UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2020 Anežka Blažková

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

OVLÁDACÍ SW ROBOTA UR3 PRO AUTOMATIZOVANÉ HRANÍ PIŠKVOREK

Anežka Blažková

Bakalářská práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Anežka Blažková**
Osobní číslo:	**I18296**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Řízení procesů**
Téma práce:	**Ovládací SW robota UR3 pro automatizované hraní piškvorek**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování

Cílem bakalářské práce je tvorba ovládacího SW robota Universal Robots UR3 pro automatizované hraní piškvorek. Ovládací software bude vytvořen v prostředí MATLAB a bude obsahovat jednoduché uživatelské rozhraní. Pro účely řešení bude stanoveno hrací pole, ve kterém budou probíhat tahy piškvorkové hry mezi člověkem a počítačem. Počítač bude zpracovávat obrazová data, volit tahy pro strategické hraní hry, komunikovat s robotem, ovládat a plánovat pohyby robota. Dále musí být zajištěna bezpečnost při tazích člověka. Práce bude obsahovat rešerši algoritmů a strategií pro hraní piškvorek, popis typů pohybu robota Universal Robots UR3, bezpečnost v robotice, popis metody pro zpracování obrazových dat a popis algoritmů pro kreslení tahů robotem na hrací pole.

Rozsah pracovní zprávy: **50**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

CORKE, Peter. Robotics, vision and control. New York, NY: Springer Berlin Heidelberg, 2017. ISBN 978-3-319-54412-0.
ZÁDA, Václav. Robotika: matematické aspekty analýzy a řízení. Liberec: Technická univerzita v Liberci, 2012. ISBN 978-80-7372-882-3.
DUŠEK, F., HONC, D. Matlab a Simulink, Úvod do používání. skriptum, Univerzita Pardubice, vydání první, Pardubice, 2005.

Vedoucí bakalářské práce: **Ing. Dominik Štursa**
Katedra řízení procesů

Datum zadání bakalářské práce: **17. prosince 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Daniel Honc, Ph.D.
vedoucí katedry

V Pardubicích dne 20. ledna 2020

Prohlášení autora

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 08. 2020

Anežka Blažková

Poděkování

Ráda bych tímto poděkovala vedoucímu bakalářské práce Ing. Dominikovi Štursovi za cenné rady, konzultace a trpělivost při testování aplikace.

V Pardubicích dne 10. 08. 2020

Anežka Blažková

ANOTACE

Cílem bakalářské práce je vytvořit software pro robotické rameno UR3 v prostředí Matlab, které bude fungovat jako automatický protivník proti uživateli. Teoretická část je věnována problematice hry piškvorky, volbě programovacího prostředí, algoritmu minimax a konvolučním neuronovým sítím. Je zde také seznámení s robotem a bezpečnost při zacházení s ním. Praktická část popisuje vzhled prostředí, jak probíhá snímání hrací plochy, čtení dat a jejich vyhodnocení.

KLÍČOVÁ SLOVA

Piškvorky, Universal Robots, UR3, Matlab, minimax, konvoluční neuronové sítě

TITLE

SW FOR AUTOMATED TIC-TAC-TOE PLAYING WITH UR3 ROBOT

ANNOTATION

The objective of this work is to create software for the UR3 robotic arm in the Matlab environment, which will then work as an automated opponent against a user. The theoretical part is dedicated to problematics of the game tic-tac-toe, the choice of programming environment, minimax algorithm and convolutional neural networks. There is also an introduction to the robot and safety while working with it. The practical part describes the looks of the environment, how the playing area is scanned, reading data and their evaluation.

KEYWORDS

Tic-tac-toe, Universal Robots, UR3, Matlab, minimax, convolutional neural networks

OBSAH

SEZNAM ZKRATEK

GUI	Graphical User Interface
IP	Internet Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UR3	Universal Robots
USB	Universal Serial Bus

SEZNAM ZNAČEK (SYMBOLŮ, PROMĚNNÝCH A FUNKCÍ)

b	faktor větvení tahového stromu
d	prohledávaná hloubka tahového stromu
f	přenosová funkce neuronu
O	notace velké O neboli Landauova notace
x	vstupy neuronu
w	paměť neuronu
w_i	váhy neuronu
x_i	vstupy neuronu
y	výstup neuronu
θ	práh

SEZNAM ILUSTRACÍ

ÚVOD

Roboti během posledních let nahrazují člověka ve spoustě odvětví. Jedná se především o práce, které mohou být pro člověka nebezpečné, práce, kde je dbán veliký důraz na preciznost, anebo práce, které jsou pro člověka repetitivní a automatizace je často z dlouhodobého hlediska vhodnější. Takové využití robota je pro většinu lidí dnes již známa, ale spousta z nich si neuvědomí, jaká komplexnost se za tím vším skrývá.

Deskové hry existovaly již tisíce let před naším letopočtem. Vždy se jednalo o způsob zábavy, socializace, ale i hazardu. Od té doby však došlo ke vzniku nespočetného množství variant her a růst neustává. S příchodem internetu bylo jen otázkou času, kdy se i tato tradiční zábava přenese do digitální podoby. Díky této expanzi došlo také ke vzniku umělého protivníka ovládaného počítačem a lidem se znovu usnadnila možnost rozptýlení se od každodenního života.

Cílem práce je spojit tyto dvě zprvu odlišná odvětví a nastínit tak možnost dalšího rozšíření pro obě strany. Zavést robota do zábavního průmyslu, který je s lidmi již tisíce let, a zároveň vylepšit umělého protivníka tím, že je mu dána s pomocí robota fyzická podoba. Zároveň je zde podrobně vysvětleno, jak komplexní, ač na oko jednoduchý, takový úkon může být a co všechno je k tomu potřeba. Robotická paže bude schopna být oponentem ve hře piškvorky. Toho se dosáhne pomocí programovacího prostředí Matlab, obecně známých algoritmů pro provedení tahu a znalostí neuronových sítí. Robot je schopný rozeznat hrací plochu a podle ní provést tah v souladu s pravidly hry. Je zde také dbán velký důraz na bezpečnost, která je důležitá v jakémkoli prostředí s umělou inteligencí.

1 TEORIE HRY PIŠKVORKY

Piškvorky jsou strategická hra, ve které se střídají a soupeří dva hráči. Nejčastěji se tato hra hraje na čtverečkovaném papíře, kde hráči střídají symboly křížků a koleček. Vyhrává hráč, jenž jako první vytvoří nepřerušenou řadu pěti svých symbolů.

Mezinárodně je tato hra nazývána gomoku z japonského slova Go, tedy pět a Moku, česky průsečík.

1.1 PRAVIDLA HRY

Piškvorky se standardně hrají na ploše tvořené z 15 vodorovných a 15 svislých čar, které utvoří síť s 225 průsečíky. Hráči na začátku hry spravedlivě určí, kdo bude začínat a zvolí si buď modrý křížek nebo červené kolečko. Alternativně lze také použít černé a bílé kameny, poté bude začínat vždy hráč s černým kamenem. Hráči se střídají v zakreslování svých značek do sítě vždy poté, co druhý hráč provede svůj tah. Tahem se myslí zakreslení křížku nebo kolečka do čtverce na hrací ploše.

Cílem hry je poskládat nejméně pět svých symbolů za sebou, ať už svisle, horizontálně nebo diagonálně. Protivník se tomu snaží zabránit a sám poskládat svých alespoň pět symbolů za sebe. Hra končí, pokud se jednomu z hráčů podaří za sebe poskládat pět a více svých symbolů a stává se tak vítězným hráčem (piskvorky.cz, 2019).

1.2 HERNÍ STRATEGIE

Tato hra má matematické řešení. Pro piškvorky na omezené i neomezené hrací ploše existuje neprohrávající strategie pro začínajícího hráče. Toto plyne z argumentu o kradení strategie, který tvrdí, že v každé silné poziční hře má začínající hráč neprohrávající strategii.

Tvrzení o tom, že začínající hráč na ploše 15x15, ale i větší, má vždy vítěznou strategii dokázal holandský expert L. Victor Allis. Velikost pole zde hraje velkou roli, neboť ji nemůže druhý hráč využít k vynucení remízy (Allis, 1994).

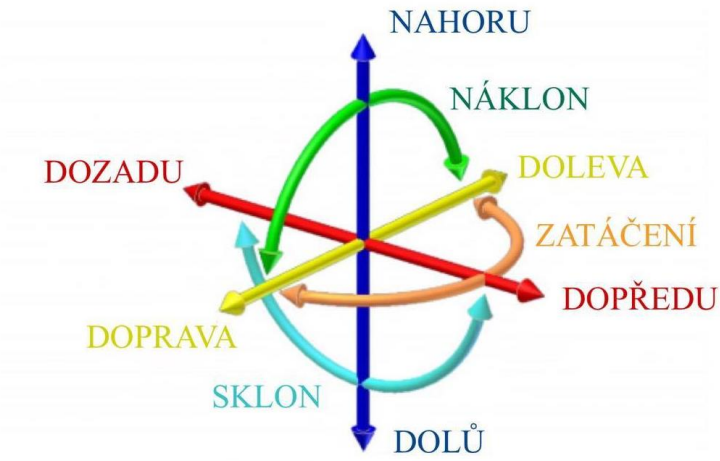
2 UNIVERSAL ROBOTS UR3

Tento typ sériových robotů je vytvářen dánskou společností Universal Robots a jedná se o jeden z prvních a nejmenších modelů. Své využití nachází v odvětví elektrotechnickém, vědeckém, zemědělském nebo farmaceutickém. Dalšími typy ze stejné rodiny jsou UR5 a UR10, kdy každý disponuje větším dosahem a schopností zvedat těžší předměty. Robot UR3 je schopný zvedat předměty do hmotnosti 3 kilogramy, což je uvedeno v jeho názvu. Jeho pracovní prostor má dosah 500 milimetrů od kloubu základny a robot váží 11 kilogramů. Daň za jeho malou váhu a velikost si vzala přesnost pohybu, která je pouze 0,1 milimetrů.

Robot má speciální bezpečnostní funkce primárně určené ke kolaboraci s člověkem, ale jen v rámci aplikací, které nepředstavují riziko při použití. Díky této spolupráci se dá využít pro různé montážní práce potřebné ve všech odvětvích. Pokud se jedná o práce jednotvárné nebo v nebezpečném prostředí, lze robota použít i samostatně jen s pomocí naprogramování (Universal Robots, 2019).

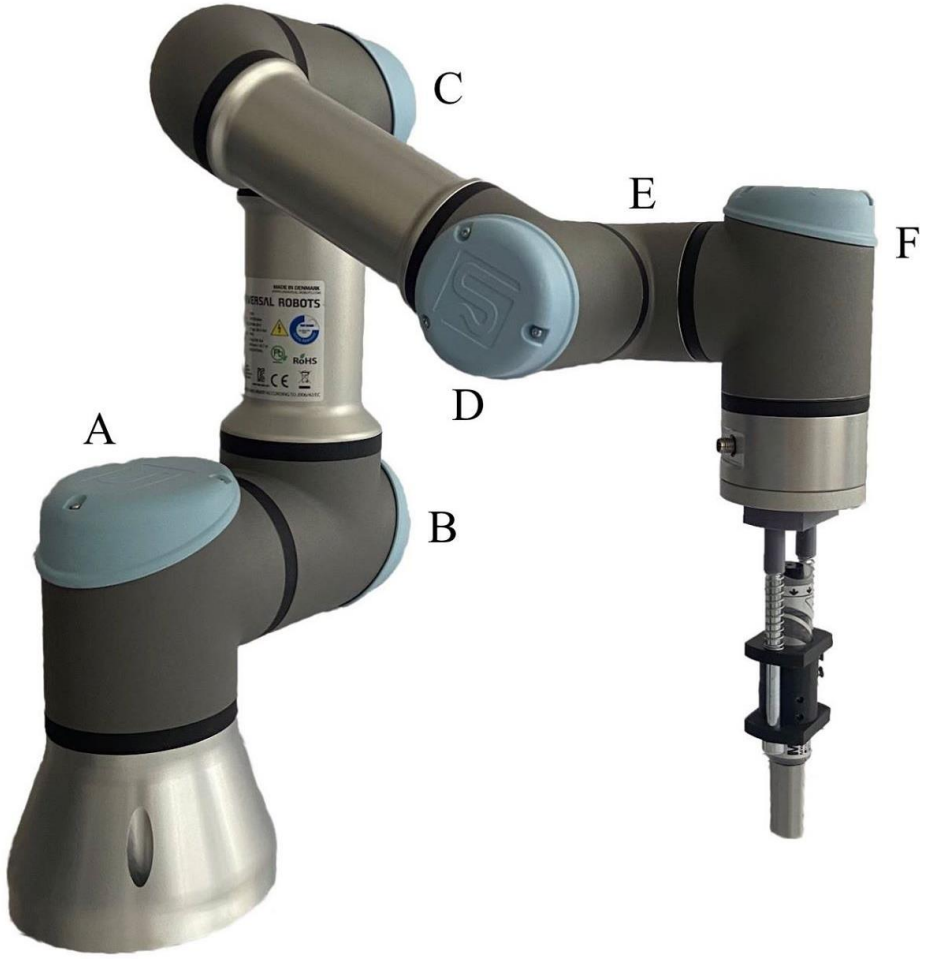
2.1 POPIS POHYBŮ ROBOTA

Robot má celkem šest otáčivých stupňů volnosti pro zajištění bezproblémového pohybu, viz obr. 1.



Obr. 1 - 6 otáčivých stupňů volnosti

Má schopnost pohybovat každým svým kloubem v mezích ±360°. Tyto klouby se nacházejí mezi ramenem, loktem a dvěma zápěstími, jak lze vidět na obr. 2. Třetí zápěstí je vybaveno kloubem s nekonečným otáčením, aby se dosáhlo jeho maximálního využití. Všechny části mají společnou rychlost otáčení 360°·s^{-1} (Universal Robots, 2019).

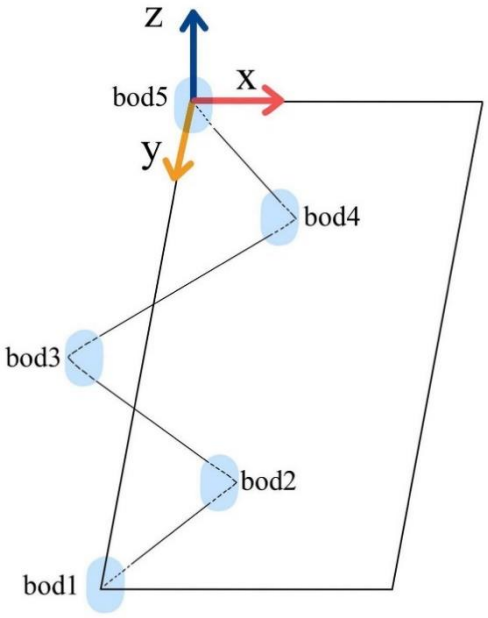


Obr. 2 - Popis robota UR3-A: Základna, B: Rameno, C: Loket a D, E, F: Zápěstí

Aby se dosáhlo vyšší efektivity pohybu, dochází také k hodnocení toho, jakým způsobem se robot dostane do požadované pozice. Jedná se o pohyb lineární, kloubový a smíšený.

Lineární pohyb

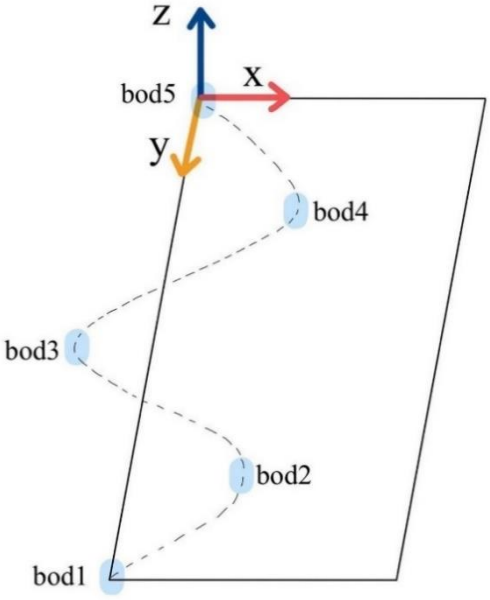
Pohyb lze jednoduše vysvětlit jako pohyb po přímce z jednoho bodu do druhého. Jedná se o nejjednodušší a nejbezpečnější z pohybů a dá se díky tomu dosáhnout preciznosti.



