

# Řízení pohybu autonomního robota pomocí umělé inteligence

Autonomous Robot Movement Control  
by Means of Artificial Intelligence

Bc. Vladimír Štusák



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2011/2012

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Vladimír Štusák**

Osobní číslo: **A11506**

Studijní program: **N 3902 Inženýrská informatika**

Studijní obor: **Informační technologie**

Téma práce: **Řízení pohybu autonomního robota pomocí umělé inteligence**

Zásady pro vypracování:

1. Vypracujte rešerši na téma využití neuronových sítí pro řízení pohybu.
2. Na základě rešerše zvolte vhodný typ neuronové sítě.
3. Navrhněte strukturu neuronové sítě tak, aby umožňovala na základě vstupů ze senzorů robota a jejich priorit provádět rozhodování o směru pohybu robota.
4. Proveďte implementaci neuronové sítě na platformě Microsoft .NET Framework v jazyce C Sharp.
5. Podrobně popište vliv dat ze senzorů na směr pohybu řízený neuronovou sítí.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. ZELINKA, Ivan. Umělá inteligence I: Neuronové sítě a genetické algoritmy. Brno: VUT v Brně, 1998, 126 s. ISBN 80-214-1163-5.
2. JONES, M. Artificial intelligence: a systems approach. Hingham, Mass.: Infinity Science Press, 2008, 498 s. ISBN 09-778-5823-5.
3. HU, Yu Hen a Jenq-Neng HWANG. Handbook of neural network signal processing. Boca Raton: CRC Press, 2002. ISBN 08-493-2359-2.
4. HEATON, Jeff. Introduction to neural networks for C Sharp. 2nd ed. St. Louis: Heaton Research Inc, 2008. ISBN 16-043-9009-3.
5. ALPAYDIN, Ethem. Introduction to machine learning. 2nd ed. Massachusetts: MIT Press, 2010, 537 s. ISBN 978-026-2012-430.

Vedoucí diplomové práce:

doc. Mgr. Roman Jašek, Ph.D.  
Ústav informatiky a umělé inteligence

Konzultant:

Ing. Lukáš Kouřil  
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

24. února 2012

Termín odevzdání diplomové práce:

21. května 2012

Ve Zlině dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.  
děkan



doc. Mgr. Roman Jašek, Ph.D.  
ředitel ústavu

## **ABSTRAKT**

Tato práce se věnuje umělé inteligenci realizované pomocí neuronových sítí a jejich využití na reálném problému. Cílem práce bylo zajištění ovládní autonomního robota pomocí umělé inteligence na základě získaných dat o jeho okolí. Tato data pocházela z laserového skeneru, který je součástí robota a v pravidelných intervalech zasílá hlavní řídicí aplikaci informace o nejbližších překážkách ve směru jízdy. Na základě těchto informací bude hlavní aplikace ovládat robota a jeho další směr jízdy. Zjednodušeně by se dalo říci, že vytvořená neuronová síť bude přímo řídit pohyb robota. Cílem bylo navrhnout vhodný typ sítě, který by tento problém řešil, a na základě rozboru dat ze senzoru zvolenou neuronovou síť implementovat. Součástí implementace bylo také provést rozbor dat z laserového senzoru a transformovat je na data, která by byla vhodná pro vložení na vstup do neuronové sítě.

Klíčová slova:

Neuronová síť, Učení neuronové sítě, Backpropagation, Robot, Řízení neuronovou sítí, Senzory, C#, .NET, Microsoft Visual Studio, Implementace

## ABSTRACT

This thesis deals with artificial intelligence which is implemented by means of neural networks and the use of neural networks in an actual problem. The problem whose solution was the task of this thesis was the control of an autonomous robot by means of artificial intelligence based on acquired data about its surroundings. These data came from a laser scanner which is a part of the robot and which, in regular intervals, sends to the controlling application information about any close obstacles in the course of movement. This information forms the basis for decisions of robot's main controlling application about robot's course of motion. In a simplified way it is possible to say that the created neural network will directly control robot's motion. The aim was to project a suitable network type which would solve the problem and, on the basis of sensor data analysis, implement the chosen neural network. The implementation also included laser sensor data analysis and their transformation into such data that would be suitable for neural network input.

### Keywords:

Neural network, Learning of neural network, Backpropagation, Robot, Movement control by neural network, Sensors, C#, .NET, Microsoft Visual Studio, Implementation of neural network

Chtěl bych velmi poděkovat mému vedoucímu diplomové práce doc. Mgr. Romanu Jaškovi Ph.D. za odborné vedení mé diplomové práce a Ing. Lukáši Kouřilovi za mnohé konzultace, rady, tipy a připomínky, které mě vedly správným směrem během práce. Velké poděkování také patří Ing. Zuzaně Oplatkové Ph.D. za spoustu času, který mi věnovala během studia problematiky neuronových sítí a jejich implementace, a za spoustu užitečných rad, které mi pomohly úspěšně a včas dokončit tuto práci. Také bych chtěl poděkovat Ing. Pavlu Neckářovi za představení robota a konzultace spojené s návrhem neuronové sítě na míru vyvíjenému robotu. Děkuji.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- § že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- § že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

|   |           |
|---|-----------|
| <b>ÚVOD.....</b>  | <b>10</b> |
| <b>I TEORETICKÁ ČÁST.....</b>                                   | <b>12</b> |
| <b>1 AUTONOMNÍ ROBOT.....</b>                                   | <b>13</b> |
| 1.1 SOFTWARE VYBAVENÍ ROBOTA.....                               | 14        |
| 1.2 HARDWARE VYBAVENÍ ROBOTA .....                              | 15        |
| 1.3 POUŽITÉ SENZORY .....                                       | 15        |
| 1.3.1 Infračervená čidla.....                                   | 15        |
| 1.3.2 Ultrazvukové senzory.....                                 | 16        |
| 1.3.3 Laserový senzor (aktuální).....                           | 17        |
| 1.3.3.1 Výstup z laserového senzoru .....                       | 18        |
| 1.3.4 Laser senzor (plánovaný) .....                            | 18        |
| <b>2 NEURONOVÉ SÍTĚ.....</b>                                    | <b>20</b> |
| 2.1 ZÁKLADNÍ DRUHY NEURONOVÝCH SÍTÍ .....                       | 23        |
| 2.1.1 Perceptron .....  | 23        |
| 2.1.1.1 Určení a typ sítě .....                                 | 23        |
| 2.1.1.2 Schéma sítě .....                                       | 23        |
| 2.1.2 Adaline a Madaline .....                                  | 24        |
| 2.1.2.1 Určení a typ sítě .....                                 | 24        |
| 2.1.2.2 Schéma sítě .....                                       | 25        |
| 2.1.3 Feedforward s algoritmem Backpropagation .....            | 25        |
| 2.1.3.1 Určení a typ sítě .....                                 | 25        |
| 2.1.3.2 Schéma sítě .....                                       | 26        |
| 2.1.4 Kohonenova mapa – samoorganizující se síť .....           | 27        |
| 2.1.4.1 Určení a typ sítě .....                                 | 27        |
| 2.1.4.2 Schéma sítě .....                                       | 29        |
| 2.1.5 Hopfieldova síť .....                                     | 30        |
| 2.1.5.1 Určení a typ sítě .....                                 | 30        |
| 2.1.5.2 Schéma sítě .....                                       | 30        |
| 2.1.6 CLN.....  | 31        |
| 2.1.6.1 Určení a typ sítě .....                                 | 31        |
| 2.1.6.2 Schéma sítě .....                                       | 31        |
| 2.1.7 Neuronová síť BAM.....                                    | 32        |
| 2.1.7.1 Určení a typ neuronové sítě .....                       | 32        |
| 2.1.7.2 Schéma sítě .....                                       | 32        |
| 2.2 VHODNÉ NEURONOVÉ SÍTĚ PRO ŘÍZENÍ AUTONOMNÍHO ROBOTA .....   | 33        |
| 2.3 APLIKOVANÁ NEURONOVÁ SÍŤ PRO REALIZOVANÉHO ROBOTA.....      | 34        |
| <b>3 NEURONOVÁ SÍŤ A DATA ZE SENZORŮ .....</b>                  | <b>37</b> |
| <b>4 POUŽITÉ TECHNOLOGIE K IMPLEMENTACI NEURONOVÉ SÍTĚ.....</b> | <b>39</b> |
| 4.1 VISUAL STUDIO 2010 .....                                    | 39        |
| 4.2 JAZYK C# .....  | 40        |
| 4.3 WINDOWS PRESENTATION FOUNDATION.....                        | 40        |
| 4.4 NÁSTROJE JETBRAINS .....                                    | 40        |
| 4.4.1 ReSharper 6.1.1 .....                                     | 41        |
| 4.4.2 dotTrace 5.0 Performance .....                            | 41        |



|           |   |           |
|-----------|---|-----------|
| 4.4.3     | dotTrace 3.5 Memory .....   | 41        |
| <b>II</b> | <b>PRAKTICKÁ ČÁST .....</b>   | <b>42</b> |
| <b>5</b>  | <b>IMPLEMENTACE NEURONOVÉ SÍTĚ .....</b>                            | <b>43</b> |
| 5.1       | DLL KNIHOVNA .....  | 43        |
| 5.2       | OBSAH KNIHOVNY .....  | 44        |
| 5.3       | IMPLEMENTACE NEURONOVÉ SÍTĚ .....                                   | 45        |
| 5.3.1     | Neuronová síť jako matice .....                                     | 46        |
| 5.3.2     | Neuronová síť jako objekty .....                                    | 46        |
| 5.4       | ÚPRAVA NEURONOVÉ SÍTĚ .....   | 46        |
| 5.5       | VYTVOŘENÍ NEURONOVÉ SÍTĚ .....                                      | 48        |
| 5.5.1     | Nová neuronová síť a její učení .....                               | 48        |
| 5.5.2     | Předem naučená neuronová síť .....                                  | 51        |
| 5.6       | POUŽÍVÁNÍ NEURONOVÉ SÍTĚ .....                                      | 52        |
| 5.6.1     | Předem naučená neuronová síť .....                                  | 52        |
| 5.6.2     | Učení neuronové sítě .....  | 52        |
| 5.6.2.1   | Algoritmus učení neuronové sítě – Backpropagation .....             | 54        |
| 5.6.3     | Vkládání dat na vstup neuronové sítě .....                          | 57        |
| 5.7       | VÝSTUP Z NEURONOVÉ SÍTĚ .....                                       | 58        |
| 5.7.1     | Získání výstupu z neuronové sítě .....                              | 60        |
| <b>6</b>  | <b>UŽIVATELSKÉ ROZHRANÍ PRO TESTOVÁNÍ .....</b>                     | <b>61</b> |
| 6.1       | UŽIVATELSKÉ ROZHRANÍ PRO VYTVOŘENÍ UČÍCÍ MNOŽINY .....              | 61        |
| 6.2       | UŽIVATELSKÉ ROZHRANÍ PRO TESTOVÁNÍ KNIHOVNY S NEURONOVOU SÍTÍ ..... | 63        |
| 6.2.1     | Načtení dat a naučení neuronové sítě .....                          | 64        |
| 6.2.2     | Testování neuronové sítě .....                                      | 65        |
| <b>7</b>  | <b>VÝSLEDNÁ NEURONOVÁ SÍŤ .....</b>                                 | <b>67</b> |
| <b>8</b>  | <b>VYHODNOCENÍ VÝSTUPŮ Z NEURONOVÉ SÍTĚ .....</b>                   | <b>69</b> |
| <b>9</b>  | <b>DALŠÍ MOŽNOSTI VÝVOJE .....</b>                                  | <b>71</b> |
|           | <b>ZÁVĚR .....</b>  | <b>72</b> |
|           | <b>ZÁVĚR V ANGLIČTINĚ .....</b>                                     | <b>74</b> |
|           | <b>SEZNAM POUŽITÉ LITERATURY .....</b>                              | <b>76</b> |
|           | <b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>                     | <b>77</b> |
|           | <b>SEZNAM OBRÁZKŮ .....</b>   | <b>78</b> |
|           | <b>SEZNAM TABULEK .....</b>   | <b>80</b> |
|           | <b>SEZNAM PŘÍLOH .....</b>  | <b>81</b> |

## ÚVOD

Cílem této práce bylo vytvořit fungující umělou inteligenci, kterou by bylo snadné začlenit do projektu vytvoření autonomního robota, který by měl mít na starost bezpečnost a měl by nahradit noční strážce ve velkých objektech, jako jsou obchodní centra nebo továrny. Tento robot se skládá z velkého počtu subsystémů, hardwarových i softwarových. Tato práce tvoří jeden z podstatných softwarových subsystémů, bez kterého by tento robot nebyl schopen samostatného pohybu. Zjednodušeně by se dalo říci, že vytvořená neuronová síť bude přímo řídit pohyb robota.

Než bylo možné navrhnout neuronovou síť, bylo potřeba se důkladně seznámit s řešeným problémem a s tím, zda byl podobný problém již řešen. Bylo potřeba zjistit, na jakém principu fungují různé druhy umělé inteligence a jak se případně implementují. Toto porovnání zpracovávám v teoretické části. S ohledem na vybranou neuronovou síť, která by měla implementovat umělou inteligenci robota, bylo potřeba nastudovat, jaká data dostaneme z laserového senzoru, a posoudit, zda je nutné tato data případně nějakým způsobem pro neuronovou síť upravit nebo ne, aby bylo dosaženo rozumného kompromisu mezi rychlostí odezvy neuronové sítě a aby komprese dat nebyla příliš velká a nedocházelo ke ztrátě důležitých informací z dat.

Existuje velké množství typů neuronových sítí, ale každý typ je nějakým prvkem pro danou neuronovou síť specifický a tím určuje její zaměření. Bylo důležité pochopit problém, který máme řešit, protože to velmi usnadní konečný výběr neuronové sítě. Nesmí se opomenout fakt, že vždy velmi záleží na přesné konfiguraci neuronové sítě pro daný problém. Během implementace bude tedy nutné se zaměřit hlavně na testování různých konfigurací neuronové sítě a zjistit, která z nich bude mít největší úspěch v řešení našeho problému.

Součástí hlavního výstupu z této práce (DLL knihovny) by měly být také aplikace, které zjednoduší práci s implementovanou neuronovou sítí. Je zapotřebí hlavně aplikace, ve které bude možné vytvořit trénovací množinu dat pro neuronovou síť a případně aplikace, ve které bude možno neuronovou síť naučit a vyzkoušet, protože konečné použití knihovny v hlavní řídicí aplikaci tyto prvky přímo nepotřebuje.

Výstupem práce by měla být DLL knihovna, která se jednoduše vloží do hlavní řídicí aplikace autonomního robota a bude možné ji co nejsnáze zařadit do hlavního cyklu programu. Cílem této umělé inteligence bude v první řadě zajistit, aby nedošlo k žádné

kolizi robota s jeho okolím, a navádět ho vždy do prostoru, kterým bude moci bezpečně projet.

Tato práce byla pro mě velkou motivací vyzkoušet si pokročilé techniky v programování a umožnila mi seznámit se velmi podrobně s problematikou v oblasti umělé inteligence. Velký přínos a motivaci jsem také viděl v tom, že výstup této práce bude použitý v reálném robotovi a nebude se jednat pouze o teoretickou práci, i když bez teorie by nebylo možné tento problém vyřešit.

Zpracování zadaného úkolu bylo potřeba naplánovat ve strategických krocích, které by na sebe navazovaly, aby například během implementace nenastala situace, kdy by nebylo možné naučit neuronovou síť pro řešení daného problému a nebyla by k tomu známa příčina nebo pokud by byl špatně vybrán typ neuronové sítě a při nasazení do reálného provozu by se zjistilo, že umělá inteligence pracuje naprosto jinak, než se očekávalo, nebo nebude možné neuronové síti zajistit dostatečný počet informací k určení správné cesty

Zpracování probíhalo v následujících bodech:

- Vypracování rešerše na téma využití neuronových sítí pro řízení pohybu.
- Zvolení vhodného typu neuronové sítě s ohledem na dostupné prostředky autonomního robota.
- Navrhnout strukturu neuronové sítě tak, aby umožňovala na základě vstupu ze senzorů robota provádět rozhodování o směru pohybu robota.
- Provést implementaci neuronové sítě na platformě Microsoft .NET Framework v jazyce C#.

## **I. TEORETICKÁ ČÁST**

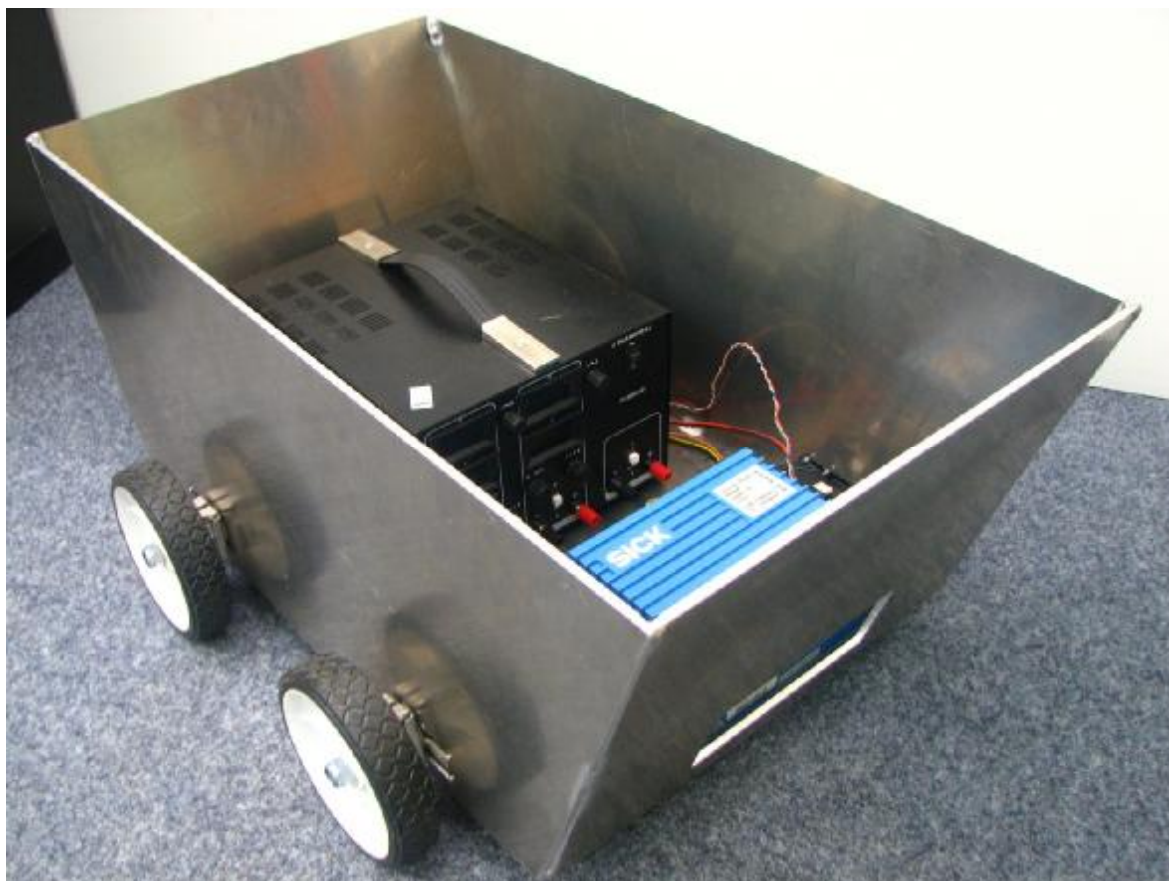
## 1 AUTONOMNÍ ROBOT

Autonomní robot, pro kterého byla implementována v rámci této diplomové práce umělá inteligence pro pohyb ve vnitřních prostorech (místnosti, haly, obchodní centra), byl primárně určen pro ochranu a zabezpečení prostor, ve kterých se má pohybovat, například velká obchodní centra, která jsou v noci zavřená.

Tento robot by měl vlastníkově chráněného objektu minimalizovat náklady na zabezpečení a nahradit většinu osob, které se na zabezpečení podílejí. Avšak stále bude při nasazení potřeba i lidský faktor kvůli dvěma základním situacím, které mohou nastat:

1. Robot v autonomním režimu detekuje narušení zabezpečení (například cizí osobu v objektu). V této situaci upozorní ostrahu ve službě. Pokud bude robot vybaven prostředky ne eliminaci pachatele (slzný plyn, vystřelovací síť, ...), použije je.
2. Dozorci v objektu pomocí kamer zjistí narušení objektu ozbrojenými zločinci. V tomto případě by mohl využít robota, kterého by přepnul na manuální ovládání a navedl ke zločincům. Tento robot by byl opět vybaven prostředky pro eliminaci zločinců. Tato varianta by také poskytla dozorcům možnost zavolat další pomoc a policii.

Bezpečnostní autonomní robot byl v době psaní této diplomové práce stále ve vývoji. Ovšem nebyl již pouze na „papíře“, ale blížil se svému dokončení. Aktuální podoba autonomního robota je vidět na Obrázek 1.



Obrázek 1 – Autonomní robot konstruován na FAI UTB ve Zlíně

Autonomní robot jako celek se skládá z několika funkčních subsystémů, jak softwarových, tak hardwarových. Schématické znázornění propojení jednotlivých subsystému je možné si prohlédnout v příloze P I.

### 1.1 Softwarové vybavení robota

Pro řízení a ovládání robota jako celku by mělo být použito aplikace na platformě .NET Framework, která poběží přímo na operačním systému Windows, který bude v počítači robota nainstalován. Takto navrhnuté řešení umožní jednoduchý přímý zásah do ovládání a konfigurace robota v případě, že by došlo ke ztrátě spojení mezi robotem a jeho obsluhou, která by se snažila komunikovat s robotem bezdrátově.

Aplikace by standardně měla běžet v autonomním režimu, tedy v cyklu, kdy:

1. zkontroluje své okolí z hlediska zabezpečení budovy,
2. naskenuje a zpracuje údaj z laserového senzoru, pro měření vzdálenosti překážek,
3. pomocí neuronové sítě vyhodnotí, kterým směrem se dále může vydat,
4. zvoleným směrem bude pokračovat.

