FAKULTA STROJNÍ TUL

Bakalářská práce

Využití neuronové sítě k optickému snímání polohy bodu v prostoru.

Studijní program:	B0715A270008 Strojírenství
Autor práce:	**Martin Kolář**
Vedoucí práce:	Ing. Michal Sivčák, Ph.D. Katedra mechaniky, pružnosti a pevnosti

Liberec 2024

FAKULTA STROJNÍ TUL

Zadání bakalářské práce

Využití neuronové sítě k optickému snímání polohy bodu v prostoru.

Jméno a příjmení:	**Martin Kolář**
Osobní číslo:	S21000046
Studijní program:	B0715A270008 Strojírenství
Zadávající katedra:	Katedra mechaniky, pružnosti a pevnosti
Akademický rok:	2023/2024

Zásady pro vypracování:

Proveďte rešerši dostupných řešení pro optické snímání polohy bodu v prostoru. Zvolte si vhodný programovací jazyk a navrhněte neuronovou síť schopnou na základě informací z kamery (případně kamer) trasovat předem připravený geometrický bod v prostoru. Experimentálně vyhodnoťte přesnost použité metody.

Rozsah grafických prací:	
Rozsah pracovní zprávy:	30
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

HAYKIN, Simon S. *Neural networks and learning machines*. 3rd ed. New York: Prentice Hall, 2009. ISBN 978-0-13-147139-9.
CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, TensorFlow*. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora. ISBN 978-80-247-3100-1.

Vedoucí práce:	Ing. Michal Sivčák, Ph.D. Katedra mechaniky, pružnosti a pevnosti
Datum zadání práce:	6. listopadu 2023
Předpokládaný termín odevzdání:	31. května 2025

L.S.

doc. Ing. Jaromír Moravec, Ph.D. děkan	doc. Ing. Jaromír Moravec, Ph.D. garant studijního programu

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci 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.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

9. května 2024 Martin Kolář

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Michalu Sivčákovi, Ph.D. za cenné rady, pravidelné konzultace a pomoc při vedení této práce. Dále děkuji Bc. Lubomíru Sivčákovi za asistenci s výpočetním serverem katedry. V neposlední řadě bych rád vyjádřil vděčnost své rodině za jejich neustálou podporu, zejména své sestře.

Anotace

V této bakalářské práci se zabýváme vývojem neuronové sítě pro trasování geometrického bodu v prostoru pomocí kamerového systému. Zkoumáme možnosti jejího využití pro určování souřadnic bodu. V teoretické části je analyzována architektura a funkce neuronových sítí. Praktická část popisuje provedené testy za účelem nalezení optimální topologie sítě. Výsledky byly experimentálně ověřeny a porovnány s triangulační metodou.

Klíčová slova

neuronové sítě, strojové vidění, kamerové systémy, trasování bodu v prostoru

Anotation

In this bachelor thesis, we develop a neural network to trace a geometric point in space using a camera system. We investigate the possibilities of using it for determining the coordinates of a point. In the theoretical part, the architecture and functions of neural networks are analyzed. The practical part describes the tests performed in order to find the optimal network topology. The results were experimentally verified and compared with the triangulation method.

Keywords

neural networks, machine vision, camera systems, spatial point tracking

Obsah

Seznam použitých zkratek, symbolů **10**

Úvod a cíle bakalářské práce **12**

1 Metody snímání polohy a teorie **14**
1.1 Snímání polohy bodu v prostoru 14
1.1.1 Kamerovým systém 14
1.1.2 Pomocí signálu 15
1.1.3 Implementace neuronové sítě 15
1.2 Architektura neuronové sítě 17
1.2.1 Formální neuron 17
1.2.2 Neuronová síť 18
1.2.3 Organizační dynamika 18
1.2.4 Aktivní dynamika 19
1.2.5 Adaptivní dynamika 21
1.3 Backpropagace 21
1.4 Regulace 22

2 Získávaní dat **23**
2.1 Vývojové prostředí 23
2.2 Sběr dat 24
2.2.1 Kamery 24
2.2.2 Pracovní prostor 25
2.2.3 Poloha kontrastního bodu 27
2.2.4 Snímání dat 28
2.3 Předzpracování dat 30
2.3.1 Různorodost dat 33

3 Trénování sítě **34**
3.1 Osa x 38
3.2 Rovina xy 40
3.2.1 Porovnání aktivačních funkcí 40
3.2.2 Vliv regulace 42
3.2.3 Přidání skryté vrstvy 45
3.3 Prostor xyz 50
3.3.1 Porovnání s triangulací 52
3.3.2 Výsledek experimentu 54

3.3.3 Výpočet okamžité rychlosti 56
3.3.4 Výkon neuronové sítě 57

Závěr **58**

Literatura **59**

Seznam obrázků **62**

Seznam tabulek **64**

Seznam kódů **65**

Přílohy **66**

Seznam použitých zkratek, symbolů

$f()$	Aktivační funkce.	
b	Bias neuronu.	
RGB	Barevný model červená - zelená - modrá.	
HSV	Barevný model odstín - sytost - jas.	
t	Čas mezi jednotlivými body v prostoru.	$[s]$
Δz_n	Chyba mezi skutečnou polohou a polohou získanou pomocí regrese.	$[m]$
$MP4$	Formát multimediálního kontejneru definovaný standardem ISO/IEC 14496-14:2003.	
GHz	Frekvence procesoru.	
$z_{(t)}$	Funkce popisující polohu v ose z za čas.	$[m]$
fps	Jednotka snímkové frekvence videa (frame per second).	
k	Konstantní vzdálenost senzoru od ohniska kamery.	$[px]$
A, B, C	Koeficienty kvadratické lineární regrese.	
U_L	Levá projekce skutečného bodu.	$[px]$
$max\Delta xy$	Maximální absolutní chyba v rovině xy.	$[mm]$
$max\Delta xyz$	Maximální absolutní chyba v prostoru xyz.	$[mm]$
ms	Milisekunda.	
F	Ohnisko kamery.	
$v_{(t)}$	Okamžitá rychlost.	$[m/s]$
$v_{x(t)}$	Okamžitá rychlost v ose x.	$[m/s]$
$v_{y(t)}$	Okamžitá rychlost v ose y.	$[m/s]$
$v_{z(t)}$	Okamžitá rychlost v ose z.	$[m/s]$
C	Optický střed.	
O_i	Optická osa kamery.	
f_z	Optimalizovaná funkce regrese pro osu z.	
S_i	Označení senzoru kamery.	
n	Počet prvků.	
$300 - 150$	Počet neuronů ve skrytých vrstvách.	
z_n	Poloha n-tého bodu v ose z.	$[m]$
y_{s_i}	Poloha okrajového pixelu bodu na snímku.	$[px]$
P_i	Poloha středu bodu na snímku v ose x.	$[px]$
P_{y_i}	Poloha středu bodu na snímku v ose y.	$[px]$
R	Poloměr sledovaného bodu.	$[mm]$
U_P	Pravá projekce skutečného bodu.	$[px]$

CPU	Procesor.	
$\Delta\overline{xy}$	Průměrná absolutní chyba v rovině xy.	$[mm]$
$\Delta\overline{xyz}$	Průměrná absolutní chyba v prostoru xyz.	$[mm]$
L	Rozteč mezi kamerami.	$[mm]$
$CMOS$	Snímač.	
T	Snímaný bod pomocí triangulace.	
P	Skutečná poloha bodu při triangulační metodě.	$[mm]$
x_c	Souřadnice optického středu v ose x.	$[mm]$
y_c	Souřadnice optického středu v ose y.	$[mm]$
x_t	Souřadnice snímaného bodu pomocí triangulace v ose x.	$[mm]$
y_t	Souřadnice snímaného bodu pomocí triangulace v ose y.	$[mm]$
z_t	Souřadnice snímaného bodu pomocí triangulace v ose z.	$[mm]$
$RMSE$	Střední kvadratická odchylka nebo střední kvadratická chyba.	
$\sum \Delta z$	Suma chyb při regresy.	$[m]$
w_n	Synaptická váha.	
φ_i	Úhel natočení kamer v rovině xy.	$[rad]$
α_i	Úhel záběru bodu.	$[rad]$
Ψ_i	Úhel záběru bodu v ose x.	$[rad]$
γ_i	Úhel záběru bodu v ose z.	$[rad]$
β_i	Úhel sklonu kamery v rovině xz.	$[rad]$
ξ	Vnitřní potenciál neuronu.	
x_n	Vstupní signál.	
y	Výstup funkce.	
H	Výška kamery od stolu.	$[mm]$
x_{g_i}	Vzdálenost bodu od skla kamery.	$[mm]$
m	Vzdálenost skla od ohniska kamery.	$[mm]$

Úvod

V současném době, kdy se neuronové sítě stávají stále významnějším nástrojem v mnoha odvětví, začíná být tento trend více aplikován i v oblasti strojírenství. Zde nabízí možnosti pro automatizaci různých procesů pomocí strojového učení. Volba tématu naší práce je inspirována spoluprací TUL a MB na projektu, který je zaměřen na vývoj a prodej univerzálních robotických buněk, často využívaných pro technologie svařování. Tento projekt si klade za cíl pomocí strojového vidění zjednodušit proces programování robotů pro svařování, což představuje značnou úsporu času a finančních prostředků, zvláště v kontextu malosériové výroby. Aktuální přístup v robotickém svařování se různě zaměřuje na využití rozšířené reality, dálkově ovládané systémy a pokročilé metody sledování svarových spojů. Tyto metody, i přes svůj inovativní přístup a přínosy pro praxi, stále čelí výzvám spojeným s přesností, komplexností nastavení a integrací lidského know-how do automatizovaných systémů [1] [2] [3] [4].