Obr. 3 - Lineární pohyb

Kloubový pohyb

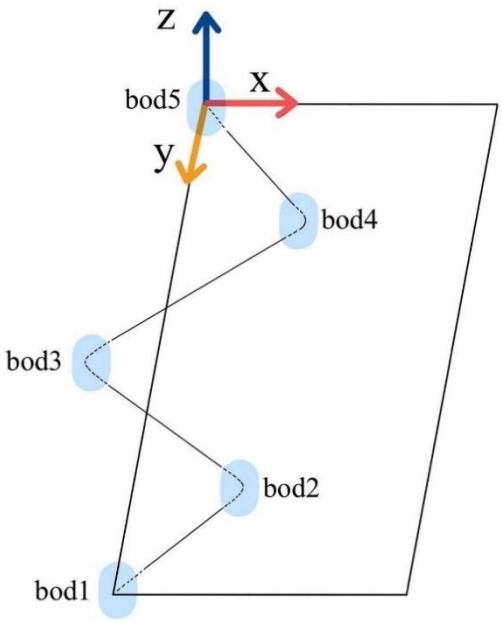
Důležité při tomto pohybu je, aby se všechny klouby dostaly do požadované polohy ve stejný moment. Tento pohyb je nejrychlejší díky tomu, že se pohyb vykoná v nedefinované trajektorii.



Obr. 4 - Kloubový pohyb

Smíšený pohyb

Jedná se o lineární pohyb, který se mezi jednotlivými body ovšem neohýbá ostře, ale v obloucích, a tím se značně zvýší jeho efektivita v podobě rychlosti.



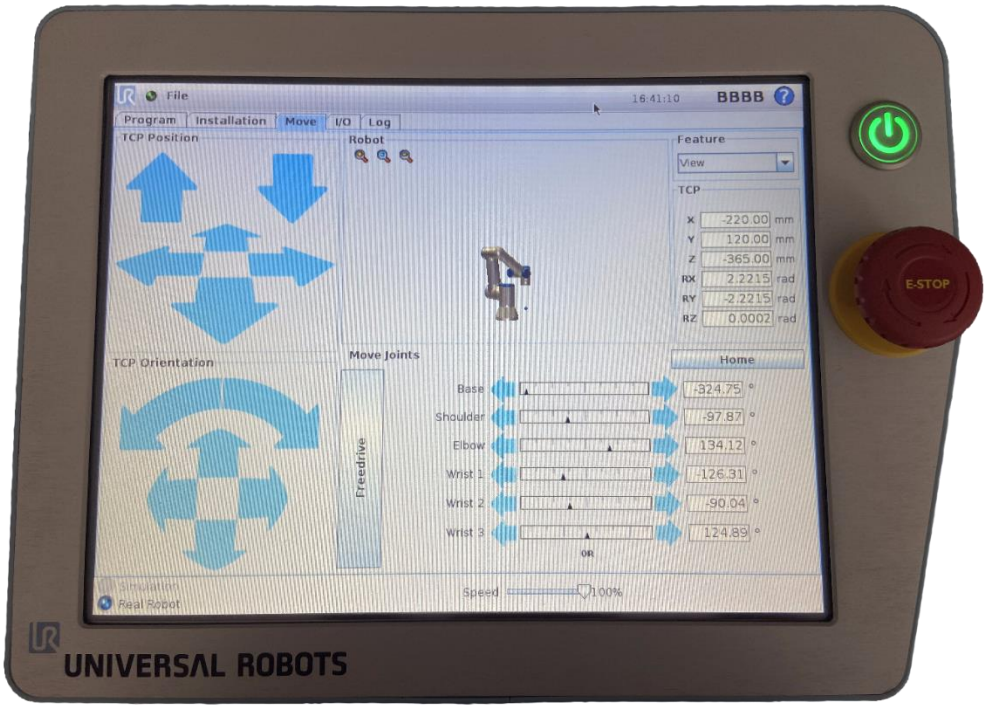
Obr. 5 - Smíšený pohyb

2.2 POLYSCOPE

Veškeré ovládání, plánování pohybů a programování robota je zprostředkováno prostředím PolyScope na řídicí jednotce. Je graficky zpracováno pro snadné použití do několika záložek pro jednotlivé operace.

Dotykové zařízení nebo také Teach Pendant slouží k jednoduchému ovládání celého ramena, zároveň nabízí tlačítko pro spuštění volného pohybu robota nebo tlačítko E-STOP, které při nouzovém stavu zastaví robota.

V záložce pohyb lze ovládat jednotlivé klouby robota a lze tak provádět pohyb posuvný nebo rotační. Pracuje se zde s veličinou aktuální polohy, která je složena ze šesti souřadnic pro každý stupeň volnosti. Jednotlivé hodnoty lze zadat manuálně, ale mohou se také měnit ovládáním robota pomocí šipek podle potřeb uživatele, viz obr. 6.



Obr. 6 - Možnosti pohybu robota

Záložka pro program umožňuje vytvoření programu od začátku, anebo editaci již nahraného programu do robota. Pro bezpečnost používání je zde také umístěný posuvník pro ovládání rychlosti pohybu, jsou zde také tlačítka pro pozastavení a spuštění robota.

Rameno UR3 však není možné ovládat přímo programovacím prostředím Matlab. Je proto nejprve nutné vytvořit program v PolyScope a následně obě prostředí propojit pomocí TCP/IP komunikace (Universal Robots, 2019).

3 MATLAB

Jedná se o programovací prostředí s vlastním jazykem pro technické výpočty v řadě odvětví. Jazyk je interpretovaný, což znamená, že nekompiluje kód předem, ale postupně ho překládá.

Využívá se především pro různé simulace, měření, analýzu dat, vizualizaci a vývoj algoritmů. Nejčastěji se používá v oblastech robotiky, aplikované matematiky, zpracování signálu a komunikací. Zasíláním jednoduchých příkazů je možné ovládat rameno UR3 díky zavedené komunikaci.

Matlab obsahuje základní uživatelské rozhraní, do kterého lze zadávat jednotlivé operace, uchovávat data ve workspace a následně je uložit nebo načíst podle potřeby (Dušek, 2000). Pro vytvoření plnohodnotné aplikace je však potřeba použití GUI.

3.1 GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ

GUI sestává z grafických objektů, které usnadňují práci s programem. S použitím tlačítek, posuvných seznamů, grafů apod. je možnost přiblížit složitý algoritmus i lidem s minimálními znalostmi práce s Matlabem. GUI se může vytvořit dvěma způsoby. Liší se od sebe jak způsobem vizualizace, tak i využitelností u různých typů aplikací.

Vytvoření rozhraní pomocí GUIDE

První možností je použít vývojové prostředí pro grafické uživatelské rozhraní. Zadáním příkazu se spustí základní pracovní oblast. Z lišt se vybírají komponenty pro vizualizaci programu a jeho následnou úpravu a nastavení. Celé použití GUIDE je velice snadné a rychlé a má za výhodu možnost dosáhnout téměř totožného výsledku s návrhem. Je proto vhodnou volbou jak pro jednoduché, tak i pro rozsáhlé aplikace.

Vytvoření rozhraní programováním

Tento druhý způsob využívá syntaxí k vytvoření vlastního GUI pomocí pouze jednoho souboru M-file. Tento soubor obsahuje kód, který ovládá jak grafické rozhraní, tak i popis grafické části. Tento způsob se však stává velice komplexním pro větší projekty, proto se využívá především v případech, kdy není potřeba velké množství objektů. Pokud je ovšem třeba obsluhovat větší počet M-files nebo vyskakovací nabídky, je tento způsob lepší právě kvůli přímé programovatelnosti (Mathworks, 2020).

4 ALGORITMY

Algoritmus je postup nebo návod pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků. Je jím myšlen teoretický princip řešení problémů.

4.1 ALGORITMUS BRITSKÉHO MUZEA

Algoritmus Britské muzeum je obecný přístup k nalezení nějakého řešení zkontrolováním jednotlivých možností, počínaje tou nejmenší. Termín odkazuje na teoretickou, nikoli praktickou, techniku, kdy je obrovský počet možností.

4.2 MINIMAX A JEHO OPTIMALIZACE

Minimax je algoritmus, který je používaný pro hraní strategických her mezi dvěma hráči. Určuje nejlepší tah na základě prozkoumání herního stromu vycházejícího z aktuální pozice a je založený na prohledávání do hloubky. Principem je procházení a minimalizace maximálních možných ztrát.

Minimax prochází všechny možné tahy, které lze v daný moment, respektive v dané hloubce (úrovni), provést. Zároveň následně řeší, jaké tahy lze v návaznosti provést až do doby, kdy se dostane ke konci hry. Výsledek hry je číselně ohodnocen, podle toho, zda došlo k výhře jednoho z hráčů, nebo k remíze. Algoritmus pracuje na principu, kdy se střídá vyhodnocení tzv. maximalizujícího a minimalizujícího hráče.

Výhře maximalizujícího hráče odpovídá návratová hodnota heuristické funkce 10, hráč se tedy bude po celou dobu hry snažit maximalizovat své skóre. Výhře minimalizujícího hráče odpovídá hodnota -10 a tento hráč se naopak snaží minimalizovat hodnotu skóre. Remíze odpovídá neutrální hodnota 0.

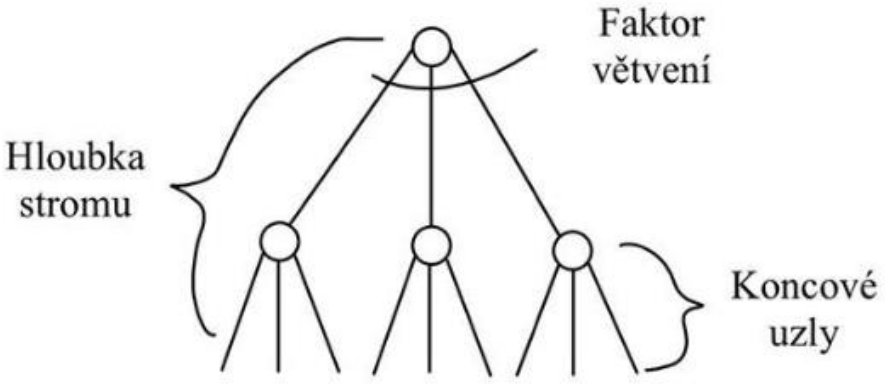
Časová složitost algoritmu je $O(b^d)$, kde b je faktor větvení odpovídající validním tahům a d je prohledávaná hloubka (resp. délka hry).

Na tomto principu lze optimalizovat určitými způsoby již vytvořený strom. Jedním ze způsobů je odstranění větví, které se nemusí procházet, protože nemohou jakkoli ovlivnit výsledek. Tento algoritmus se nazývá alfa-beta ořezávání (Winston, 2010).

4.2.1 Praktický příklad minimaxu

Tahový strom

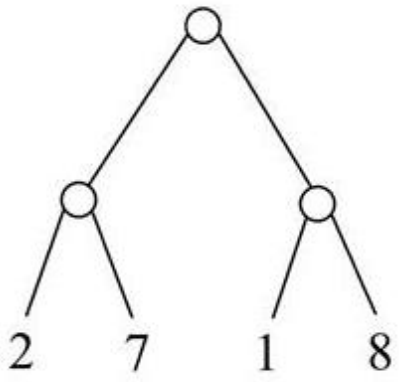
V tomto grafickém teoretickém znázornění existuje určitý počet výběrů a úrovní. První z veličin popisujících tahový strom se nazývá faktor větvení *b* udávající počet větví, ve kterých existují možné tahy. Druhá je hloubka *d* vyjadřující, kolikrát dojde k rozvětvení do hloubky. Takovýto strom má následně daný počet koncových uzlů, který se vypočítá jako mocnina výše uvedených veličin. Tento způsob je vhodný k ověření výběru algoritmu (Winston, 2010).



Obr. 7 - Popis částí tahového stromu

Princip algoritmu Minimax

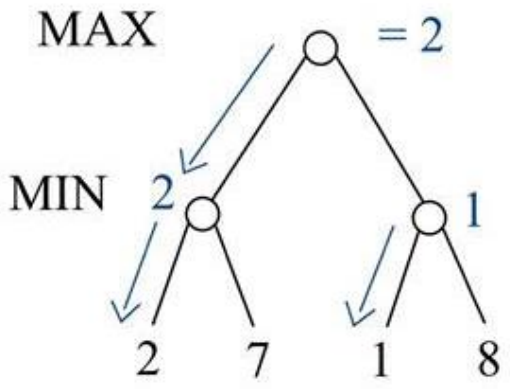
Odvíjí se od tahového stromu, kdy se jako příklad uvádí dva hráči. První chce dosáhnout vždy co největší hodnoty, a druhý hráč co nejmenší hodnoty.



Obr. 8 - Příklad tahového stromu

Následné vybírání začíná u konečných hodnot, ze kterých hráč na tahu vybere tu pro něj nejvýhodnější a posune ji o jednu výběrovou úroveň výš, aby svůj tah mohl provést i druhý hráč, viz obr. 9. Každý z hráčů vždy sníží hloubku stromu o jednu úroveň, až na vrchol

do výsledného tahu, který se provede a zaznamená se cesta, jakou se k němu došlo (Winston, 2010).



Obr. 9 - Příklad tahového stromu po aplikaci algoritmu minimax

4.3 ALFA-BETA OŘEZÁVÁNÍ

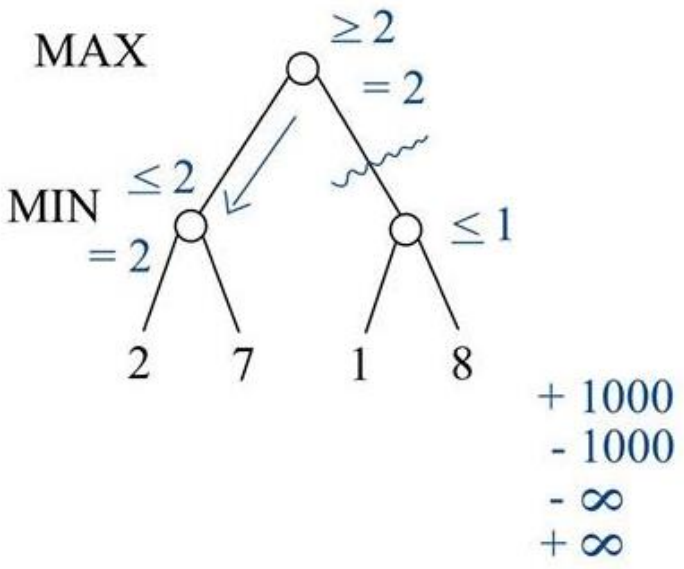
Alfa-beta ořezávání je vylepšením algoritmu minimax zrychlující jeho běh tím, že neprohledává podstromy herního stromu, které by neměly vliv na nalezení optimálního tahu. Princip použití tohoto algoritmu spočívá v použití nových proměnných. Proměnné alfa udávající nejlepší nalezenou hodnotu MAX úrovně, inicializovaná na hodnotu $-\infty$ a proměnné beta pro nejlepší hodnotu MIN úrovně, inicializovaná na hodnotu $+\infty$. Následně dojde k prohledávání tahů, dokud nedojde k situaci, že je alfa větší nebo rovna betě, což znamená, že už není možnost lepšího tahu (Bartel, 2014).

Nejvhodnější situace pro použití nastává, pokud jsou tahy ideálně uspořádané. V tomto případě je časová složitost $O(b^{d/2})$. Větvící faktor b odpovídá validním tahům a d je hloubka, do které se prohledává. Oproti algoritmu Minimax se tedy prohledá za stejný čas dvakrát větší hloubka, ale algoritmus nalezne stejné řešení. Avšak v jiných situacích může být časová složitost stejná jako u minimaxu. Proto je nejlepší se vždy co nejvíce přiblížit ideálnímu uspořádání, čehož se dosáhne zavedením heuristické funkce, která bude určovat pořadí, ve kterém se tahy prohledají (Winston, 2010).