Tento hlavní cyklus programu mohou ovlivnit přerušení ze sekundárních senzorů, jako jsou infračervená čidla, případně přerušení obsluhou kvůli manuálnímu ovládání nebo konfiguraci robota.

Jak již bylo zmíněno v předchozím odstavci, aplikaci robota bude možno přepnout z autonomního pohybu na manuální dálkové ovládání.

## **1.2 Hardwarové vybavení robota**

Autonomní robot bude vybaven počítačem na platformě Intel Atom, který by měl zajistit dostatečně výkonný základ pro běh řídicí aplikace robota. Vzhledem k řídicí aplikaci, která bude postavena na vyšším programovacím jazyku z důvodů možností, které má poskytovat (zpracování obrazu, zpracování údajů z laserového senzoru, ...), je pro provoz robota nezbytný počítač s vysokým výkonem a se standardním operačním systémem. Aby bylo těchto technologických požadavků možno dosáhnout, je aplikace implementována na platformě .NET Framework a z toho důvodu bude počítač v autonomním robotu vybaven i operačním systémem Windows.

## **1.3 Použité senzory**

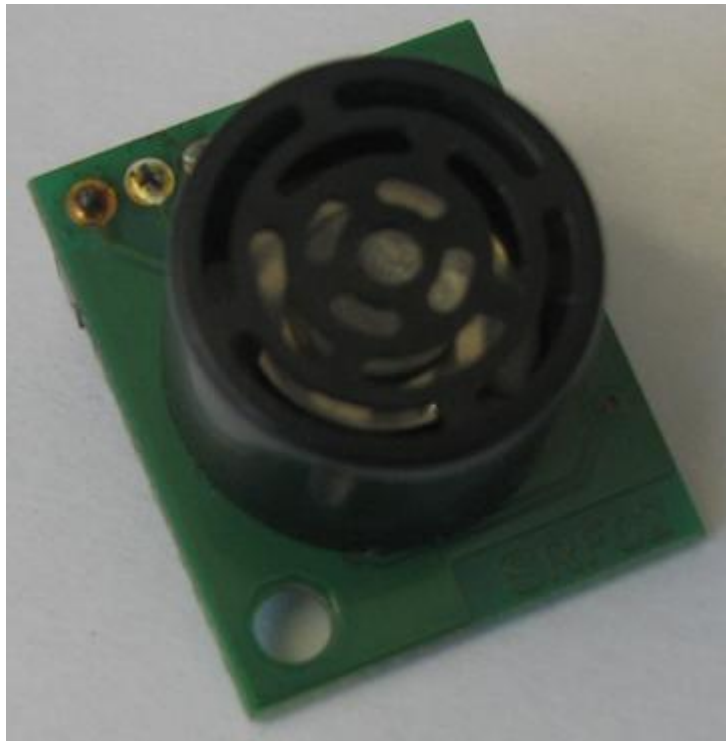
Na robotovi je použito několik typů senzorů, které by měly robotovi umožňovat bezpečný pohyb v prostoru bez kolizí. V průběhu vývoje robota došlo, nebo v nejbližší době dojde, k několika změnám v jeho vybavení. Hlavní změna se týká laserového senzoru z důvodu přechodu na technologicky lepší a spolehlivější zařízení.

### **1.3.1 Infračervená čidla**

Po obvodu celého autonomního robota jsou umístěna bezpečnostní infračervená čidla. Tato čidla slouží jako poslední záchrana v případě, kdy se v těsné blízkosti robota vyskytne nějaká překážka. V případě aktivace některého z těchto čidel dojde k okamžitému přerušení pohybu robota, aby se zamezilo kolizi. Tato kolize by mohla vést buď k poškození robota samotného, nebo k poškození okolního majetku. V nejhorším případě by mohlo dojít k nechtěnému zranění člověka. Tato čidla reagují do maximální vzdálenosti 20 centimetrů v okolí autonomního robota.

### 1.3.2 Ultrazvukové senzory

Autonomní robot je vybaven ultrazvukovými senzory typu SRF02. Tyto senzory tvoří subsystém, který podporuje svou činností laserový senzor. Využitím těchto senzorů bylo zamýšleno vyřešení hlavních nedostatků použitého laserového senzoru. Ultrazvukové senzory by měly pokrýt slepé pole viditelnosti laserových senzorů. Použité ultrazvukové senzory dosahují měřitelné vzdálenosti až 6 metrů a jsou aktivovány a použity právě ve chvíli, kdy dojde k nespolehlivému měření laserového senzoru, které může být způsobeno špatným odrazem laserového paprsku od skla, vodní hladiny či jiného prostředí, které láme světelný paprsek.



Obrázek 2 – Vybava robota – ultrazvukový senzor

Princip měření tohoto ultrazvukového senzoru je založen na měření času mezi vysláním signálu a jeho zpětném přijetím po odrazu o překážku. Při měření pomocí ultrazvukových technologií je využíváno zvukových vln, které nijak neovlivňují měření laserového senzoru.



### 1.3.3 Laserový senzor (aktuální)

Jako hlavní prostředek pro orientaci autonomního robota v prostoru bude sloužit laserový skener SICK LMS 400. Tento laserový skener patří do kategorie laserových bezkontaktních měřicích přístrojů. Slouží k měření vzdáleností a zjišťování tvaru ve velkém rozsahu. Měření je realizováno pomocí červeného laseru o vlnové délce 650 nanometrů. Tento laserový senzor má však i svá technologická omezení. Nejmenší možná měřená vzdálenost je od 0,7 metrů a maximální rozlišitelná vzdálenost je 3 metry. Úhel měření, který je možno snímat, je nastaven na 70°. Konstrukce tohoto laserového skeneru umožňuje nastavit rozsah měření od 55° do 125°.



Obrázek 3 – Výbava robota – laserový senzor (aktuální)

Tento laserový skener není pro použití pro autonomního robota zrovna nejvhodnějším, právě z důvodu jeho maximálního měřitelného dosahu, stejně tak z důvodu jeho maximálního měřitelného úhlu. Primárně byl tento laserový skener navržen pro použití ve výrobním sektoru na automatických výrobních linkách například pro kontrolu naplnění krabic nebo jiných pevných obalů, ve kterých se skladují výrobky.

### ***1.3.3.1 Výstup z laserového senzoru***

Laserový skener SICK LMS 400 komunikuje s okolními prvky systému přes Ethernetové rozhraní a skener má svou vlastní IP adresu. Komunikace je realizována pomocí protokolu UTP. Komunikace s laserovým skenerem probíhá podle klasického modelu „dotaz – odpověď“.

Komunikace je zahájena nejdříve několika dotazy ze strany řídicí aplikace na stav skeneru, jestli je v pořádku nebo došlo nějakému poškození či chybě. Po navázání spojení a po inicializační komunikaci laserový skener začne posílat naměřené hodnoty v opakujících se cyklech.

Všechny instrukce, pomocí kterých probíhá komunikace mezi laserovým skenerem a počítačem mají hexadecimální tvar (například: 0F34h)

### **1.3.4 Laser senzor (plánovaný)**

Aktuálně použitý laserový senzor je však pouze dočasným řešením pro získávání údajů o okolním prostředí pro autonomního robota. Svými technologickými vlastnostmi není pro využití na autonomním robotovi nejvhodnější, ale dočasně pouze dostačující.

V době realizace a implementace knihovny s umělou inteligencí pro řízení robota probíhalo teprve výběrové řízení pro dodavatele nového a ve všech směrech výkonnějšího laserového skeneru. Jedná se o laserový skener SICK produktové řady LMS500 se středním dosahem. Tento laserový skener je primárně určen do vnitřních prostor budov a stupeň krytí má podle standardu IP 65. Maximální měřitelná vzdálenost je až 80 metrů s maximálním pozorovacím úhlem až 190°. Tento laserový skener dosahuje frekvence snímání až 100 Hz a využívá infračerveného světla o vlnové délce 905 nanometrů.



Obrázek 4 – Vybava robota – laserový senzor (plánovaný)

Tento laserový skener by měl být pro použití ve vnitřních prostorech dostačující a hlavní předpoklad je, že z něj budou získávána mnohem kvalitnější data, než z aktuálního senzoru, viz 4. kapitola, kde jsou problémy aktuálního laserového senzoru popsány.

## 2 NEURONOVÉ SÍTĚ

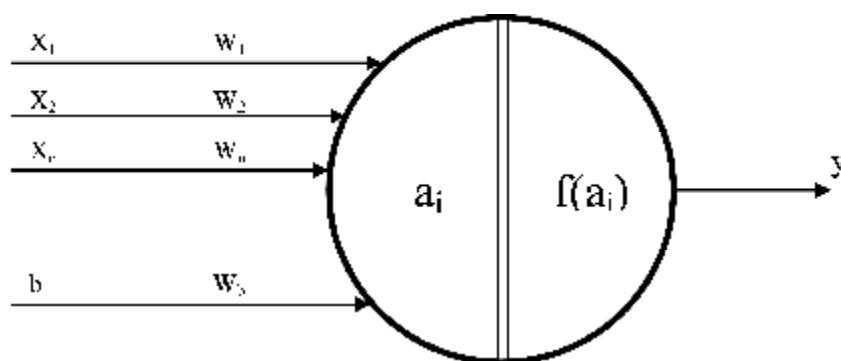
Umělá inteligence je ve své podstatě tvořena ve většině případů pomocí neuronových sítí. Neuronové sítě umožňují dle specifikovaných pravidel rozpoznávat vzory, dělit vzory do skupin, předpovídat budoucí vývoj, na základě získaných dat v minulosti (například vývoj ceny ropy za barel), aproximovat funkce, hledat lokální maxima nebo minima a mnohé další využití, se kterými se můžeme setkat v každodenním životě téměř na každém kroku. Úkolem neuronové sítě a umělé inteligence je napodobovat předpokládané chování na základě poskytnutých informací, tím je myšleno především rozhodování. U neuronových sítí je kladen důraz na výkon a rychlost rozhodování za cenu minimálních nároků na sdílené zdroje.

Každá neuronová síť se skládá z neuronů. Tyto neurony jsou vždy uspořádány do topologie dle typu neuronové sítě například do kruhu nebo do vrstev.

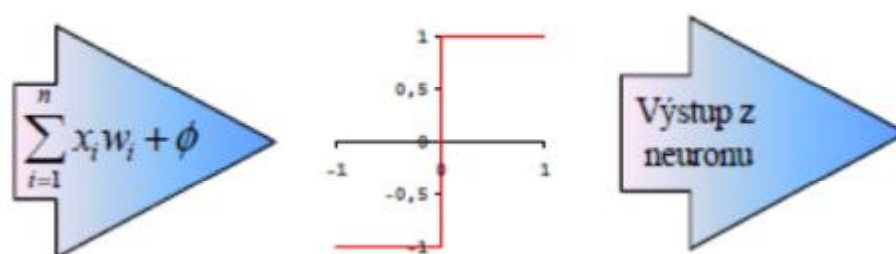
Paměť neuronové sítě je tvořena pomocí vah mezi jednotlivými neurony. Existují dva možné způsoby, jak naučit neuronovou síť vzorům (nastavit váhy mezi neurony), které má rozeznávat:

1. **Učení bez učitele** – probíhá tak, že jsou neuronové sítě předloženy vzory, které si má zapamatovat a kterým bude přiřazovat následně vzory, které má neuronová síť rozeznat. Jednoduše řečeno se jedná o rozřazování vstupních vzorů do skupin a tyto skupiny tvoří základní vzory zadané neuronové síti během učení.
2. **Učení s učitelem** – na rozdíl od učení bez učitele, tady je neuronové síti předán ke každému vstupnímu vzoru i očekávaný výsledek. Učení následně probíhá tak, že se vloží na vstup učený vzor a aktivuje se neuronová síť. Výsledek neuronové sítě se porovná s referenčním výsledkem a na základě rozdílu mezi těmito údaji se případně upravují váhy mezi neurony.

Obrázek 5 znázorňuje obecné schéma neuronu. Vstupem do neuronu je vždy výstup předchozího neuronu vynásobený vahou. Vnitřní potenciál neuronu se pak skládá ze všech vstupů do neuronu, tedy ze součtu všech vážených výstupů předcházející vrstvy neuronové sítě a k této hodnotě bývá přičten práh „b“ (standardně hodnota 1), který je také vážený.



Obrázek 5 – Obecné schéma neuronu



Obrázek 6 – Matematické znázornění neuronu (převzato z [1])

Definice vnitřního potenciálu neuronu

$$a = \sum_{i=1}^{n+1} x_i w_i$$

(3 – 1)

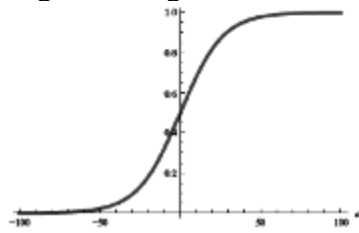
Definice vnitřního potenciálu neuronu s prahem uvnitř neuronu

$$a = \sum_{i=1}^{n+1} x_i w_i + b w_b$$

(3 – 2)

Pro přenos vnitřního potenciálu uvnitř neuronu na výstup neuronu se používá transformační funkce, jejímž cílem je transformovat hodnotu vnitřního potenciálu do zvoleného intervalu. Standardně se využívá intervalu  $\langle 0;1 \rangle$  nebo  $\langle -1;1 \rangle$ . Je možno využít následujících přenosových funkcí:

- Logistická sigmoida

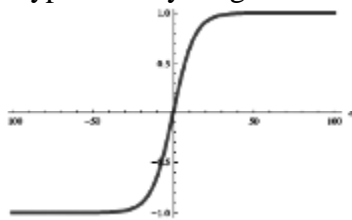


$$f(a) = \frac{1}{1 + e^{-\lambda a}}$$

$\lambda$  - směrnice

(3 – 3)

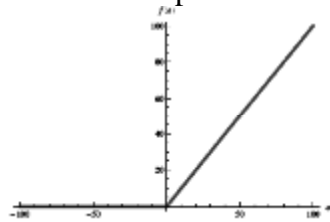
- Hyperbolický tangens



$$f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

(3 – 4)

- Funkce Perceptron



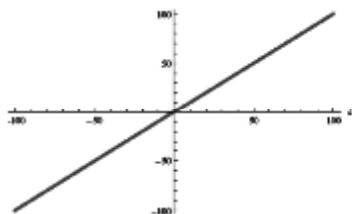
$$f(a) = a$$

$$pro\forall a > 0, jinak$$

$$f(a) = 0$$

(3 – 5)

- Lineární

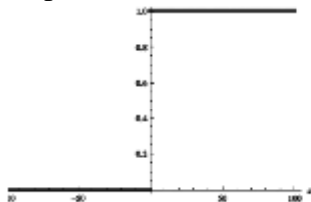


$$f(a) = ka$$

$k$  - směrnice přímky

(3 – 6)

- Binární (unipolární nebo bipolární), skoková, Heavisidova (lze pohybovat i s prahem):



$$f(a) = 1$$

$$pro\forall a > 0, jinak$$

$$f(a) = 0(nebo -1)$$

(3 – 7)

## 2.1 Základní druhy neuronových sítí

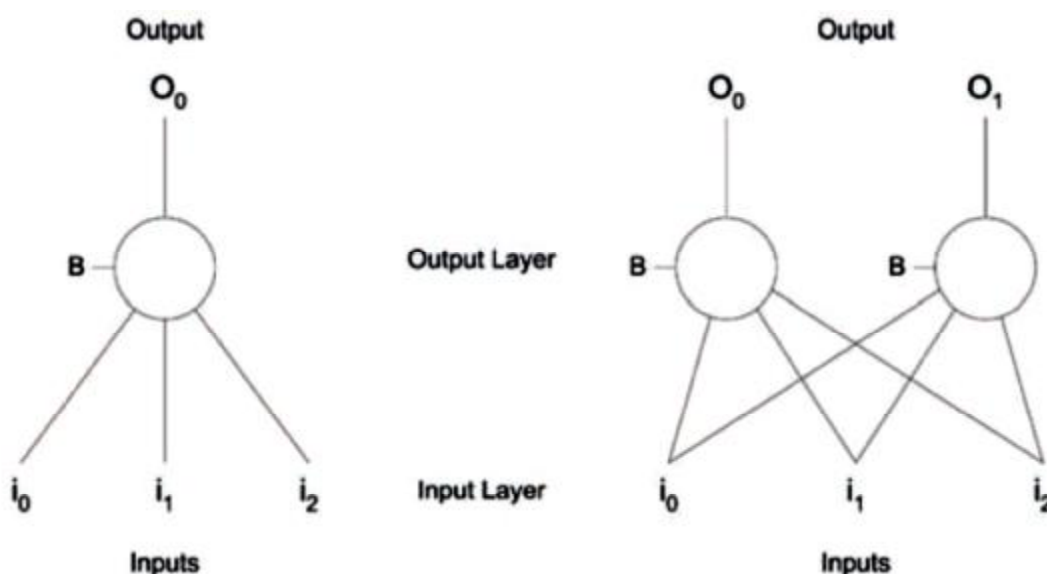
### 2.1.1 Perceptron

#### 2.1.1.1 Určení a typ sítě

Tato neuronová síť umí řešit pouze lineárně separabilní problémy (více viz [1]). Počet výstupních skupin, do kterých bude tato neuronová síť schopna rozdělit vstupní vzory, záleží na počtu neuronů ve výstupní vrstvě. Použití této sítě je vhodné v aplikacích, kde potřebujeme vstupy rozdělit do kategorií podle podobnosti tvaru nebo podobného umístění. V závislosti na implementaci je možné v případě potřeby nebo v případě velmi odlišných vstupních dat vytvářet za běhu i nové výstupní skupiny vzorů. Jednalo se o první neuronovou síť, která byla schopna učení. Učení této sítě probíhá bez učitele pomocí přenastavování vah. U Perceptronu se využívá skokové přenosové funkce

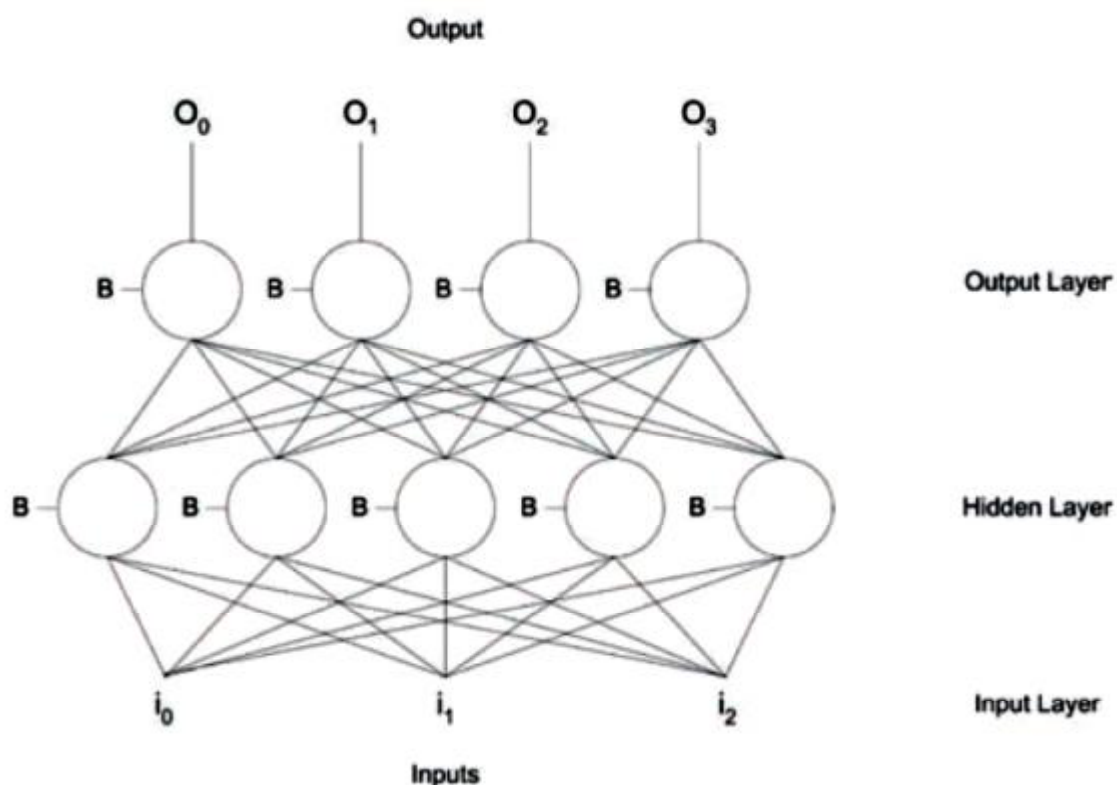
#### 2.1.1.2 Schéma sítě

Jedná se o vícevrstvou neuronovou síť. Tato neuronová síť má dvě pevné vrstvy. Jednu vstupní, do které vkládáme data, a druhou výstupní. U výstupní vrstvy je možné přenastavovat váhy. Vazby mezi neurony jsou v této síti realizovány pouze jednosměrně mezi vrstvami a jsou vždy propojeny všechny neurony se všemi v následující vrstvě.



Obrázek 7 – Ukázka jednovrstvé neuronové sítě Perceptron se třemi vstupy a jedním nebo dvěma výstupy (převzato z [2])

Časem byl implantován i vícevrstvý Perceptron, který umožnil implementovat podrobnější možnosti separability.



Obrázek 8 – Vícevrstvý Perceptron (převzato z [2])

U uvedených schémat s příklady neuronové sítě Perceptron označují symboly „ $i_k$ “ vstupy do neuronové sítě, symboly „ $O_k$ “ označují výstupy z neuronové sítě. Index „ $k$ “ označuje, o kolikátý neuron ve vrstvě se jedná. Všechny vstupy do neuronů (mimo vstupní vrstvy) jsou váženy a k sumě vážených výstupů z nižší vrstvy je přičítán práh ( $B$ , bias).