Katedra mechaniky, pružnosti a pevnosti v současnosti pro určování polohy bodu v prostoru využívá triangulační metodu, která se opírá o analýzu kontur. U této metody se odchylky přesnosti pohybují do 5 mm. Přesnost výrazně ovlivňují vnější vlivy (poloha zdroje světla, teplota barvy, denní doba, atd.). To je zásadní limitace při aplikacích, kde nelze zajistit konstantní světelné podmínky, jako je například v robotickém svařování. Tato závislost na světelném prostředí brání širšímu využití triangulační metody v průmyslových aplikacích, kde jsou světelné podmínky proměnlivé.

Cílem naší práce je vyvinout a otestovat neuronovou síť schopnou určovat polohu předem stanoveného geometrického bodu v prostoru s využitím kamerového systému. Pro testování přesnosti sítě budeme používat dvě běžně dostupné webkamery, což nám umožní provádět testy s daty v základní kvalitě, obsahující různé šumy, nedokonalosti a proměnlivé světelné podmínky. Průběh práce bude zahrnovat teoretické představení architektury neuronové sítě. Shromažďování trénovacích dat, která budou získávána pomocí kamer a následně zpracována tak, aby byla vhodná pro trénování neuronové sítě. Vyvineme topologii neuronové sítě a provedeme její trénink. Během tohoto procesu realizujeme sérii testů, při kterých budeme hodnotit vliv různých úprav topologie na přesnost určování polohy. Porovnáme výsledky naší neuronové sítě s metodou triangulace, která se aktuálně používá na katedře, abychom vyhodnotili přínosy a omezení obou přístupů.

Při řešení problémů v terminálu programovacího prostředí budeme využívat ChatGPT 3.5 [5] jako rychlý a efektivní zdroj nápovědy. Tento nástroj nám umožní získávat okamžité odpovědi na dotazy týkající se programování, což značně zefektivní proces odstraňování chyb a optimalizaci našeho kódu.

Naše motivace pro využití neuronové sítě pro úspěšné trasování vyplynula ze dvou studií, které testovaly neuronové sítě ve svých automatizačních systémů. V těchto studiích bylo demonstrováno, že neuronové sítě jsou schopné se naučit predikovat polohu s určitou přesností v rámci různých oblastí aplikace. Tato pozitivní zjištění nás vedla k přesvědčení, že podobný přístup by mohl být úspěšně aplikován i v rámci naší práce [6] [7].

Očekáváme, že výsledky této práce dokážou, že neuronové sítě jsou vhodné pro predikci polohy bodu v prostoru. Zároveň, že bude naše práce základem pro vývoj metody určování polohy v rámci projektu společnosti Machine Building s.r.o.

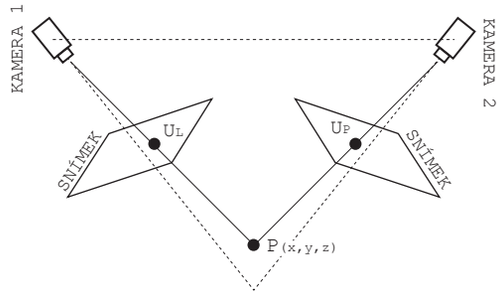
Kapitola 1

Metody snímání polohy a teorie

1.1 Snímání polohy bodu v prostoru

1.1.1 Kamerovým systém

Jedním ze známých způsobů optického snímání polohy v prostoru je triangulační metoda. Tato technika využívá dvě nebo více kamer, které jsou od sebe umístěny ve stanovené vzdálenosti. Tyto kamery zabírají stejnou scénu, ale každá pod jiným úhlem. Díky této rozdílné perspektivě je možné pomocí matematické analýzy, konkrétně triangulace, přesně určit polohu bodu v prostoru. Ilustrační schéma triangulační metody je zobrazeno na obrázku 1.1.



Obrázek 1.1: Schéma triangulační metody s dvěma kamerami zachycující skutečný pod P, který má svoje projekce U_L (levá projekce) a U_P (pravá projekce) na snímcích.

Proces funguje tak, že se identifikuje určitý bod v obrazech pořízených oběma kamerami. Vzhledem k tomu, že se jedná o stejný bod zachycený z různých úhlů, lze vytvořit trojúhelník, jehož základnu tvoří vzdálenost mezi kamerami a vrcholy jsou pozice kamer a hledaný bod. Pomocí geometrických principů a trigonometrických výpočtů lze pak určit přesnou polohu tohoto bodu v trojrozměrném prostoru [8] [9].

Optické sledování je metoda, která pro sledování pohybu a polohy objektů ve 3D prostoru využívá optických senzorů (především kamerových systémů). K sledování využívá algoritmus Motion Estimation, což je proces identifikace a modelování trajektorií pohybu sledovaných objektů na základě rozdílů mezi snímky pořízenými v různých časových okamžicích [10].

Další možnou metodou pro určení polohy bodu je strukturované osvětlení. Tato metoda využívá promítání známého vzoru světla (například mřížky nebo proužků) na scénu pro 3D skenování a mapování. Tato technika měří deformaci promítaného vzoru způsobenou různými povrchy objektů, což umožňuje rekonstruovat trojrozměrnou strukturu scény [11] [12].

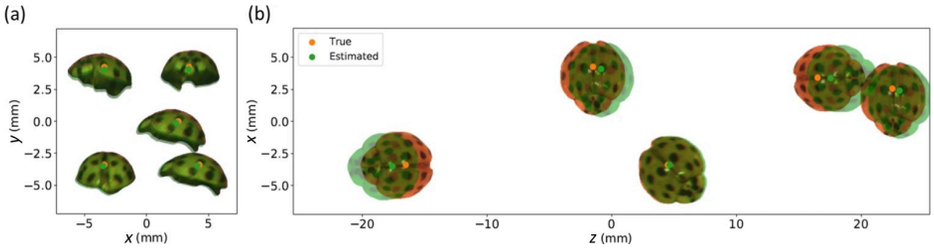
1.1.2 Pomocí signálu

Metoda času letu (Time-of-Flight) umožňuje určit polohu bodu pomocí měření času, který signálu (světelné impulsy, ultrazvukové vlny) trvá k cestě od zdroje k cíli a zpět. Základem je znalost rychlosti signálu a času, což umožňuje vypočítat vzdálenost mezi zdrojem a cílem. S kombinací informací o vzdálenosti, znalostí polohy a orientace senzoru lze získat prostorovou informaci o poloze bodů vzhledem k senzoru [13] [12].

Na podobném principu funguje metoda LIDAR (Light Detection and Ranging). Ta umožňuje určit polohu bodu v prostoru pomocí vysílání laserových paprsků a měření času, který trvá odraženému signálu k návratu zpět k senzoru. Tímto způsobem je možné měřit vzdálenost mezi senzorem a povrchem cíle [14].

1.1.3 Implementace neuronové sítě

V rámci studie *Neural network based 3D tracking with a graphene transparent focal stack imaging system* byla implementována vícevrstvá neuronová síť, která byla využita pro sledování bodových a více bodových objektů. Zde byly sítě trénovány pro odhad x, y a z souřadnice z obrazových dat. Data získávána pomocí konvenčních CMOS kamer, která byla převedena do zásobníků zaostření (focal stacks). Více bodový objekt představovala beruška, u které se určovala poloha a orientace.

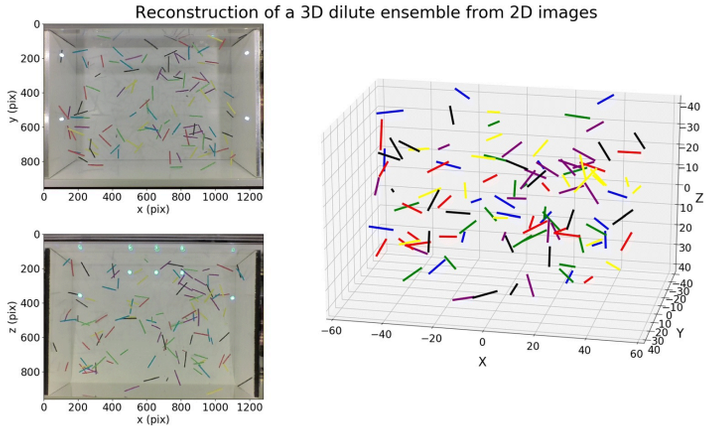


Obrázek 1.2: Výsledky odhadované (skutečné) polohy a orientace berušky jsou vyznačeny zelenými (oranžovými) body a zeleně (oranžově) překrytými obrázky berušky. Obrázky berušek (více bodový objekt) nejsou výstupem neuronové sítě, slouží pouze jako ilustrace [6].
(a) Výsledky v perspektivě roviny xy, (b) Výsledky v perspektivě roviny xz

Neuronové sítě byly navrženy s využitím aktivační funkce ReLU a byly optimalizovány pomocí Adam optimizeru. Výsledky (obr. 1.2) ukázaly, že pro jednobodové objekty bylo dosaženo kořenové střední kvadratické odchylky (RMSE) v řádu desetin milimetrů pro osy x, y a z. U vícebodových objektů přesnost sledování rostla

s rozlišením senzorů, což naznačuje, že vyšší rozlišení vede k lepší schopnosti sledování složitějších scén [6].