4.3.1 Praktický příklad Alfa-Beta ořezávání

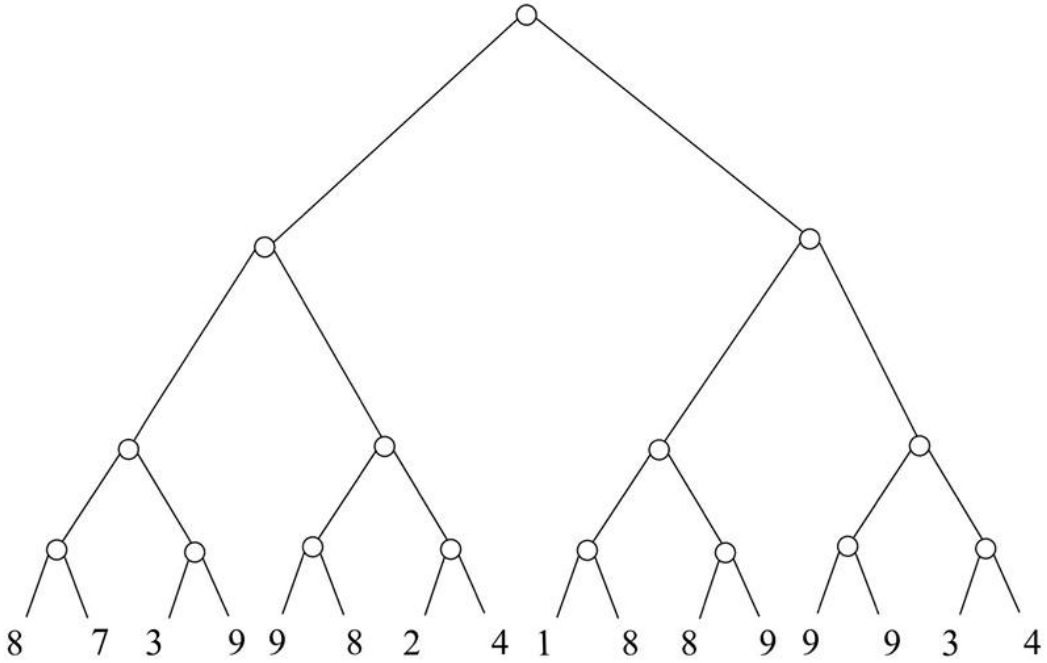
Lze to znovu vysvětlit na principu dvou hráčů, v tomto případě ale první hráč vybírá nejlepší tah alfa, který může teoreticky zvolit. Druhý hráč tedy už dopředu vidí pro něj nejlepší

možnost beta a může se tak celá nevyhovující větev možností vynechat a pracovat bez ní, jako kdyby neexistovala.



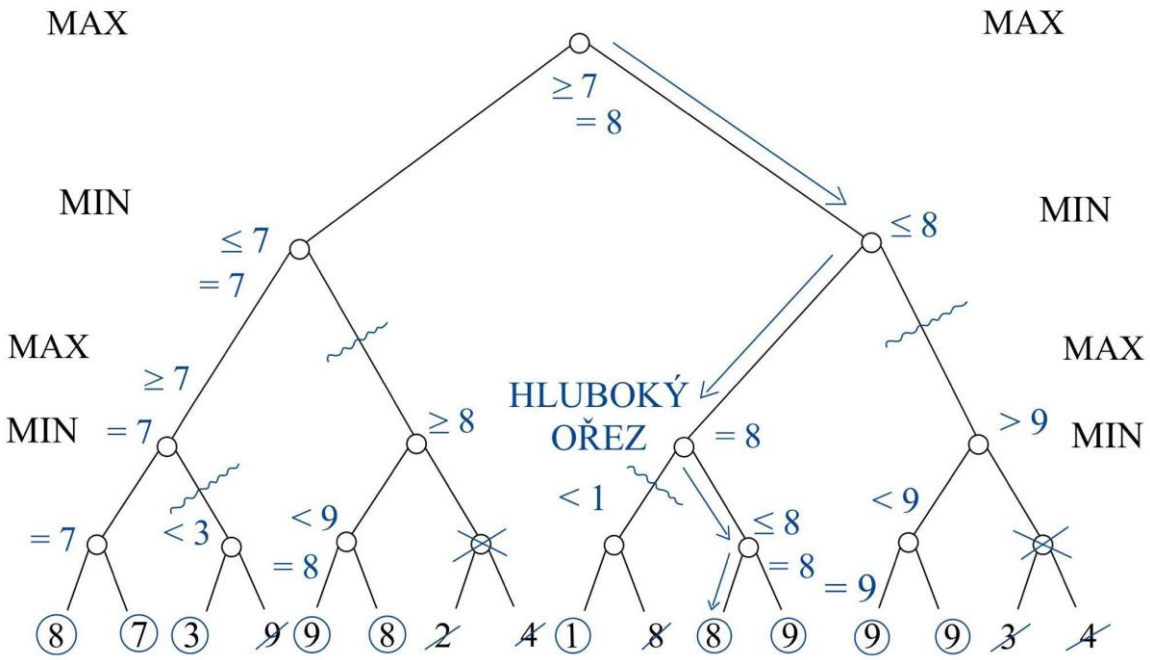
Obr. 10 - Tahový strom s příkladem využití alfa-beta ořezávání

Avšak nejlepší využití má Alfa-beta algoritmus v komplexnějších případech, kdy se díky jeho použití může předejít až několika výpočtům, a to vynecháním větví s konečnými hodnotami, u kterých se ani nemusí volba zhodnotit, protože se už dopředu ví, že tam směřovat nebude.



Obr. 11 - Příklad rozsáhlejšího tahového stromu

Dochází zde také k tzv. hlubokým ořezům, které se vyskytnou v případě lepšího teoretického výsledku z jedné strany a ušetří se tím proces od zbytečných výpočtů, jak lze vidět na obr. 12. Proces střídání hráčů se opakuje jako u lehčích případů až do doby, kdy není nalezen vhodný tah (Winston, 2010).



Obr. 12 - Tahový strom optimalizovaný pomocí alfa-beta ořezávání

4.4 ITERATIVNÍ PROHLUBOVÁNÍ

V iterativním prohlubování je principem procházení herního stromu do předem stanovené hloubky. Strom vždy postupně prochází od první do konečné hloubky, v každé iteraci minimaxu se uloží průběžný optimální tah a přepíše předchozí. Tento postup je zvolen z důvodu časového kritéria. U této metody je možnost nastavit časový limit pro hledání tahu. Když tento definovaný čas vyprší, algoritmus je schopný vrátit tah z nejhlubší iterace, kterou stihl do té doby vypočítat (Bartel, 2014).

Použitím heuristické funkce u této metody jsme schopni zavést průběžnou hodnotící funkci stavu hry pro každý uzel. Při současném použití transpoziční tabulky a správném seřazení prohledávaných uzlů je možné celý cyklus zrychlit (Winston, 2010).

4.5 POROVNÁNÍ ALGORITMŮ A JEJICH VHODNOST

Použití britského muzea v praxi je už z jeho podstaty nereálné, jelikož se jedná o tzv. algoritmus hrubé síly a bylo by třeba obrovského výpočetního výkonu pro jeho provedení v reálném čase. V neposlední řadě se tento algoritmus používá pouze na teoretické bázi.

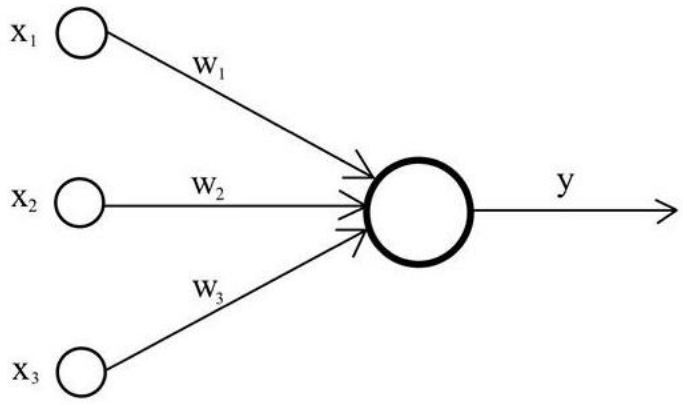
Jelikož jsou piškvorky jednoduchou hrou dvou hráčů, je minimax logickou volbou. Je však třeba ho optimalizovat pomocí alfa-beta ořezávání a iterativního prohlubování s pevně daným časovým limitem.

5 KONVOLUČNÍ NEURONOVÉ SÍTĚ

Konvoluční neuronové sítě jsou nástrojem pro práci s rozpoznáním obrazu nebo zvuku. Jsou navrženy pro rozpoznávání snímků přímo z jednotlivých pixelů s předzpracováním obrazu. Předzpracovaný obraz se předá ve formě vektoru do neuronové sítě na vstup. Neuronové sítě jsou schopny efektivně zpracovávat vstupy velkých rozměrů za použití mnohem menšího množství parametrů než obecná konvoluční síť. Díky tomu je snazší takovou síť učit. Cílem sítě je rozpoznávání objektů nezávisle na jejich deformaci, velikosti a změně polohy (Goodfellow, 2016; Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, 2020).

5.1 NEURONOVÉ SÍTĚ

Neuronové sítě jsou inspirovány principem a poznatky živých organismů a jejich schopnostmi. Základem přirozené i umělé neuronové sítě je neuron. Neurony jsou navzájem propojeny, předávají si signály a platí, že každý neuron může mít více vstupů, ale pouze jeden výstup a tento výstup může být poslán i více než jednomu dalšímu neuronu (QCExpert, 2020).



Obr. 13 - Model neuronu, x: vstupy, w: paměť neuronu, y: výstup

Matematicky lze vyjádřit neuronovou síť jako neuron, který poté, co přijme vstupy, vynásobí jejich hodnoty váhami, které má každý neuron odlišné na vstupu. Následně tyto hodnoty sečte, a pokud je výsledek větší než stanovený práh, výsledek se transformuje předem danou přenosovou funkcí a pošle na výstup dle vztahu

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i - \theta\right), \tag{1}$$

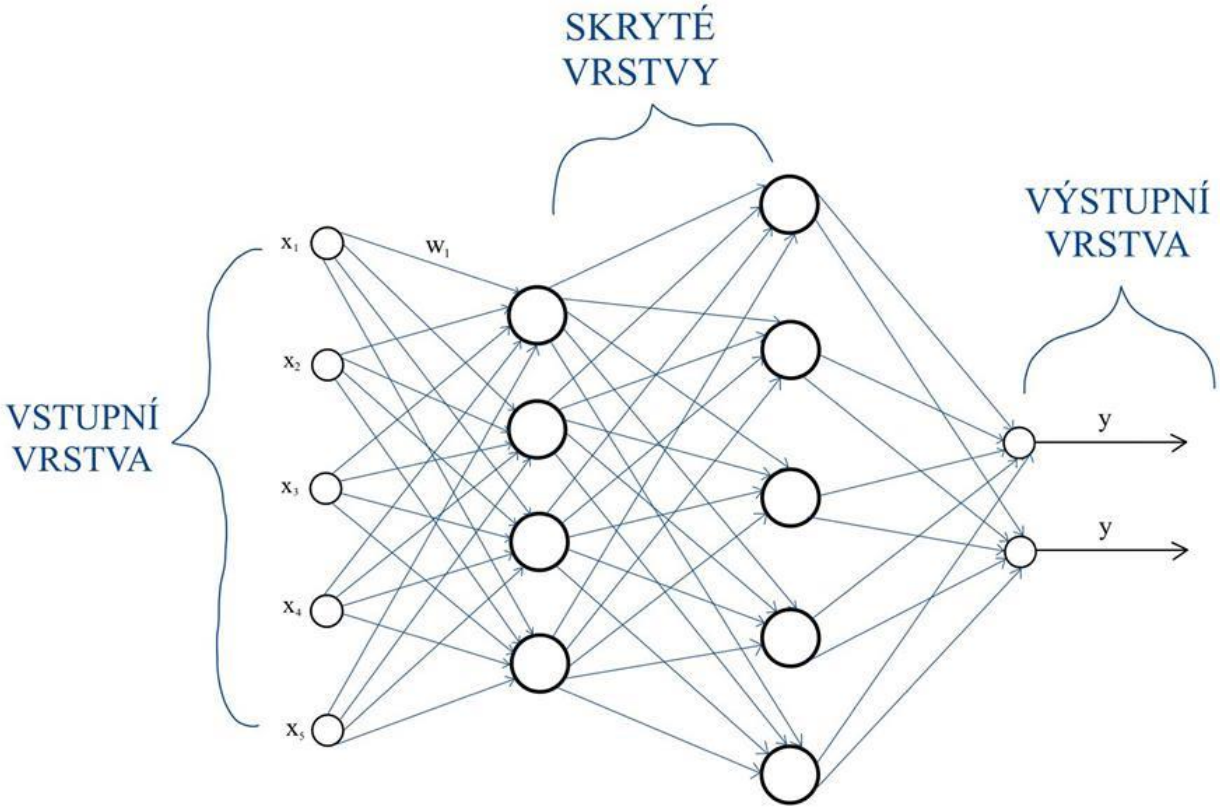
kde x_i *jsou vstupy,*
w_i *– váhy,*

θ – práh,
f – přenosová funkce,
y – výstup.

Vstupní vrstva zobrazí hodnoty vstupů a předá je všem neuronům v dalších vrstvách, těm se říká vrstvy skryté. Neurony výstupní vrstvy naopak mají pouze jeden výstup a jejich hodnoty pak kódují celou neuronovou síť. Neuronová síť má schopnost učit se a přizpůsobovat se vstupním datům (Durčák, 2017).

5.2 VRSTVY KONVOLUČNÍ NEURONOVÉ SÍTĚ

Konvoluční neuronová síť obsahuje vrstvy, z nichž každá má svou specifickou funkci.



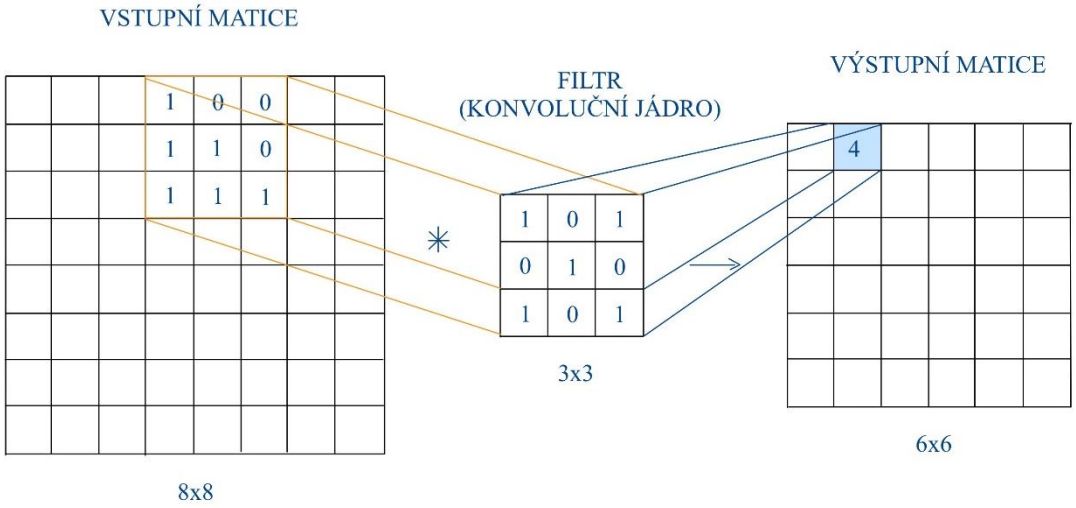
Obr. 14 - Příklad konvoluční neuronové sítě

5.2.1 Vstupní vrstva

První z vrstev je vrstva vstupní. Jejím vstupem je matice hodnot obrazových bodů ve formátu výška × šířka × hloubka. Tato vrstva obsahuje hodnoty pixelů z původního obrázku, které poté přivede do konvoluční vrstvy.

5.2.2 Konvoluční vrstva

Následuje konvoluční vrstva a každá taková operace má definované konvoluční jádro nebo též filtr, což je matice o relativně malých rozměrech, nejčastěji ve formě obrázku. Filtr postupně prochází celý obraz a postupně se přikládá na vstupní matici, násobí se jednotlivé pixely a tyto výsledky se sečtou. Výsledkem bude jedno číslo a jejich součet se zapíše do výstupní matice. Poté se posune jádro o daný krok a pokračuje (Karn, 2016).