## 2.1.2 Adaline a Madaline

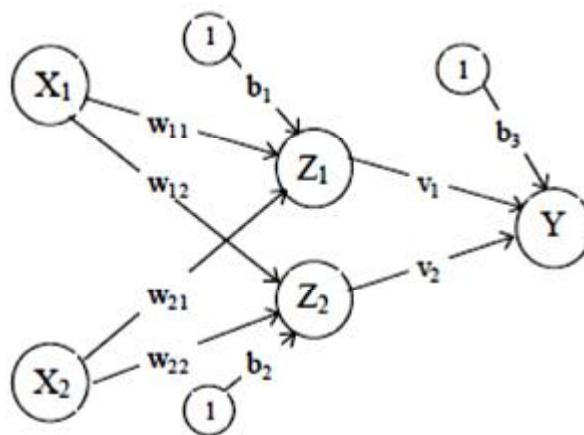
### 2.1.2.1 Určení a typ sítě

Adaline je základním stavebním prvkem pro neuronovou síť Madaline. Pod pojmem Adaline si můžeme představit jinou formu jednoho neuronu z neuronové sítě Perceptron. Rozdíl mezi Adaline a neuronem Perceptron je v tom, že Adaline používá lineární pravidla. Pro vstup i výstup používají bipolární pravidla a stejně jako u neuronové sítě Perceptron každý neuron Adaline obsahuje svou prahovou hodnotu.



### 2.1.2.2 Schéma sítě

Pro neuronovou síť Madaline představuje Adaline základní stavební prvek (neuron). Jednoduchá architektura sítě Madaline je zobrazena na Obrázek 9. (čerpáno z [12])



Obrázek 9 – Madaline se dvěma skrytými neurony Adaline a jedním výstupním neuronem Adaline (převzato z [12])

U této neuronové sítě je výhoda ve velmi jednoduché hardwarové realizaci právě díky použitému lineárnímu bipolárnímu pravidlu uvnitř neuronu. Vzhledem k podobě mezi neurony Adaline a Perceptron, je i použití neuronové sítě Madaline velmi podobné neuronové síti Perceptron.

### 2.1.3 Feedforward s algoritmem Backpropagation

#### 2.1.3.1 Určení a typ sítě

Jedná se o neuronovou síť, která principiálně funguje podobně jako neuronová síť Perceptron. Je tu ale hlavní rozdíl v tom, že tato neuronová síť používá k učení metodu s učitelem, tedy k naučení této neuronové sítě jsou potřeba v trénovací množině jak samotná data, tak i očekávaný výstup neuronové sítě. Na základě rozdílu mezi očekávaným výstupem a reálným výstupem na zadaný vstup je možno tuto síť naučit našim potřebám.

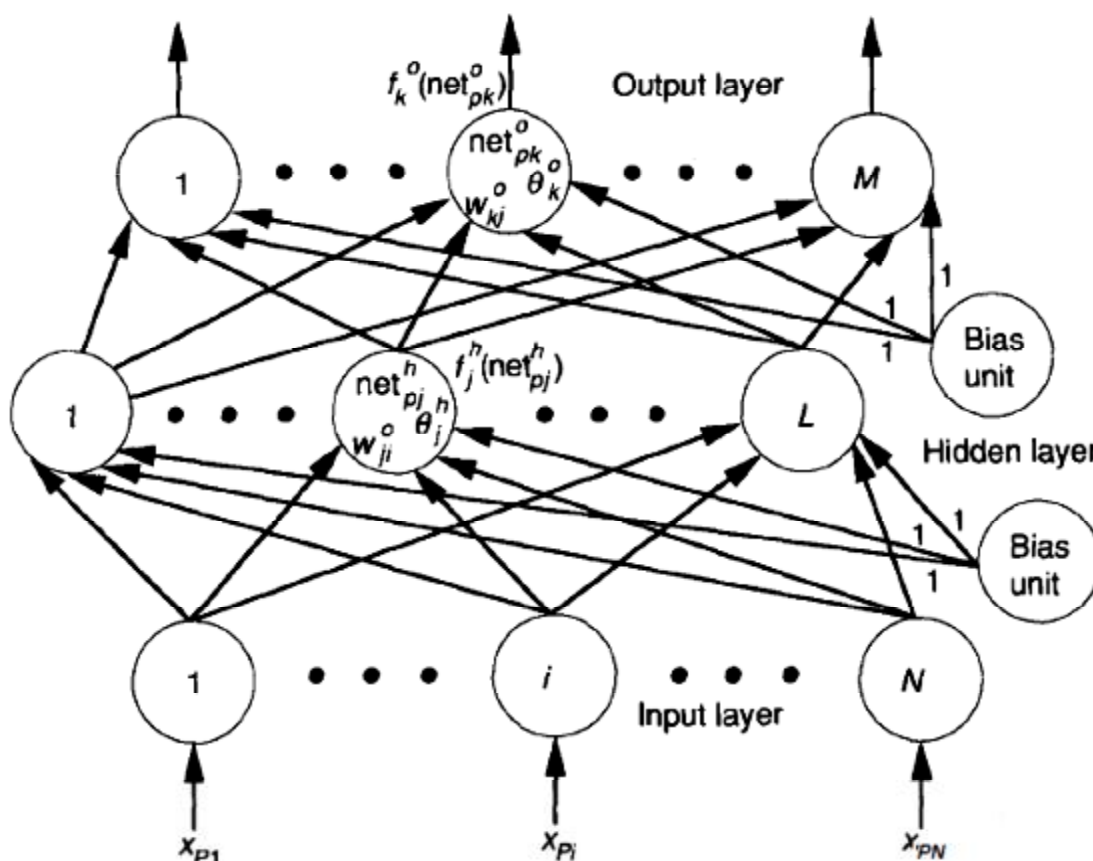
Tato neuronová síť je použita právě v místech, kde potřebujeme předem rozdělit vstupní vzory do konkrétních výstupních skupin a potřebujeme mít tyto skupiny předem nějakým způsobem označené.

Pro zopakování by se dalo říci, že Perceptron pouze dělí vstupní vzory do skupin na základě podobnosti vzorů v dané skupině, kdežto neuronová síť s algoritmem

Backpropagation přiřazuje vstupní vzory na základě podobnosti se vzory z trénovací množiny přímo do předem definovaných skupin. Algoritmus Backpropagation získal své jméno díky jeho použití. Data standardně procházejí neuronovou sítí ze vstupu na výstup, kdežto během učení se lokální chyba a její oprava šíří z výstupu směrem ke vstupu.

### 2.1.3.2 Schéma sítě

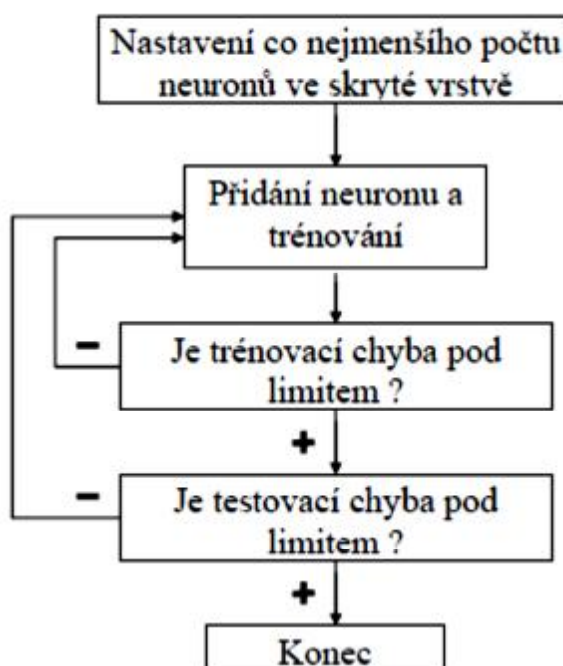
Tato neuronová síť je tvořena vrstvami. Může obsahovat více vrstev a všechny neurony v jedné vrstvě jsou vždy spojeny se všemi neurony v následující vrstvě. Vrstvy jsou propojeny vždy směrem ze vstupu na výstup a nikdy nejsou propojeny neurony uvnitř jedné vrstvy. Během učení se váhy neuronů upravují směrem z výstupu na vstup. Tímto způsobem je šířena lokální chyba každého výstupního neuronu a je možné na jejím základě pozměnit váhy všech neuronů, aby byla konečná odezva neuronové sítě pro daný vzor shodná s požadovanou referenční hodnotou.



Obrázek 10 – Schéma neuronové sítě s algoritmem Backpropagation  
(převzato z [11])

Kolmogorův teorém, který se týká vícevrstvých sítí, říká, že maximálně dvě skryté vrstvy v neuronové síti jsou dostatečné pro řešení jakéhokoli problému. Ohledně určení přesného počtu neuronů ve skrytých vrstvách neexistuje žádné striktní pravidlo, které by říkalo, kolik má být neuronů. Existují pouze vzorce, pomocí kterých se dá spočítat počet neuronů, které by měly být dostatečné pro řešení problému. Bez testování ale nelze na tento výpočet přesně spoléhat. Tyto vzorce jsou přesněji rozebrány v praktické části diplomové práce v kapitole 8 a doporučený postup ohledně nastavení nejvhodnějšího počtu neuronů ve skryté vrstvě je znázorněn schématem na Obrázek 11.

Zda jsou potřeba dvě nebo jedna skrytá vrstva, a jaký je potřeba přesný počet neuronů ve skrytých vrstvách, vždy záleží na konkrétním problému a testování úspěšnosti neuronové sítě.



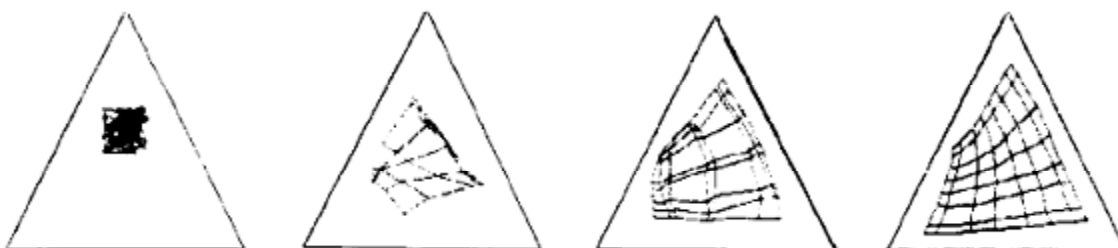
Obrázek 11 – Schéma pro postup volby správného počtu neuronů ve skryté vrstvě  
(převzato z [1])

## 2.1.4 Kohonenova mapa – samoorganizující se síť

### 2.1.4.1 Určení a typ sítě

Kohonenova mapa pracuje na principu vyhledávání závislostí v datech v trénovací množině. V podstatě se jedná o shlukovou analýzu dat s podobnými vlastnosti. Tímto

chováním jsou Kohonenovy mapy, které jsou svým chováním přímo předurčeny pro aplikaci v problémech jako je rozhodování, rozlišování nebo třídění objektů. Často se využívají i v oblasti rozpoznávání řeči nebo zpracování psaného textu do elektronické podoby. Výhodou této neuronové sítě je, že není potřeba mít o zpracovávaném signálu hluboké znalosti a SOM si je automaticky zpracuje a roztrídí. Tato neuronová síť nepotřebuje ke svému učení učitele.



Obrázek 12 – Aplikace Kohonenovy mapy, sekvence diagramů ilustrující evoluci mapy podle předlohy (převzato z [11])



Obrázek 13 – Aplikace Kohonenovy mapy, rozpoznání vzoru (převzato z [11])



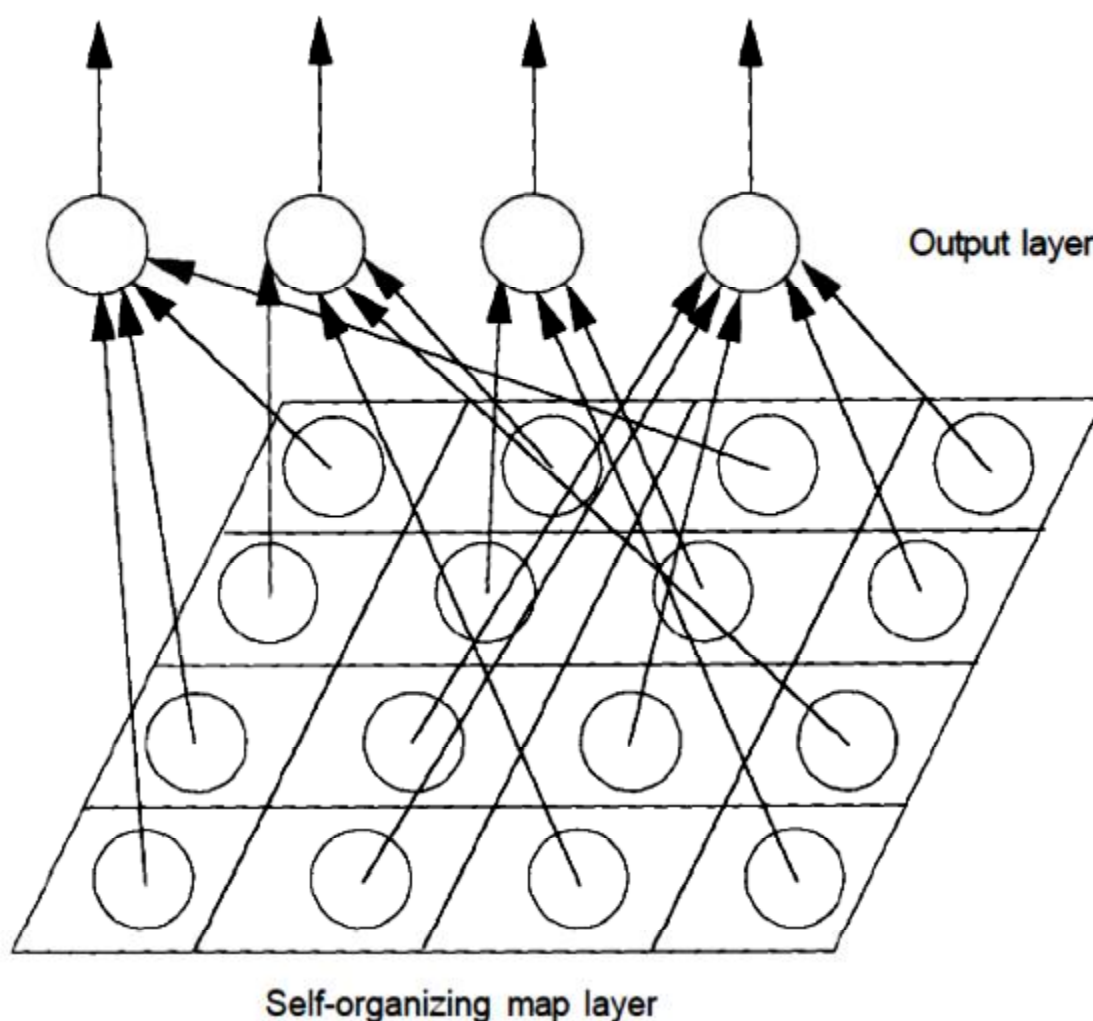
Obrázek 14 – Aplikace Kohonenovy mapy, rozpoznání vzoru, přeneseně například rozpoznání cesty (převzato z [11])

Vzhledem ke svému potenciálu a zaměření je tato neuronová síť také vhodným prvkem pro aplikaci řízení autonomního robota. Rozpoznání trasy v mapě s překážkami se téměř

shoduje s použitím této neuronové sítě u rozpoznání písma, které je na Obrázek 14. Stačí si představit, že jednotlivé rohy písmene „K“ jsou body, které musí autonomní robot navštívit. Pomocí Kohonenovy mapy je možné získat přesnou trasu se souřadnicemi, kterou autonomní robot musí projet.

#### 2.1.4.2 Schéma sítě

Kohonenova síť má obvykle jednu vrstvu představující plošné uspořádání neuronů. V této vrstvě jsou vždy všechny neurony propojeny se všemi ostatními a pomocí vah se udržuje informace o trénovací množině. Rychlost této neuronové sítě je dána počtem neuronů, obvykle se používá  $17 * 17$  neuronů s odchylkou 8 neuronů. Neurony v této neuronové síti neobsahují přenosovou funkci, ale zpracovávají vstupní vzor na základě vzdálenosti do vzoru, který byl naučen (ve smyslu Hemmingovy vzdálenosti). Čerpáno z [1].



Obrázek 15 – Schéma samoorganizující se mapy (převzato z [11])

## 2.1.5 Hopfieldova síť

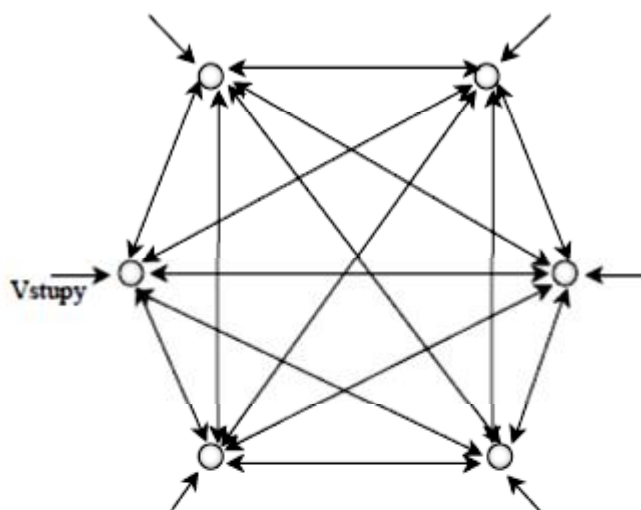
### 2.1.5.1 Určení a typ sítě

Hopfieldova neuronová síť se ve většině aplikací používá ke klasifikaci nebo k optimalizaci. Tato neuronová síť má autoasociativní paměť, což v praxi znamená, že je tuto neuronovou síť možné použít pro rozeznávání poškozených vzorů. Například rozpoznání poškozeného textu či obrázku. Tato neuronová síť pracuje podobně jako lidský mozek, který také disponuje hlavně autoasociativní pamětí. Na základě předchozí zkušenosti je člověk schopný rozeznat poškozený text nebo obrázek. U Hopfieldovy sítě je princip podobný, protože zkušenosti předáváme síti během učení a základní vzory, které bude Hopfieldova síť rozeznávat, ji naučíme.

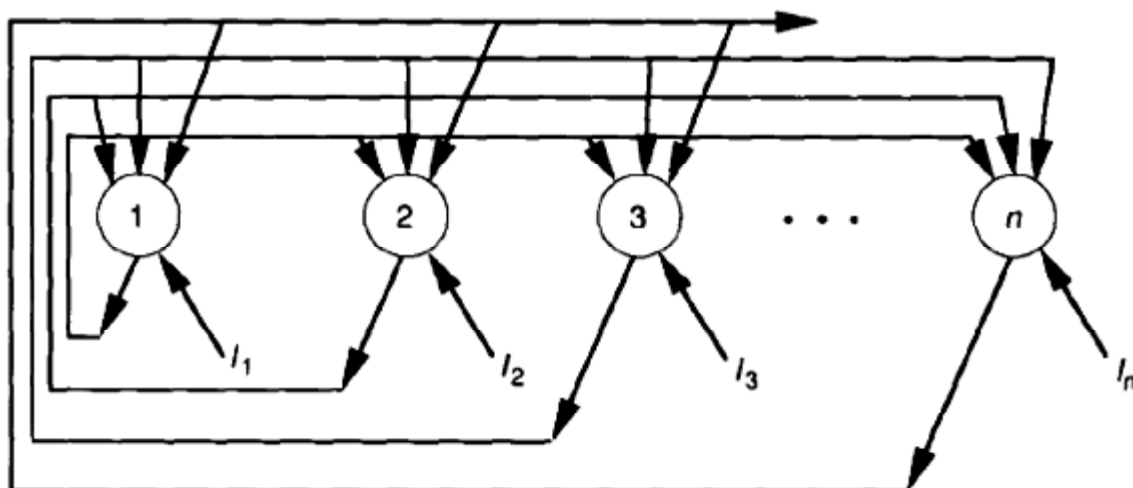
Učení této neuronové sítě probíhá bez učitele. Hopfieldově neuronové síti předložíme během učení vzory, které chceme, aby si zapamatovala. Při vyhodnocení poškozeného vzoru je žádoucí, aby Hopfieldova neuronová síť rekonstruovala vzor a přiřadila ho k některému z naučených na základě podobnosti.

### 2.1.5.2 Schéma sítě

Hopfieldova síť je pouze jednovrstvá a, jak již bylo zmíněno, využívá algoritmu pro učení bez učitele. Tato jednovrstvá neuronová síť je specifická tím, že ve vrstvě je propojen každý neuron s každým a využívá přenosové funkce skokového charakteru (čerpáno z [1]). Schéma Hopfieldovy neuronové sítě je vidět na Obrázek 16.



Obrázek 16 – Hopfieldova neuronová síť (převzato z [1])



Obrázek 17 – Hopfieldova síť – jiné znázornění (převzato z [11])

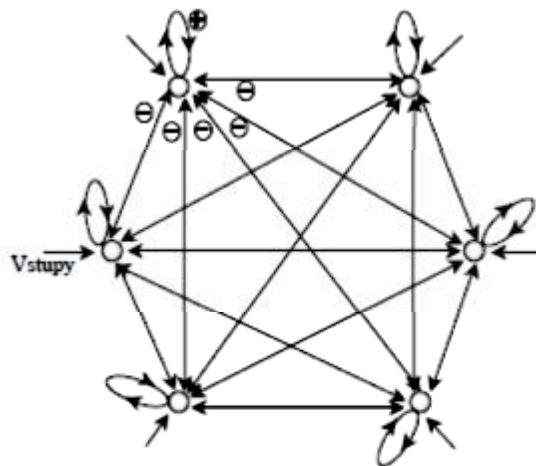
## 2.1.6 CLN

### 2.1.6.1 Určení a typ sítě

Síť CLN (Competitive Learning Network) je velmi podobná Hopfieldově neuronové síti. Jedná se také o neuronovou síť s jedinou vrstvou, kde jsou neurony uvnitř vrstvy propojeny každý s každým. Jediný rozdíl v topologii sítě je takový, že každý neuron má navíc zpětnou vazbu i sám na sebe. To v konečném použití neuronové sítě znamená, že každý neuron ve výsledku podporuje navíc sama sebe. Sítě tohoto typu bývají používány převážně k nalezení tříd v množině vstupních dat. Během vyhodnocování se využívá Hammingovy vzdálenosti a vítězný neuron je ten, který představuje třídu, ke které má vstupní vzor nejmenší vzdálenost. Učení této neuronové sítě probíhá pomocí algoritmu bez učitele (čerpáno z [1]).