Další studie, která implementovala neuronovou síť je *Machine Learning for 3D Particle Tracking in Granular Gases*. Studie se věnovala automatizovanému sledování a analýze 3D pozic a orientaci částic v granulárních plynů. Data pro trénování sítě byla experimentálně vizualizována a anotována s přesnými pozicemi konců částic, což umožnilo efektivní učení sítě.



Obrázek 1.3: Výsledky automatické rekonstrukce 3D scény pro zředěný soubor tyčí na základě dvou na sebe kolmých pohledů kamery [7].

Výsledky (obr. 1.3) studie ukázaly vysokou přesnost detekce částic srovnatelnou s manuální analýzou. Systém byl schopen zpracovat velké množství dat s minimálními požadavky na manuální korekce, což výrazně zefektivnilo analýzu [7].

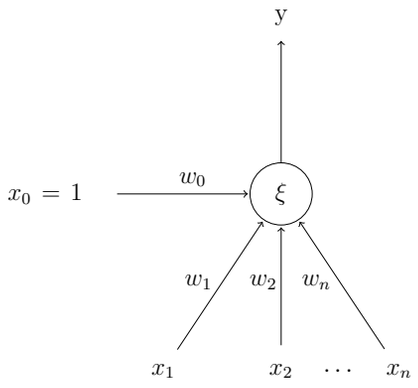
Článek *3D Object Detection and Tracking Methods using Deep Learning for Computer Vision Applications* se zabývá metodami určování polohy v prostoru pro detekci a sledování 3D objektů. Autoři poukazují na význam přesného určení 3D polohy objektů, které je zásadní pro vylepšení funkcionality autonomních systémů, jako jsou autonomní vozidla a robotika. Použité neuronové sítě byly optimalizovány s využitím technik jako Adam optimizer a trénovány na datech z LIDARu a RGB kamer. Výsledky naznačují, že tyto metody zlepšují schopnost počítačového vidění ve složité scéně [15].

Z provedeného rešerše jsme zjistili, že efektivita snímání polohy pomocí kamerových (optických) systémů je ovlivněna okolním osvětlením. Triangulační metoda například čelí výzvám při změnách osvětlení, kdy nedokáže správně určit polohu. Schopnosti těchto optických systémů jsou omezeny podmínkami prostředí, ve kterém dochází k měření. V této oblasti by využití neuronové sítě mohlo představovat vhodné řešení. Neuronové sítě nabízí flexibilitu, adaptabilitu a mohou účinně řešit problémy spojené s osvětlením. Díky své schopnosti učit se a přizpůsobovat se různým světelným podmínkám na základě tréninkových dat. Z rešerše relevantních studií vyplývá, že neuronové sítě jsou často optimalizovány pomocí Adam optimizeru. Oblíbenou aktivační funkcí je ReLU pro její efektivitu v trénování hlubokých neuronových sítí. Tyto prvky architektury neuronové sítě plánujeme implementovat do naší práce. Informace o specifické topologii sítě jsme v dostupných studiích nenalezli, proto uskutečníme sérii testů k určení optimální topologie pro naši práci.

1.2 Architektura neuronové sítě

1.2.1 Formální neuron

Formální neuron je základem matematického modelu neuronové sítě. Struktura formálního neuronu (dále jen neuronu) je zobrazena na obrázku 1.4.



Obrázek 1.4: Struktura formálního neuronu.

Do neuronu obecně vstupuje n reálných vstupů $x_1, ..., x_n$, které modelují dentrity. Tyto vstupy jsou ohodnoceny reálnými synaptickými váhami $w_1, ..., w_n$, ty určují propustnost vstupů. Vnitřní potenciál neuronu je vyjádřen váženou sumou vstupních hodnot:

$$\xi = \sum_{i=1}^{n} w_i x_i. \tag{1.1}$$

Bias (prahová hodnota) lze do vztahu implementovat přidáním členu $x_0 = 1$. Bias je ve vztahu dále zpracováván jako váha $w_0 = b$. Po úpravě je vstup do neuronu ξ dán vztahem:

$$\xi = \sum_{i=0}^{n} w_i x_i = w_0 + \sum_{i=1}^{n} w_i x_i = b + \sum_{i=1}^{n} w_i x_i. \tag{1.2}$$

Při dosažení hodnoty interního potenciálu ξ, neuron indikuje výstup y. Bias b se přidává k vážené sumě vstupů, čímž ovlivňuje úroveň ξ a určuje, kdy neuron aktivuje svůj výstup. Nelineární nárůst výstupní hodnoty $y = f(\xi)$ nastává v reakci na hodnotu ξ, kterou ovlivňuje i b. Tento nárůst je dán aktivační (přenosovou) funkcí f. Pro představu je zde ukázka na nejjednodušší aktivační funkci typu ostrá nelinearita:

$$f(\xi) = \begin{cases} 1 & \text{pokud } \xi \geq 0 \\ 0 & \text{pokud } \xi < 0 \end{cases}. \tag{1.3}$$

1.2.2 Neuronová síť

Neuronová síť se skládá z několika neuronů, které jsou propojené mezi sebou tak, že výstup jednoho neuronu se stává vstupem pro jeden nebo více dalších. Jejich počet a způsob propojení určuje architekturu sítě. V dané síti rozlišujeme vstupní, skrytý a výstupní neuron. Vstupní neurony přejímají informace. Propojením skrytých neuronů dochází k síření informací na výstupní neuron.

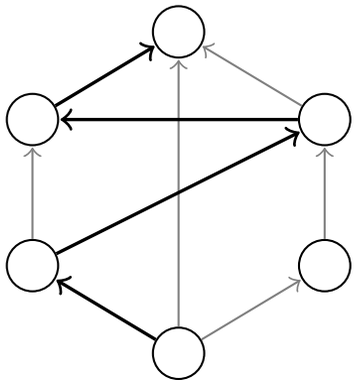
Neuronové sítě se v čase vyvíjejí, jejich propojení a stav neuronů se mění, dochází k adaptaci vah mezi nimi. Změnou těchto charakteristik v čase je vhodné rozdělit dynamiku sítě do tří skupin: *organizační (topologie), aktivní (změna stavu) a adaptivní (změna konfigurace)* (Šíma, 1996, s. 29).

Stanovením jednotlivých dynamik získáváme různé modely neuronových sítí. Každý model je vhodný pro řešení odlišných úloh. Pro specifikaci modelu nám tedy stačí definovat jeho tři dynamiky. Dále v práci popíšeme ty dynamiky, které byly použity při tvorbě našeho modelu neuronové sítě [16].

1.2.3 Organizační dynamika

U organizační dynamiky předpokládáme takovou architekturu sítě, která se v čase nemění. Mluvíme o dvou typech: *cyklická (rekurentní) a acyklická (dopředná) síť* (Volná, 2008, s.14).

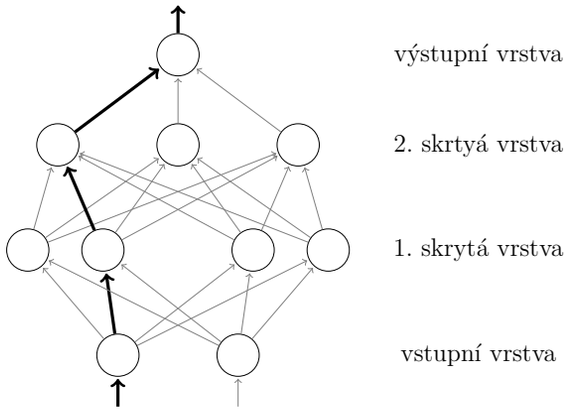
Při tvorbě našeho modelu byla využita acyklická dynamika. Na rozdíl od cyklické, v acyklických neexistuje cyklus (skupina neuronů spojena do kruhu), tedy všechny cesty vedou jedním směrem (obr. 1.5).



Obrázek 1.5: Příklad acyklické architektury neuronové sítě s vyznačenou nejdelší cestou.

Využitím tohoto předpokladu rozdělujeme neuronové sítě do vrstev. Ty jsou uspořádané tak, že jednotlivé spoje neuronů vedou ze spodních vrstev do vrstev horních (je možné přeskočit jednu nebo více z nich). Jedním z těchto typů architektury je vícevrstvá neuronová síť. V síti jsou tři kategorie vrstev. První je vstupní vrstva složena z vstupních neuronů. Další je známá jako výstupní vrstva, která obsahuje výstupní neurony. A mezi nimi se nachází ostatní vrstvy, které označujeme jako skryté. V této topologii jsou neurony bezprostředně spojeny se všemi neurony, které

se nachází v další vrstvě. Na příkladu vícevrstvé neuronové sítě (obr. 1.6) je vyznačena cesta, která vede všemi vrstvami.



Obrázek 1.6: Příklad architektury vícevrstvé neuronové sítě s vyznačenou jednou cestou.