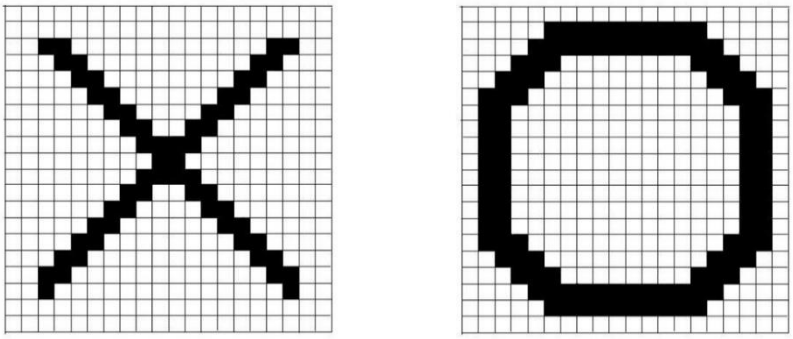
Obr. 15 - Konvoluční vrstva

Konvoluční vrstvy jsou tvořeny určitým počtem filtrů. Výstup z konvoluční vrstvy tvoří výstupy všech daných filtrů a každý z filtrů slouží k detekci určitého tvaru obrázku. Více použitých filtrů bude mít schopnost naučit se více rozdílných tvarů, nevýhodou je zvýšený počet parametrů.

Součástí konvoluční vrstvy může být i aktivační vrstva, ta se používá k určení výstupu neuronové sítě. Také ji lze připojit mezi dvě různé neuronové sítě (Moujahid, 2016).

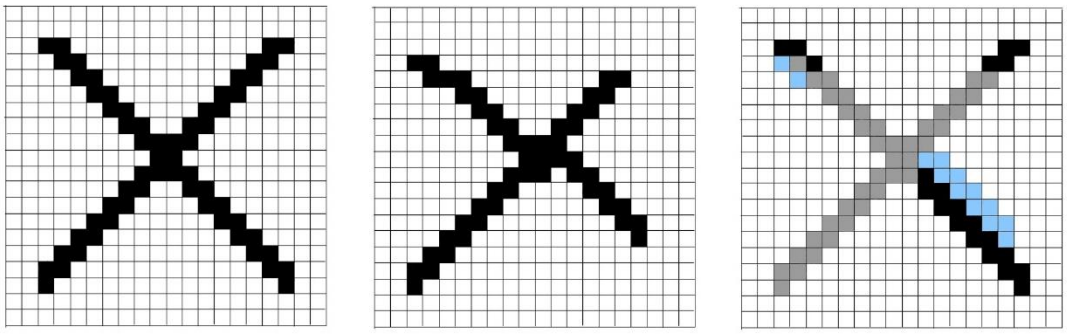
Příznaková mapa

Konvoluční vrstva se skládá z několika příznakových map. Používá se na rozpoznání znaku či symbolu ze snímku. Je ideální, aby oříznutá políčka měla stejnou velikost, proto je zde použita velikost 32 × 32.



Obr. 16 - Vstup pro síť písmene X a O

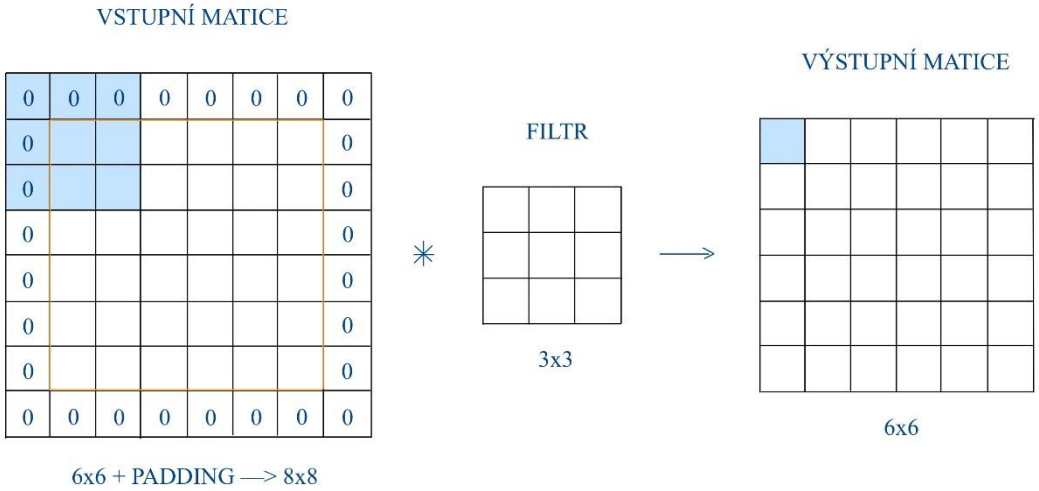
Principem je použít ideální symbol jako vzor, poté různě mírně deformovaný, ale přesto stejný symbol jako vstupní obrázek a zaznamenávat rozdíl, kde se pixely obou symbolů neshodují.



Obr. 17 - Vzor, vstupní obrázek a jejich rozdíl

5.2.3 Padding

Parametr padding neboli obtékání je umělé zvětšení vstupní matice tak, aby byly ve všech směrech přidány na okraje hodnoty 0. Díky tomu nejsou ztraceny informace o okrajových hodnotách a zároveň dojde k zachování původní velikosti vstupní matice. Pokud by nebyl tento parametr nastaven, z okrajových hodnot by se mohla tato informace ztrácet (Karn, 2016).

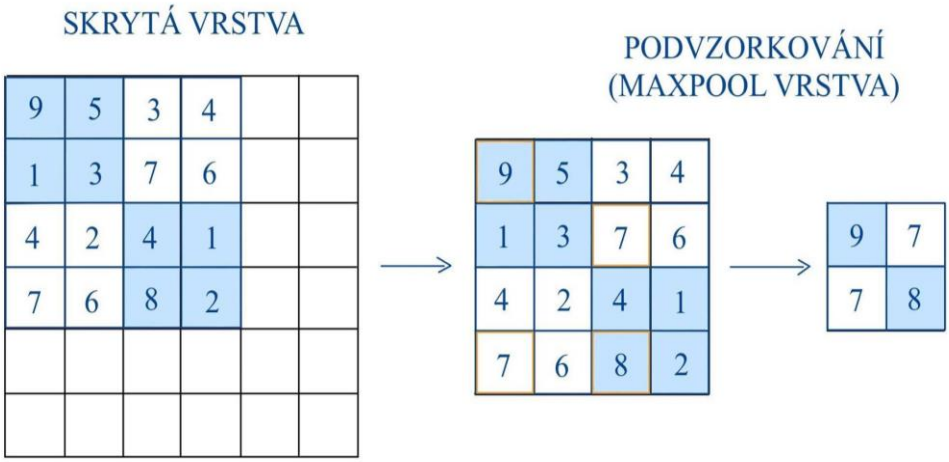


Obr. 18 - Konvoluční vrstva a padding

5.2.4 Pooling vrstva

Podvzorkování neboli pooling vrstva slouží k redukci velikosti pro snížení počtu parametrů sítě a výpočetní náročnosti, což pomáhá i proti přetrénování nebo přeučení sítě. Nejpoužívanější metodou podvzorkování je maxpool vrstva.

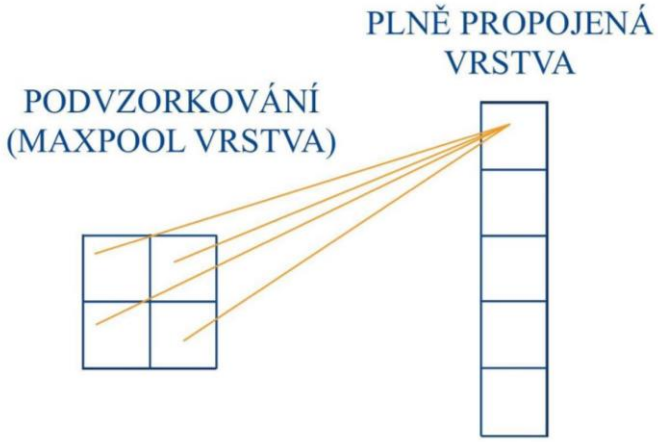
Maxpool vrstva slouží ke snížení velikosti vstupů, a tedy nepřímo i ke zlepšení distribuce informace. Dále snižuje výpočetní náročnost učení. Vrstva vybere maximální hodnotu dané podoblasti a zapíše ji na výstup, poté se posune dále (Moujahid, 2016).



Obr. 19 - Maxpool vrstva konvoluční sítě

5.2.5 Plně propojená vrstva

Další z vrstev je vrstva plně propojená, kdy je vytvořeno spojení mezi každým neuronem v jedné vrstvě a každým neuronem v jiné vrstvě. Pro každou vrstvu je potřeba definovat rozměr vstupu a výstupu a na vstupu je potřeba mít vektor (Karn, 2016).



Obr. 20 - Plně propojená vrstva

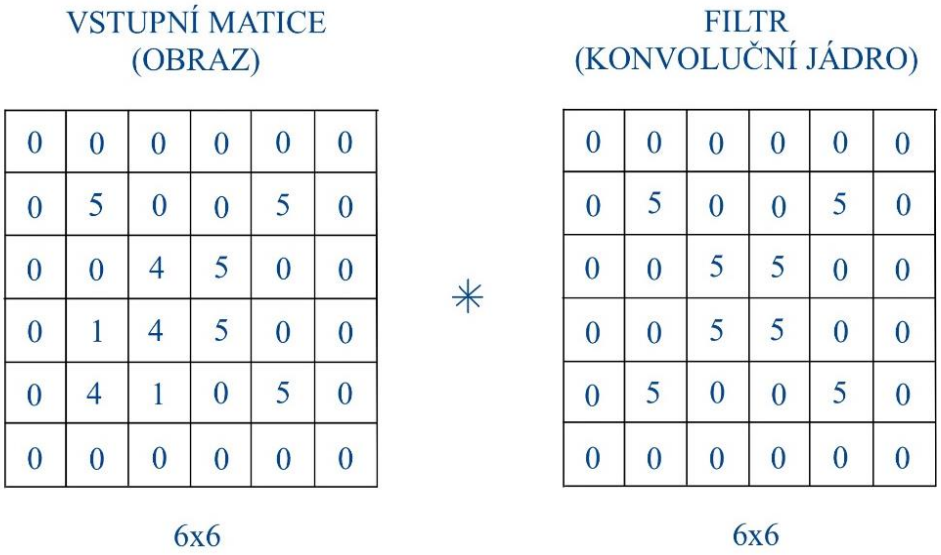
5.2.6 Výstupní vrstva

Poslední z vrstev je výstupní vrstva. Počet klasifikačních tříd a neuronů je zde stejný a vrstva je zcela propojena s vrstvou, která ji předchází.

5.3 FILTRY

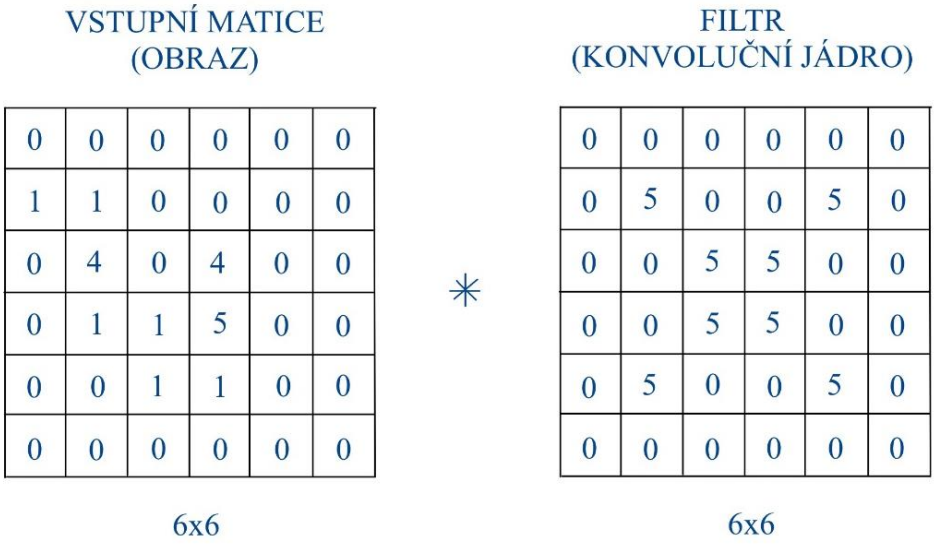
Nastavením filtrů je rozuměno nastavení velikosti matice a její počet. Nejvhodnější je volba čtvercového tvaru. Určení těchto hodnot u všech filtrů probíhá tak, že každý filtr má využití na více místech v daném obrázku, protože v obrázku se nalézají opakující se tvary. Pokud bude zvolena menší velikost filtru, v obrázku bude hledat v první konvoluční vrstvě jemnější tvary (Karn, 2016).

Na obr. 21 je vstupní matice a filtr velmi podobný tvarem, to je zřejmé z toho, že se hodnoty filtru značně shodují se vstupní maticí. Při vynásobení shodných hodnot se získá vysoké číslo váhy.



Obr. 21 - Vstupní matice a filtr s podobným tvarem

Naproti tomu na obr. 22 je nepodobný tvar filtru a vstupní matice. Při vynásobení vstupní matice s filtrem se získá nízká hodnota váhy.



Obr. 22 - Vstupní matice a filtr s nepodobným tvarem

6 UČENÍ NEURONOVÉ SÍTĚ

Odborně se dá toto popsat jako soubor výpočetních jednotek, které zpracovávají data, vzájemně mezi sebou komunikují a pracují paralelně. Neuronové sítě se používají mimo jiné i pro rozpoznávání a kompresi, neboli ve zmenšování poměru velikosti obrazů nebo zvuků. Cílem učení neuronové sítě je nastavit síť, aby vykazovala co nejpřesnější výsledky.

6.1 POSTUP UČENÍ

Učení neuronové sítě se může dělit na samotný postup učení, ve kterém je popsáno trénování sítě, určení její chyby, jak chybu minimalizovat pomocí zpětné propagace a následné optimalizace a jaké jevy mohou dále nastat.

6.1.1 Trénování sítě

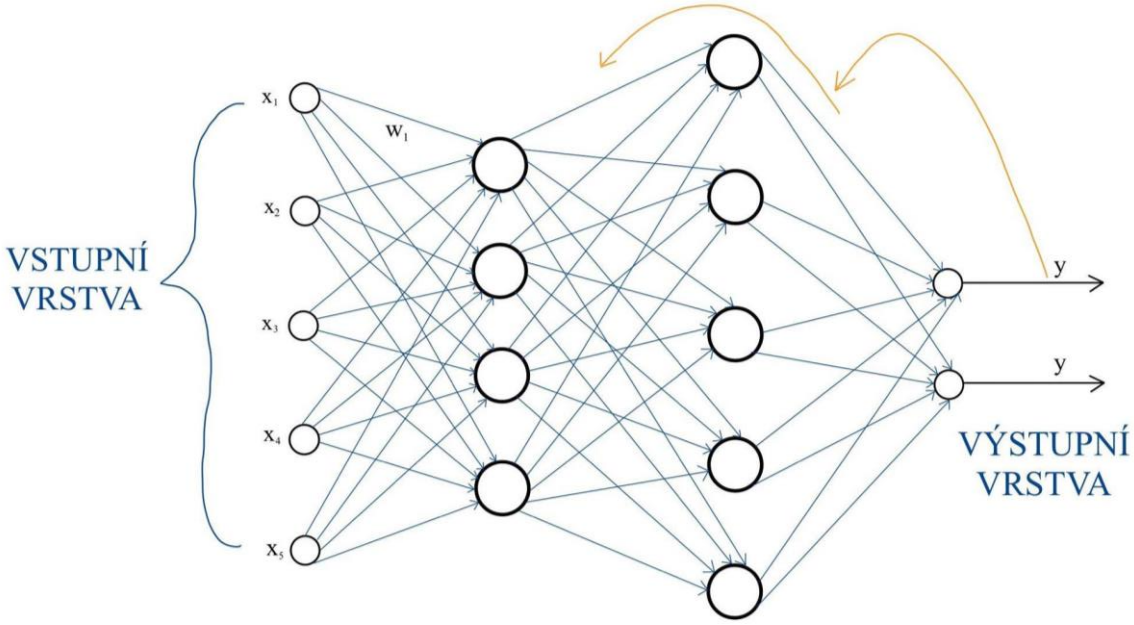
Trénování neuronových sítí probíhá experimentálně a cílem je optimalizace jejich vnitřních parametrů s cílem minimalizovat chybu predikce. V tomto procesu se nalézá množina trénovací a testovací. Testovací množina reprezentuje reálná data, trénovací slouží k samotnému učení sítě. V jednotlivých částech se počítá chyba na testovací i trénovací množině. Při výpočtu chyby na testovací množině je vynechána fáze zpětné propagace a optimalizace parametrů. V této závislosti při minimálním či nedostatečném počtu dat dochází k přeučení sítě z důvodu nedoučení (Feldman, 1996; Srivastava, 2014).