### 2.1.6.2 Schéma sítě

Jak již bylo zmíněno v předchozí kapitole, síť CLN se od Hopfieldovy sítě odlišuje pouze v topologii a to tak, že kromě toho, že je každý neuron spojený s každým uvnitř vrstvy, má také každý neuron navíc zpětnou vazbu sám na sebe, viz Obrázek 18.



Obrázek 18 – Neuronová síť CLN (převzato z [1])

## 2.1.7 Neuronová síť BAM

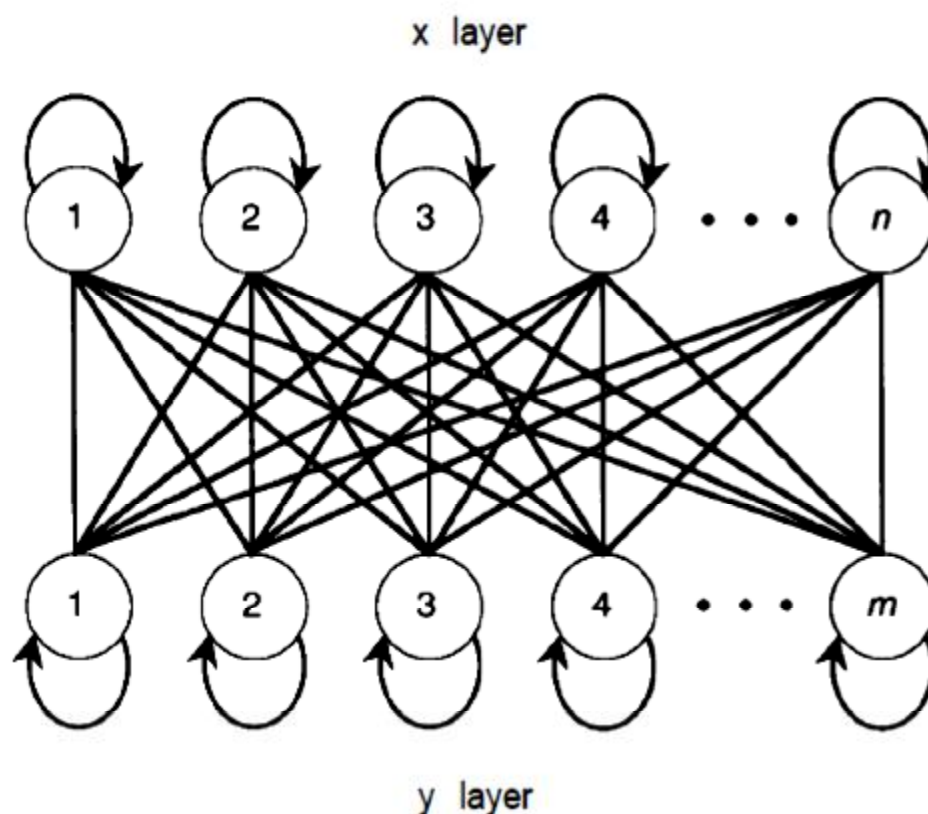
### 2.1.7.1 Určení a typ neuronové sítě

Neuronová síť BAM (Bidirectional Associative Memory) obsahuje dvousměrnou asociativní paměť, sloužící pro heteroasociativní přiřazování vzorů. Na rozdíl do Hopfieldovy sítě, která je autoasociativní, nedokáže z poškozeného vzoru rekonstruovat původní vzor. Tento druh sítě se vyznačuje tím, že vstupnímu vzoru nepřirazuje originální, ale pouze asociovaný vzor. Neurony jsou stejné jako v Hopfieldově síti.

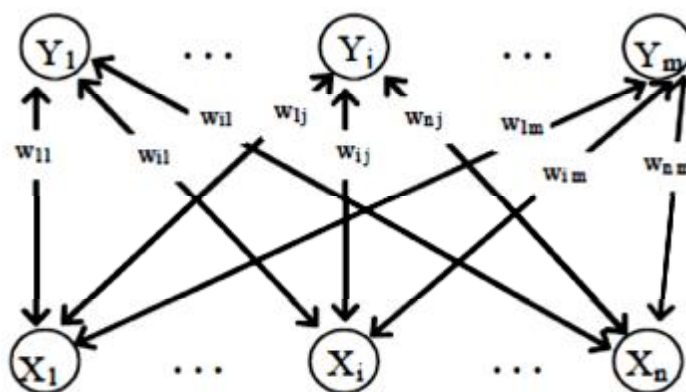
### 2.1.7.2 Schéma sítě

Topologie neuronové sítě BAM je dvouvrstvá, kdy každý neuron v jedné vrstvě je propojený s každým neuronem v druhé vrstvě. Propojení neuronů je obousměrné.





Obrázek 19 – Neuronová síť BAM (převzato z [11])



Obrázek 20 – Jiné schéma neuronové sítě BAM s dvousměrnou asociativní pamětí (převzato z [13])

## 2.2 Vhodné neuronové sítě pro řízení autonomního robota

Z přehledu základních neuronových sítí v předešlé kapitole je snadné vybrat, které neuronové sítě by byly vhodné pro řešení řízení autonomního robota pomocí umělé

inteligence. Z těchto neuronových sítí bylo potřeba vybrat takové, které by umožnily naučit neuronovou síť správnému chování, tedy aby bylo možné naučit síť správným výstupům v závislosti na vstupu. To se jedná o učení s učitelem. Nebo takovou síť, která je schopná dodat hlavní řídící aplikaci informace o trase, kterou robot musí ujet, například pomocí souřadnic, vzdáleností mezi souřadnicemi a úhlem otočení v bodech cesty.

Na základě výše napsaných podmínek na neuronovou síť jsou vyhovující je možné za vyhovující sítě považovat neuronovou síť s algoritmem učení Backpropagation a Kohonenovu mapu (samoorganizující se síť).

Obě tyto sítě splňují požadavky na řešení řízení autonomního robota a daný problém vyřeší. Obě tyto sítě byly už pro podobné problémy použity. Před implementací bylo potřeba důkladně nastudovat, jak dané neuronové sítě pracují a jak je možné využít jejich potenciál, aby byl výsledek co nejefektivnější. Princip a vlastnosti těchto dvou neuronových sítí jsou popsány v předchozí kapitole.

Neuronová síť s algoritmem učení Backpropagation je pro tento problém vhodná hlavně z důvodu, že její učení probíhá s učitelem. To znamená, že my můžeme tuto neuronovou síť naučit, jakou má mít odezvu podle několika základních vstupních vzorů a při použití naučené sítě bude vždy odezva taková, podle toho, kterému vzoru bude vstupní vzor nejpodobnější.

Kohonenova mapa je vhodná při potřebě plánování celé trasy. Ovšem k tomuto řízení je potřeba znát celé prostředí, ve kterém se bude robot pohybovat, včetně polohy překážek, které by autonomní robot mohl při svém pohybu potkat. Tato situace by byla řešitelná například tak, že by robot po vpuštění provedl seznamovací jízdu a nasnímané obrazy by si uložil do mapy. Následně by už stačilo pouze znát aktuální pozici robota a určit, které místo v mapě bude označeno jako cíl. Laserový skener by potom při průchodu prostředím už sloužil pouze jenom jako pojistka, pokud by se v prostoru vyskytla nová překážka, tak by mohlo dojít k přerušení programu a naplánovat, jakým způsobem bude překážka překonána.

### **2.3 Aplikovaná neuronová síť pro realizovaného robota**

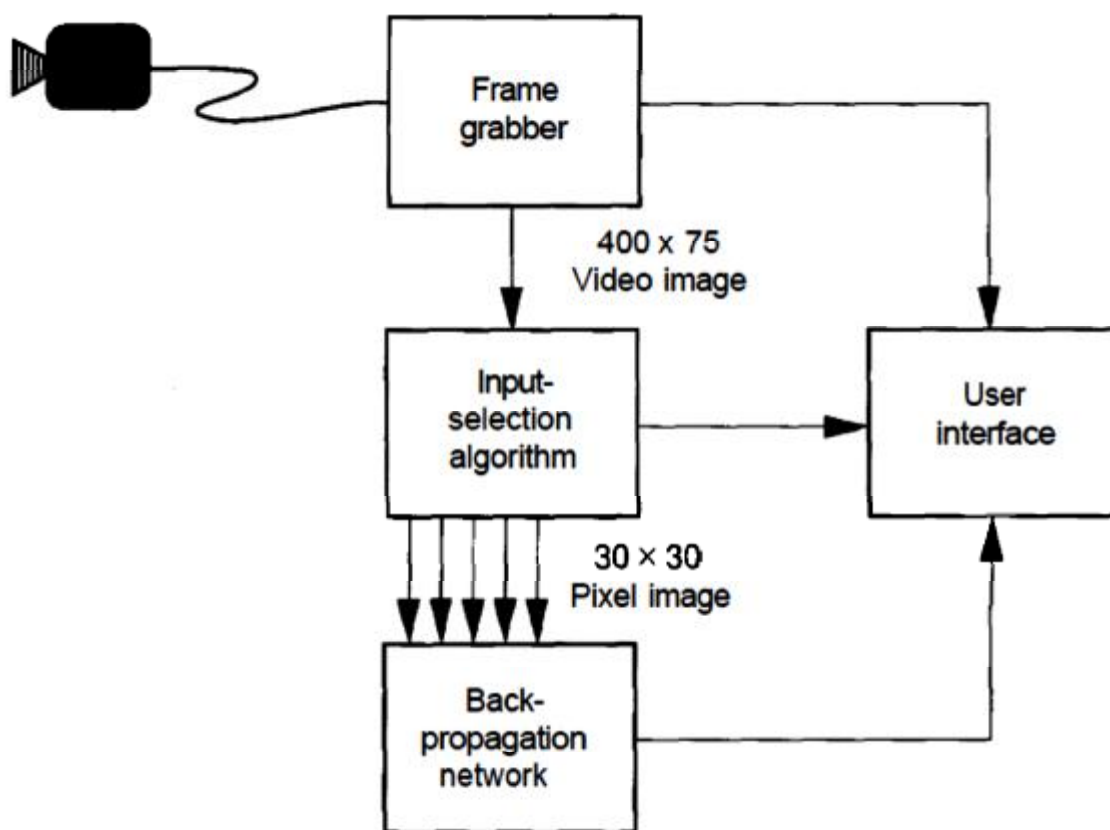
Vzhledem k prostředkům získávání informací, které autonomní robot nabízí, byla zvolena neuronová síť s algoritmem učení Backpropagation. Cílem bylo vytvořit neuronovou síť, která by vždy reagovala v danou chvíli na překážky, které má aktuálně před sebou. Pro

autonomního robota bezpečnostní služby je důležité náhodně jezdit po objektu a vyhýbat se překážkám v trase.

Autonomní robot disponuje laserovým skenerem. Údaje z toho skeneru jsou s drobnými úpravami (převod do desítkové soustavy, ořezání nadbytečných informací) předávány přímo neuronové síti na vyhodnocení. Autonomní robot už neuronové síti nepředává žádné informace o aktuální pozici nebo o ujeté vzdálenosti od posledního vyhodnocení směru. Z těchto důvodů je neuronová síť s algoritmem Backpropagation nejvhodnější, protože se rychle rozhoduje na základě aktuálně vložených dat na vstup neuronové sítě a nezávisle na předchozích výsledcích či aktuální pozice se data vyhodnotí.

Kohonenova mapa nebyla pro tuto realizaci umělé inteligence vhodná, právě z důvodu přesné specifikace jejího použití. Kohonenova mapa je vhodná v případě, pokud bychom znali prostředí a potřebovali bychom, aby si autonomní robot sám naplánoval cestu z bodu A do bodu B. K tomuto by bylo potřeba si uchovávat informace o prostředí, ve kterém se robot pohybuje. Tyto informace by bylo možné ukládat do mapy, ale aby tuto mapu bylo možné vytvořit, bylo by potřeba získávat od robota i informace o pozici nebo ujeté vzdálenosti a směru jízdy. Zadáním umělé inteligence bylo, aby robot nezávisle a náhodně jezdil po střeženém objektu, což Kohonenova mapa neumožňuje, protože jejím úkolem je právě plánování nejvhodnější trasy autonomního robota.

Následné schéma (Obrázek 21) použití neuronové sítě s algoritmem Backpropagation je velmi podobné případu, který byl potřeba řešit pro ovládání autonomního robota.



Obrázek 21 – Schéma zpracování obrazového vstupu do neuronové sítě  
(převzato z [11])

Jak je ze schématu patrné, pomocí nějakého snímacího zařízení je průběžně pořizován obraz. Tento obraz je zobrazen v uživatelském rozhraní a zároveň kopie tohoto obrazu je pomocí určité kompresní metody transformována na vhodná data, která by bylo možné předat neuronové síti. Aby bylo možné provést kontrolu transformovaného obrazu metodou porovnání, je tento transformovaný obraz opět poslán do uživatelského rozhraní. Transformovaný obraz je vyhodnocen neuronovou sítí a výsledek je předán k dalšímu zpracování. V rámci tohoto schématu je také zobrazen v uživatelském rozhraní.

V rámci této práce bylo hlavním úkolem implementovat právě tu část schématu, kde je neuronová síť. Analogicky je možné si domyslet jako snímací zařízení použitý laserový senzor na autonomním robotu. Vyhodnocený výsledek z neuronové sítě bude potom zpracovávat řídicí aplikace, která právě na základě výstupu z neuronové sítě, bude dále ovládat řízení autonomního robota. Součástí této diplomové práce budou také uživatelská rozhraní, která budou umožňovat otestovat implementovanou neuronovou síť právě tak, jak je znázorněno v předešlém schématu (Obrázek 21).

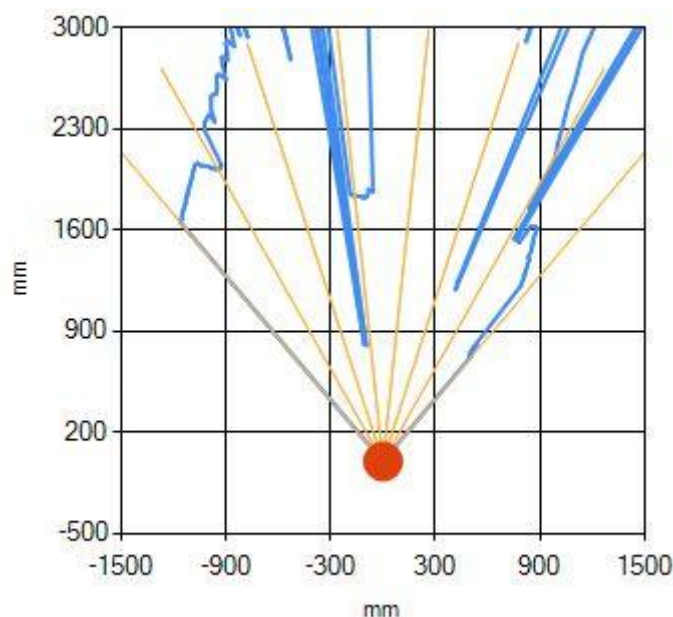
### 3 NEURONOVÁ SÍŤ A DATA ZE SENZORŮ

Před vložením dat z laserového senzoru do neuronové sítě je potřeba tato data zkontrolovat a celkově zjistit, jak daný laserový senzor funguje, aby bylo možné mu přesně přizpůsobit neuronovou síť pro vyhodnocení směru.

Jak bylo zmíněno v kapitole 1.3.3, aktuální laserový senzor vzhledem k jeho dosahu a rozptylu není nejvhodnější pro použití při řízení autonomního robota. Fakt, že použitý laserový skener je určen opravdu pouze pro měření na krátké vzdálenosti (do tří metrů) byl při testování velmi znát a naměřená data tím byla velmi ovlivněna.

V základním nastavení se laserový senzor choval tak, že pokud nezjistil překážku v daném směru měření do tří metrů, automaticky pro daný směr zaznamenal ve spektru hodnotu 0. Opravit tuto informaci byla minimální implementační záležitost, pokud budeme předpokládat, že pokud bude ve spektru hodnota 0, byl ve skutečnosti v daném směru dostatek místa. Změna hodnoty 0 na vyšší hodnotu byla volena z důvodu snadného přechodu na laserový senzor, který je pro tento účel určen, takže neuronové síti jsou už předkládána data, která jsou transformována tak, že vypadají jako data z nového laserového senzoru. Samozřejmě by mohla nastat situace, kdy před robotem nebude opravdu žádné místo, ale to pouze tehdy, pokud by obsluha umístila robota před spuštěním přímo do této pozice. Během řízení robota pomocí implementované neuronové sítě k takové situaci nedojde a neuronová síť by měla vždy včas dát řídicímu systému pokyny, aby bylo možné se překážce vyhnout.

Kdyby laserový senzor fungoval reálně tak, jak je výše popsáno, nebyl by se zpracováním dat ze senzoru problém. Bohužel použitý laserový senzor nebyl tak spolehlivý, jak se očekávalo. Tento fakt se nejvíce projevoval při průjezdech robota oblastmi, kde byl právě otevřený prostor, nebo překážky byly dále než maximální měřitelná vzdálenost senzoru. Pokud se tak stalo (například při průjezdu chodbou), tak v oblasti změřeného spektra, kde byl volný prostor, bylo spousta špatně změřených údajů, viz obrázek spektra (Obrázek 22), které bylo změřeno v chodbě před laboratoří s autonomním robotem.



Obrázek 22 – Naměřené spektrum vykreslené do grafu – chodba

Použít filtr na tato chybná data by bylo samozřejmě realizovatelné, ale bohužel vzhledem k tomu, jak takto znehodnocená data vypadají a v jakém prostředí má být robot nasazen do služby, nebylo možné naměřená data opravit. Protože pokud by se před robotem vyskytla úzká překážka, například sloupek, snadno by se mohlo stát, že by byl tento sloupek filtrem odstraněn, což je nežádoucí a mohlo by to zavinit srážku robota s překážkou.

Laserový senzor v rámci jednoho měření ukládá spektrum o 280 hodnotách. Vzhledem k rozptylu celého spektra na  $70^\circ$  to znamená, že laserový senzor snímá vždy po  $0,25^\circ$ . Taková data jsou pro neuronovou síť zbytečně velmi podrobná a znamenalo by to velký počet neuronů ve vstupní vrstvě neuronové sítě. Data jsou tedy průměrována po osmi hodnotách tak, aby nedocházelo ke ztrátě důležitých informací ve spektru.

## 4 POUŽITÉ TECHNOLOGIE K IMPLEMENTACI NEURONOVÉ SÍTĚ

### 4.1 Visual Studio 2010

*Microsoft Visual Studio je vývojové prostředí pro aplikace od firmy Microsoft. Má aplikace byla vytvářena právě v tomto vývojovém prostředí, které mi svými vlastnostmi ulehčilo spoustu práce.*

*Ve Visual Studiu je možné vytvářet konzolové aplikace i aplikace s grafickým uživatelským prostředím, například s Windows Forms nebo pomocí Windows Presentation Foundation. Visual Studio má velké možnosti rozšiřitelnosti pomocí zásuvných modulů, tzv. pluginů, a případně pomocí šablon vytvářených projektů.*

*Hlavní prvky, které Microsoft Visual Studio podporuje, jsou editor kódu, debugger (pro ladění vyvíjené aplikace) a designér (pro návrh formulářů nebo jiných grafických objektů). Editor kódu podporuje tzv. „IntelliSense“. Jedná se o automatické doplňování psaného zdrojového kódu. Tato funkce Visual Studia programátorovi usnadňuje spoustu práce. IntelliSense doplňuje názvy proměnných, funkcí a metod jednotlivých tříd, které jsou z daného místa programově dostupné (například zobrazuje pouze veřejné proměnné a funkce jiných tříd, privátní proměnné a funkce programátorovi vůbec nenabídne, pokud nejsou součástí stejné třídy, kde je právě psán zdrojový kód). Další velmi užitečná vlastnost editoru kódu je podpora refaktorování. Jedná se o proces provádění změn ve zdrojovém kódu tak, že výsledné změny nemají vliv na vnější chování kódu. Používá se pro zpřehlednění zdrojového kódu. V praxi to znamená především to, že pokud je třeba změnit název nějaké proměnné, která je použita na více místech zdrojového kódu, tak není nutné ručně procházet celý kód a hledat všechna místa, kde byla proměnná použita, ale refaktorizace tuto činnost udělá za programátora. Tímto se eliminuje případné vnášení chyb do zdrojového kódu a následně jejich složité hledání.*

*Visual Studio podporuje velké množství programovacích jazyků. Mezi hlavní vestavěné jazyky patří C/C++, VB.NET a C#. Podpora dalších jazyků může být přidána pomocí jazykových služeb, které musí být doinstalovány.*

*Microsoft Visual Studio 2010 je v rámci programu Microsoft Academy pro studenty zdarma, což je velkou výhodou tohoto vývojového prostředí. Toto je přínosem zejména při vyvíjení aplikací pro nekomerční využití. Adaptováno z [14].*

## 4.2 Jazyk C#

Jedná se o objektově orientovaný programovací jazyk. Tento jazyk vyvinula firma Microsoft společně s platformou .NET Framework. Jazyk C# je založen na jazyce C++.

Programovací jazyk C# lze použít pro tvorbu konzolových aplikací, formulářových aplikací ve Windows (Windows Forms, Windows Presentation Foundation), webových aplikací a stránek, databázových programů a software pro mobilní zařízení jakou jsou PDA či mobilní telefony (adaptováno z [7]).