1.2.4 Aktivní dynamika

Aktivní dynamika určuje počáteční stav neuronové sítě a pravidla pro změnu stavu v průběhu času, přičemž zůstává stejná topologie a konfigurace sítě. Na začátku je nastavená konkrétní hodnota vstupních neuronů na tzv. vstup sítě. Ostatní neurony jsou v daném počátečním stavu. Všechna tato počáteční nastavení a změny sítě tvoří vstupní prostor neuronové sítě. Po inicializaci nastává vlastní výpočet sítě. Pokud obecně předpokládáme spojitý vývoj stavu neuronové sítě v čase, hovoříme o spojitém modelu, který obvykle modelujeme pomocí diferenciálních rovnic. V praxi se většinou setkáváme s předpokladem diskrétního stavu sítě, kdy se stav sítě mění v určitých časových okamžicích. V každém tomto okamžiku je podle stanoveného pravidla vybrán jeden neuron (sekvenční výpočet) nebo více neuronů (paralelní výpočet), které podle jejich vstupů aktualizují svůj stav. Stav výstupních neuronů, který se časem mění, vytváří výstup neuronové sítě. V běžných situacích uvažujeme o tom, že aktivní dynamika vede ke konstantnímu výstupu neuronové sítě, který nastane po určité době. V takovém případě neuronová síť realizuje určitou funkci na vstupním prostoru v aktivním stavu. Tato specifická funkce neuronové sítě je dána pravidly aktivní dynamiky, která je závislá na topologii a konfiguraci sítě. Tyto parametry zůstávají neměnné, což umožňuje síti v aktivním režimu provádět vlastní výpočty.

Aktivní dynamika neuronové sítě specifikuje funkce jednotlivých neuronů, kromě vstupních neuronů, které mají obvykle stejný matematický vzorec (předpis). To je charakteristické pro homogenní neuronovou síť. Při tvorbě našeho modelu, se setkáme s následujícími aktivačními funkcemi [17]:

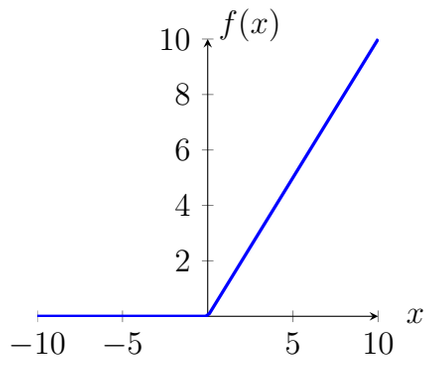
Aktivační funkce ReLU (Rectified Linear Unit) (1.4) je jedna z nejpoužívanějších funkcí při hlubokém učení, tato funkce nemění velikost daného vstupu a je dána:

$$f(x) = max(0, x). \tag{1.4}$$

To v praxi znamená, že pokud je na vstupu hodnota x kladná, funkce vrátí hodnotu x (1.5). Bude-li hodnota záporná, funkce vrátí nulu:

$$f(x) = \begin{cases} x & \text{pokud } x \geq 0 \\ 0 & \text{pokud } x < 0 \end{cases}. \tag{1.5}$$

Graf této aktivační funkce je znázorněn na obrázku **??**.



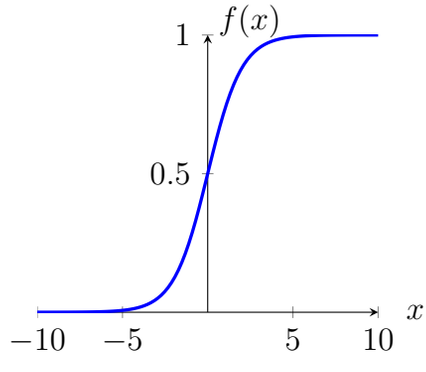
Obrázek 1.7: Graf aktivační funkce ReLU.

ReLU je nelineární funkce, která je vhodná pro učení složitých vzorů v datech [18]. Zároveň zmírňuje problém s mizejícím gradientem, který má vliv během backpropagace. Je výpočetně efektivnější, jak z hlediska paměti, tak výpočetního času. Dochází pouze k jednoduchému porovnání s nulou a není potřeba výpočet exponenciálních funkcí jako například při použití funkce sigmoid (1.6).

Aktivační funkce sigmoid (1.6) se řadí též mezi nejpoužívanější funkce a je dána:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1.6}$$

Patří mezi nelineární funkce, která oproti funkci ReLU vrací hodnoty v rozmezí 0 až 1. Není symetrická kolem nuly, což znamená, že všechny výstupní hodnoty budou mít stejná znaménka. Graf této funkce je znázorněn na obrázku 1.8 [19].



Obrázek 1.8: Graf aktivační funkce sigmoid.

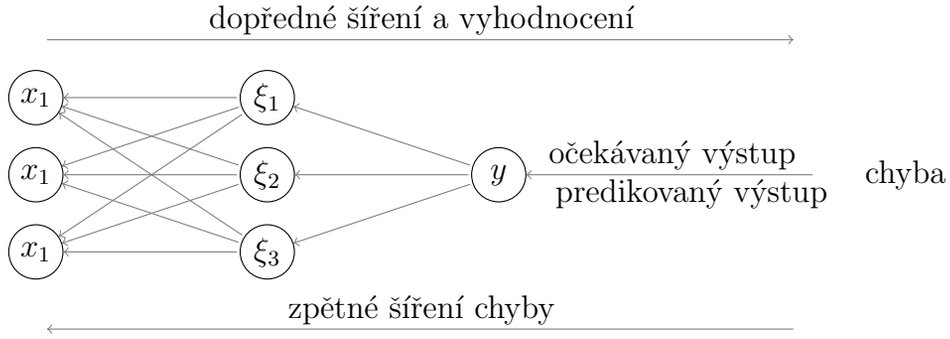
1.2.5 Adaptivní dynamika

Adaptivní dynamika neuronové sítě se zabývá počáteční konfigurací sítě a změnou váhových hodnot v čase mezi neurony. Všechny konfigurace tohoto procesu tvoří váhový prostor neuronové sítě. Na začátku adaptivního režimu se nastaví počáteční konfigurace vah všech spojů v síti. Po dokončení inicializace sítě probíhá vlastní adaptace. Obdobně jako v případě aktivní dynamiky uvažujeme o spojitém modelu, kde se konfigurace sítě a váhy mezi neurony mění v čase dle spojité funkce. V tomto případě spojitá funkce je zadaná diferenciální rovnicí. Přesto se většinou předpokládá adaptace vah v diskrétním čase. Cílem adaptace je objevit právě takovou konfiguraci sítě ve váhovém prostředí, aby realizovala danou funkci v aktivním režimu. Můžeme říci, že pokud je aktivní režim využíván pro realizaci dané funkce, tak adaptivní režim se využívá k učení (programování) této funkce.

Funkci, kterou chceme, aby neuronová síť vykonávala, se typicky definuje pomocí trénovací množiny dvojic vstupů nebo výstupů sítě. Při adaptaci našeho modelu neuronové sítě je využito učení s učitelem (supervised learning). Učitel předepisuje správné hodnoty na výstupu pro konkrétně zadané vstupy a tímto způsobem adaptuje chování sítě. [17].

1.3 Backpropagace

Neuronová síť využívá algoritmus zpětného šíření chyb (backpropagace) k učení na základně optimalizace vah jednotlivých neuronů. Ten probíhá vypočítáním chyby (například pomocí střední kvadratické chyby) mezi výstupy sítě a očekávanými hodnotami. Tento proces trénování probíhá ve vícero cyklech neboli epochách. Síť během epoch opakovaně zpracovává tréninková data a upravuje hodnoty vah mezi jednotlivými neurony. Zároveň se snaží snižovat celkovou chybu výstupu. Tento proces představuje adaptaci mechanismu sítě, který se snaží dosáhnout stanovené tolerance [20]. Na obrázku 1.9 je ilustrační schéma zpětného šíření chyby.



Obrázek 1.9: Zpětné šíření chyby neuronovou sítí (backpropagace), chyba = predikovaný výstup - očekávaný výstup.

Vypočítaná chyba představuje rozdíl mezi očekávanými a predikovanými výstupy. Ta je následně šířena zpět sítí skrze každou vrstvu. Pro jednotlivé vrstvy jsou vypočítány derivace ztrátové funkce. Tyto funkce nám umožní vypočítat gradienty, které jsou důležité pro úpravu vah. Váhy se upravují způsobem, který minimalizuje ztrátovou funkci. Upravené váhy se následně ukládají a znovu se upravují do té doby, dokud se nenalezne jejich optimální velikost [21].

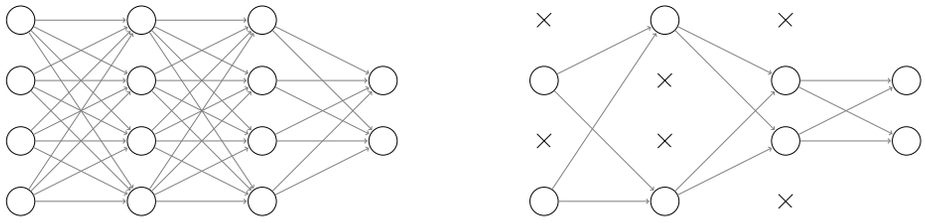
Pro nalezení minimální ztrátové funkce se používá základní optimalizační algoritmus, který funkci minimalizuje. Gradient ztrátové funkce ukazuje směr, ve kterém ztrátová síť nejrychleji roste. Tento algoritmus pracuje na principu pohybu v opačném směru tohoto gradientu. Opačný směr je chápán jako směr k nejrychlejšímu poklesu ztrátové funkce. Pro nastavení optimálních parametrů neuronové sítě se alegorismus snaží najít minimum. Tento proces probíhá opakovaně na základě stanoveného počtu epoch [22].