6.1.2 Chyba učení sítě

Chybu učení lze vypočítat jako rozdíl mezi požadovaným výstupem a současným výstupem sítě. Daná síť se učí generovat vstupní data, aby co nejlépe odpovídala požadovanému výstupu.

6.1.3 Metoda zpětné propagace

Tato metoda je založena na minimalizaci chyby. Ta je dána rozdílem mezi skutečným a požadovaným výstupem neuronové sítě neboli podílem jednotlivých vah na výsledné chybě (Vondrák, 1994).



Obr. 23 - Metoda zpětné propagace

6.1.4 Optimalizace parametrů

Pro snížení hodnoty chyby je potřeba řešit optimalizaci parametrů. K tomu je zapotřebí nastavení vah, aby v dalším kroku byla snížena celková chyba sítě a bylo postupně dosaženo globálního minima (QCExpert, 2020). Nejčastěji je využívána některá z gradientních metod (Bouvrie, 2006).

6.1.5 Přeučení sítě

Komplikací může být neúplná informace o daném průběhu učení. Informaci poskytují veličiny, které jsou trénovací a testovací nula. Ideální případ by byl, kdyby obě veličiny současně konvergovaly k nule. Reálně ve fázi učení nastává chvíle, kdy se klesající trend testovací chyby zastaví a chyba začne stoupat, zatímco trénovací chyba klesá. Toto se nazývá jako jev přeučení (QCExpert, 2020).

Pokud budou na vstupu ručně psané křížky a kolečka, bude výstupem vektor hodnot. Síť se naučí, které hodnoty odpovídají danému symbolu a v případě, že se poté objeví nový

obrázek se symbolem, síť je schopná rozeznat a dodat obrázku význam a zařadit ho do jedné ze skupin.

Výstup, který je požadovaný, může být přiřazen člověkem. V tom případě se jedná o učení se s učitelem. Pokud ale nejsou k dispozici požadované výsledky, jedná se o učení bez učitele.

6.1.6 Předčasné ukončení

Nejideálnějším způsobem, jak zamezit předčasnému ukončení, je zastavit učení, a to ve chvíli, kdy testovací chyba roste. Při zastavení se uloží parametry sítě, protože je možnost, že testovací chyba začne opět klesat a díky tomu se mohou po určitém počtu opakování určit záchytné body. Následně je třeba vybrat nejlepší výsledky (Mathworks, 2016).

6.1.7 Dropout

Dropout je parametr, který zabraňuje přeučení sítě. Podle náhodného výběru s určitou pravděpodobností v průběhu učení přerušuje spoje mezi neurony. Následkem toho průběžně mění váhy, které přispívají do ztrátové funkce a díky tomuto kroku se rovnoměrněji rozloží vliv vah, který ovlivňuje výsledek predikce (Srivastava, 2014).

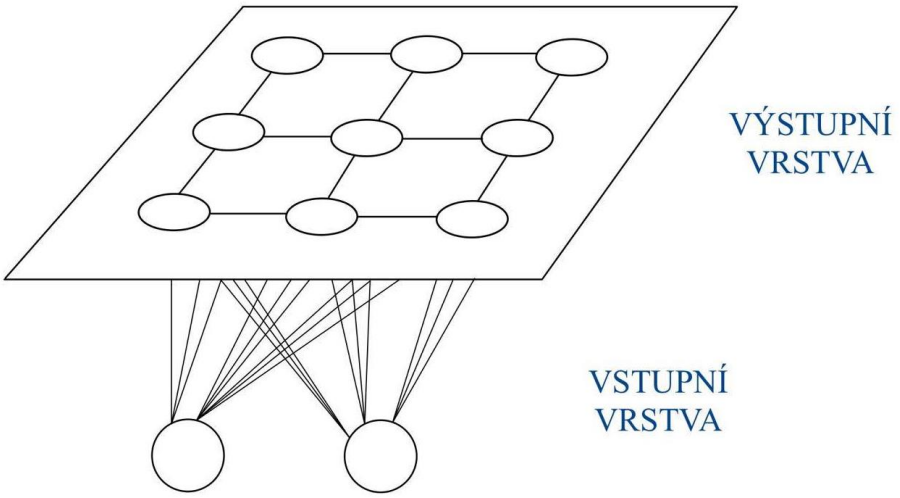
6.2 METODY UČENÍ

Metody učení jsou rozděleny do dvou skupin, na metody nekontrolovaného učení, což lze dále nazývat jako učení bez učitele a na kontrolované učení, také nazývané jako učení s učitelem. Tyto metody se potom dále dělí na různé metody a modifikace.

6.2.1 Metoda nekontrolovaného učení

Toto učení lze také nazývat učení bez učitele. Tato metoda pracuje na principu shlukové analýzy. Cílem je v dané množině elementů nalézt její podmnožiny neboli shluky objektů. Dále jsou tříděny do skupin, které mohou, ale i nemusí být známy předem, ve kterých mají objekty podobné vlastnosti. V této metodě a jejím průběhu učení není tedy žádná kontrola, zda dané učení postupuje správně. Celé toto učení vychází pouze z informací získaných ze vstupní množiny dat a je trvalého charakteru, jelikož se neví, zda jsou výstupní data správná. Do této skupiny patří tzv. Kohonenova síť, nebo též nazývána Kohonenovy samoorganizační mapy (Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, 2020).

Kohonenova síť je určena pro rozhodování, rozlišování dat a třídění různých objektů, signálů apod. Vychází z Kohonenova učení. Možné aplikace jsou například hledání a detekce osob podle fotografií, zpracování obrazu, videa, fotografií, ale i úprava zvuku, zpracování řeči, přepis psaného textu na tištěný aj (Vojáček, 2006).



Obr. 24 – Kohonenova síť

6.2.2 Metoda kontrolovaného učení

Toto učení existuje také pod pojmem učení s učitelem. Tato metoda strojového učení využívá sady trénovacích dat.

Tato metoda má množinu dat, která je rozdělena na dvě části, a to tréninkovou a testovací. Obvykle je tréninková množina značně větší a obsahuje zhruba tři čtvrtiny dat z celkové množiny dat.

Algoritmus prochází postupně jednotlivé prvky a zjišťuje odchylku od očekávaného výstupu a následně provádí korekci. K naučení určité sítě je potřeba i stovky až tisíce postupů, které se nazývají epocha, tedy proces, než se projde celá testovací množina. Zastavení učení je nejčastěji tehdy, kdy je dosaženo určité hodnoty celkové chyby. Tehdy je možné prohlásit síť za naučenou. Zastavení učení je také možné, pokud se celková chyba ustálí na hodnotě a dále již neklesá. Pomocí testovací množiny poté lze ověřit výkonnost naučené sítě. Nejčastějším kritériem správného naučení je střední kvadratická odchylka mezi obdrženými a očekávanými výstupy. Pokud je výkonnost nad testovací množinou adekvátní, dá se předpokládat, že bude přibližně obdobně dobrá i u vstupů mimo danou množinu (Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity, 2020).

7 BEZPEČNOST

7.1 BEZPEČNOST V ROBOTICE

Bezpečnost v robotice, kde je využívána automatizace, je v dnešní době na prvním místě.

7.2 IDENTIFIKACE NEBEZPEČÍ

Pokud při posouzení rizika dojde ke zjištění nebezpečí, musí dojít k procesu jeho omezení. Jestliže na stroji není žádné ochranné opatření, je předpokladem úraz jeho obsluhy. Mezi nebezpečí při práci s roboty se může například řadit mechanická nebezpečí, elektrická nebezpečí, tepelná nebezpečí a jejich kombinace. Musí se brát také v potaz zdroje rizik, ke kterým může dojít neúmyslným spuštěním stroje, chybou obsluhy, pohybem robota nebo nezabezpečeným prostředím. Aby nedošlo k úrazu, musí se dodržovat ochranná opatření a uživatel stroje musí být náležitě proškolen (ČSN EN ISO 12100, 2010).

7.3 BEZPEČNÉ PROSTŘEDÍ

Robot musí mít kolem sebe bezpečný pracovní prostor. Nesmí mít možnost narazit do obsluhy ramenem, proto se v pracovním prostředí vymezuje, kam robot dosáhne a kde je tedy potřeba dbát zvýšené opatrnosti. Je potřeba, aby robot nezasahoval do činnosti obsluhy. Důležité je i zapojení ochranného obvodu s tlačítkem STOP, který umožňuje okamžité zastavení robota v případě potřeby (ČSN EN ISO 12100, 2010).

7.4 BEZPEČNOST PŘI TAZÍCH ČLOVĚKA

V průběhu hry by se mělo dbát na dodržení základních opatření. Tah provádět až po úspěšném dokončení pohybu ramene robota a zapisovat s určitou přesností, aby byl správně zpracován. Důležité je také po stisku tlačítka pro tah robota nijak nezasahovat do pracovního prostoru a snažit se udržet kvalitní osvětlení hrací plochy pro nejlepší kvalitu pořízeného snímku (ČSN EN ISO 12100, 2010).

7.5 ŠKOLENÍ

Je prováděno před začátkem práce s robotickým zařízením a je obnovováno každé dva roky. Školení probíhá v souladu všeho uvedeného v zákoně o zajištění dalších podmínek bezpečnosti a ochrany zdraví při práci. Konkrétně se jedná o základní předpisy o bezpečnosti,

povinnosti při vzniku pracovního úrazu, zásady poskytnutí první pomoci a dalšími, které vyplývají z charakteru pracoviště a činností v něm prováděných (pyroservis.cz, 2020).

8 GRAFICKÉ ROZHRANÍ HRY

8.1 VLASTNÍ PRAVIDLA HRY

Tato hra je určena pro dva hráče, a to robota a uživatele. Ti se střídají v tazích a umisťují na hrací plochu pomocí fixu psané symboly X nebo O. Hra probíhá na poli o velikosti 36 políček. Vždy začíná hráč, který si na začátku hry zvolil symbol X. Cílem a vítězstvím této hry je umístit vodorovně, svisle, nebo diagonálně řadu o 4 stejných symbolech.

8.2 POPIS JEDNOTLIVÝCH TLAČÍTEK

Obr. 25 - Grafické uživatelské rozhraní

Symbol X a O

Před samotným začátkem hry je třeba zvolit symbol, kterým bude uživatel po celou dobu hrát a druhý symbol po vybrání získá robot. Je třeba dbát na herní pravidla a symbol X tedy vždy začíná hru.

Start

Pro umožnění začátku hry se musí stisknout tlačítko Start. Po předchozím vybrání symbolu nebo při automatickém výběru tlačítka X a O zešednou, aby nebyla možnost změny. Dále se rozsvítí zeleně symbol X, který hru začíná.

Po tomto úkonu začne robot vykreslovat předem definované pole 6x6 v rozmezí 4x4 cm. Nejprve se vykreslují linie vertikální a poté horizontální. Po dokončení tohoto procesu tlačítko zešedne a nelze na něj dále klikat, aby nebyla možnost několikrát po sobě zakreslovat pole.

Následně poté, pokud má robot přidělený symbol X, provede první tah hry.

Předat tah

Po dokončení tahu hráče je třeba stisknout tlačítko Předat tah. To umožňuje povolit robotovi, aby mohl provést svůj tah. Jeho tah se skládá z rozsvícení jemu přiděleného symbolu, aby bylo zřejmé, že je na tahu a uživatel se v tomto prostoru nesmí pohybovat. Dále podle vyhodnocení hracího pole pomocí algoritmu provede tah a přesune se do jeho výchozí pozice.

Reset

Tímto tlačítkem se ukončí hra v jejím průběhu. Po stisknutí se hra dostane na samotný začátek, znovu nabídne výběr možnosti symbolu X a O a probíhá dále úplně stejným způsobem jako dříve.

Příkladem případu, kdy je zapotřebí reset, může být nechtěné pohnutí s vykresleným hracím polem a tím znemožnění přesné hry robota.

8.3 BEZPEČNOST POHYBU A OCHRANA UŽIVATELE

8.3.1 Ošetření stisknutím tlačítka

Pro větší bezpečnost uživatele i robota byla zavedena určitá opatření. Jedním z nich je ošetření při stisknutí tlačítka Předat tah. Toto tlačítko bylo vytvořeno pro umožnění celého tahu robota. Tedy před stisknutím tohoto tlačítka uživatel kontroluje, zda není v hracím poli zanechaný fix či nějaký jiný předmět, který by znemožnil tah robota. Po zmáčknutí je dán prostor robotovi k jeho vyhodnocení a následnému tahu. Tím se získá ošetření proti jeho nechtěnému pohybu nebo podobným událostem.

8.3.2 Vizuální ošetření

Vizuální ošetření je primárně zavedeno proto, aby bylo na první pohled zřejmé, zda je na tahu symbol X nebo O a tím ověření, zda probíhá hra nadále v pořádku a v jaký moment se robot dostane do výchozí pozice a umožní tím provést tah uživatele. Toto ošetření je viditelné zezelenáním symbolu, který je aktuálně na tahu.

8.4 INFORMATIVNÍ STAVOVÉ ZPRÁVY

Informativní stavové zprávy jsou doplňkem celé hry, aby byl uživatel informován o průběhu hry a jednotlivých krocích systému i robota.

Obr. 26 - Příklad stavové zprávy při kreslení hracího pole

9 ZPRACOVÁNÍ OBRAZU HRACÍ PLOCHY

9.1 KAMERA

Kamera pro pořízení snímku hrací plochy je umístěná přímo nad hracím polem. Pro nejefektivnější využití je použito vyšší rozlišení, aby se ze snímku získaly podstatné detaily. Dále je třeba zajistit, aby nebyl obraz zašuměný z nedostatku světla nebo nečitelný z důvodu přímého světla a tím neznehodnotil potřebné vyhodnocení. Pro optimální zpracování obrazu je třeba získat snímky za stejných podmínek. Pokud by tomu tak nebylo, může potom trénovací množina získat nepřesné nebo špatné výsledky. Pro práci s webkamerou v Matlabu je zapotřebí mít nainstalovaný balíček pro práci s USB webkamerami.

9.1.1 Pořízení snímku

Pořízení snímku je provedeno tehdy, kdy je stisknuto tlačítko Předat tah. Snímek je uchován v programu jako proměnná, dokud není přepsán při pořízení dalšího snímku.

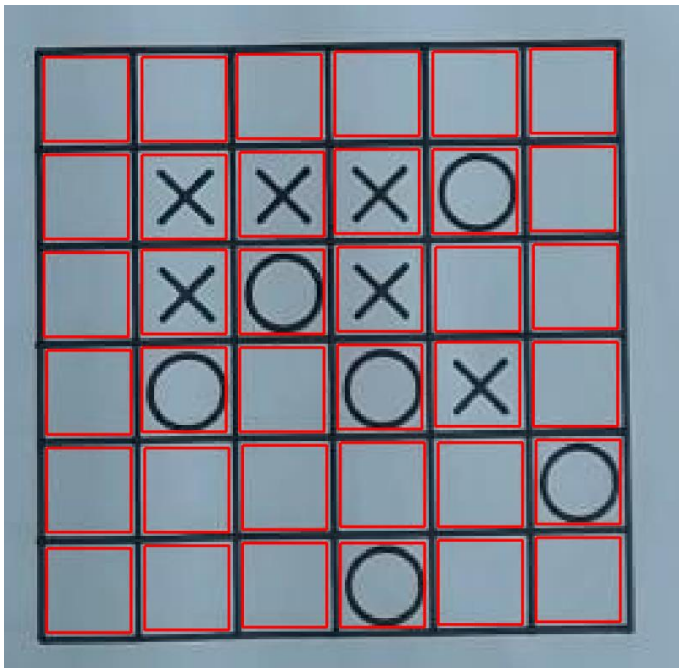
Pokud se jedná o první tah a hru začíná robot, pořízení snímku se vynechá. K pořízení snímku dochází před tahem robota a nemá tedy smysl kontrolovat prázdné pole. Pořídí se nový snímek a předá se funkci checkField, která provede ořez a kontrolu jednotlivých políček hracího pole. Navrací se nový stav hracího pole a počet zaznamenaných nových tahů. Jestliže přibylo nových tahů více než 1, hráč porušil pravidla a hra je restartována.

```
r = handles.Robot;
if (~firstTurn)
    cam = webcam('Logitech Webcam C930e');
    cam.Resolution = '1280x720';
    camSnap = snapshot(cam);
    [handles.Game.Grid, added] = checkField(handles.Game.Grid,r.net,camSnap);

    if (added > 1)
        handles = init(handles);
        set(handles.statusText,'String','Porusil jsi pravidla hry! Hra byla restartovana.');
        h = handles;
        return;
    end
end
```

9.1.2 Ořez snímku

Každý snímek je ořezán po jednotlivých políčkách hracího pole, které má pěvně dané souřadnice.