Tento programovací jazyk jsem použil pro vytvoření knihovny DLL obsahující neuronovou síť a pro vytvoření aplikací, které jsou nutné pro otestování implementované neuronové sítě.

## 4.3 Windows Presentation Foundation

WPF (Windows Presentation Foundation) je jedno z rozšíření rozhraní .NET Framework. Jedná se o knihovnu pro tvorbu grafického uživatelského rozhraní aplikací. Dříve bylo využíváno ovládacích prvků Windows Forms, které byly nativními ovládacími prvky systému Windows. Používaly popisovače okna, které byly založené na pixelech.

WPF je založeno na rozhraní DirectX. Tato aplikace uživatelského rozhraní popisovače okna nepoužívá a velmi snadno se s ním může měnit velikost uživatelského rozhraní (přizpůsobuje se velikosti aplikačního okna) a podporuje zvuk a video.

Jednou z velkých výhod rozhraní WPF je, že umožňuje snadno rozdělit práci na projektu mezi více lidí a mezi návrháře a vývojáře. Vývojář může pracovat na kódu, aniž by potřeboval hotové uživatelské rozhraní a návrhář naopak nepotřebuje při vývoji funkční a datovou část aplikace (čerpáno z [8]).

## 4.4 Nástroje JetBrains

Nástroje této firmy slouží především pro optimalizaci zdrojového kódu. Tyto nástroje jsou používány v rámci Visual Studia jako jeho doplňky. Jsou to velmi silné nástroje, které umožňují zvýšit přehlednost zdrojového kódu a jeho efektivitu, případně pomáhají lokalizovat náročné části programu na sdílené prostředky, jako je čas procesoru. Všechny níže uvedené nástroje jsou chráněné licenci a pro použití jsou placené. Každý nástroj je možné si vyzkoušet v rámci zkušební verze, která je časově omezená. Těchto zkušebních verzí bylo využito pro závěrečné čištění zdrojového kódu a optimalizace problémových



částí. Tyto nástroje jsou velmi užitečným pomocníkem každého programátora a jejich použití v praxi je nenahraditelné.

#### **4.4.1 ReSharper 6.1.1**

Hlavním nástrojem firmy JetBrains je doplněk Visual Studia ReSharper. Tento doplněk slouží ke zvýšení přehlednosti kódu a k rychlé refaktorizaci celých bloků zdrojového kódu. Díky tomuto nástroji je rychlejší a efektivnější všechna práce s Visual Studií. Umožňuje důkladné vyhledávání uvnitř zdrojového kódu napříč všemi projekty v rámci jednoho celkového řešení. Tento nástroj také Visual Studio rozšiřuje o spoustu nových klávesových zkratk a dalších drobných detailů, které z Visual Studia dělají mnohem silnější nástroj, než se na první pohled zdá. Čerpáno z [15].

#### **4.4.2 dotTrace 5.0 Performance**

Tento nástroj umožňuje vyhledávání částí zdrojového kódu v aplikaci, ve kterých dochází k největšímu časovému zdržení. Práce s ním spočívá ve vytvoření snímku paměti aplikace před a po spuštění určité činnosti v aplikaci a následné analýze rozdílů mezi jednotlivými stavy. Díky zpětné kompilaci je možné zjistit přesně, která metoda, funkce nebo část kódu způsobovala největší zdržení a využila nejvíce sdílených zdrojů. Díky tomuto nástroji je možné tato slabá místa aplikace nalézt a následně opravit. Čerpáno z [15].

#### **4.4.3 dotTrace 3.5 Memory**

S tímto doplňkem Visual Studia je možné rychle profilovat paměť, která je využita běžící aplikací. Je možné lokalizovat místa v kódu, ve kterých je špatně pracováno s pamětí, a následně tyto problémy vyřešit. Korektní práce aplikace s pamětí zajišťuje stabilitu aplikace a předchází neočekávanému chování aplikace. Čerpáno z [15].

## **II. PRAKTICKÁ ČÁST**

## 5 IMPLEMENTACE NEURONOVÉ SÍTĚ

Stěžejní částí této diplomové práce bylo vytvoření části obsluhujícího softwaru pro autonomního robota. Konkrétně se jednalo o umělou inteligenci, která by na základě dat získaných z laseru určila další směr, kterým by měl robot pokračovat v pohybu, aby se vyhnul překážkám. I když robot bude obsahovat relativně výkonný počítač (pro toto použití), je potřeba, aby rozhodnutí o následujícím směru proběhlo co nejrychleji. Zároveň bylo potřeba dbát na snadné použití této části aplikace, aby bylo její použití a vložení do hlavní aplikace co nejjednodušší.

Celé řešení<sup>1</sup> obsahuje tři projekty. Prvním projektem je DLL knihovna, která obsahuje implementaci neuronové sítě a je klíčová pro použití v hlavní aplikaci autonomního robota. Dalšíma dvěma projekty jsou WPF aplikace, které slouží pro vytvoření trénovací množiny pro naučení neuronové sítě a otestování neuronové sítě.

### 5.1 DLL knihovna

Aby bylo možné opravdu co nejsnáze implementovat do hlavní aplikace autonomního robota umělou inteligenci pro samovolné řízení, zvolila se metoda vytvoření DLL knihovny obsahující implementaci neuronové sítě a rozhraní pro zpracování bloků surových dat z laserového senzoru do vhodné podoby pro vstup do neuronové sítě.

Tento způsob distribuce umožňuje vložit celou neuronovou síť do hlavní aplikace jako jeden kousek velké skládky. DLL knihovnu může autor finální podoby aplikace pro autonomního robota vložit do svého projektu jako další zdroj<sup>2</sup> objektů a tříd.

V praxi se tento způsob doplňování aplikací standardně používá pro svou jednoduchost během implementace a hlavně možnosti aktualizace pouze části aplikace. Distribuat pouze aktualizace knihoven je mnohem jednodušší, bezpečnější a rychlejší, než opětovná distribuce celé aplikace, jejíž velikost se může pohybovat v řádech megabytů až gigabytů.

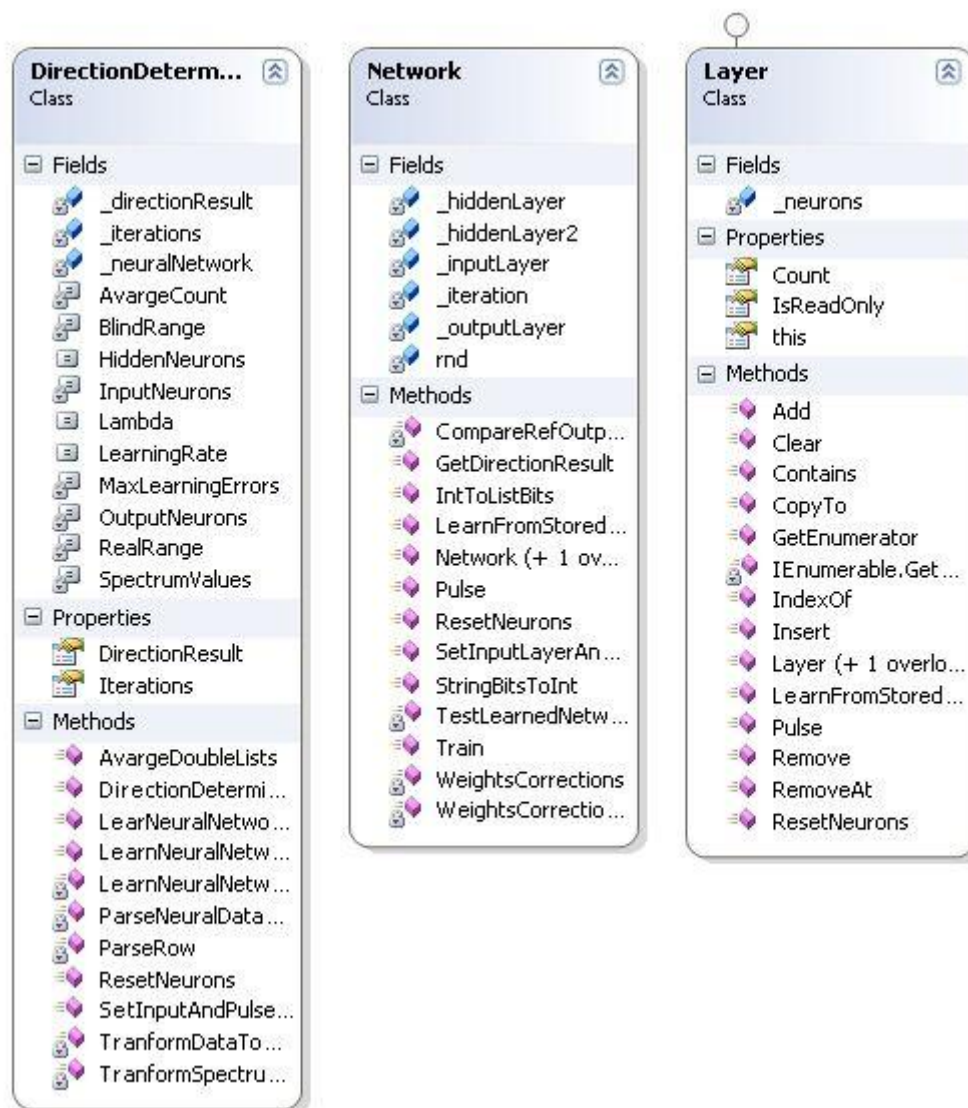
---

<sup>1</sup> Aplikační řešení – v původním názvosloví (v anglickém jazyce) v rámci Visual Studio – „Solution“

<sup>2</sup> Zdroj – v původním názvosloví (v anglickém jazyce) v rámci Visual Studio – „Resources“

## 5.2 Obsah knihovny

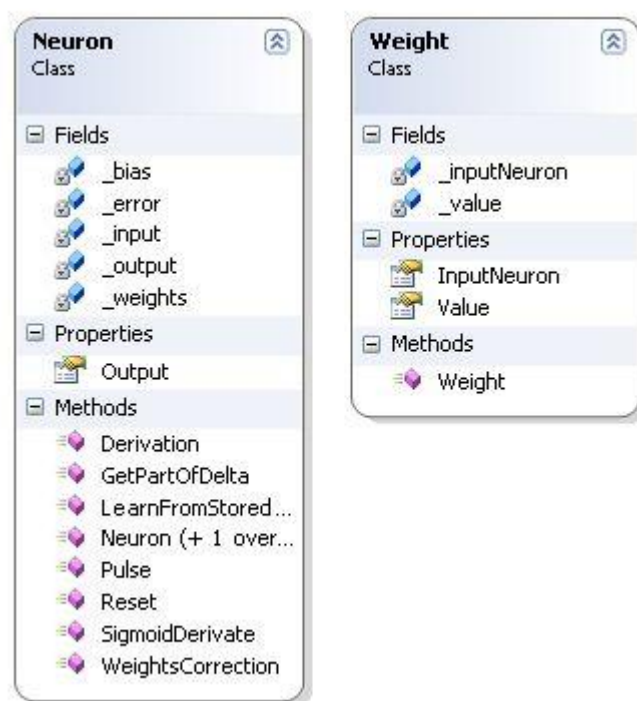
Knihovna s neuronovou sítí obsahuje jednu hlavní třídu „**DirectionDetermination**“, která bude použita v hlavní aplikaci pro vytvoření objektu s neuronovou sítí. V knihovně jsou také další třídy, které obsahují jednotlivé implementační části neuronové sítě:



Obrázek 23 – Třídy DLL knihovny – „DirectionDetermination“, „Network“ a „Layer“

- **Network** – jedná se o třídu, která vytváří neuronovou síť jako takovou, vytváří vrstvy a zajišťuje vyvolání činnosti neuronové sítě, tzv. pulz.
- **Layer** – slouží pro logické rozdělení neuronů do vrstev neuronové sítě (vstupní, skrytá a výstupní).

- **Neuron** – základní prvek neuronové sítě. Jsou rozděleny do vrstev, kde každý neuron obsahuje tolik vah, kolik je neuronů ve vyšší vrstvě.
- **Weight** – v hierarchii objektů použitých v neuronové síti se jedná o nejnižší třídu. Obsahuje informaci o váze a o neuronu z vyšší vrstvy, na který se aplikuje daná váha.
- **NeuralData** – třída, která zapouzdřuje kolekci spekter z laserového senzoru s potřebnými informacemi ke každému spektru.
- **Spectrum** – každé spektrum obsahuje kolekci hodnot o vzdálenosti. Pokud se jedná o trénovací množinu, obsahuje i správný směr, který by neuronová síť měla vybrat.



Obrázek 24 – Třídy DLL knihovny – „Neuron“ a „Weight“

Ve své podstatě hlavní třída DirectionDetermination vytváří neuronovou síť uvnitř sama sebe a zaštiťuje funkce nad neuronovou sítí.

### 5.3 Implementace neuronové sítě

Před samotnou implementací bylo nutné zvolit správný přístup a metodu, kudy se bude projekt ubírat. Jak již bylo zmíněno, bylo potřeba klást důraz na rychlost odezvy neuronové sítě.

### 5.3.1 Neuronová síť jako matice

Jednou z implementačních možností je implementovat neuronovou síť jako soubor několika matic. Tvorba neuronové sítě by tímto způsobem nebyla složitá, jednalo by se pouze o několik matic, které by obsahovaly jednotlivé váhy. Po několika drobných testech a zkušenostech s operacemi nad maticemi obsahující neuronovou síť z odlišných aplikací (Mathematica, Matlab), byla tato varianta implementace zavrhnuta s tím, že se pokusíme k neuronové síti přistoupit jako k objektům a tím bude neuronová síť opravdu existovat a bude snazší pro představu a následnou úpravu v případě potřeby nebo nějaké vnější změny (například změna laserového senzoru). Čerpáno z [4].

### 5.3.2 Neuronová síť jako objekty

Vzhledem k potřebě snadné představitelnosti a přehlednosti v kódu a v neuronové síti jako takové, byla zvolena implementace pomocí OOP. Tato potřeba tu je především z toho důvodu, že se předpokládá využití této implementace i v případě, že dojde u autonomního robota ke změně senzorů nebo jeho zaměření. Proto bylo cílem, aby byla možná snadná úprava počtu vrstev neuronové sítě nebo počtu neuronů ve vrstvách. Důvod k tomuto návrhu byl zmíněn již v předchozích kapitolách, tedy při sebemenší změně parametrů robota (změna laserového senzoru, rozsahu a maximální vzdálenosti viditelnosti) bude potřeba otestovat výsledky neuronové sítě a případně v síti upravit počet skrytých vrstev či počet neuronů v nich. Výsledkem tedy je, že v tomto řešení je každý prvek z oblasti neuronových sítí opravdu reálným objektem a ne pouze číslem ve složité skupině matic.

## 5.4 Úprava neuronové sítě

V hlavní třídě implementace neuronové sítě jsou k dispozici hlavní konstanty, které mohou velmi ovlivnit výslednou neuronovou síť. Díky tomu lze snadno dělat základní úpravy neuronové sítě a jejího chování.

K dispozici jsou následující konstanty:

- **InputNeurons** – počet vstupních neuronů do neuronové sítě. Datový typ konstanty je integer, primárně nastaveno a otestováno jako nejvhodnější je 35 vstupních neuronů.
- **HiddenNeurons** – počet neuronů ve skryté vrstvě. Pokud jsou dvě skryté vrstvy, jedná se o první skrytou, tedy první po vstupní vrstvě. Datovým typem konstanty je integer. V případě jedné skryté vrstvy byl otestovaný vhodný počet neuronů 18,

i přesto, že podle vzorce pro výpočet odhadu nejlepšího počtu by mělo být neuronů 11. V předem naučené neuronové síti je použito v této vrstvě 7 neuronů.

- **HiddenNeurons2** – počet neuronů v druhé skryté vrstvě, pokud je použita neuronová síť s dvěma skrytými vrstvami. Datovým typem konstanty je integer.
- **OutputNeurons** – počet neuronů ve výstupní vrstvě neuronové sítě. Datovým typem konstanty je integer a je použito třech výstupních neuronů, protože je potřeba osm různých výstupních směrů (počet kombinací jednoho neuronu umocněný počtem neuronů = výsledný počet možných kombinací výstupu,  $2^3 = 8$ ).
- **AverageCount** – jedná se o konstantu, která se nastavuje podle poměru počtu hodnot spektra z laserového skeneru k počtu vstupních neuronů do neuronové sítě. S touto hodnotou následně pracují metody v třídách, které zajišťují transformaci spektra pro neuronovou síť. Standardně tuto hodnotu není potřeba upravovat, pokud se nemění zásadně způsob práce neuronové sítě. Datový typ konstanty je integer.
- **SpectrumValues** – udává počet hodnot, které jsou ve spektru. U použitého laserového skeneru se jedná o 280 hodnot. Tato konstanta je také potřeba při transformaci dat pro neuronovou síť. Datovým typem konstanty je integer.
- **BlindRange** – tzv. slepá oblast, udává jakou nejmenší vzdálenost je laserový skener změřit. Tato konstanta je typu double.
- **RealRange** – hodnota, která se ve skutečnosti nastaví, pokud je při zpracování spektra ze senzoru některá hodnota ve slepé oblasti. Každý laserový senzor má svá technologická omezení na jakou délku je schopný rozeznat překážky. Pokud je mezi senzorem a překážkou větší vzdálenost, senzor překážku nerozezná a automaticky vrátí hodnotu 0. Proto jsou při zpracování dat všechny hodnoty ve slepé oblasti invertovány na velkou vzdálenost, tedy že překážka je daleko. Toto si můžeme dovolit díky tomu, že předpokládáme, že díky řízení autonomního robota neuronovou sítí by se neměla žádná překážka ve slepé oblasti reálně vyskytnout.
- **MaxLearningErrors** – počet přípustných chyb v testovací množině během učení neuronové sítě, datovým typem konstanty je integer.
- **Lambda** – určuje úhel náklonu směrnice v transformační funkci na výstupu neuronu. Jako transformační funkce je použita logistická sigmoida. Transformační funkce byly popsány v teoretické části práce.

- **LearningRate** – stupeň rychlosti učení neuronové sítě. Touto hodnotou se násobí odchylka přesnosti každého neuronu z důvodu zpomalení konvergence hledaného řešení, aby nedošlo k překročení optimálního řešení hledání správné hodnoty váhy.

## 5.5 Vytvoření neuronové sítě

Každou neuronovou síť je potřeba naučit pomocí vzorů, které musí umět rozeznávat. Vzhledem k typu použité neuronové sítě s algoritmem učení Backpropagation musí vzory také obsahovat správnou kategorii, do které vzor patří. Implementace neuronové sítě pro autonomního robota nabízí možnost použít předem naučenou neuronovou síť nebo pomocí vytvořené vlastní trénovací množiny naučit novou neuronovou síť.

### 5.5.1 Nová neuronová síť a její učení

Třída „DirectionDetermination“ obsahuje dva konstruktory, které vytvoří v aplikaci objekt obsahující neuronovou síť. Bezparametrový konstruktor této třídy v sobě vytvoří novou nenaučenou neuronovou síť s parametry, které jsou nastaveny v této třídě pomocí konstant (první region uvnitř implementace této třídy „CONSTANTS (ReadOnly variables)“).

Parametry neuronové sítě nejsou předávány pomocí konstruktoru z důvodu jednoduchosti implementace této neuronové sítě v hlavním programu autonomního robota. Změna těchto konstant je potřeba pouze v nejkrajnějších případech, jako je například změna laserového senzoru, tudíž i změna celé neuronové sítě. Při běžném použití knihovny v hlavní aplikaci není potřeba tyto parametry nijak měnit a jsou přednastaveny optimálně pro aktuální hardwarové vybavení autonomního robota.

Naučení neuronové sítě je možné pomocí dvou metod implementovaných ve třídě DirectionDetermination:

```
1 public bool LearnNeuralNetwork(NeuralData learningData,  
                                NeuralData testingData)
```

Tato metoda naučí neuronovou síť z předaných dat z hlavní aplikace. Pomocí dat uložených v „**learningData**“ bude neuronová síť učena a pomocí dat uložených v „**testingData**“ bude testována úspěšnost učení. Druhou možností je předat cestu k textovým souborům, které obsahují příslušná data pro naučení a otestování neuronové sítě pomocí metody:



```
1 public bool LearnNeuralNetworkFiles(string learningFile,  
                                     string testingFile)
```

Algoritmy pro načtení dat z těchto textových souborů jsou implementovány v rámci hlavní třídy „DirectionDetermination“. Data se načítají do kolekce v rámci datových objektů vytvořených z třídy „NeuralData“ a následně je volána předchozí metoda „LearnNeuralNetwork()“ pro učení sítě, které se tyto datové objekty předají.

Tohoto způsobu učení neuronové sítě také využívá přiložená WPF aplikace sloužící pro testování a pokusy s implementovanou neuronovou sítí.

Obě tyto metody používají jako návratový datový typ „bool“. Vždy vracejí bez výjimky hodnotu „true“, aby hlavní aplikace měla odezvu od neuronové sítě o dokončení učení, kdyby bylo učení například realizováno v rámci druhého vlákna nebo procesu.

Během prvních pokusů u nově navrhnuté neuronové sítě je vhodné předat neuronové síti data pro učení i testování stejná, v případě úspěchu můžeme otestovat naučení neuronové sítě na odlišných datech.

Ukázka zdrojového kódu pro volání naučení sítě s předanými daty z laserového skeneru, jedná se o zjednodušený zdrojový kód:

```
1  /// <summary>
2  /// Zpracuje data pro nauceni neuronove site do spravneho formatu a
    naucí sit
3  /// </summary>
4  /// <param name="learningData">Data pro uceni</param>
5  /// <param name="testingData">Data pro otestovani uceni</param>
6  /// <returns>Vždycky TRUE, je tu pokud bude potreba hlavni aplikaci
    pozdrzet (if), dokud se nenauci</returns>
7  public bool LearnNeuralNetwork(NeuralData learningData,
    NeuralData testingData)
8  {
9      //1. prevest na data pro neuronku
10     NeuralData learningNetworkData =
        TranformDataToNeuralSpectrum(learningData);
11     NeuralData testingNetworkData =
        TranformDataToNeuralSpectrum(testingData);
12     //2. predat neuronce learning i testing
13     _iterations = _neuralNetwork.Train(learningNetworkData,
        testingNetworkData, MaxLearningErrors);
14     return true;
15 }
```

Po každé epoše učení (neboli iteraci, jeden průchod celou testovací množinou) je proveden test úspěšnosti učení. V testu se počítá počet chyb oproti referenčním hodnotám ke každému spektru dat.