1.4 Regulace

Modely neuronových sítí, které pracují s velkým množství dat, se mohou v některých případech přizpůsobit trénovacím datům. To znamená, že nedochází k úpravám vah a síť se přestává učit. Tento jev se nazývá přeučení modelu (overfitting). Nastává tehdy, když si neuronová síť dobře zapamatuje trénovací data včetně jejich náhodných šumů a detailů. Dochází tím ke špatnému chování sítě na dosud neviděných datech. Aby se zabránilo přeučení neuronové sítě, používají se regulační metody [23]. Pro náš experiment jsme testovali pouze dvě: L2 regulace a dropout.

1. L2 regulace (váhový útlum): Přidává penalizační hodnotu k původní ztrátové funkci. Tímto způsobem tlačí na to, aby velikosti vah zůstaly malé [24].

2. Dropout: Při aplikaci na vrstvy dochází k náhodnému vynechání neuronů ve skrytých vrstvách. Jejich vnitřní potenciál je následně roven 0 a neprobíhá jeho aktivace [25].



Obrázek 1.10: Ukázka principu regulace dropout, porovnání neuronové sítě bez a s regulací.

Kapitola 2

Získávaní dat

2.1 Vývojové prostředí

K vývoji potřebných programů a skriptů jsme zvolili editor Visual Studio Code pro jeho flexibilitu a snadné instalovaní uživatelských balíčků.

Jako programovací jazyk jsme zvolili Python 3.11.5, který je populární v oblasti analýzy dat a strojového učení. Tím se pro nás stává vhodnou volbou pro řešení našeho experimentu. Během vývoje jsme využili několik klíčových knihoven, které Python umožňuje využívat při řešení různých problémů. K práci na našem experimentu jsme nainstalovali knihovny, které jsou uvedeny v tabulce 2.1, ostatní použité knihovny jsou součástí Pythonu.

Tabulka 2.1: Použité knihovny a jejich verze.

OpenCV	4.8.1.78	[26]
NumPy	1.26.0	[27]
Pandas	2.2.2	[28]
Pillow	10.0.1	[29]
OpenPyXL	3.1.2	[30]
Tensorflow	2.15.0 později 2.16.1	[31]
Keras	2.15.0 později 3.1.1	[32]

Pro tvorbu modelu neuronové sítě jsme zvolili knihovnu TensorFlow od vývojářů Google Brain. Jako rozhraní jsme použili open-source knihovnu Keras (tab. 2.1). Na změnu verze Keras bude upozorněno v kapitole 3 této práce.

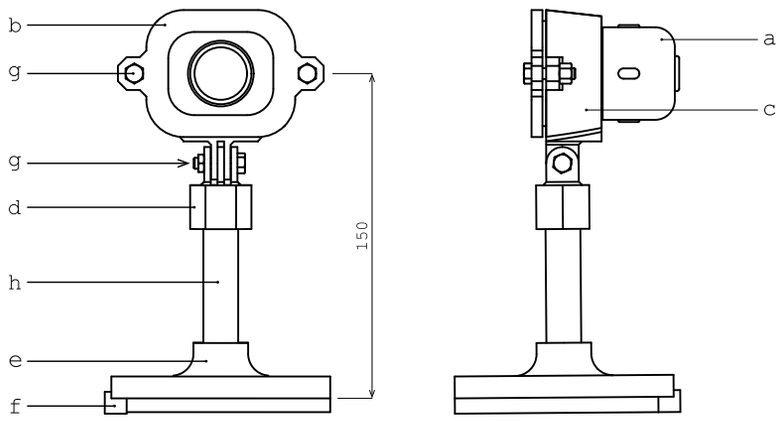
2.2 Sběr dat

2.2.1 Kamery

K získání potřebných dat o poloze sledovaného bodu jsme zvolili dvojici kamer Logitech C980 StreamCam. Tento výběr byl z dostupných kamer na katedře. To umožňuje otestování neprůmyslových kamer v naší problematice neuronové sítě.

Za účelem zvýšení přesnosti snímání a zajištění konzistence získávaných dat, jsme vytvořili speciální stativ (příloha 01). Originální držák dodávaný s kamerou nesplňoval potřebné požadavky na tuhost. Docházelo zde k výraznému pohybu kamery. Nově vytvořený stativ kamery má výrazně větší tuhost a je bez vůlí v uloženích. Tím jsme splnili jednu z podmínek pro zachování co možná nejvyšší možné přesnosti a spolehlivosti při sběru dat.

Pro zajištění pevného spoje mezi kamerou a hlavou stativu, jsme vytvořili konstrukci na principu dvou šroubů a využití geometrie těla kamery. Šrouby přitlačují víko hlavy stativu na tělo kamery. Tím získáváme vysokou stabilitu spoje a zaručenou míru přesnosti při reprodukovatelnosti dat získaných během záznamu. Popis stativu kamery je na obrázku 2.1.



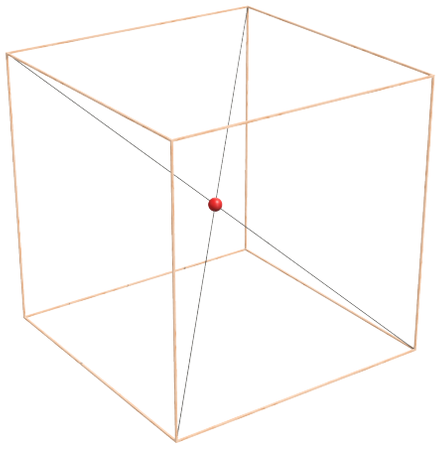
Obrázek 2.1: Kamera umístěna do stativu, pro stabilní přichycení ke kraji stolu. Vyznačena výška, ve které se nachází střed snímače od stolu.

a - kamera, b - víko hlavy stativu, c - tělo hlavy stativu, d - kloub, e - základna, f - doraz, g - šroubový spoj, h - trubka

Za účelem širší manipulace s kamerou jsme vytvořili kloub ve tvaru šestihranu, který vytváří spojení kompatibilní se systémem používaným například s kamery GoPro. Toto řešení umožňuje široký rozsah nastavení pozice kamery. K potřebám našeho experimentu jsme vyrobili stativ, který umožňuje pevně uchytit kameru ke kraji pracovního stolu. Tato konstrukce zajišťuje jednoduché sestavení experimentu a stabilní umístění během záznamu dat.

2.2.2 Pracovní prostor

Abychom získali maximálního využití plochy snímače kamery, vytvořili jsme model pracovního prostoru s vyznačeným středem (obr. 2.2). Tento krok nám dovoluje orientovat kamery tak, aby střed pracovního prostoru mířil do optického středu snímaného obrazu. Tím také získáváme informaci, zda je zabrán celý pracovní prostor pro nejlepší dosažitelnost výsledků.



Obrázek 2.2: Pracovní prostor o velikosti 300x300x300 mm s vyznačeným středem pro nastavení kamer.

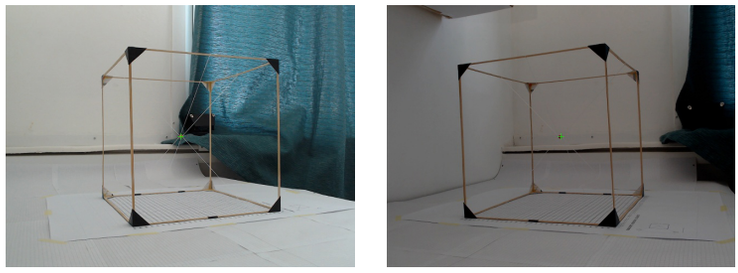
Specifikace středu nám usnadňuje nastavení obou kamer pod stejným úhlem. Umožňuje vysokou možnost reprodukce a konzistenci dat při přemístění nebo změně pracovního prostředí. Tím jsme schopni zachovat vyšší úroveň přesnosti a spolehlivosti měření, což je pro náš experiment zásadní.