Obr. 27 - Znázornění ořezu jednotlivých políček hracího pole

Funkce postupně prochází políčka v hracím poli, dle předem definovaných souřadnic jednotlivá políčka ořízne a pomocí konvoluční neuronové sítě zjistí, jaký symbol se na políčku nachází, případně zda je prázdné. Políčka, ve kterých se již nějaký symbol nachází, se přeskočí. Pro práci s obrázky je zapotřebí mít v Matlabu nainstalovaný Image Processing Toolbox.

```
function [grid,added] = checkField(grid,net,img)

    cutPoints = [
        ...
    ];

    added = 0;

    for i=1:36
        x = cutPoints(i,1);
        y = cutPoints(i,2);

        gY = floor((i-1)/6) + 1;
        gX = i - (gY - 1) * 6;

        if (grid(gY,gX) ~= 0)
            continue;
        end
```

```
        field = imcrop(img,[floor(x) floor(y) 31 31]);
        symbol = classify(net,field);

        if (symbol == 'x')
            grid(gY,gX) = 'X';
            added = added + 1;
        elseif (symbol == 'o')
            grid(gY,gX) = 'O';
            added = added + 1;
        end
    end
end
```

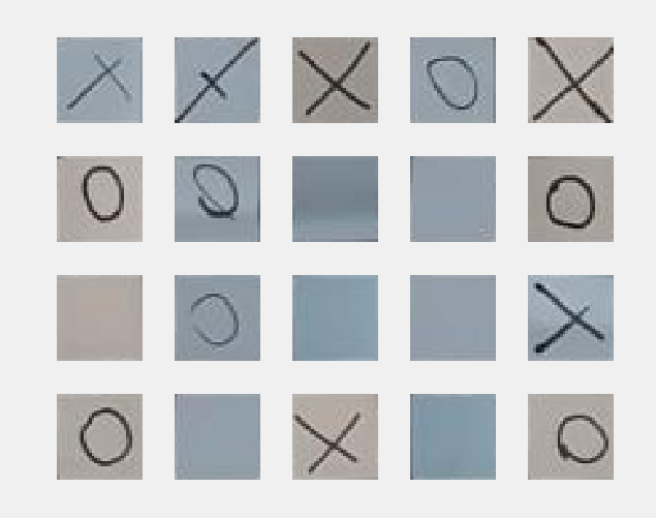
9.2 VYHODNOCENÍ OBRAZU POMOCÍ KONVOLUČNÍ NEURONOVÉ SÍTĚ

Pro vyhodnocení obrazu je použita konvoluční síť zpracovávající každé políčko samostatně. K tomu je třeba vytvořit trénovací množinu. Ta je vytvořena tak, že jsou z pořízeného snímku vyříznuta jednotlivá políčka. Ořezaná políčka mají stejnou velikost. Ořezaná políčka jsou pak uložena jako kolečko, křížek nebo prázdné pole. Tímto postupem vzniknou údaje, kdy ořezaný snímek je vstupem a příslušný křížek, kolečko nebo prázdné pole výstupem. Konvoluční neuronová síť byla v Matlabu vytvořena a trénována pomocí oficiálního balíčku Deep Learning Toolbox.

9.2.1 Učení konvoluční neuronové sítě

Učení konvoluční neuronové sítě se rozděluje na více částí. Samotným cílem učení je nastavit síť, aby vykazovala co nejpřesnější výsledky. Nejrozsáhlejší část učení je trénování sítě, která obsahuje množinu trénovací a testovací.

Použitá trénovací množina se skládala ze 125 snímků prázdných polí, křížků a koleček. Na obr. 28 je zobrazených několik náhodně zvolených snímků použitých k učení.



Obr. 28 - Část trénovací množiny konvoluční neuronové sítě

Z celkového počtu 125 snímků se náhodně zvolí 100 k učení a zbylých 25 k ověření přesnosti rozpoznávání. Použitá konvoluční síť má konvoluční vrstvy s filtry o velikosti 3 × 3 s využitím padding pro zachování velikosti. Učení proběhlo v 10 epochách s tempem učení 0,01. Výsledná přesnost byla 98,67 %.

```
imds = imageDatastore('dataset', ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

numTrainFiles = 100;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

layers = [
    imageInputLayer([32 32 3])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
```

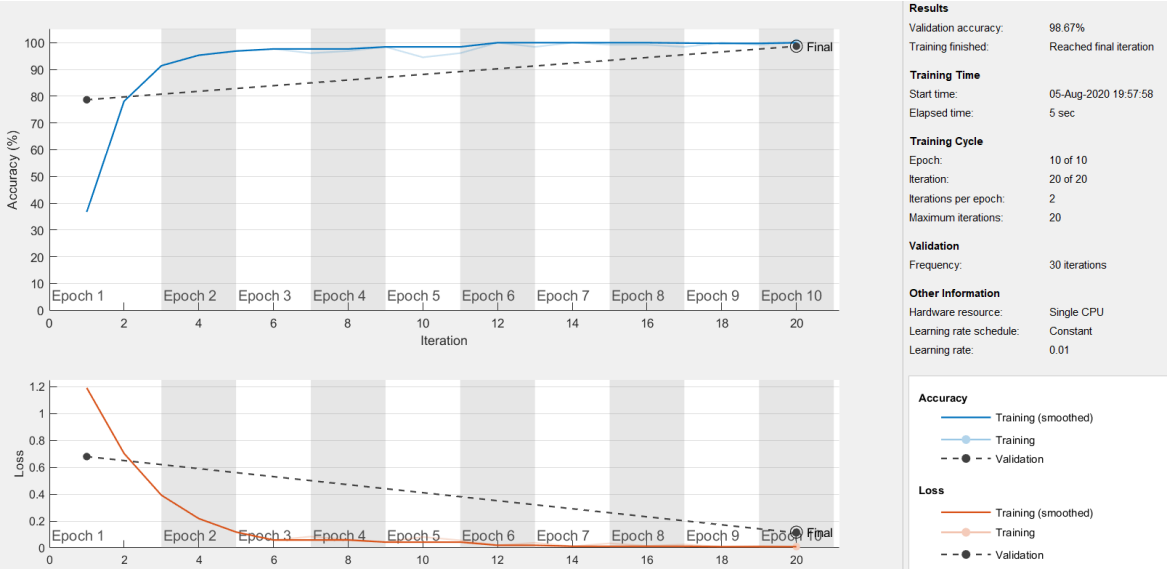
```
    fullyConnectedLayer(3)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',10, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);
```

9.2.2 Znázornění v grafu

Na obr. 29 je vynesený graf průběhu učení konvoluční neuronové sítě. V grafu je viditelné, že v tomto případě došlo k vrcholu přesnosti už v šesté epoše a následně kolem deváté epochy došlo k nepatrnému poklesu přesnosti. To mohlo být způsobeno přeučením sítě. Výsledná přesnost je dostačující. Při testovacích hrách robot při rozpoznávání pole již neudělal jedinou chybu.



Obr. 29 - Průběh učení konvoluční neuronové sítě

10 ZPRACOVÁNÍ DAT

10.1 KOMUNIKACE S ROBOTEM

Komunikace s robotem je prováděna pomocí prostředí PolyScope a programu Matlab.

Před začátkem programování je potřeba si uvědomit rozsah a osy robota. Dále je třeba určit, kde se aktuálně robot nachází a jaké má možnosti pohybu vůči danému prostoru.

Aby byl uživatel schopný softwarové komunikace s robotickým ramenem, je třeba vytvořit základní program s navázáním komunikace.

10.1.1 Navázání komunikace

Mezi programem Matlab a Polyscope probíhá softwarová komunikace a její navázání se provádí pomocí komunikačního protokolu TCP/IP. Je nutné definovat připojení k serveru s IP adresou 192.168.1.10. Pro připojení k robotovi z prostředí Matlab je vyžadován balíček Instrument Control Toolbox.

```
byteSize = 1116;
r.s = tcpip('192.168.1.20', 30003, 'NetworkRole', 'client', 'InputBufferSize', byteSize*5,
'BytesAvailableFcnCount', byteSize, 'BytesAvailableFcnMode', 'byte', 'BytesAvailableFcn',
@cti);
```

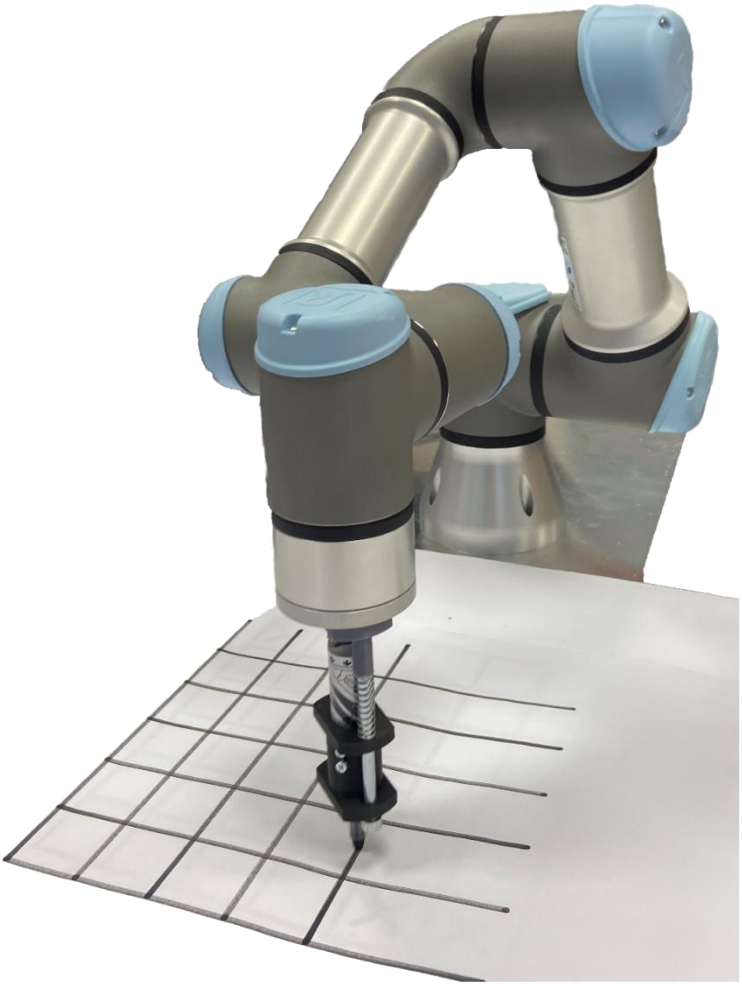
10.1.2 Výchozí pozice robota

Výchozí pozice robota je místo, kde se robot nachází ve chvíli, kdy není na tahu. V této pozici musí být robot připravený co nejméně komplikovanými pohyby zahrát tah. Zároveň musí být tato pozice v místě, kde nebude robot zasahovat do hracího pole a tím ani kameře, která snímá hrací plochu.

```
pause(0.5);
fprintf(r.s, 'movel(p[-0.15, -0.19, 0.035, -3.1415, 0, 0], a=1.4, v=0.15, t=0, r=0)');
```

10.2 ZAKRESLENÍ HRACÍHO POLE

Při stisknutí tlačítka Start se spustí zakreslení hracího pole. Robot vykresluje předem definované pole 6x6 v rozmezí 4x4cm.



Obr. 30 - Robot vykreslující hrací pole

Pro vykreslování je použita pomocná funkce add, která provádí lineární pohyb relativní vůči současné pozici robota.

```
% values in mm [x, y, z]
function add(values, v, s, t)
    x = s.UserData.zprava(56) + values(1)/1000;
    y = s.UserData.zprava(57) + values(2)/1000;
    z = s.UserData.zprava(58) + values(3)/1000;
    rx = s.UserData.zprava(59);
    ry = s.UserData.zprava(60);
    rz = s.UserData.zprava(61);
    a=2.5;
    r=0;
    text = sprintf('movel(p[%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f], a=%1.4f, v=%1.4f,
t=%1.4f, r=%1.4f)',x,y,z,rx,ry,rz,a,v,t,r);
    fprintf(s, text);
    pause(t+0.25)
end
```

Nejprve se robot posune do počáteční pozice, odkud postupně kreslí jednotlivé horizontální a vertikální části hracího pole. Kreslení hracího pole je počítané relativně vůči počáteční pozici.

```
% Init position
text = sprintf('movel(p[%1.4f, %1.4f, %1.4f, -3.1415, 0, 0], a=1.4, v=0.15, t=0, r=0)', xBase-(boxLength*boxNumbers)/2000, yBase, zBase);
fprintf(s, text);

pause(3);
k = 6;
% Horizontal lines
for i=1:boxNumbers/2
    add([0, -boxLength, 0], v, s, t/k);
    add([0, 0, -h], v, s, t/k);
    add([boxLength*boxNumbers, 0, 0], v, s, t);
    add([0, 0, h], v, s, t/k);
    add([0, -boxLength, 0], v, s, t/k);
    add([0, 0, -h], v, s, t/k);
    add([-boxLength*boxNumbers, 0, 0], v, s, t);
    add([0, 0, h], v, s, t/k);
    disp(i)
end
% Vertical lines
for i=1:boxNumbers/2
    add([0, 0, -h], v, s, t/k);
    add([0, boxLength*boxNumbers, 0], v, s, t);
    add([0, 0, h], v, s, t/k);
    add([boxLength, 0, 0], v, s, t/k);
    add([0, 0, -h], v, s, t/k);
    add([0, -boxLength*boxNumbers, 0], v, s, t);
    add([0, 0, h], v, s, t/k);
    add([boxLength, 0, 0], v, s, t/k);
    disp(i)
end
% Finishing
add([0, 0, -h], v, s, t/k);
add([0, boxLength*boxNumbers, 0], v, s, t);
add([-boxLength*boxNumbers, 0, 0], v, s, t);
add([0, 0, h], v, s, t/k);
```

10.3 POPIS ALGORITMU

K vyhodnocení optimálního tahu robota je tedy použit minimax. Konkrétně je maximalizujícím hráčem robot a minimalizujícím hráčem člověk (uživatel/hráč). Dostupným tahem, který se vyhodnocuje, jsou všechna prázdná políčka hracího pole.

Je definovaná funkce bestMove, minimax a hodnotící funkce bestScore. Funkce bestMove postupně projde všechna prázdná políčka hry s cílem najít tah, se kterým by se

dosáhlo nejvyššího skóre. Do hracího pole se na dané políčko zapíše symbol robota a upravené hrací pole se předá funkci minimax, která navrací skóre, jakého se takovým tahem dosáhne.

```
best = -Inf;
if (hrac=='X'); aip='O'; else; aip='X'; end
for i=1:radky
    for j=1:sloupce
        if (board(j,i) == 0)
            board(j,i) = aip;
            tic;
            skore = minimax(board,0,-Inf,Inf,false,hrac,aip,remainingFields);
            board(j,i) = 0;
            if (skore > best)
                best = skore;
                x = j;
                y = i;
            end
        end
    end
end
```

Ve funkci minimax se nejprve zavolá hodnotící funkce checkScore. Jestliže došlo ke konci hry, dojde k návratu hodnoty odpovídající výsledku hry.

```
function best=minimax(board,depth,alpha,beta,maximizingPlayer,hrac,aip,remainingFields)
    vysledek = checkScore(board);
    if (vysledek ~= -1)
        switch(vysledek)
            case aip
                best = 10;
                return;
            case hrac
                best = -10;
                return;
            case 0
                best = 0;
                return;
        end
    end

…
```

V opačném případě se pro změnu ze zbylých volných hracích polí hledá optimální tah pro člověka, tedy pro minimalizujícího hráče. Minimax se tedy dále rekurzivně volá a v závislosti na vstupním parametru funkce se střídá výběr optimálního tahu robota, tedy maximalizujícího hráče, a člověka, tedy minimalizujícího hráče. K rekurzivním voláním dochází až do momentu, kdy funkce bestMove na počátku minimaxu vyhodnotí konec hry.

```
if (maximizingPlayer)
```

```
    best = -Inf;
    for i=1:radky
        for j=1:sloupce
            if (board(j,i) == 0)
                board(j,i) = aip;
                skore = minimax(board,depth+1,alpha,beta,false,hrac,aip,remainingFields);
                board(j,i) = 0;
                best = max(skore, best);
                alpha = max(alpha,skore);
                if (beta <= alpha)
                    return;
                end
            end
        end
    end
else
    best = Inf;
    for i=1:radky
        for j=1:sloupce
            if (board(j,i) == 0)
                board(j,i) = hrac;
                skore = minimax(board,depth+1,alpha,beta,true,hrac,aip,remainingFields);
                board(j,i) = 0;
                best = min(skore, best);
                beta = min(beta, skore);
                if (beta<=alpha)
                    return;
                end
            end
        end
    end
end
```

Jednoduše řečeno se hledá optimální tah robota za předpokladu, že i člověk bude hrát optimálně.