Učení skončí, pokud je počet chyb z testu nižší než maximální dovolený počet chyb uložený v konstantě „**MaxLearningErrors**“ nebo pokud učení přesáhne určitý počet epoch. S tímto nastavením je potřeba experimentovat v závislosti na trénovací množině.

Pro otestování naučení sítě slouží metoda „**TestLearnedNetwork**“, která je ve zjednodušené podobě zobrazena níže:

```

1  /// <summary>
2  /// Otestuje, jestli je neuronova sit naucena, tim ze predlozi
    vstupy a kontroluje vystupy s referencnimi daty
3  /// Primarne by se melo testovat odlišnými daty než jsou použity pro
    nauceni neuronove site,
4  /// ale pri zmene implementace/poctu neuronu ve vrstvach nebo jinych
    zmenach je vhodne nejdrive zkusit,
5  /// jestli projde testovani primo na datech pomoci kterych byla
    neuronova sit naucena (predpokladana 100% uspesnost)
6  /// </summary>
7  /// <param name="testingNetworkData">Testovací data</param>
8  /// <returns>Pocet vzoru, u kterych se neuronova sit spletla ve
    vysledku oproti referencni hodnote</returns>
9  private int TestLearnedNetwork(NeuralData testingNetworkData)
10 {
11     int errorsCount = 0;
12     foreach (Spectrum spectrum in testingNetworkData)
13     {
14         List<double> refOutput =
            IntToListBits(spectrum.CorrectDirection);
15         SetInputLayerAndPulse(spectrum.ValuesSpectrum);
16         if (!CompareRefOutputWithNetworkOutput(refOutput))
17             errorsCount++;
18     }
19     return errorsCount;
20 }

```

### 5.5.2 Předem naučená neuronová síť

Knihovna umožňuje také vytvořit předem naučenou neuronovou síť, která měla během učení 100% úspěšnost. Vytvoření této neuronové sítě umožňuje konstruktor objektu hlavní třídy „DirectionDetermination“ s parametrem „True“, díky kterému se zavolá přetížený konstruktor, který vytvoří předem definovanou a naučenou neuronovou síť.

```

1  public DirectionDetermination(bool learned)
2  {
3      _neuralNetwork = new Network(35, 7, 3);
4      LearnNeuralNetworkStored();
5  }

```

Jak je vidět v ukázce konstruktoru naučené neuronové sítě, je vytvořena klasická neuronová síť, stejně jako kdyby byla vytvářena nenaučená síť. Ale jsou přímo definované počty neuronů pro jednotlivé vrstvy a po vytvoření sítě je volána metoda „LearnNeuralNetworkStored“ která nastaví předem připravené váhy pro každý neuron.

## **5.6 Používání neuronové sítě**

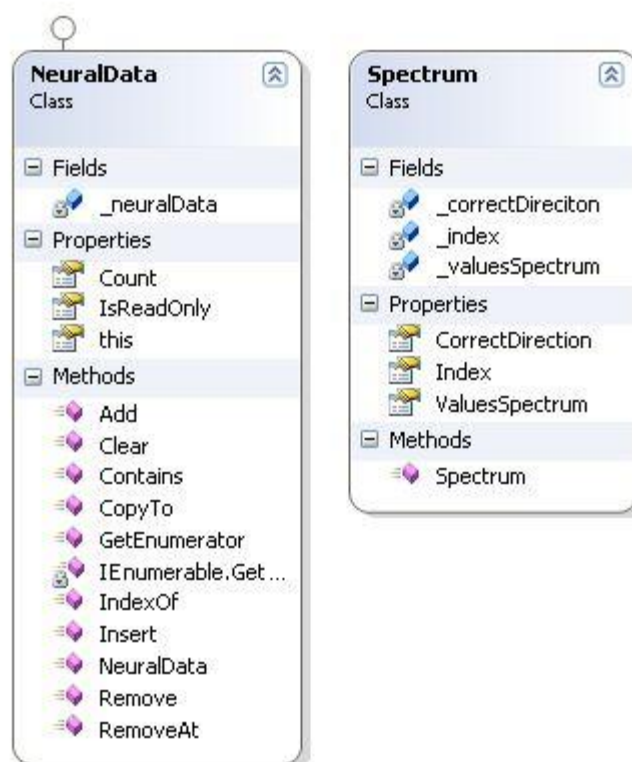
Základní princip použití neuronové sítě spočívá v tom, že v hlavní aplikaci se vytvoří nová instance třídy „DirectionDetermination“, která je obsahem DLL knihovny a veškerá komunikace a práce s neuronovou sítí probíhá pomocí tohoto objektu, což vyplynulo z předchozí kapitoly.

### **5.6.1 Předem naučená neuronová síť**

U předem naučené neuronové sítě není třeba už žádného dodatečného volání metod, které by zajišťovaly, aby byla neuronová síť funkční, a je tedy možné rovnou pomocí vytvořené instance třídy „DirectionDetermination“ začít předávat data z laserového senzoru ke zpracování a vyhodnocení.

### **5.6.2 Učení neuronové sítě**

Pokud je v hlavní aplikaci z nějakého důvodu využita neuronová síť, která není předem naučena, je potřeba tuto síť naučit a předat jí k naučení potřebná data nebo informaci o umístění textového souboru s těmito daty.



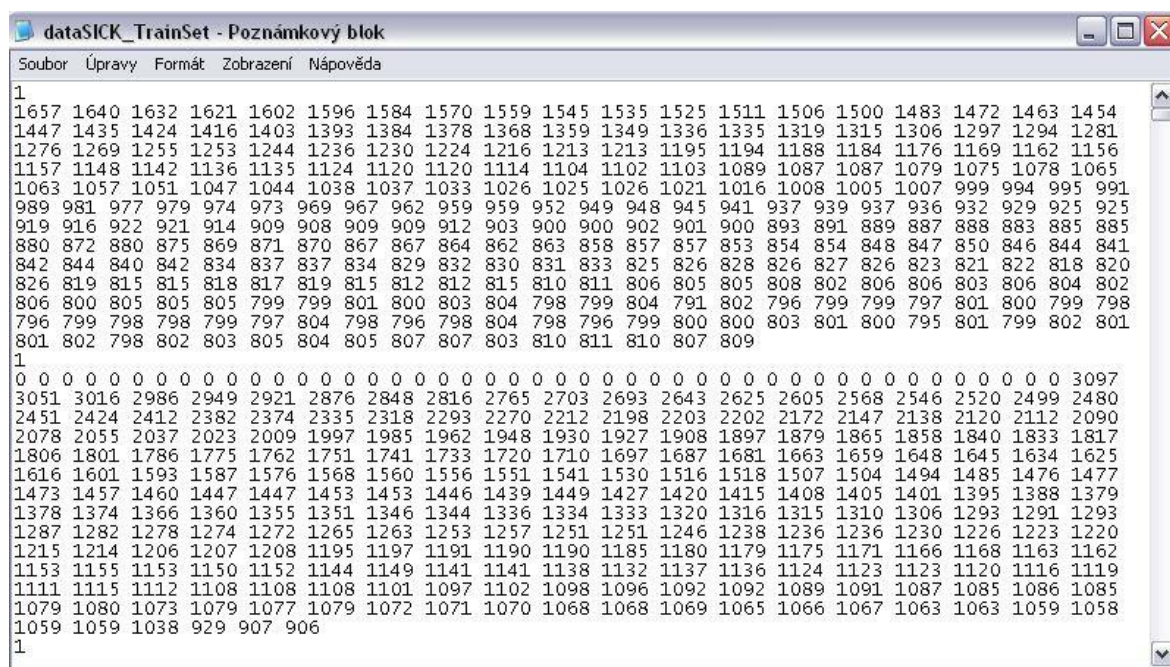
Obrázek 25 – Třídy DLL knihovny – „NeuralData“ a „Spectrum“

Data určená již přímo k učení neuronové sítě jsou předávána pomocí datového objektu třídy „NeuralData“ (Obrázek 25 vlevo), který implementuje rozhraní kolekce „IList“. Jako objekty kolekce jsou použity instance třídy „Spectrum“ (Obrázek 25 vpravo).

Čím více spekter tato kolekce obsahuje, tím lépe a přesněji je možné neuronovou síť naučit. Zároveň je důležité, aby byl poměr mezi počtem spekter pro každý směr rovnoměrný.

Pokud bude hlavní aplikace předávat metodě pro učení pouze cestu k textovému souboru, který obsahuje trénovací množinu, je důležité, aby data v textovém souboru byla ve správném formátu. Cílem návrhu této knihovny nebylo ošetření, zda jsou načítaná data ve správném formátu, ale naopak aby byla práce s neuronovou sítí co nejrychlejší a nejoptimálnější. I přes tento fakt je v kódu implementováno zabezpečení, kdyby bylo načteno chybné spektrum, aby toto spektrum nebylo při učení použito a nenarušilo integritu učení.

Trénovací množina dat v textovém souboru by měla být uložena ve formátu, kde se vždy opakují dva řádky. Na prvním řádku by se mělo nacházet číselné označení směru a druhý řádek obsahuje hodnoty z laserového senzoru oddělené mezerami. Těchto hodnot musí být 280 (podle aktuálně používaného laserového senzoru a nastavení neuronové sítě).



Obrázek 26 – Ukázka trénovací množiny dat

### 5.6.2.1 Algoritmus učení neuronové sítě – Backpropagation

Pro učení neuronové sítě je implementován algoritmus Backpropagation, z důvodů, které byly uvedeny v přechozích kapitolách v teoretické části této diplomové práce.

Hlavní princip tohoto algoritmu spočívá ve vložení spektra s referenčním výsledkem do neuronové sítě, kde se následně počítá rozdíl mezi referenční hodnotou a výsledkem neuronové sítě. Tento rozdíl se počítá pro každý neuron zvlášť, proto jsou hodnoty v ukázce zdrojového kódu uloženy v kolekci. Ukázka zdrojové kódu pro naučení neuronové sítě ve zjednodušeném tvaru:

```
1  /// <summary>
2  /// Naučí a otestuje úplnost naučení neuronové sítě, učení trvá,
3  /// dokud počet chyb při testování neklesne pod požadovanou hodnotu
4  /// </summary>
5  /// <param name="learningNetworkData">Trenovací množina dat</param>
6  /// <param name="testingNetworkData">Testovací množina dat</param>
7  /// <param name="maxLearningErrors">Max. počet příp. chyb</param>
8  /// <returns>Počet iterací učení</returns>
9  public int Train(NeuralData learningNetworkData,
                  NeuralData testingNetworkData,
                  int maxLearningErrors)
10 {
11     int error;
12     do
13     {
14         if (_iteration >= 500)
15         {
16             ResetNeurons();
17             _iteration = 0;
18         }
19         lock (this)
20         {
21             foreach (Spectrum spectrum in learningNetworkData)
22             {
23                 List<double> refOutput =
24                     IntToListBits(spectrum.CorrectDirection);
25                 List<double> delta = new List<double>();
26                 SetInputLayerAndPulse(spectrum.ValuesSpectrum);
27                 for (int i = 0; i < _outputLayer.Count; i++)
28                 {
29                     delta.Add(refOutput[i] - _outputLayer[i].Output);
30                 }
31                 WeightsCorrections(delta);
32             }
33             _iteration++;
34         }
35         error = TestLearnedNetwork(testingNetworkData);
36     } while (error > maxLearningErrors || _iteration < 400);
37     return _iteration;
38 }
```

Úspěšnost učení neuronové sítě z velké části také závisí na vygenerovaných vahách při vytváření každého neuronu. Ve většině případů stačí pro naučení sítě 200 iterací s úspěšností 70 %. Pokud jsou váhy při vytváření neuronů vygenerovány velmi daleko od potřebných cílových hodnot, může nastat situace, kdy bude k učení sítě potřeba i tisíce epoch (iterací). Z tohoto důvodu je v učení neuronové sítě implementovaná podmínka, která hlídá počet iterací. Pokud počet iterací přesáhne 500, je to znamení právě toho, že váhy nebyly vygenerovány nejvhodněji. V tuto chvíli je výhodnějším a rychlejším řešením váhy všech neuronů vygenerovat znovu a začít učit od začátku, než nechat běžet učení neuronové sítě po dobu, která by se počítala již v řádech minut.

Neuronovou síť je možné naučit pro řešený problém i s vyšší přesností, ale za podmínky, že by byl dostatek času pro učení. Bylo potřeba zvolit vhodný poměr mezi potřebným časem k naučení neuronové sítě a úspěšností učení.

V případě, že jsou vygenerovány váhy neuronu blízko řešení a teoreticky by stačilo například i pouhých 10 iterací učení pro úspěšnost 70 %, tak i přesto je ve zdrojovém kódu podmínka, aby s tímto nastavením vah proběhlo minimálně 400 iterací učení. Celkovou dobu učení to výrazně neprodlouží, ale je velká šance, že se podaří dosáhnout úspěšnosti větší než 85 % – 90 %.

Pro autonomního robota bylo cílem co nejrychlejší učení. Je důležité poznamenat, že i když úspěšnost učení v tak krátkém čase je okolo 70 % – 80 %, tak to neznamená, že je neuronová síť naučená špatně. Vždy záleží na testovací množině a algoritmem testovaná úspěšnost učení je velmi relativní.

Algoritmus testuje, jestli byl na daném spektru zvolen správný směr vůči referenčnímu směru, avšak pro autonomního robota je dostačující, pokud zvolený směr nenavede robota přímo proti překážce. Po důkladných testech bylo zjištěno, že právě u přesnosti učení 70 % dojde k reálné chybě určení směru pouze průměrně v 3 % – 5 %. Při přesnosti učení 80 % je reálná chyba minimalizována až na 1 % – 3 %.

Výše uvedená reálná přesnost učení je už pro autonomního robota, pro kterého je tato neuronová síť určena, více než dostatečná, a to z důvodu, že k vyhodnocení dalšího směru neuronovou sítí bude docházet ve velmi krátkých časových intervalech



### 5.6.3 Vkládání dat na vstup neuronové sítě

Z hlavní řídicí aplikace autonomního robota můžeme předat data z laserového senzoru do instance třídy „DirectionDetermination“ s neuronovou sítí pomocí dvou metod:

```
1 public void SetInputAndPulse(List<double> input)
```

nebo pomocí přetížené metody:

```
1 public void SetInputAndPulse(List<double> input,  
                               List<double> controlInput)
```

Jak již bylo v předchozích kapitolách zmíněno, ani jedna z těchto metod nemá žádnou návratovou hodnotu. To umožňuje v případě nutnosti zpracovávat operace s neuronovou sítí paralelně s hlavní řídicí aplikací a ta nemusí čekat na výstup z neuronové sítě. Výstup si zkontroluje až ve chvíli, kdy jej bude potřebovat.

Vstupem do těchto metod jsou vždy kolekce hodnot datového typu „double“. Kolekce obsahuje 280 hodnot, které jdou přímo z laserového skeneru. Všechny hodnoty jsou vždy v intervalu od 0 do maximální vzdálenosti, kterou je laserový senzor schopen rozeznat. U aktuálně použitého senzoru se jedná o čísla do 3100 (milimetrů). V tomto případě by pro tyto hodnoty stačilo použití datového typu „integer“, ale z důvodu plánovaného přechodu na výkonnější laserový senzor, byl zvolen bez odkladu datový typ „double“.

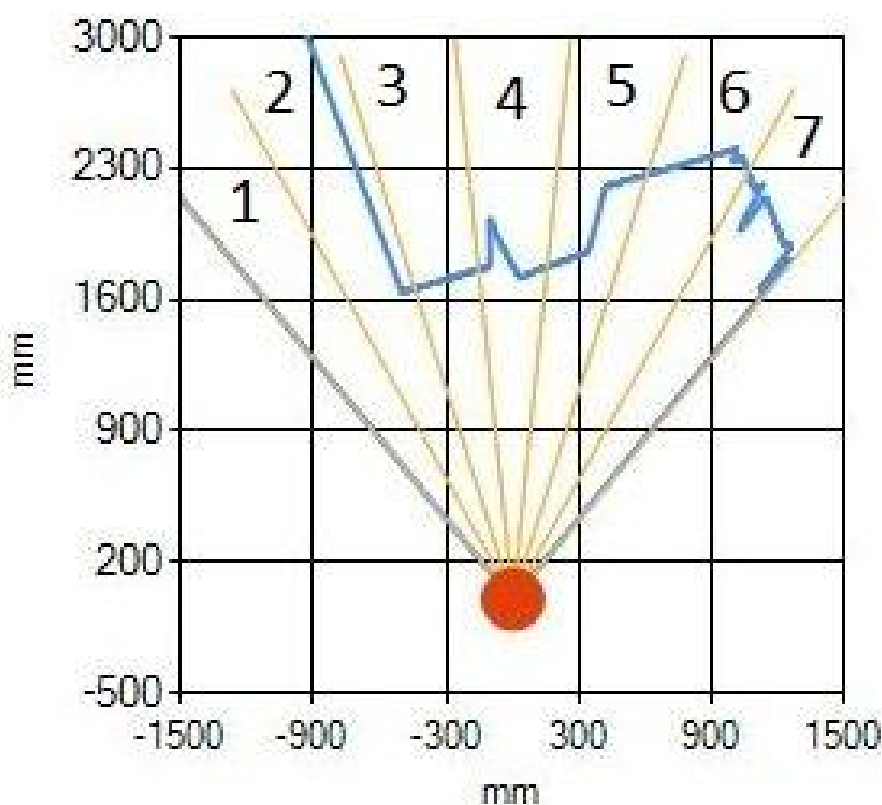
Obě metody pro vkládání dat do neuronové sítě pracují stejně:

1. **Transformují výstup z laserového senzoru na vstup neuronové sítě** – jedná se o změnu počtu hodnot v daném spektru. Spektrum obsahuje 280 hodnot, které se musí zprůměrovat na počet neuronů ve vstupní vrstvě neuronové sítě. Vstupní vrstva neuronové sítě v základním nastavení obsahuje 35 neuronů. Transformace probíhá tak, že se vždy 8 hodnot z původního spektra zprůměruje na jednu hodnotu.
2. **Transformované spektrum je vloženo do vstupní vrstvy** – hodnota se přiřazuje jako výstup daného neuronu.
3. **Aktivace neuronové sítě** – tento krok je implementačně řešen už v rámci druhého kroku. Přímě v třídě „Network“ na závěru metody pro naplnění neuronů ve vstupní vrstvě zavolá metodu „Pulse()“, která aktivuje celou neuronovou síť.
4. **Získají výstup z neuronové sítě** – výstup z neuronové sítě je potřeba vložit do proměnné, do které má přístup vnější okolí, tedy hlavní řídicí aplikace.

Druhé přetížené metodě se v parametrech předává ještě kontrolní vstup. Mělo by se jednat o data ze senzoru ze stejné pozice jako hlavní vstupní data. Hlavní vstup a kontrolní vstup je následně zprůměrován. Cílem tohoto postupu je eliminovat co nejvíce chyb v měření vzdálenosti, které mohly nastat vlivem odrazu nebo lomu laserového paprsku.

## 5.7 Výstup z neuronové sítě

Výstup z hlavního objektu zapouzdřujícího celou neuronovou síť je řešen pomocí datového typu integer, kde každé celé číslo v rozmezí od 0 do 7 označuje směr vyhodnocený neuronovou sítí:



Obrázek 27 – Možné výsledné směry pro pohyb robota

Tabulka 1 – možné výsledné směry pro pohyb robota

| Výstup z neuronové sítě | Pokyn pro robota |
|-------------------------|------------------|
| 0                       | Otočit o 90      |
| 1                       | Vlevo            |
| 2                       | Více vlevo       |
| 3                       | Mírně vlevo      |
| 4                       | Rovně            |
| 5                       | Mírně vpravo     |
| 6                       | Více vpravo      |
| 7                       | Vpravo           |

Jak je patrné z Obrázek 27 a Tabulka 1, jsou směry označeny číselně zleva. V případě, že se robot vyskytne v situaci, kdy nebude žádný z těchto směrů vyhovovat, bude neuronová síť automaticky vracet hodnotu „0“, která bude pro řídicí aplikaci robota znamenat, že se ocitl ve slepé uličce nebo naproti zdi a má se otočit o 90° doleva. V tomto případě je relativně jedno, jestli se bude při hodnotě otáčet doleva nebo doprava, důležité je, aby zvolená strana byla v každém případě stejná, aby nedošlo k uvíznutí robota ve slepé uličce.

Neuron jako takový by sám o sobě nebyl schopný tolik směrů roztrždit. Byla by možnost použít nějakou pozvolnou spojitou přenosovou funkci a interval od 0 do 1 rozdělit na 8 částí. Při tomto řešení by ale docházelo k velkým nepřesnostem a omylům ve výsledném zvoleném směru.

Uvnitř neuronové sítě je tento problém řešen pomocí tří výstupních neuronů, aby dělicí interval mezi dvěma hodnotami byl větší. Tři výstupní neurony s binární klasifikací umožňují rozdělení vzorů do osmi výstupních skupin, což je pro náš řešený problém ideální.

Vhodnou transformační funkcí uvnitř výstupních neuronů by pro toto použití neuronové sítě a požadovaný formát výstupu byla binární skoková funkce. Při použití neuronové sítě s algoritmem učení Backpropagation ale není možné použít nespojitou transformační funkci z důvodu výpočtu derivací uvnitř neuronů. Nespojitou funkci nelze derivovat.