K zjednodušení a zpřesnění nastavování orientace kamer jsme napsali jednoduchý program. Ten jsme navrhli tak, aby zobrazoval stejný obraz ze dvou kamer zároveň a zelený křížek, který označuje střed snímače (obr. 2.3). Tento program zajišťuje rychlé a efektivní nastavení orientace kamer, či jejich případnou kalibraci. Pomocí implementace tohoto programu jsme zajistili, že osa kamery bude vždy mířit na střed pracovního prostoru. Zde je ukázka kódu. Jedná se o funkci, která vizualizuje značky na středu (kód 2.1). Celý program nalezneme v příloze 02.

```
1  def mark(fixed_width, fixed_height, frame_1, frame_2):
2    # poloha znacky
3      markerX = fixed_width // 2
4      markerY = fixed_height // 2
5    # nastaveni znacky
6      markerType = cv.MARKER_CROSS  # typ
7      markerColor = (0, 255, 0)     # barva
8      markerSize = 10               # velikost
9      markerThickness = 1           # tloustka
10   # vykresleni znacky
11     cv.drawMarker(frame_1, (markerX, markerY),
12     markerColor, markerType, markerSize, markerThickness)
13     cv.drawMarker(frame_2, (markerX, markerY),
```

```
14      markerColor, markerType, markerSize, markerThickness)
```

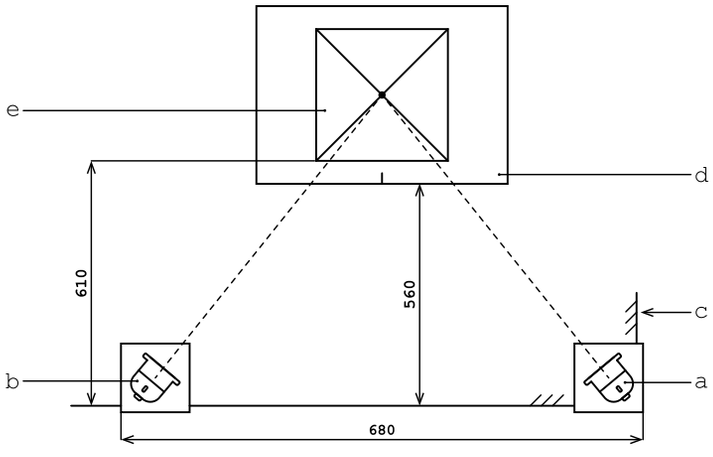
Kód 2.1: Funkce pro vytvoření zeleného křížku na středu obrazů z kamer, která mají velikost `fixed_width` a `fixed_height`. Obrazy z kamer jsou `frame_1` a `frame_2`. Funkce slouží pro nastavení směru kamer do optického středu pracovního prostoru



Obrázek 2.3: Výstup z programu (příloha 02), který zobrazuje stejný obraz ze dvou různých kamer s vyznačením zeleným křížkem, pro nastavení kamer.

Pro dosažení dostatečné přesnosti při pohybu posuvného držáku bodu jsme vytvořili čtvercovou síť (příloha 03) s rastrem 10 mm, kterou jsme vytiskli na formát A2. Tímto se můžeme opakovaně a efektivně vracet na již snímanou polohu, což nám umožňuje získat více dat z jedné polohy v různých časových intervalech. Celkově tento koncept usnadňuje práci při opakovaném získávaní potřebných dat.

K získání dat pro naši neuronovou síť jsme definovali specifické parametry pracovního prostředí (obr. 2.4), který je definováno následovně: Vzdálenost mezi vnějšími okraji stativů kamer je pevně nastaveno na 680 mm. Hranu čtvercové sítě jsme umístili 560 mm od okraje stolu. Zároveň jsme ji nastavili na osu kamer, pro zajištění optimalizace přesnosti snímání. Pracovní prostor jsme dali do vzdálenosti 610 mm od okraje stolu a tím poskytuje dostatečný prostor pro manipulaci se sledovaným bodem.



Obrázek 2.4: Definice pracovního prostředí, s vyznačenými vzdálenostmi: mezi kamerami, od okraje stolu, pracovního prostoru.

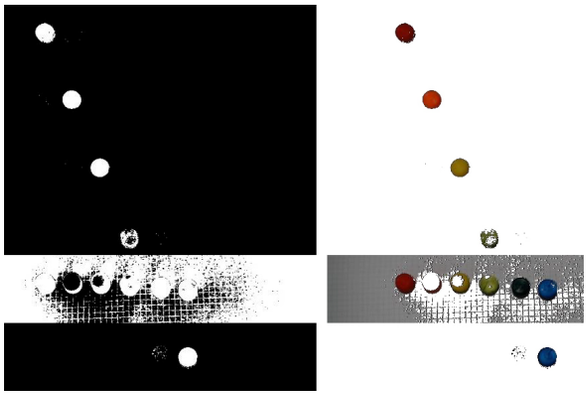
a - kamera 1, b - kamera 2, c - okraj stolu, d - čtvercová síť,
e - pracovní prostor

2.2.3 Poloha kontrastního bodu

Pro simulaci bodu v prostoru jsme vytvořili posuvný držák, který je určen k manipulaci kontrastním předem daným geometrickým bodem. Tento kontrastní bod nám reprezentuje sledovaný bod ve snímaném prostoru, jehož polohu chceme určit. Při návrhu jsme zvažovali několik klíčových faktorů, počínaje barvou bodu. Zde je nutný dostatečný kontrast s pozadím pracovního prostředí pro snadné rozpoznání kamerou. Další byl na řadě mechanismus, který umožňuje dostatečně přesné nastavení polohy středu bodu.

Při volbě optimální barvy bodu jsme použili poupravený program (příloha 04) *finding HSV value of target object(Pen)* [33], který nám pomohl určit vhodný rozsah hodnot pro vytvoření kontur v obraze. Pomocí něho jsme prováděli několik testování výběru správné barvy. Jeden z těchto testů je zobrazen na obrázku 2.5, vytvořená maska a samotná barva bodu.

Vytvoření kontur ovlivňovalo několik faktorů. Jelikož bod byl vytištěn na 3D tiskárně, objevoval se zde vliv povrchu tištěného materiálu. Lesklý povrch vytvářel odlesky - světlejší místa něž na zbytku bodu. To vytvářelo problém, který mohl vést ke špatnému vytvoření kontury, mohlo docházet k záměně s pozadím nebo jinou barvou.

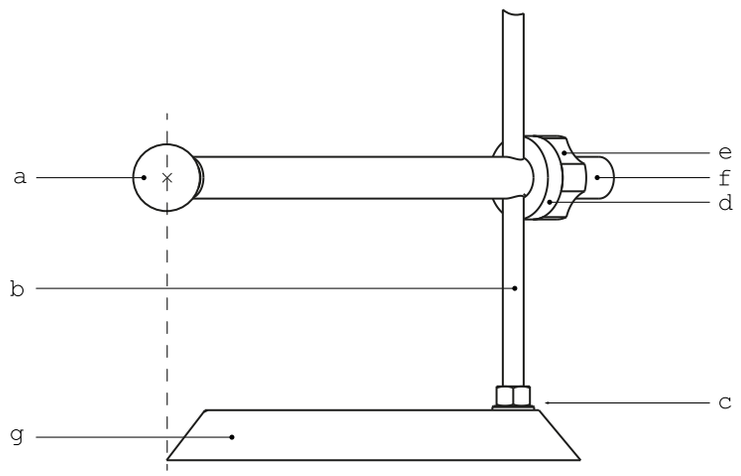


Obrázek 2.5: Porovnání vlivu barvy kontrastního bodu na vytvoření masky pomocí barevného modelu HSV k nalezení kontur a určení přibližného středu kontrastního bodu.

Následně jsme věnovali pozornost i samotné světlosti barvy. Při čemž jsme zjistili, že tmavší barvy, jako je tmavě zelená, byly často obtížně rozpoznatelné od tmavších míst v pracovním prostředí. Naopak při světlejších barvách docházelo k sjednocení více barev. To vykazovala světle zelená, která byla brána v rozsahu žluté. Během sérií testů, které zahrnovaly různé pozadí a světelné podmínky, se ukázalo, že oranžová barva vykazuje nejlepší kompromis mezi kontrastem a viditelností za různých světelných podmínek.

Přípravek posuvného držáku (příloha 05) jsme navrhli a vyrobili s ohledem na jeho požadavky: stabilitu, určení středu sledovaného bodu a snadné manipulace. Tyto požadavky byly zásadní pro zajištění vhodnosti držáku pro náš experiment. Klíčovým požadavkem na mechanismus byla schopnost spolehlivě a rychle určit střed bodu. Za tímto účelem jsme navrhli základnu ve čtvercovém tvaru, která intuitivně umožňuje snadný pohyb snímaného bodu. Jeden z rohů základny byl určen

jako referenční bod, který slouží k určení středu a při pohyby po čtvercové síti. Takto docílíme určení polohy v osách x a y. Pro určení polohy v ose z, jsme zvolili vertikální tyč kolmou k základně (obr. 2.6). Po této tyči můžeme pohybovat horizontální tyčí s přišroubovaným kontrastním bodem. Uchycení jsme zde provedli speciálně tvarovanou podložkou a maticí upravenou pro snadnou manipulaci. Tyto dva prvky vytvářejí tlakový spoj, který zajišťuje, že bod zůstane stabilně na svém místě během snímání.



Obrázek 2.6: Posuvný držák kontrastního bodu pro získávaní dat k učení neuronové sítě, který umožňuje nastavení polohy bodu v prostoru xyz.

a - kontrastní bod, b - osa z, c - šroubový spoj, d - podložka,
e - utahovací matice, f - tyč, g - základna

Tento návrh posuvného držáku umožňuje pohyb bodem ve všech třech osách. To má velkou výhodu při reprodukci snímání a získávaní dat.

