Mechanical Datasheet*. 2012.  
URL [www.spezial.cz/pdf/em\\_ds\\_oemspa\\_310.pdf](http://www.spezial.cz/pdf/em_ds_oemspa_310.pdf)
- [7] connectBlue AB: *Product Brief OEMSPA310*. 2012.  
URL [www.spezial.cz/pdf/ProductBrief\\_OEMSPA310.pdf](http://www.spezial.cz/pdf/ProductBrief_OEMSPA310.pdf)

- [8] Devantech Ltd: *SRF05 - Ultra-Sonic Ranger Technical Specification*. 2012.  
URL [www.robot-electronics.co.uk/htm/srf05tech.htm](http://www.robot-electronics.co.uk/htm/srf05tech.htm)
- [9] Dumeck V.: *Programování v jazyku C*. Fakulta strojního inženýrství VUT v Brně, 2004.
- [10] Fojtík R.: *Programování v C*. Ostravská univerzita, 2004.
- [11] Grill P.: EEG – Elektroencefalografe, 2005, prezentace, ČVUT, FEL, Katedra kybernetiky.
- [12] Havel I. M.: *Robotika - úvod do teorie kognitivních robotů*. SNTL Nakladatelství technické literatury, 1980.
- [13] iRobot: iRobot ChemBot dokáže měnit tvar jako Terminátor T-1000, 2012, Článek je dostupný na adrese [irobot.cz/cz/press-centrum](http://irobot.cz/cz/press-centrum).
- [14] Janoušek J. a Voženílek P.: Stejnosměrný motor s cizím buzením, 2011, text je dostupný na stránkách [motor.feld.cvut.cz](http://motor.feld.cvut.cz) jako zdroj ke cvičení z předmětu „Základy silnoproudé elektrotechniky (X14ZSE)“.
- [15] Koller M.: Směr Ammán. *Robot Revue*, 2010.
- [16] Koller M.: Robot Špulka. *Robot Revue*, 2011.
- [17] Kraus A.: *Návrhový systém EAGLE*. 2012.  
URL [web.quick.cz/chmelar.t/eagle/manual.htm](http://web.quick.cz/chmelar.t/eagle/manual.htm)
- [18] Littschwager T.: Časová osa: Roboti, 2010, Článek portálu Chip.cz dostupný na [earchiv.chip.cz/cs/earchiv](http://earchiv.chip.cz/cs/earchiv).
- [19] Lutonský M.: Nový robot Recon Scout Throwbot: leze, leze po železe. *Mladá fronta E15 VTM.cz*, 2011.
- [20] Šíma J. & Neruda R.: *Teoretické otázky neuronových sítí*. Matfyzpress, 1996, ISBN 80-85863-18-9.
- [21] Mařík V.: *Umělá inteligence 1*. ACADEMIA, 1993, ISBN 80-200-0496-3.
- [22] Mařík V.: *Umělá inteligence 2*. ACADEMIA, 1997, ISBN 80-200-0504-8.

- [23] Matoušek D.: *C pro mikrokontroléry PIC*. BEN - technická literatura, 2011, ISBN 978-80-7300-413-2.
- [24] McGrath M.: *C programming in easy steps*. Computer Step, 2002, ISBN 1-84078-203-X.
- [25] Microchip Technology Inc.: *In-Circuit Serial Programming (ICSP) Guide*. 2012.  
URL [ww1.microchip.com/downloads/en/devicedoc/30277d.pdf](http://ww1.microchip.com/downloads/en/devicedoc/30277d.pdf)
- [26] Microchip Technology Inc.: *PIC16F87XA Data Sheet*. 2012.  
URL [ww1.microchip.com/downloads/en/devicedoc/39582b.pdf](http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf)
- [27] Mikulčák J.: *Matematické, fyzikální a chemické tabulky pro střední školy*. Státní pedagogické nakladatelství Praha, 1988, ISBN 80-85849-84-4.
- [28] Olšák P.: Lineární algebra, 2000-2007, scripta jsou dostupná na adrese [math.feld.cvut.cz/pub/olsak/linal/linal.pdf](http://math.feld.cvut.cz/pub/olsak/linal/linal.pdf).
- [29] Pěchouček M.: Úvod do filosofie umělé inteligence. *katedra kybernetiky ČVUT v Praze*, 2012.
- [30] Russell S. a Norvig P.: *Artificial Intelligence A Modern Approach*. Prentice Hall, 2010, ISBN 0-13-604259-7, 978-0-13-604259-4.
- [31] samtec: *FSI-1XX-03-X-D-X-XX-XX*. 2012.  
URL [www.samtec.com/ftppub/cpdf/FSI-1XX-03-X-D-X-XX-XX-MKT.pdf](http://www.samtec.com/ftppub/cpdf/FSI-1XX-03-X-D-X-XX-XX-MKT.pdf)
- [32] Schildt H.: *Java 7 - Výukový kurz*. Computer Press, 2012, ISBN 978-80-251-3748-2.
- [33] Snail Instruments: Katalogový list - podvozek UMU-01, 2012, [shop.snailinstruments.com](http://shop.snailinstruments.com).
- [34] Sony Computer Science Laboratory in Paris: Can a dog tell the difference?, 2012, Článek je dostupný na adrese [www.csl.sony.fr/items/2000/dog-versus-aibo](http://www.csl.sony.fr/items/2000/dog-versus-aibo).
- [35] STMicroelectronics: *POSITIVE VOLTAGE REGULATORS*. 2012.  
URL [www.datasheetcatalog.org/datasheets/270/9138\\_DS.pdf](http://www.datasheetcatalog.org/datasheets/270/9138_DS.pdf)

- [36] Suchan J.: Co je to stavový diagram? [www.minmax.cz](http://www.minmax.cz), 2007.
- [37] Thrun S., Burgard W. and Fox D.: *Probabilistic Robotics*. MIT Press, 2005, ISBN 9780262201629.
- [38] Toal R.: Problem Solving. *Loyola Marymount University Los Angeles*, 2012.
- [39] Vondra M. a Lank V.: *Fyzika v kostce pro střední školy*. Fragment, 2002, ISBN 80-7200-335-6.
- [40] Vít Olmr: *HW server představuje - Sériová linka RS-232*. 2012.  
URL [www.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html](http://www.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html)
- [41] Zunt D. a Dušáková E.: Karel Čapek, 2012, internetový portál věnovaný Karlu Čapkovi [capek.misto.cz](http://capek.misto.cz).