Uvnitř každého neuronu je na výstupu transformační funkce logistická sigmoida. Tato transformační funkce má nastavenou dostatečně velkou konstantu lambda (běžně se používá hodnota 1) Takto zvolená hodnota lambdy zajistí, že se navenek bude neuron

chovat jako s binární skokovou transformační funkcí. V implementaci je použita hodnota 6. Ovšem ani takto vysoká hodnota nezajistí přesnou binární skokovou funkci a během učení by stále nebylo možné porovnat výstup z neuronové sítě s referenčním výstupem. Tato situace je řešena až přímo v podmínce ve zdrojovém kódu, kde dochází k porovnání referenční hodnoty s výstupem z neuronové sítě. Jednotlivé **výstupy z neuronové sítě jsou zaokrouhleny na celé číslo**, spojením všech výstupních neuronů vznikne binární reprezentace číselného označení zvoleného směru, který je následně porovnán s referenční hodnotou z trénovací množiny dat.

### 5.7.1 Získání výstupu z neuronové sítě

Výsledek z neuronové sítě není předáván přímo pomocí návratové hodnoty metody ale pouze pomocí veřejně<sup>3</sup> přístupné proměnné pouze pro čtení, která je ovšem uvnitř objektu<sup>4</sup> třídy přístupná bez omezení.

Důvodem této volby řešení je nutnost, aby autonomní robot, respektive jeho řídicí aplikace, téměř okamžitě reagoval na přerušení, které by bylo způsobeno nějakou bezpečnostní výstrahou. Pokud by k takové situaci došlo, je potřeba, aby řídicí aplikace reagovala okamžitě na vyvolané přerušení a nemusela čekat, až neuronová síť zpracuje požadavek na následující směr cesty. Implementace této situace je tedy myšlena tak, že řídicí aplikace neustále posílá do neuronové sítě data a ve chvíli, kdy potřebuje výslednou hodnotu neuronové sítě, si ji načte z proměnné objektu třídy „DirectionDetermination“, která je veřejnému prostředí přístupná pouze pro čtení. Když informace z neuronové sítě není vyžadována, tak jednoduše není ani volána metoda „get“ pro získání obsahu proměnné uvnitř objektu s neuronovou sítí.

Proměnná obsahující výsledek neuronové sítě je v základním nastavení pouze jako privátní a je implementovaná jako vlastnost objektu třídy, která je pouze ke čtení, tedy má implementovaný pouze tzv. „getter“. „Setter“ není potřeba, protože řídicí aplikace by neměla do této proměnné mít přístup (a ani by jej neměla potřebovat) pro zapsání dat.

---

<sup>3</sup> Veřejně přístupná proměnná, myšleno „public“.

<sup>4</sup> Proměnná, která je přístupná pouze uvnitř objektu je označována jako „private“

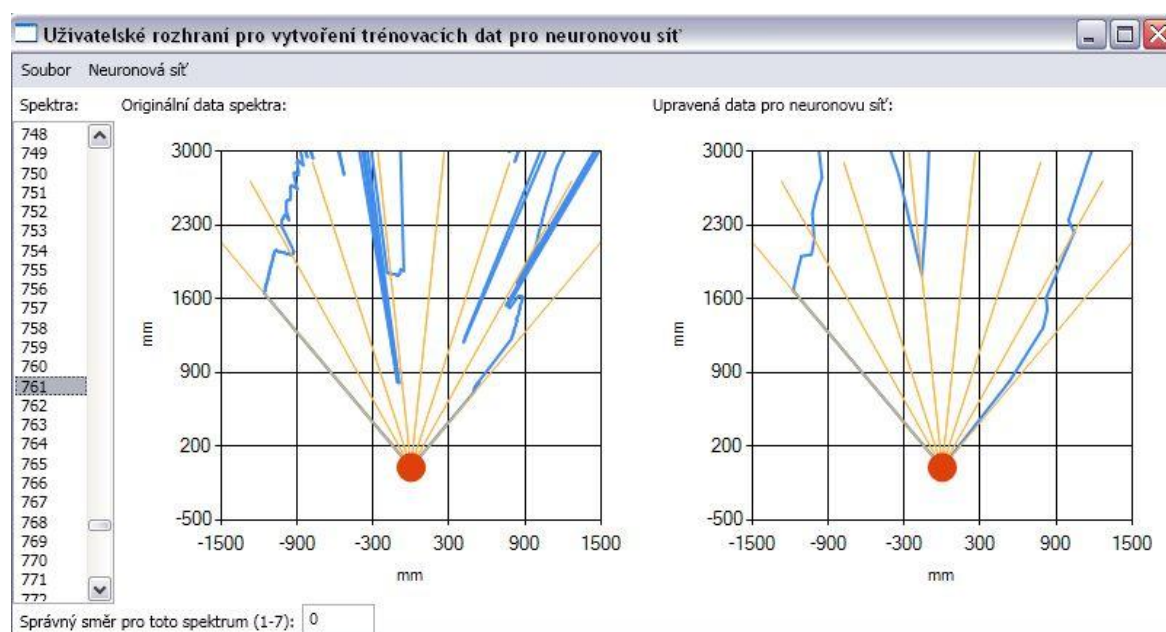
## 6 UŽIVATELSKÉ ROZHRANÍ PRO TESTOVÁNÍ

Cílem této práce bylo vytvořit DLL knihovnu, kterou by bylo možné využít při automatickém řízení robota pomocí neuronové sítě, která by byla obsahem právě zmiňované DLL knihovny.

Ovšem autonomní robot a jeho hlavní řídicí aplikace v době vývoje umělé inteligence pro řízení nebyly dostupné. Bylo proto potřeba vytvořit aplikace, které by danou neuronovou sítí mohly otestovat jak z pohledu komunikace s neuronovou sítí v DLL knihovně, tak z pohledu porovnání výsledku neuronové sítě s obrazem z laserového senzoru. Druhou aplikací, která je potřeba pro používání neuronové sítě, je aplikace, ve které bude možné snadno vytvořit trénovací množinu dat pro naučení neuronové sítě, pokud by uživatel nechtěl z jakéhokoli důvodu využít předem naučenou neuronovou sít' a chtěl by si neuronovou sít' naučit pro jiný případ užití, než bylo primárně zamýšleno.

### 6.1 Uživatelské rozhraní pro vytvoření učící množiny

První z podpůrných aplikací pro výslednou DLL knihovnu je aplikace, která umožňuje vytvořit snadno a rychle trénovací množinu dat pro naučení neuronové sítě. Tato aplikace se jmenuje „UICreateLearnigData“ (uživatelské prostřední vytvářející trénovací množinu dat) a její zdrojové kódy je možné nalézt v rámci celého řešení jako jeden z projektů.



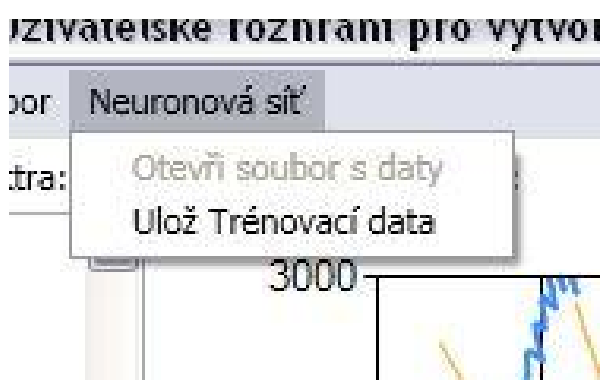
Obrázek 28 – Aplikace UICreateLearningData

Jak je vidět na Obrázek 28, aplikace obsahuje několik základních částí. Data, která načítáme do aplikace, budou ve většině případů obsahovat více než jedno spektrum hodnot z laserového skeneru. Důvodem je, že trénovací množina dat pro neuronovou síť musí obsahovat co nejvíce vzorků dat z každé skupiny, do kterých bude neuronová síť následně přichozí vzory dat třídit. Proto je v levé části umístěn seznam, do kterého se načtou jednotlivá spektra, která jsou načtena z textového souboru.

Vždy po vybrání konkrétního spektra je toto spektrum vykresleno do dvou grafů. V prvním (levém) grafu je vykresleno přímo to, co reálně „viděl“ laserový skener. V druhém (pravém) grafu je spektrum již transformované do podoby, ve které bude vloženo na vstup neuronové sítě. Proces transformace mezi daty z laserového skeneru na data pro neuronovou síť je implementován shodně s DLL knihovnou, aby uživatel viděl, jak tato transformace ovlivnila data pro neuronovou síť.

Po zobrazení vybraného spektra ze seznamu určíme vhodný směr, který by byl pro toto spektrum správný, a neuronová síť by se měla tento směr naučit, a napíšeme ho do textového pole v dolní části aplikačního okna. Textové pole obsahuje kontrolu zadané hodnoty a dovolí vložit pouze hodnoty, které označují směry, tedy hodnoty od 0 do 7.

Bylo žádoucí, aby zadávání správných směru probíhalo co nejrychleji, proto vždy po vepsání správné hodnoty a potvrzením pomocí klávesy Enter je hned vybráno a vykresleno další spektrum v seznamu a následně se aktivuje kurzor v textovém poli pro vepsání další hodnoty.



Obrázek 29 –

Menu aplikace UICreateLearningData

Přes hlavní menu aplikace jsou dostupné pouze tři základní akce. První je ukončení aplikace, které je v menu „Soubor“. Další dva ovládací prvky jsou v menu

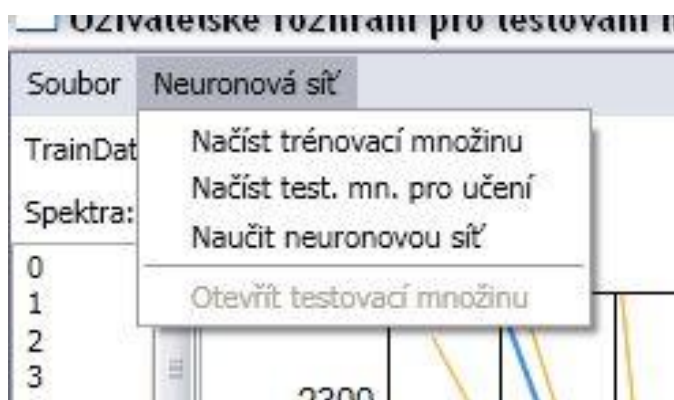
„**Neuronová síť**“. Jsou to ovládací prvky pro načtení textového souboru se spektry z laserového skeneru („**Otevři soubor s daty**“) a pro uložení vytvořené trénovací množiny dat do textového souboru („**Ulož trénovací data**“).

Textové soubory, které je možné načíst, mají přesně definovaný formát obsahu. Jedná se vždy o 280 hodnot oddělených mezerou na jednom řádku. Pokud řádek obsahuje jiný počet hodnot, je tento řádek ignorován a není načten. Prázdné řádky algoritmu pro načítání textových souborů nevadí, stejně tak nevadí, pokud je mezi jednotlivými hodnotami více mezer (například kvůli formátování textového souboru s daty pro větší přehlednost).

Trénovací množina je ukládána také v předem definované formě a soubory vytvořené touto aplikací jsou přímo určené pro aplikaci, která testuje DLL knihovnu s neuronovou sítí nebo přímo pro aplikaci, která tuto knihovnu využívá k řízení autonomního robota. Takto vytvořená trénovací množina je určena pouze pro učení neuronové sítě, případně pro kontrolu naučení. Není určena pro testování již naučené neuronové sítě. Pro testování již naučené neuronové sítě slouží surová data z laserového skeneru, tedy shodná data, která jsou vstupem do této aplikace.

## 6.2 Uživatelské rozhraní pro testování knihovny s neuronovou sítí

Druhou z podpůrných aplikací pro výslednou DLL knihovnu je aplikace, která umožňuje vyzkoušet funkčnost neuronové sítě v DLL knihovně. Je možné vyzkoušet učení neuronové sítě i otestovat odezvu naučené neuronové sítě. Tato aplikace se jmenuje „**UIDirectionDeterminationByAI**“ (uživatelské prostředí testující rozhodování o směru) a její zdrojové kódy je možné nalézt v rámci celého řešení jako jeden z projektů.



Obrázek 30 –

Menu aplikace UIDirectionDeterminationByAI

### 6.2.1 Načtení dat a naučení neuronové sítě

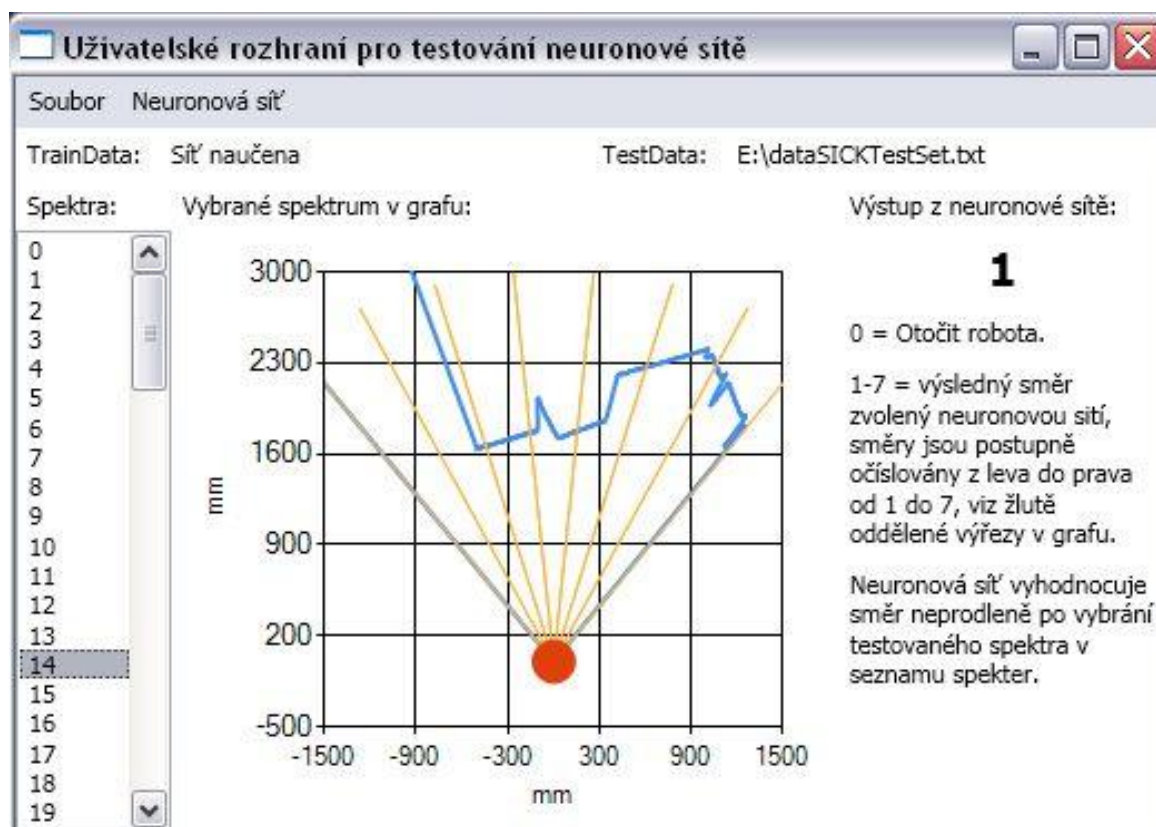
Při učení dat se využívá tři tlačítek v hlavním menu „**Neuronová síť**“.

- **Načíst trénovací množinu** – otevře textový soubor s trénovací množinou dat pro neuronovou síť. Je nutné, aby tato data byla přesně ve formátu, v jakém je uloží aplikace pro vytvoření trénovací množiny. Jedná se o tvar, kdy se střídají dva řádky. První řádek obsahuje informaci o správném směru ve spektru, které bude následovat, a druhý řádek obsahuje samotné spektrum, tedy 280 hodnot oddělených mezerami.
- **Načíst test. mn. pro učení** – otevře textový soubor s množinou dat, která neslouží přímo pro učení neuronové sítě, ale pro ověření stupně naučení sítě během učení. Na základě tohoto testování, jakmile klesne počet chyb určení směru v tomto souboru spekter, je učení neuronové sítě zastaveno. Formát v tomto souboru je shodný jako u dat trénovací množiny. Pokud se dělají pokusy s neuronovou sítí, je vhodné při prvních testech nově navrhnuté neuronové sítě použít data shodná s trénovací množinou.
- **Naučit neuronovou síť** – Předá cesty k souborům s trénovací a testovací množinou instanci třídy „DirectionDetermination“ s neuronovou sítí, která se na základě předaných dat naučí.

Dokončení učení neuronové sítě je signalizováno přepsáním cesty k textovému souboru s trénovací množinou na „Sít' naučena“.

Během učení se může uživateli zdát, že aplikace „zamrzla“. Není tomu tak, pouze probíhá učení neuronové sítě, které je velmi náročné na výkon procesoru a celého počítače. Doba trvání učení aplikace je také velmi závislá na tom, jak se vygenerují základní nenaucené váhy při vytváření neuronové sítě. Pokud se tyto váhy vygenerují velmi špatně a mohlo by se stát, že řešení by nevedlo ke chtěným výsledkům, jsou váhy neuronů automaticky resetovány a učení probíhá znovu.





Obrázek 31 – Aplikace UIDirectionDeterminationByAI

### 6.2.2 Testování neuronové sítě

Testování neuronové sítě probíhá už pouze tak, že si v hlavním menu „Neuronová síť“ načteme pomocí ovládacího prvku „**Otevřít testovací množinu**“ testovací data z textového souboru. Data v textovém souboru jsou v surové podobě, tedy 280 hodnot na řádku oddělených mezerami. Jedná se o stejný formát, který je vstupem do aplikace pro vytvoření trénovací množiny.

Načtená data jsou zobrazena v levém seznamu. Vždy po výběru konkrétního spektra je toto spektrum vykresleno do grafu uprostřed okna aplikace, aby uživatel mohl sám intuitivně vyhodnotit nejlepší směr. Automaticky po výběru spektra ze seznamu je toto spektrum odesláno neuronové síti vytvořené z třídy v DLL knihovně. Neuronová síť zpracuje vstupní vzor tak rychle, že je možné se okamžitě dotázat na výsledný směr určený pomocí neuronové sítě. Tento směr je zobrazen v pravé části aplikace tučně.

Při tomto testování je potřeba brát v úvahu, že neuronová síť nemusí nutně zvolit stejný směr, který si myslí uživatel, že je nejvhodnější. Směr, který zvolí neuronová síť, vždy

závisí na datech, která jsou neuronové síti předložena k naučení. Je možné, že neuronová síť nedosáhne 100% úspěchu při učení nebo že načtený vzor neodpovídá žádnému vzoru z trénovací množiny dat. V takovém případě se může stát, že neuronová síť určí směr špatně. Ovšem s ohledem na reálné použití, není tato výjimečná situace problémem. Data z laserového senzoru budou neuronové síti posílána ve velmi krátkých cyklech, a pokud dojde k chybnému určení směru, autonomní robot nestihne ujet ani několik centimetrů a už bude mít k dispozici novou a správnou informaci o dalším směru.

## 7 VÝSLEDNÁ NEURONOVÁ SÍŤ

Pro určení, jak bude vypadat schéma finální neuronové sítě, bylo potřeba projít celou řadou testování. Testování se týkalo především počtu neuronů ve skryté vrstvě.

Jaký typ neuronové sítě zvolit napovídal už samotný typ problému, který bylo potřeba řešit. Jedná se tedy o neuronovou síť s učením pomocí algoritmu „Backpropagation“.

Pro řešení jakéhokoliv problému by měly stačit maximálně dvě skryté vrstvy, ovšem počet neuronů není přesně specifikován a vždy záleží přímo na řešené úloze. Pro návrh neuronové sítě bylo možné použít buď jednu skrytou vrstvu s počtem neuronů uvnitř této sítě podle vzorce 8 - 1, nebo s dvěma skrytými vrstvami, kde se počet neuronů v každé z vrstev určí podle vzorců 8 - 2 a 8 - 3.

Výpočet vhodného počtu neuronů ve skryté vrstvě v neuronové síti s jednou skrytou vrstvou:

$$N_{skrytá} = \sqrt{N_{vstup} * N_{výstup}} \quad (8 - 1)$$

Výpočet vhodného počtu neuronů ve skrytých vrstvách v neuronové síti se dvěma skrytými vrstvami:

$$N_{skrytá1} = N_{výstup} * \sqrt[3]{\left(\frac{N_{vstup}}{N_{výstup}}\right)^2} \quad (8 - 2)$$

$$N_{skrytá2} = N_{výstup} * \sqrt[3]{\frac{N_{vstup}}{N_{výstup}}} \quad (8 - 3)$$

Cílem celého zadání bylo, aby neuronová síť měla hlavně co nejrychlejší odezvu. Už toto zadání směřovalo k tomu, aby v neuronové síti bylo použito co nejméně neuronů, jak to jen bude možné, ale nesmělo to být na úkor kvality výstupu z neuronové sítě.

Nejdříve byla vyzkoušena neuronová síť se dvěma skrytými vrstvami. Vzhledem k 35 neuronům ve vstupní vrstvě, byl pro první pokus nastaven počet neuronů ve skrytých vrstvách na 16 neuronů v první skryté vrstvě a na 7 neuronů v druhé skryté vrstvě. Ve výstupní vrstvě jsou tři neurony. Po několika testech i s různým počtem neuronů ve vrstvách nebylo dosaženo rozumné úspěšnosti v rozumném čase. Proto tato varianta byla po těchto testech ihned zavrhnuta a nebyla ani důkladně testována. I přes to byly ve

zdrojovém kódu nechány prvky, které by umožnily snadno vytvořit neuronovou síť s dvěma skrytými vrstvami.

Mnohem lepších výsledku pro řešený problém bylo dosaženo při testování neuronové sítě s jednou skrytou vrstvou. Testování začalo s 11 neurony ve skryté vrstvě. Na tomto počtu neuronů se začínalo jako na doporučené hodnotě spočítané pomocí vzorce 8 - 1.