2.2.4 Snímání dat

K účinnému učení a trénování neuronové sítě je důležité disponovat dostatečným množstvím dat. Nastavení našeho experimentálního prostředí, které zahrnuje prvky:

1. Kamery se stativy
2. Posuvný držák kontrastního bodu
3. Čtvercová síť s rastrem 10 mm

Na základě těchto dat se síť učí rozpoznávat vzory a získávat užitečné vlastnosti, které jsou potřebné pro plnění stanoveného úkolu. Pro zjednodušení a zefektivnění procesu získávání dat byl vytvořen program, který je popsán v příloze 06. Tento program je nástrojem pro polo-automatizaci sběru dat, urychlení celého procesu a ukládání potřebných informací. Program je složen z následujících kódů 2.2, 2.3 a 2.4.

```
camera = [1, 2]              # urceni kamer
resolution = [1920, 1080]    # nastaveni rozliseni
main_folder = "DATA"         # slozky pro ukladani
camera_folder = ["camera_1", "camera_2"]
working_space = [10, 310]    # pracovniho prostredi
# nastaveni vychozi polohy kulicky
position_x = 0
position_y = 0
position_z = 0
# nastaveni cest pro ulozeni
data_folder_1 = f"{main_folder}\{camera_folder[0]}"
data_folder_2 = f"{main_folder}\{camera_folder[1]}"
```

Kód 2.2: Nastavení potřebných proměněných pro získávání dat: volba kamer, rozlišení snímků, cesty pro ukládání, velikost pracovního prostoru.

```
# obraz (video) z kamer
cam1 = cv.VideoCapture(camera[0])
cam2 = cv.VideoCapture(camera[1])
# nastaveni pozadovaneho rozliseni
cam1.set(cv.CAP_PROP_FRAME_WIDTH, resolution[0])
cam1.set(cv.CAP_PROP_FRAME_HEIGHT, resolution[1])
cam2.set(cv.CAP_PROP_FRAME_WIDTH, resolution[0])
cam2.set(cv.CAP_PROP_FRAME_HEIGHT, resolution[1])
```

Kód 2.3: Získání obrazu z jednotlivých kamer (řádek 2,3) a nastavení požadovaného rozlišení snímků pro každou kameru.

```
if cv.waitKey(1) == ord('c'): # pri stisknuti 'c'
  position_x += working_space[0]
  if position_x == working_space[1]:
    position_y += working_space[0]
    position_x = 0
    if position_y == working_space[1]:
      position_z += working_space[0]
      position_y = 0
  # pojmenovani snimku
  snapshot_1 = os.path.join(
  f"{main_folder}\{camera_folder[0]}",
  f"cam1_{position_x}_{position_y}_{position_z}_.jpg")
  snapshot_2 = os.path.join(
  f"{main_folder}\{camera_folder[1]}",
  f"cam2_{position_x}_{position_y}_{position_z}_.jpg")
  # ulozeni snimku
  cv.imwrite(snapshot_1, frame_1)
  cv.imwrite(snapshot_2, frame_2)
```

Kód 2.4: Při stisknutí klávesy 'c' dochází připsání aktuální polohy do názvu, která se automaticky ukládá při každém stisku. Poloha je zapsána do názvu snímku, který se uloží do zvolených složek.

V kódu 2.4 je jednoduchý algoritmus, který postupně při zachycení snímku přiřazuje dané souřadnice do názvu. Tento formát nám umožňuje jednoznačnou identifikaci a klasifikovat každý snímek podle zdroje (kamery) a polohy. Tento krok je nezbytný pro efektivní trénink a analýzu neuronové sítě. Struktura názvu snímku s informacemi o poloze je následující:

```
cam1_10_30_80_.jpg
```

Prvním prvkem (cam1) je identifikátor kamery, který je důležitý pro rozdělování dat při zpracovávání. Dalším je informace o pozici v ose x (10), což v případě našeho experimentálního prostředí představuje horizontální pozici. Hloubku nám představuje osa y (30) a vertikální pozici nám udává osa z (80). Pro rozdělení názvu byl zvolen znak '_', ten je klíčový pro další zpracování snímků. Všechny informace o poloze sledovaného bodu v kartézském souřadnicovém systému jsou v jednotkách mm. Snímky získané pomocí programu nelezeme na obrázku 2.7.

Obrázek 2.7: Snímky jedné polohy posuvného držáku ze dvou kamer, při různých světelných podmínek. Výstup z programu (příloha 06) při získávaní dat.

2.3 Předzpracování dat

Po dokončení procesu snímání přecházíme ke zpracování dat pro učení neuronové sítě. Vzhledem k tomu, že obrazový pixel chápe jako jeden neuron na vstupní vrstvě. Každý snímek (obr. 2.7) obsahuje tři barevné kanály červená - zelená - modrá (barevný model RGB). Při rozlišení 1920x1080 pixelů, které odpovídá plné kvalitě, by celkový počet neuronů na vstupní vrstvě dosáhl: $1920 \cdot 1080 \cdot 3 = 6\ 220\ 800$ neuronů. Takto velký počet neuronů by představoval významný výpočetní nárok, který by se promítl na celkový čas a efektivitu učení neuronové sítě.

V rámci našeho experimentu zaměřeného na určení polohy bodu není potřeba získávat informace v celém rozsahu snímku, ale stačí nám malá část kolem něho. To nám umožní výrazně snížit počet obrazových dat pro trénování, což urychlí celý proces učení neuronové sítě.

Pro tento krok jsme napsali program (příloha 07), který detekuje sledovaný bod na základě jeho barvy a pomocí kontur určí jeho přibližný střed, kolem kterého následně vytvoří výřez. Program je složen z následujících kódů 2.5, 2.6 a 2.7.

```
for filename in os.listdir(input_dir):
    if filename.endswith(".jpg"):
        img_path = os.path.join(input_dir, filename)
        # nacteni snimku
        img = cv.imread(img_path)
        # prevedeni z RBG do HSV
        hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
        # HSV rozsah pro ornazovou barvu
        lower_color = np.array([3, 120, 150])
        upper_color = np.array([40, 255, 240])
        # vytvoreni masky
        mask = cv.inRange(hsv, lower_color, upper_color)
        # vytvoreni kontur - rohove body
        contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.
    CHAIN_APPROX_SIMPLE)
```

Kód 2.5: Vytvoření masky na základě nalezené kontury (kód 2.6), která je určena volbou rozsahu v barevném modelu HSV. Ten je nastaven na řádku 9 a 10.

```
if contours:
  # kontura s nejvetsi nalezenou plochou
  largest_contour = max(contours, key = cv.contourArea)
  rectangle = cv.minAreaRect(largest_contour)
  position, size, angle = rectangle
  x, y = position
  w, h = size

  # vyvolani funkce pro vytvoreni vyrezu
  cropped_image, x1, y1 = crop_img(img, position, [70,70], min_val,
    max_val, val)
  # zapsani informaci o poloze vyrezu do nazvu
  output_filename = os.path.join(output_dir, f"{filename}{x1}_{y1}_
    .jpg")
  cv.imwrite(output_filename, cropped_image)
```

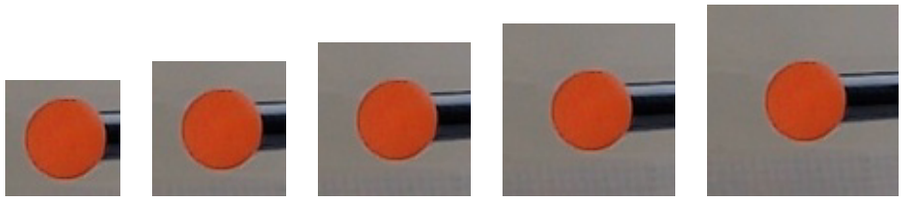
Kód 2.6: Po nalezení kontur, dochází k určení přibližného středu bodu (`position`), který je výstupem funkce `cv.minAreaRect` - kontura s nejmenší plochou. A následné uložení této polohy do názvu výřezu.

```
def crop_img(img, position, desired_size):
    x, y = position        # priblizny stred kulicky
    w, h = desired_size    # velikost vyrezu
    x1 = int(x - w / 2)    # levy roh
    y1 = int(y - h / 2)
    x2 = int(x1 + w)       # pravy roh
    y2 = int(y1 + h)
    cropped_image = img[y1:y2, x1:x2]
    return cropped_image, x1, y1
```

Kód 2.7: Funkce pro vytvoření výřezu, na základě vstupů: snímek, poloha středu, velikost výřezu. Výřez je tvořen na základě dvou bodů `[y1:y2, x1:x2]`.

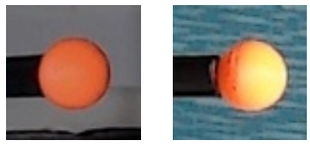
K určení optimální velikosti výřezu jsme provedli několik testů (obr. 2.8), při kterém se experimentovalo s různými hodnotami. Docházelo ke změně velikosti snímaného bodu na snímku v důsledku pohybu v ose y, která má vliv na hloubko v pracovním prostoru.

Takto jsme získali optimální velikost, při která je sledovaný bod vždy viditelný na výřezu v celém rozsahu pracovního prostoru.

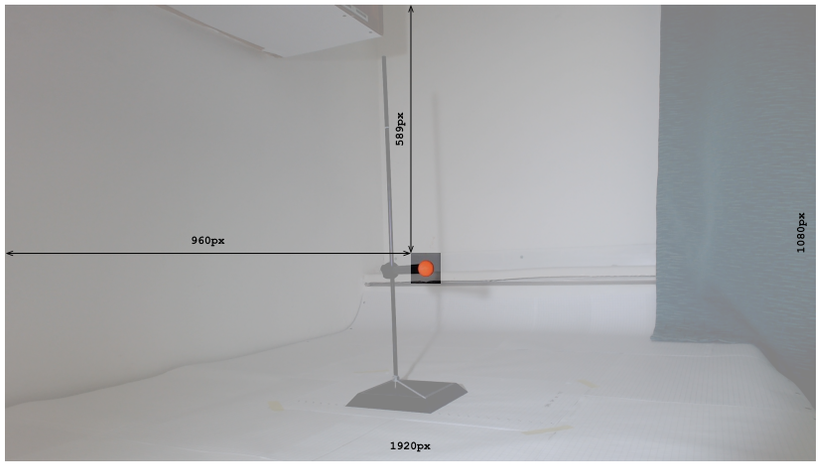


Obrázek 2.8: Porovnání různých velikostí výřezu sledovaného bodu ze snímků pro nalezení optimální velikosti. Velikost je volena v pixelech.