10.3.1 Způsoby optimalizace

V případě, kdy úvodní tah provádí robot a pole je prázdné, je zbytečné kontrolovat celé pole pomocí algoritmu. Vybere se tedy náhodně jedno z polí s výjimkou okrajových.

```
if (max(max(board))==0)
    x = round(rand()*3) + 2;
    y = round(rand()*3) + 2;
    best = 0;
    return;
end
```

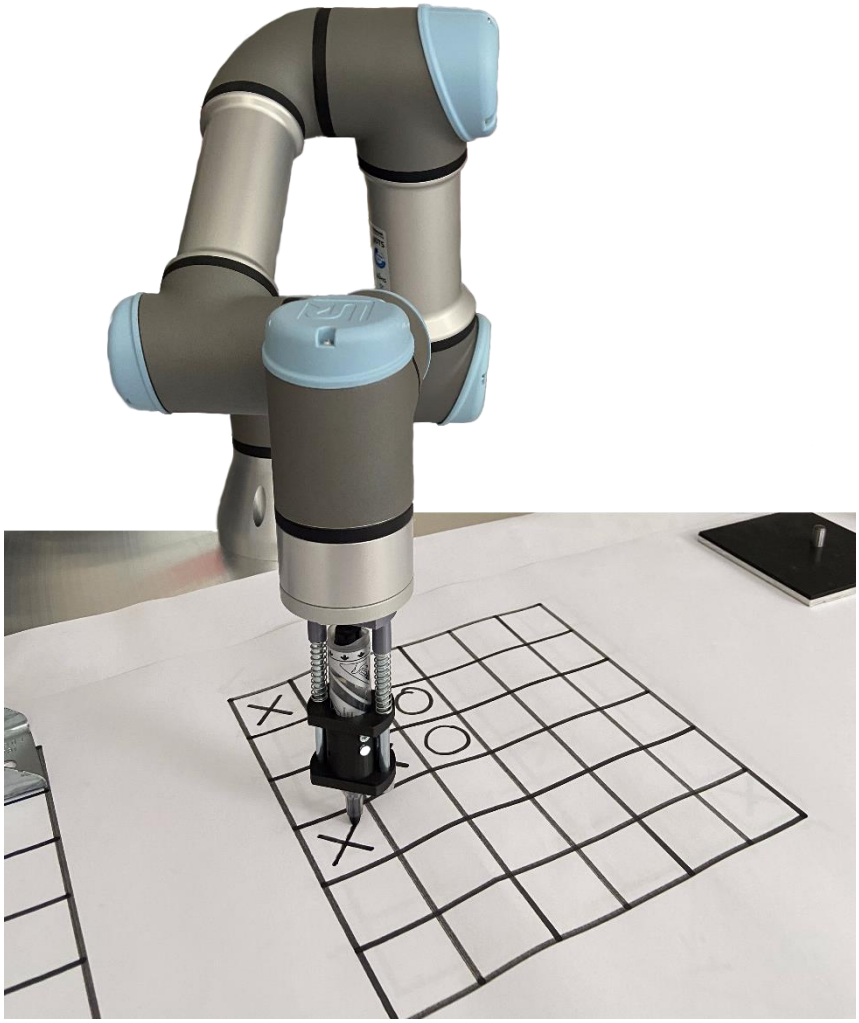
Vzhledem k narůstající časové složitosti minimaxu s velikostí hrací plochy je třeba algoritmus optimalizovat. Optimalizace je provedena zavedením proměnných alfa a beta, tedy tzv. alfa-beta ořezáváním. V případě pole 6 × 6 je však i po zavedení alfa-beta ořezávání časová složitost stále příliš vysoká, bylo tedy zavedeno časové kritérium, po jehož dosažení se výpočet předčasně ukončí. Časový limit je stanovený 3 sekundy na celé vyhodnocení, limit na jedno políčko v sekundách je tedy 3/(počet zbývajících políček).

Jestliže algoritmus nestihne zkontrolovat hrací pole do hloubky 5, je časové kritérium ignorováno.

```
diff = toc;
if (diff >= 3/(remainingFields) && depth == 5)
    best = 0;
    return;
end
```

10.4 ZAKRESLENÍ TAHŮ

Robot zakresluje do hracího pole symbol X nebo O. Výběr symbolu závisí na uživateli, který si na začátku hry volí svůj symbol, druhý je přidělen robotovi. Robot kreslí symbol pomocí fixu připevněného k ramenu pomocí speciálního držáku. Po zakreslení symbolu se rameno přesune na výchozí pozici.



Obr. 31 - Robot zakreslující tah v průběhu hry

10.4.1 Zakreslení tahu X

Robot nejprve pomocí lineárního pohybu posune rameno na roh políčka, do kterého bude kreslit symbol X. Následně postupně nakreslí symbol X lineárními pohyby relativními vůči předchozí pozici ramene.

```
text = sprintf('movel(p[%1.4f, %1.4f, %1.4f, -3.1415, 0, 0], a=1.4, v=0.15, t=0, r=0)',
xBase-(boxLength*(boxNumbers))/2000 + column*boxLength/1000 + 30/1000,
yBase-boxLength*row/1000-30/1000, zBase);
fprintf(s, text);
```

```
pause(1);
t = 0.5;
add([0, 0, -h], v, s, t);
add([-20, 20, 0], v, s, t);
add([0, 0, h], v, s, t);
add([0, -20, 0], v, s, t);
add([0, 0, -h], v, s, t);
add([20, 20, 0], v, s, t);
add([0, 0, h], v, s, t);
```

10.4.2 Zakreslení tahu O

Před zakreslením symbolu O se robot nejprve lineárním pohybem posune do středu levé hrany políčka, odkud kloubovým pohybem zakreslí nejprve první polovinu kolečka a následně i druhou polovinu.

```
text = sprintf('movel(p[%1.4f, %1.4f, %1.4f, -3.1415, 0, 0], a=1.4, v=0.15, t=0, r=0)',
xBase-(boxLength*(boxNumbers))/2000 + column*boxLength/1000 + boxLength/2000,
yBase-boxLength*row/1000-boxLength/2000-d/2000, zBase);
fprintf(s, text);

pause(1);
t = 0.3;

add([0, 0, -h], v, s, t);
addC([d/2, d/2, 0],[-d/2, d/2, 0], v, s);
addC([-d/2, -d/2, 0],[d/2, -d/2, 0], v, s);
add([0, 0, h], v, s, t);
```

Pomocná funkce addC kloubovým pohybem kreslí polovinu kružnice.

```
function addC(valuesVia, valuesTo, v, s)
    x = s.UserData.zprava(56) + valuesVia(1)/1000;
    y = s.UserData.zprava(57) + valuesVia(2)/1000;
    z = s.UserData.zprava(58) + valuesVia(3)/1000;
    xT = x + valuesTo(1)/1000;
    yT = y + valuesTo(2)/1000;
    zT = z + valuesTo(3)/1000;
    rx = s.UserData.zprava(59);
    ry = s.UserData.zprava(60);
    rz = s.UserData.zprava(61);
    a=2.5;
    r=0;
    text = sprintf('movec(p[%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f], p[%1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f], a=%1.4f, v=%1.4f, r=%1.4f, mode=%d)',
x,y,z,rx,ry,rz,xT,yT,zT,rx,ry,rz,a,v,r,0);
    fprintf(s, text);
    pause(1/v/20);
end
```

10.5 VYHODNOCENÍ STAVU HRY

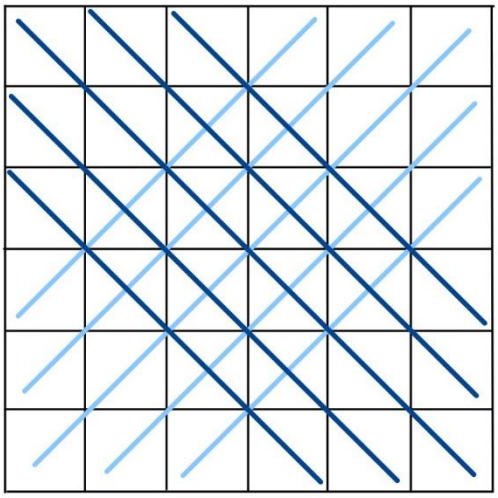
Při vyhodnocení stavu hry lze dosáhnout tří stavů a to výhry, remízy nebo reakce na prohru. Robot či uživatel zvítězí tehdy, kdy dostáhne vodorovně, svisle nebo diagonálně čtyř po sobě jdoucích stejných symbolů. Stav hry vyhodnocuje funkce checkScore.

Funkce postupně prochází zvlášť řádky, kdy na každém řádku zvlášť volá pomocnou funkci, která projde celý řádek a zkontroluje, jestli je počet spojených symbolů bez přerušení v daném řádku roven 4.

```
function vyhra = kHorizontal(board,y,spoj)
    [radky, sloupce] = size(board);
    spojene = zeros(2);
    vyhra = -1;
    for i=1:sloupce
        if (board(y,i) == 'X')
            spojene(1) = spojene(1) + 1;
            if (spojene(1) == spoj)
                vyhra = 'X';
                return;
            end
        else
            spojene(1) = 0;
        end
        if (board(y,i) == 'O')
            spojene(2) = spojene(2) + 1;
            if (spojene(2) == spoj)
                vyhra = 'O';
                return;
            end
        else
            spojene(2) = 0;
        end
    end
end
```

Následně se prochází jednotlivé sloupce a dochází ke kontrole, zda v daném sloupci je již spojena čtveřice symbolů. Vertikální kontrola probíhá obdobně jako horizontální.

Kontrola spojení diagonálně probíhá pouze v diagonálách, kde to má smysl. Kontrolované diagonály jsou znázorněné na obr. 32.



Obr. 32 - Znázornění kontrolovaných diagonál

Obecně má smysl v diagonále, pro jejíž počátek platí: řádek ≤ (počet řádků – počet spojovaných + 1), nebo sloupec ≤ (počet sloupců – počet spojovaných + 1). Pomocná funkce na kontrolu diagonály se volá v průběhu kontroly řádků a sloupců.

```
function vyhra = kDiagonal(board,add,x,y,spoj)
    [radky, sloupce] = size(board);
    spojene = zeros(2);
    vyhra = -1;
    for i=0:sloupce
        if (add)
            if ((x+i) > sloupce || (y+i) > radky)
                vyhra = -1;
                return;
            else
                if (board(y+i,x+i) == 'X')
                    spojene(1) = spojene(1) + 1;
                    if (spojene(1) == spoj)
                        vyhra = 'X';
                        return;
                    end
                else
                    spojene(1) = 0;
                end
                if (board(y+i,x+i) == 'O')
                    spojene(2) = spojene(2) + 1;
                    if (spojene(2) == spoj)
                        vyhra = 'O';
                        return;
                    end
                else
                    spojene(2) = 0;
                end
```

```
            end
        else
            if ((x-i)  < 1 || (y+i) > radky)
                vyhra = -1;
                return;
            else
                if (board(y+i,x-i) == 'X')
                    spojene(1) = spojene(1) + 1;
                    if (spojene(1) == spoj)
                        vyhra = 'X';
                        return;
                    end
                else
                    spojene(1) = 0;
                end
                if (board(y+i,x-i) == 'O')
                    spojene(2) = spojene(2) + 1;
                    if (spojene(2) == spoj)
                        vyhra = 'O';
                        return;
                    end
                else
                    spojene(2) = 0;
                end
            end
        end
    end
end
```

V momentě, kdy dojde k nalezení čtveřice spojených symbolů, okamžitě dojde k nastavení návratové hodnoty na daný symbol a návratu. K další kontrole již nedochází. Pokud se v poli žádná čtveřice nenachází a zároveň jsou v poli stále volná políčka, dojde k návratu hodnoty -1, která značí, že hra stále probíhá. Jestliže již v poli žádné volné políčko není, navrací se hodnota 0 vyjadřující remízu.

10.5.1 Reakce na konec hry

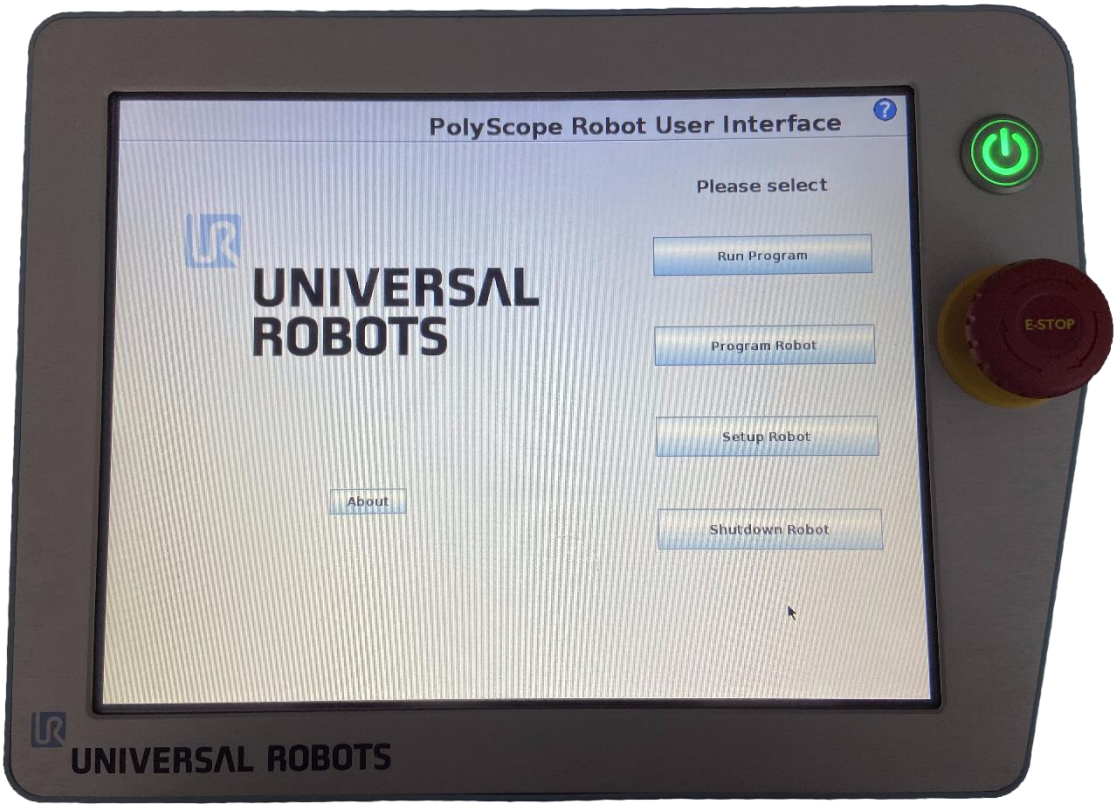
Pokud dojde ke konci hry, grafické rozhraní se resetuje podobně, jako při stisku tlačítka reset a stavová informativní zpráva vypíše, zda vyhrál robot, hráč, nebo došlo k remíze.

```
vysledek = checkScore(board);
if (vysledek ~= -1)
    hrac = handles.Game.HumanSymbol;
    handles = init(handles);
    switch(vysledek)
        case hrac
            set(handles.statusText,'String','Vyhral jste hru!');
        case 0
```

```
            set(handles.statusText,'String','Remiza!');
        otherwise
            set(handles.statusText,'String','Robot vyhral hru!');
    end
    h = handles;
    return;
end
```

10.5.2 Vypnutí robota

Nejsnadnější volbou vypnutí robota a celého systému, do kterého se zahrnuje robotické rameno, ovládací jednotku a program PolyScope, je možností v systému PolyScope použitím tlačítka Vypnout robota. Další možností je odpojení robota pomocí programového příkazu.