U požadované neuronové sítě, která má 35 vstupních neuronů a 3 výstupní neurony, byla testována rychlost a úspěšnost učení se 3 až 38 neurony ve skryté vrstvě.

Nejvhodnější výsledný počet neuronů ve skryté vrstvě byl po testování stanoven na 18. Výsledky testování pro tento počet neuronů jsou v následující kapitole.

## 8 VYHODNOCENÍ VÝSTUPŮ Z NEURONOVÉ SÍTĚ

S finální neuronovou sítí byla provedena skupina výkonnostních testů, které měly ukázat na efektivitu použití neuronové sítě při řízení autonomního robota.

Většina testů byla prováděna během učení neuronové sítě, neboť při samotném zpracování vstupu a jeho vyhodnocení bylo nereálné jakýmkoliv způsobem měřit čas zpracování vstupu kvůli velmi velké rychlosti neuronové sítě.

Tabulka s měřením, jak dlouho trvala jedna iterace učení (včetně otestování dosavadní úspěšnosti učení) a jak dlouho trvalo 500 iterací učení do resetu vah (důvod vysvětlen v jedné z předchozích kapitol) pro finální neuronovou síť (jedna skrytá vrstva, počet neuronů ve skryté vrstvě: 18):

Tabulka 2 – Délka učení neuronové sítě v závislosti na počtu iterací

| Počet iterací | Potřebený čas [s] |
|---------------|-------------------|
| 1             | 0,031             |
| 500           | 15,680            |

Následující tabulka obsahuje údaje o tom, kolik bylo potřeba iterací pro naučení neuronové sítě tak, aby chyba při testování úspěšnosti naučení klesla pod 30 %. Počet iterací je vždy (0 nebo 500) + 400, kvůli resetování vah po 500 iterací a následně minimálně 400 iterací pro dosažení co nejlepšího výsledku učení (viz předchozí kapitoly). Testování probíhalo s finální neuronovou sítí (jedna skrytá vrstva, počet neuronů ve skryté vrstvě: 18)

Tabulka 3 – Počet iterací a potřebné délky k učení k dosaženému výsledku učení

| Počet iterací | Celkový čas [mm:ss,00] | Výsledek učení [% chyb/100] |
|---------------|------------------------|-----------------------------|
| 3900          | 02:03,28               | 0,23                        |
| 1900          | 01:04,51               | 0,2                         |
| 1900          | 01:05,67               | 0,17                        |
| 400           | 00:13,62               | 0,27                        |
| 400           | 00:12,83               | 0,27                        |
| 2400          | 01:19,18               | 0,25                        |
| 2400          | 01:13,59               | 0,13                        |

Poslední tabulka obsahuje obecné statistické informace o rychlosti vyhodnocování neuronové sítě a o rychlosti jejího učení. Údaje, které jsou uvedeny v procentech, uvádí informaci o tom, jaký podíl času byl pro danou úlohu spotřebován z celkového času učení. Výsledky jsou opět pro finální neuronovou síť (jedna skrytá vrstva, počet neuronů ve skryté vrstvě: 18):

Tabulka 4 – Obecné informace o výkonu neuronové sítě

|  |                          |
|--|--------------------------|
| Učení neuronové sítě   | 100%                     |
| Načtení trénovací a testovací množiny ze souboru                         | 0,11%                    |
| Trénování neuronové sítě:  | 98,13%                   |
| Opravy vah neuronů:  | 54,76%                   |
| Nastavování vstupu do neuronové sítě (učení i testování):                | 21,57%                   |
| Testování úspěšnosti naučení neuronové sítě:                             | 21,43%                   |
| Ostatní operace  | 1,76%                    |
|  |                          |
| Získání výstupu z neuronů (sigmoida):                                    | 7%                       |
|  |                          |
| Aktivace neuronové sítě (bez nastavování vstupu, pouze získání výstupu): | 0,50854 ms               |
| Nastavení vstupu do neuronové sítě:                                      | $6,727 \cdot 10^{-5}$ ms |

## 9 DALŠÍ MOŽNOSTI VÝVOJE

Do algoritmu neuronové sítě by bylo možné dále implementovat paměť, která by umožnila realizovat pokročilejší chování umělé inteligence.

Konkrétně by se jednalo například o tyto situace či chování:

- Mapa prostoru, který robot procestoval, což by robotovi umožnilo bez jakéhokoliv zásahu cestovat z bodu A do bodu B. K řešení tohoto problému by bylo potřeba implementovat typ neuronové sítě SOMA. Je důležité, aby byl tento způsob řešen v rámci nadřazené řídicí aplikace, která má přístup k ovládání motorů a má od nich zpětnou vazbu. Bez informace o pozici či o ujeté vzdálenosti není možné mapu ukládat.
- V rámci knihovny implementovat funkci, která by si pamatovala spektra laserového senzoru z pozic, ve kterých se robot již nacházel. Pokud by tato funkce vyhodnotila, že se robot příliš často a v příliš krátkých časových úsecích vrací do stejného místa, mohla by „přeučit“ neuronovou síť (neboli přímo bez neuronové sítě rozhodnout) a zkusit poslat robota jiným směrem, protože by tato funkce mohla předpokládat, že se robot na základě rozhodnutí z neuronové sítě zacyklil v nějakém malém prostoru. Pro implementaci takové funkce a paměti by bylo potřeba už reálného testování s dokončeným robotem, aby bylo možné správně určit časové úseky, ve kterých by se data mohla bez povšimnutí opakovat a kdy by se vyhodnotila jako „kroužení v kruhu“. Dokud nevíme, jak rychle bude schopen robot jezdit, nemůžeme tyto časové úseky ani odhadnout.

## ZÁVĚR

Cílem této diplomové práce bylo vytvořit umělou inteligenci pomocí neuronové sítě, která by měla na starosti rozhodování a řízení autonomního robota, který je vyvíjen v rámci Fakulty aplikované informatiky na Univerzitě Tomáše Bati ve Zlíně. Hlavním bodem zadání bylo vytvořit neuronovou síť, která by měla rychlou odezvu na vložené spektrum dat získané z laserového skeneru, který je součástí robota.

Vytvořit tuto umělou inteligenci se podařilo s velmi dobrými výsledky. Byla implementována vícevrstvá neuronová síť s algoritmem učení Backpropagation, která je obsahem hlavní DLL knihovny, která bude vložena jako celý blok do hlavní řídicí aplikace robota. Součástí řešení je také aplikace pro vytvoření trénovací množiny neuronové sítě a aplikace, ve které je možné si implementovanou neuronovou síť vyzkoušet naučit pomocí trénovací množiny a následně vyzkoušet odezvu neuronové sítě na vložené vzory dat. Vzhledem k dosaženým výsledkům je jisté, že tato DLL knihovna bude použita při konečném řešení ovládání autonomního robota pomocí umělé inteligence.

Během implementace a testování neuronové sítě bylo zjištěno, že každý problém, který je řešen pomocí neuronových sítí, potřebuje odlišné nastavení parametrů neuronové sítě. A to i v případě, že řešené problémy jsou velmi podobné. Pokud tedy dojde k plánované výměně laserového skeneru robota, bude také potřeba pozměnit parametry sítě. Během vývoje této umělé inteligence byla však tato situace známa a neuronová síť je implementována tak, aby po výměně laserového skeneru bylo potřeba co nejméně úprav v neuronové síti a aby tyto úpravy nebyly náročné ani složité. Při zohlednění této situace bylo zjištěno, že i dva velmi blízké problémy nemohou použít shodnou neuronovou síť, ale v případě mého řešení bude stačit provést pouze drobné úpravy jako je počet neuronů ve skryté vrstvě a otestovat tyto změny.

Tato práce mi dala mnoho cenných zkušeností, hlavním důvodem bylo, že výsledek mé práce bude použit na reálném projektu. Tedy, že neuronová síť, kterou jsem vytvořil, bude použita přímo k řízení autonomního robota jako umělá inteligence.

Výsledná neuronová síť splňuje požadavky pro použití v autonomním robotu, je optimalizován proces učení a co více, způsob implementace této neuronové sítě umožňuje téměř okamžitou odezvu, takže během řízení robota nebude docházet k žádnému zpoždění, které by teoreticky mohlo být rozhodováním umělé inteligence způsobeno.



Vytvořit spolehlivou umělou inteligenci bylo velmi důležité, neboť pro automatické řízení robota je právě tato neuronová síť tou nejdůležitější částí, aby byl robot schopen bezpečného pohybu.

Kromě velkého množství znalostí v oblasti umělé inteligence, neuronových sítí a jejich implementace, mi tato práce dala také velké množství zkušeností v komunikaci se členy týmu, kteří se podílejí na vývoji celého robota. Věřím, že všechny znalosti a zkušenosti získané v rámci této diplomové práce pro mě budou velkým přínosem pro mou praxi v budoucnu.

## ZÁVĚR V ANGLIČTINĚ

The aim of this thesis was to create artificial intelligence in the form of neural network which would ensure decision-making and controlling of an autonomous robot which is being developed at the Faculty of Applied Informatics of Tomas Bata University in Zlín. The main point of the assignment was to create a neural network which would have a quick response to the inserted data spectrum gained from the laser scanner which is a part of the robot.

This artificial intelligence was created successfully. A multi-layer neural network with the learning algorithm Backpropagation was implemented. This network is included in the main DLL library which will be inserted as a whole block into the robot's main controlling application. A part of the solution is also an application for creation of a training set of values and an application in which it is possible to try training of the implemented neural network with the training set of values and subsequently to test the neural network response to inserted data patterns. Considering the achieved results, it is certain that this DLL library will be used in the final solution of the autonomous robot control by means of artificial intelligence.

During implementation and testing of the neural network it was found out that each problem which is handled by means of neural networks needs a different parameter setting in the neural network. This is the case even if these handled problems are very similar. In case of planned robot's laser scanner replacement, it will also be necessary to adjust network parameters. However, during the implementation of this artificial intelligence this situation was known. Therefore, the implementation was carried out in such a way so as in the event of laser scanner replacement it is necessary to make as few changes in the neural network as possible. Moreover, these adjustments should be neither demanding nor complicated. While taking this situation into consideration, it was found out that even such similar problems cannot use the same neural network. In this case, however, it will be necessary to make only small adjustments such as the number of neurons in the hidden layer and test these changes.

I gained much valuable experience during writing this thesis. The main reason was the fact that the results of my work will be used in an actual project which means that the neural network I created will be used directly for the autonomous robot control as the artificial intelligence.

The final neural network complies with all the requirements for the use in an autonomous robot, the learning process is optimized and the way the neural network is implemented enables almost instant response so that during robot control there will be no delay which could be, in theory, caused by artificial intelligence's decision.

It was very important to create a reliable artificial intelligence because for the automatic robot control it is the neural network which is the most important part necessary for robot's safe movement.

Besides large amount of knowledge in the field of artificial intelligence, neural networks and their implementation, this project has given me also a great deal of experience with communication with members of the team who participate in the development of the robot. I believe that all the knowledge and experience gained during the work on this diploma thesis will be greatly beneficial to my future practice.

## SEZNAM POUŽITÉ LITERATURY

- [1] ZELINKA, Ivan. *Umělá inteligence I: Neuronové sítě a genetické algoritmy*. Brno: VUT v Brně, 1998, 126 s. ISBN 80-214-1163-5.
- [2] JONES, M. *Artificial intelligence: a systems approach*. Hingham, Mass.: Infinity Science Press, c2008, 498 s. ISBN 09-778-5823-5.
- [3] HU, Yu Hen a Jenq-Neng HWANG. *Handbook of neural network signal processing*. Boca Raton: CRC Press, c2002. ISBN 08-493-2359-2.
- [4] HEATON, Jeff. *Introduction to neural networks for C#*. 2nd ed. St. Louis: Heaton Research Inc, 2008. ISBN 16-043-9009-3.
- [5] ALPAYDIN, Ethem. *Introduction to machine learning*. 2nd ed. Massachusetts: MIT Press, c2010, 537 s. ISBN 978-026-2012-430.
- [6] KROSE, Ben a Patrick SMAGT. *An Introduction to Neural Networks*. The University of Masterdam, 1996, Eight edition, s. 135.
- [7] GUNNERSON, Eric. *Začínáme programovat v C#*. Praha : Computer Press, 2001. 316 s. ISBN 80-7226-525-3.
- [8] NAGEL, Christian, et al. *C# 2008 : Programujeme profesionálně*. Brno : Computer Press, 2009. 772 s. ISBN 978-80-251-2401-7.
- [9] Msdn [online]. © 2012 Microsoft Corporation. All rights reserved., 2012 [cit. 2012-05-18]. Msdn. Dostupné z WWW: <http://msdn.microsoft.com/en-us/default.aspx>
- [10] Stack Overflow [online]. 2012 [cit. 2012-05-18]. Dostupné z WWW: <http://stackoverflow.com/>.
- [11] FREEMAN, James A. *Neural networks - algorithms, applications and programming techniques*. Boston: Addison-Wesley, 1995, 401 s. ISBN 02-015-1376-5.
- [12] VOLNÁ, Eva. *Neuronové sítě 1*. Ostravská univerzita v Ostravě, 2008, s. 86.
- [13] SARANGAPANI, Jagannathan. *Neural network control of nonlinear discrete-time systems*. Boca Raton: CRC/Taylor, 2006, 602 p. Control engineering (Taylor, 21. ISBN 978-082-4726-775.
- [14] ŠTUSÁK, Vladimír. *Testovací prostředí pro výuku kurzů Microsoft IT Academy na UTB ve Zlíně*. UTB Zlín, 2011. Bakalářská. UTB Zlín. Vedoucí práce Ing. Lukáš Kouřil.
- [15] JetBrains. [online]. 2012 [cit. 2012-05-20]. Dostupné z: <http://www.jetbrains.com>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

MB      Megabyte

GB      Gigabyte

WPF    Windows Presentation Foundation.

OOP    Objektově orientované programování

NS      Neuronová síť

SOM    Samo-organizující se mapa (Kohonenova mapa)

## SEZNAM OBRÁZKŮ

|  |    |
|--|----|
| Obrázek 1 – Autonomní robot konstruován na FAI UTB ve Zlíně .....  | 14 |
| Obrázek 2 – Výbava robota – ultrazvukový senzor.....   | 16 |
| Obrázek 3 – Výbava robota – laserový senzor (aktuální).....  | 17 |
| Obrázek 4 – Výbava robota – laserový senzor (plánovaný) .....  | 19 |
| Obrázek 5 – Obecné schéma neuronu.....   | 21 |
| Obrázek 6 – Matematické znázornění neuronu (převzato z [1]).....   | 21 |
| Obrázek 7 – Ukázka jednovrstvé neuronové sítě Perceptron se třemi vstupy a jedním<br>nebo dvěma výstupy (převzato z [2]) ..... | 23 |
| Obrázek 8 – Vícevrstvý Perceptron (převzato z [2]) .....   | 24 |
| Obrázek 9 – Madaline se dvěma skrytými neurony Adaline a jedním výstupním<br>neuronem Adaline (převzato z [12]) .....          | 25 |
| Obrázek 10 – Schéma neuronové sítě s algoritmem Backpropagation<br>(převzato z [11]) .....                                     | 26 |
| Obrázek 11 – Schéma pro postup volby správného počtu neuronů ve skryté vrstvě<br>(převzato z [1]) .....                        | 27 |
| Obrázek 12 – Aplikace Kohonenovy mapy, sekvence diagramů ilustrující evoluci<br>mapy podle předlohy (převzato z [11]) .....    | 28 |
| Obrázek 13 – Aplikace Kohonenovy mapy, rozpoznání vzoru (převzato z [11]) .....  | 28 |
| Obrázek 14 – Aplikace Kohonenovy mapy, rozpoznání vzoru, přeneseně například<br>rozpoznání cesty (převzato z [11]) .....       | 28 |
| Obrázek 15 – Schéma samoorganizující se mapy (převzato z [11]).....  | 29 |
| Obrázek 16 – Hopfieldova neuronová síť (převzato z [1]).....   | 30 |
| Obrázek 17 – Hopfieldova síť – jiné znázornění (převzato z [11]) .....   | 31 |
| Obrázek 18 – Neuronová síť CLN (převzato z [1]).....   | 32 |
| Obrázek 19 – Neuronová síť BAM (převzato z [11]).....  | 33 |
| Obrázek 20 – Jiné schéma neuronové sítě BAM s dvousměrnou asociativní pamětí<br>(převzato z [13]) .....                        | 33 |
| Obrázek 21 – Schéma zpracování obrazového vstupu do neuronové sítě (převzato z<br>[11]).....                                   | 36 |
| Obrázek 22 – Naměřené spektrum vykreslené do grafu – chodba .....  | 38 |
| Obrázek 23 – Třídy DLL knihovny – „DirectionDetermination“, „Network“<br>a „Layer“ .....                                       | 44 |

|   |    |
|---|----|
| Obrázek 24 – Třídy DLL knihovny – „Neuron“ a „Weight“ .....       | 45 |
| Obrázek 25 – Třídy DLL knihovny – „NeuralData“ a „Spectrum“ ..... | 53 |
| Obrázek 26 – Ukázka trénovací množiny dat .....                   | 54 |
| Obrázek 27 – Možné výsledné směry pro pohyb robota .....          | 58 |
| Obrázek 28 – Aplikace UICreateLearningData.....                   | 61 |
| Obrázek 29 – Menu aplikace UICreateLearningData.....              | 62 |
| Obrázek 30 – Menu aplikace UIDirectionDeterminationByAI.....      | 63 |
| Obrázek 31 – Aplikace UIDirectionDeterminationByAI .....          | 65 |

**SEZNAM TABULEK**

|  |    |
|--|----|
| Tabulka 1 – možné výsledné směry pro pohyb robota .....                              | 59 |
| Tabulka 2 – Délka učení neuronové sítě v závislosti na počtu iterací.....            | 69 |
| Tabulka 3 – Počet iterací a potřebné délky k učení k dosaženému výsledku učení ..... | 69 |
| Tabulka 4 – Obecné informace o výkonu neuronové sítě .....                           | 70 |

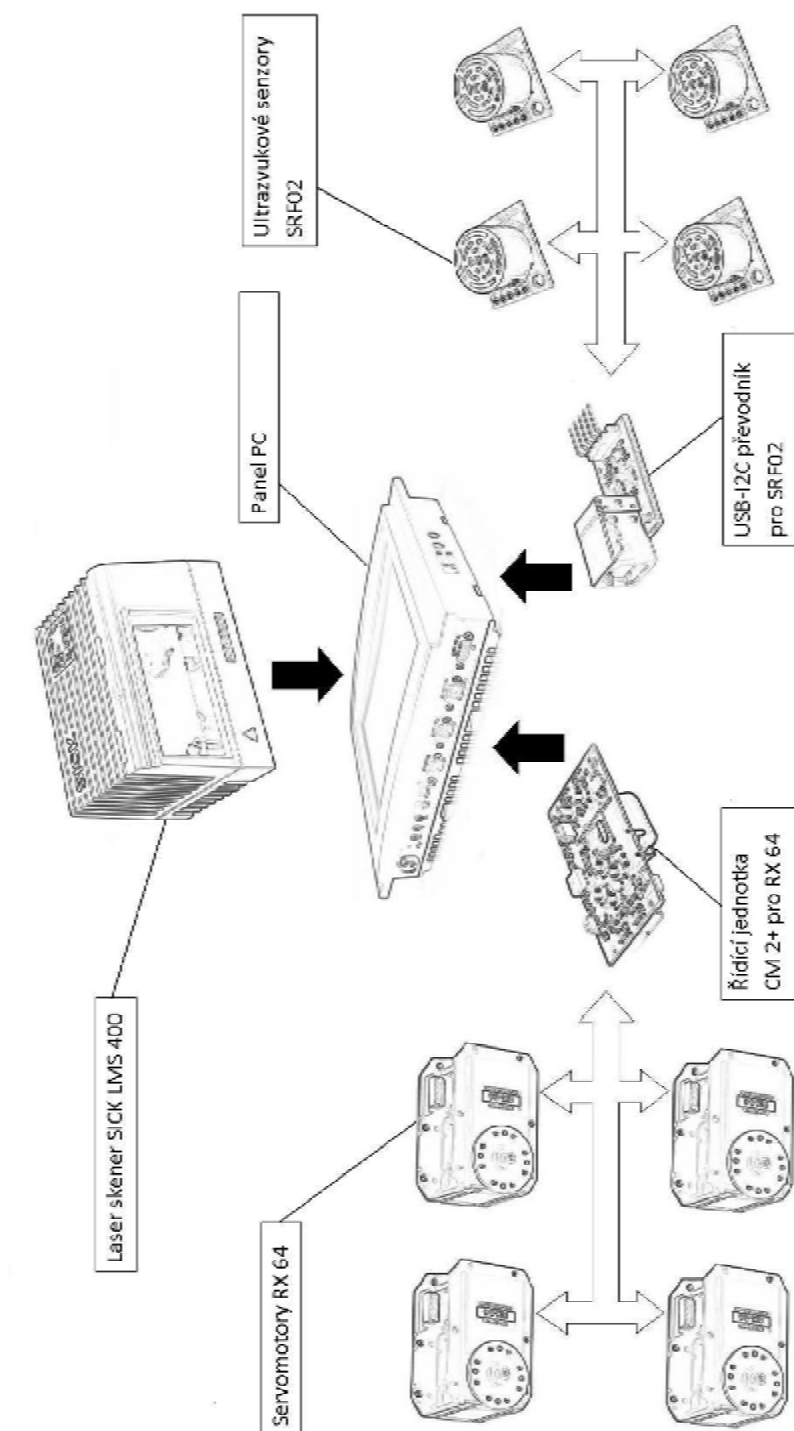


## SEZNAM PŘÍLOH

P I Schématické znázornění propojení subsystému autonomního robota

P II Třídy DLL knihovny

## PŘÍLOHA P I: SCHÉMATICKÉ ZNÁZORNĚNÍ PROPOJENÍ SUBSYSTÉMŮ AUTONOMNÍHO ROBOTA.



## PŘÍLOHA P II: TRÍDY DLL KNIHOVNY