Obr. 33 - Úvodní obrazovka uživatelského rozhraní robota PolyScope

11 ZÁVĚR

Tato práce se zabývá programováním robotického ramene UR3 v jazyce Matlab, aby bylo schopné být automatizovaným protihráčem uživateli ve hře piškvorky.

Teoreticky je zde popsána problematika hry piškvorky, programovacího prostředí a výběr správného algoritmu pro provedení tahu. Dále jsou zde vysvětleny konvoluční neuronové sítě a jak probíhají jejich jednotlivé fáze, aby došlo k co nejlepšímu vyhodnocení hrací plochy. Je zde popsán robot UR3, jeho veškerá využití, a i bezpečnost při zacházení s ním.

V práci je dále detailně popsáno grafické prostředí pro ovládání robota. Je zde popsáno zpracování plochy, které zahrnuje pořízení a ořez snímku plochy a dále komplexní vyhodnocení pomocí konvoluční neuronové sítě. V kapitole zpracování dat je již řešena samostatná hra, kde pomocí grafického rozhraní kooperuje robot s uživatelem. Je zde popsáno vykreslení hracího pole a zakreslování symbolů X, O, dále samotný algoritmus, který vyhledává nejlepší možný tah pro robota a také možnosti vyhodnocení.

Pro rozpoznávání symbolů byla využita konvoluční neuronová síť s využitím balíčku Deep Learning Toolbox, který umožňuje jednoduchým způsobem definovat jednotlivé vrstvy konvoluční sítě, aniž bychom museli veškerou funkčnost sami programovat. Přesnost konvoluční sítě je závislá na počtu dat použitých k učení. Bylo využito celkem 375 snímků, tedy 125 od každého typu. S tímto množstvím dat bylo dosaženo dostačující přesnosti.

Vhodný tah robota je řešen pomocí algoritmu minimax s využitím alfa-beta ořezávání a iterativního prohlubování. Implementace algoritmu v této aplikaci však stále není ideálně optimalizovaná a výpočet trvá relativně dlouhou dobu. Algoritmus funguje velmi dobře do momentu, kdy vyhodnotí prohru při jakémkoliv tahu, a navíc není schopný prohru ani o tah oddálit. V tomto případě dojde k vyhodnocení stejné váhy pro všechna pole, tudíž zahraje první z kontrolovaných polí.

LITERATURA A ZDROJE

ALLIS, Louis Victor. 1994. *Searching for Solutions in Games and Artificial Intelligence. Ph.D. thesis, University of Limburg, The Netherlands*. ISBN 90-900748-8-0

BARTEL, Grant. *Playing Strategy Games With The Minimax Algorithm*. [online]. freeCodeCamp, 2014 [cit. 2020-07-05]. Dostupné z: https://www.freecodecamp.org/news/playing-strategy-games-with-minimax-4ecb83b39b4b/?fbclid=IwAR296S7pggIdKsUFR85sj3V8rhnwyZlk-9BLKUNgSPSXENr-ai7jrbJa0t0

Bezpečnost práce. [online]. pyroservis.cz. [cit. 2020-07-05]. Dostupné z: http://pyroservis.cz/ozo-bezpecnost-prace/?gclid=CjwKCAjwjLD4BRAiEiwAg5NBFjLj7lGPiJN3naTL26nT8Lv01cgU1ZnUoXfPc6M6e7BNgcVOYj8hMxoCBysQAvD_BwE&fbclid=IwAR3p5Hl1XGb4fFnSqLhXRqOdEY-mAsOSpsqp5zqO7naO99ECFdKM-qPxe9Q

BOUVRIE, Jake. *Notes on convolutional neural networks*. Technická zpráva, listopad 2006.

ČESKÁ FEDERACE PIŠKVOREK A RENJU. Oficiální pravidla piškvorek, 2019. [online]. [cit. 2020-07-15]. Dostupné z: http://www.piskvorky.cz/federace/oficialni-pravidla-piskvorek-gomoku/?fbclid=IwAR1vgVVdlaTL6mo37F0lNLuPTxsHf5_HV41wfxMQ2GQQo8wNg3UNbHKibNg

ČSN EN ISO 12100. Bezpečnost strojních zařízení – Všeobecné zásady pro konstrukci – Posouzení rizika a snižování rizika. Praha: Úřadu pro technickou normalizaci, metrologii a státní zkušebnictví, 2010

DURČÁK, Pavel. *Neuronové sítě a princip jejich fungování*. [online]. NaPočítači.cz, 2017 [cit. 2020-07-05]. Dostupné z: https://www.napocitaci.cz/33/neuronove-site-a-princip-jejich-fungovani-uniqueidgOkE4NvrWuNY54vrLeM670eFNQh552VdDDulZX7UDBY/?fbclid=IwAR2xNu-8vGQytJv2Jj2FE5E6vWhjNtrO7ZWuBkGOl7EvDO1toBThCc6s2Os

DUŠEK, F. *MATLAB a SIMULINK - úvod do užívání*. Univerzita Pardubice, 2000. 147 s. ISBN 80-7194-273-1

FELDMAN, J. a ROJAS, R. *Neural Networks: A Systematic Introduction*. 1996.

GOODFELLOW, Ian; BENGIO, Yoshua and COURVILLE, Aaron. *Deep learning. Adaptive computation and machine learning.* [online] The MIT Press, Cambridge, Massachusetts, 2016. Dostupné z: https://www.deeplearningbook.org/.

Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity. *Koncept umělé neuronové sítě*. [online]. Matematická biologie. [cit. 2020-07-15]. Dostupné z : https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site

Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity. *Učení s učitelem*. [online]. Matematická biologie. [cit. 2020-07-15]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--adaptacni-dynamika-neuronu--uceni-s-ucitelem

Institut biostatistiky a analýz Lékařské fakulty Masarykovy univerzity. *Učení bez učitele*. [online]. Matematická biologie. [cit. 2020-07-15]. Dostupné z : https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--adaptacni-dynamika-neuronu--uceni-bez-ucitele

KARN, Ujjwal. *An Intuitive Explanation of Convolutional Neuaral Networks*. [online]. the data science blog, 2016 [cit. 2020-07-05]. Dostupné z: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/?fbclid=IwAR3p0NfC_catBGii1jc4ZJnamEmsBx4tWRolOXuy30kEQfHk5U6nMK-_wF0

MATHWORKS. *Improve Shallow Neural Network Generalization and Avoid Overfitting*, 2016. [online]. [cit. 2020-07-15]. Dostupné z: http://www.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html

MATHWORKS. *MATLAB GUI* [online]. [cit. 2020-07-04]. Dostupné z: https://www.mathworks.com/discovery/matlab-gui.html

MOUJAHID, Adil. *Data Analytics and more*. [online]. 2016 [cit. 2020-07-10]. Dostupné z: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

QCExpert. Neuronová síť. [online]. [cit. 2020-07-15]. Dostupné z: https://www.trilobyte.cz/downloadfree/qcemanual/neural_net.pdf

SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya a SALAKHUTDINOV, Ruslan, 2014. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research. 15, 1929–1958. [cit. 2020-07-15].

UNIVERSAL ROBOTS. User manual UR3. [online] Universal Robots, 2009-2019. [cit. 2020-07-04]. Dostupné z: https://www.axima-obchod.cz/admin-data/storage/get/3334-ur3_user_manual_cs_global.pdf?fbclid=IwAR1795QxqlTMCNSc4zsCLLUgzoIn5qgh3Ktj4L7lTJKwX689kASdPV4hZzk

VOJÁČEK, Antonín. *Samoučící se neuronová síť - SOM, Kohonenovy mapy,* [online]. 2006. [cit. 2020-07-15]. Dostupné z: https://www.kiv.zcu.cz/studies/predmety/uir/NS/Samouc_NN2.pdf

VONDRÁK, I. *Neuronové sítě*. [online] Ostrava: VŠB-TU, 1994. 56. [cit. 2020-07-15] Dostupný z: http://vondrak.cs.vsb.cz/download/Neuronove site.pdf

WINSTON, Patrick. Search: Games, Minimax, and Alpha-Beta. [online] MIT 6.034 Artificial Intelligence, 2010. [cit. 2020-07-05]. Dostupné z: https://www.youtube.com/watch?v=STjW3eH0Cik&fbclid=IwAR3pH7YPIOyx5TvDJl_39vWhytbY5Bqd1xKktCP7g5vGk7bss9VTy0SEarA

PŘÍLOHY

A – CD

B – Manuál pro hru piškvorky mezi uživatelem a robotickým ramenem UR3

Příloha A

Příloha k bakalářské práci

OVLÁDACÍ SW ROBOTA UR3

PRO AUTOMATIZOVANÉ HRANÍ PIŠKVOREK

Anežka Blažková

CD

OBSAH

1. Text bakalářské práce ve formátu PDF
2. Úplný zdrojový kód aplikace v prostředí Matlab

Příloha B

Příloha k bakalářské práci

OVLÁDACÍ SW ROBOTA UR3

PRO AUTOMATIZOVANÉ HRANÍ PIŠKVOREK

Anežka Blažková

Manuál pro hru piškvorky mezi uživatelem a robotickým ramenem UR3

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato příloha slouží jako manuál k seznámení uživatele s aplikací pro hraní piškvorek s robotickým ramenem UR3. Uživatel by si měl nejprve prostudovat uživatelský návod, aby bylo předcházeno ohrožení uživatele či poškození robota.

Obsahuje nezbytná nastavení pro připojení k robotovi, pravidla hry a popisuje způsob, jakým robot komunikuje s uživatelem prostřednictvím grafického rozhraní a zároveň, jak uživatel komunikuje s robotem.

Aplikace je napsána v prostředí Matlab, proto je tento software požadován, a to ideálně ve verzi R2019b a novější.

1 PRAVIDLA HRY

Tato hra je určena pro dva hráče, a to robota a uživatele. Ti se střídají v tazích a umisťují na hrací plochu pomocí fixu psané symboly X nebo O. Hra probíhá na poli o velikosti 36 políček. Vždy začíná hráč, který si na začátku hry zvolil symbol X. Cílem a vítězstvím této hry je umístit vodorovně, svisle, nebo diagonálně řadu o 4 stejných symbolech.

1.1 BEZPEČNOST

Je důležité před samotným začátkem hry zatížit papír a nijak s ním dále nepohybovat, aby nedošlo k nepřesnému vyhodnocování hracího pole robotem a tím znemožnění optimální hry. Dále je důležité během hry neumisťovat žádné předměty do hracího pole.

1.2 OVLÁDÁNÍ

Obr. 1 - Grafické rozhraní

1.2.1 Symbol X a O

Slouží k volbě symbolu, kterým bude uživatel po celou dobu hrát. Druhý symbol po vybrání získá robot.

1.2.2 Start

Tlačítko Start umožňuje začátek hry. Po stisknutí začne robot vykreslovat hrací plochu, případně provede svůj první tah.

1.2.3 Předat tah

Tlačítko umožňující robotovi provést svůj tah. Poté se přesune do jeho výchozí pozice.

1.2.4 Reset

Ukončí hry v jejím průběhu. Po stisknutí se hra dostane na samotný začátek, znovu nabídne výběr možnosti symbolu X a O a probíhá dále úplně stejným způsobem jako dříve.

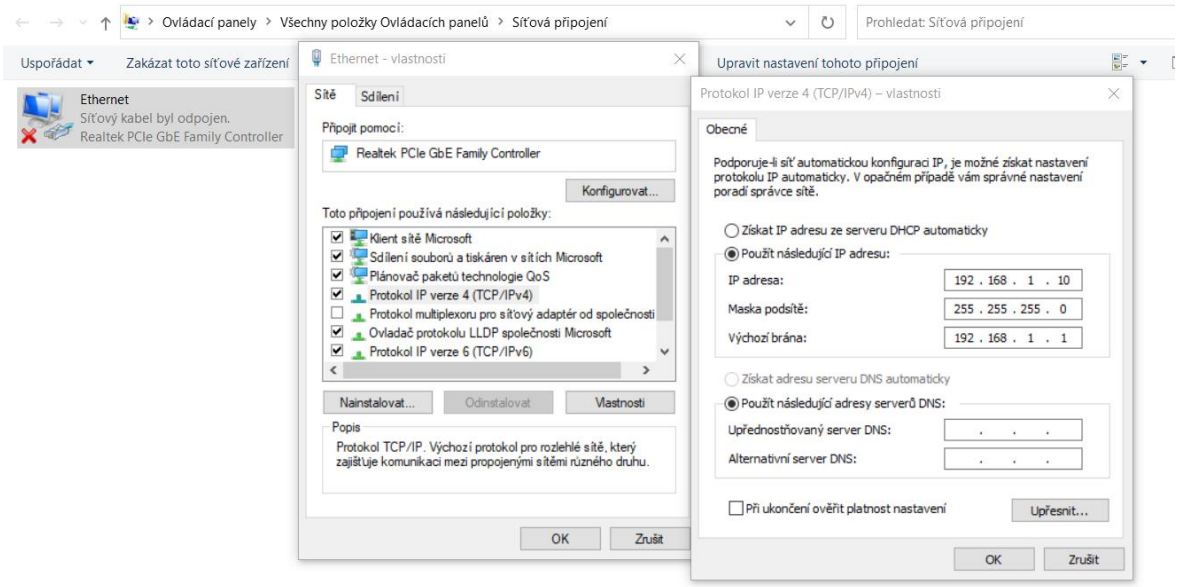
1.3 POSTUP

1.3.1 Navázání komunikace mezi počítačem a robotem

Nejprve je potřeba zapnout robota. Dále je potřeba pro samotné navázání komunikace s robotem připojit síťový kabel do svého zařízení. Poté je třeba nastavit IP adresu.

To lze najít v systému Microsoft Windows 10 následujícím postupem. Otevřeme nastavení, zvolíme položku Síť a internet, dále zvolíme sekci Ethernet, volbu Změnit možnosti adaptéru. Následně pravým tlačítkem rozklikneme ethernetový adaptér, vybereme vlastnosti, rozklikneme protokol IP verze 4 a v jeho vlastnostech nastavíme IP adresu na 192.168.1.10.

Dále je nutné otevřít program Matlab a spustit skript s grafickým uživatelským rozhraním.



Obr. 2 - Nastavení IP adresy pro komunikaci s robotem

1.3.2 Výběr symbolu

Po spuštění programu je zobrazeno grafické rozhraní, kde je v nabídce volba symbolu X nebo O. Hráč vybírá v rozhraní symbol, za který bude po celou dobu hrát, druhý získá robot. Je třeba dbát na pravidla a uvědomit si, že X vždy začíná hru.

1.3.3 Začátek hry

Na začátku hry je třeba stisknout tlačítko Start. To zamezí další změny hraného symbolu. Následně se začne vykreslovat hrací pole. Pokud zvolí uživatel symbol O, a tedy robotovi je přidělen symbol X, začíná robot po dokončení hrací plochy první tah. Po dokončení

tahu se vždy přesune do výchozí pozice. Pokud začíná hru uživatel, vyčká, než robot dokončí vykreslení hrací plochy a následně provede první tah.

1.3.4 Průběh hry

Jakmile odehraje uživatel svůj tah, stiskne tlačítko Předat tah. Tím potvrdí, že se v hracím poli již nenachází žádná jeho končetina ani jiný cizí předmět a dá robotovi možnost provést tah. V průběhu hry se tomu, kdo je aktuálně na tahu, rozsvítí zeleně přidělený symbol. Tím má uživatel větší kontrolu a přehled.

1.3.5 Reset a ukončení hry

V případě resetu hry, který může být vyžadován například v případě, že je pohnuto s hracím polem, a tedy znemožnění robotovi správně snímat, je třeba stisknout tlačítko Reset. Po stisknutí tohoto tlačítka se resetuje systém a objeví se nabídka výběru tlačítka X nebo O, dále je postup obdobný, jako v předchozí sekci Začátek hry.