Optimální velikost pro náš experiment a pracovní prostředí byla velikost výřezu na 70x70 pixelů, což nám zredukovalo počet neuronů z několika milionů na pouhých 14 700. To odpovídá ke snížení dat o 99,76 %. Ze snímků (obr. 2.7) byly vytvořeny výřezy, které zle vidět na obrázku 2.9. Pro lepší představu jsme vytvořili demonstraci výřezu ze snímku na obrázku 2.10.



Obrázek 2.9: Výřezy s kličkou při různých světelných podmínkách s různě kontrastním pozadím. Na pravém lze vidět vzniklý odlesk, důsledkem směru působení světla.



Obrázek 2.10: Demonstrace principu vytvoření výřezu 70x70 pixelů z původního snímku s rozlišením 1920x1080 pixelů. Kde je vyznačená poloha levého rohu, která je uložena do názvu snímku.

Tento proces nám umožnil vytvořit optimalizovaná a konzistentní data pro trénování neuronové sítě. Získali jsme menší počet neuronů na vstupní vrstvě a tím optimálnější architekturu. A to nám umožňuje efektivní a rychlé učení.

2.3.1 Různorodost dat

Abychom rozšířili náš dataset a zabránili přeučení sítě, poupravili jsme kód 2.7. Cílem bylo vytvoření náhodného výřezu kolem přibližného středu a tím zabránit konzistentnímu umístění bodu ve výřezu. Tento krok jsme zvolili po prvních provedených testech, které vykazovaly vysokou nepřesnost sítě. Domnívali jsem se, že neuronová síť určuje polohu pouze za pomocí polohy výřezu, jelikož bod byl na všech vstupních datech vždy uprostřed výřezu. Tato metoda zabrání chybnému učení a dovolí nám tak rozšířit náš dataset.

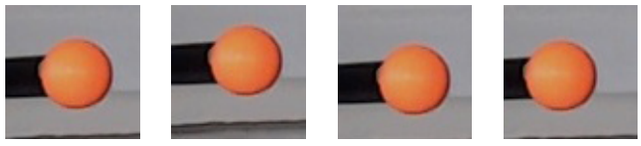
Za pomocí přidání náhodné hodnoty do polohy středu na základě kontur, jsem získali upravený kód 2.8:

```
min_val, max_val = [-1, 1]
val = 8        # velikost nahodneho posuvu

def crop_img(img, position, desired_size, min_val, max_val, val):
    x, y = position
    w, h = desired_size
    # stanoveni nahodneho posuvu v ose x a ose y
    random_x = random.uniform(min_val, max_val) * val
    random_y = random.uniform(min_val, max_val) * val
    # pridani nahodneho posuvu do tvorby vyrezu
    x1 = int(x + random_x - w / 2)
    y1 = int(y + random_y - h / 2)
    x2 = int(x1 + w)
    y2 = int(y1 + h)
    cropped_image = img[y1:y2, x1:x2]
    return cropped_image, x1, y1
```

Kód 2.8: Vytvoření náhodného výřezu využitím funkce `random`, která je dána parametry `min_val, max_val` a velikostí posuvu `val`, který jsme stanovili na 8 pixelů. Přičtením proměnných `random_x` a `random_y` dochází k posuvu výřezu vzhledem k středu bodu.

Výstupem tohoto programu (příloha 08) je několik variant výřezu, které můžeme získat z jednoho snímku. Na obrázku 2.11 je uveden příklad náhodných výřezů, první snímek je vytvořen neupraveným kódem 2.7 pro porovnání.



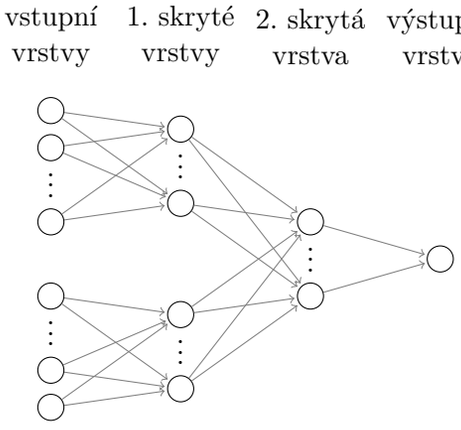
Obrázek 2.11: Porovnání náhodných výřezů z jednoho snímku vytvořené programem 08.

Kapitola 3

Trénování sítě

Z počátku není možné stanovit ideální architekturu neuronové sítě. Provedli jsme několik experimentální testů topologie sítě, kde jsme analyzovali vliv jednotlivých parametrů na efektivitu a přesnost sítě. V této kapitole společně probereme tyto parametry a nalezneme z nich tu nejvýhodnější (popř. nejlepší). Testy jsme provedli postupně od osy x, přes rovinu xy až do výsledného prostoru xyz.

První architekturu neuronové sítě jsme navrhli s tím, že pro každou kameru jsme vytvořili vstupní vrstvu, která je spojena s vlastní skrytou vrstvou. Tuto topologii jsme zvolili s cílem, aby síť nejdříve izolovaně zpracovávala data snímků z jednotlivých kamer. Od tohoto kroku očekáváme, že umožňuje předzpracování dat bez vzájemného ovlivňování. Následně po zpracování těchto dat jsme očekávali, že síť bude vyhodnocovat spojitost mezi těmito toky dat, pro přesné určení polohy bodu. Na obrázku 3.1 je naše architektura ilustrována pro lepší představu.



Obrázek 3.1: První sestavená architektura neuronové sítě pro predikci polohy kuličky na ose x. Složena ze dvou vstupních vrstev, s jednotlivými prvními skrytými vrstvami, společnou druhou skrytou a výstupní vrstvou.

Pro úspěšnost trénování naší neuronové sítě je důležité zpracování a normalizace vstupních dat. Využili jsme k tomu program pro předpracování dat, který nalezneme v příloze 07. Tento program, jak již bylo uvedeno, zpracovává obraz vytvořením výřezu kolem sledovaného bodu. A také uložením informace o poloze výřezu do názvu souboru. Abychom tyto data mohli zpracovávat, vytvořili jsme funkci (kód 3.1), která extrahuje tato data z názvu souboru, který je ve formátu:

`cam1_10_30_80_960_589_.jpg`

Poslední dva prvky jsou informací o poloze výřezu v pixelech. První informace (960 pixelů) je o horizontální poloze na snímku. Druhá informace (589 pixelů) reprezentuje vertikální polohu na snímku.

```
def load_data(folder):
    imgs = []            # obrazova data
    sqr = []             # poloha vyrezu
    coordinates = []     # souradnice kulicky

    for filename in os.listdir(folder):
        if filename.endswith('.jpg'):
            img = Image.open(os.path.join(folder, filename))
            # prevedeni obrazovych dat do rozsahu 0 az 1
            img_array = np.array(img) / 255.0
            imgs.append(img_array)
            # ziskani dat o poloze vyrezu
            sqr_1, sqr_2 = filename.split('_')[4:6]
            sqr.append([float(sqr_1), float(sqr_2)])
            # ziskani dat o souradnici v ose x
            coordinate_x = filename.split('_')[1]
            coordinates.append([float(coordinate_x)])
    return np.array(imgs), np.array(sqr), np.array(coordinates)
```

Kód 3.1: Funkce pro extrahování informací z názvu snímku, které jsou uloženy v `folder`. Pomocí funkce `split('_')` rozdělujeme název na jednotlivé prvky (informace), které jsou ve stanoveném pořadí a uloženy do seznamů: `imgs` - obrazová data, `sqr` - polohová data, `coordinates` - skutečné souřadnice sledovaného bodu.

Po zpracování dat jsme napsali část programu pro trénování naší neuronové sítě, který nalezneme v příloze 09. Jako první jsme nastavili formát vstupních dat do sítě, která rozdělena do dvou kategorií:

1. Obrazová data: Tato data obsahují informace, které jsou na snímku, v našem případě se jedná o kontrastní bod. Jejich formát jsme nastavili na `(70, 70, 3)`, což odpovídá velikosti výřezu 70x70 pixelů v barevném prostoru RBG.

2. Polohová data: Ty jsou ve formátu `(2,)`, který odpovídá horizontální a vertikální poloze výřezu na snímku.

Poté jsme začali se stavbou architektury sítě (kód 3.2), kde důležitým krokem je převést matici obrazových dat do vektoru. Poté jsme tento obrazový vektor spojili s vektorem polohových dat. Tímto jsme vytvořili vstupní vrstvu, která kombinuje obrazové a polohové informace, tvořenou 14 702 vstupními neurony (`input_merged_1` a `input_merged_2`). Dále jsme vytvořili první dvě skryté vrstvy (`dense1_img_1` a `dense1_img_2`). Tyto vrstvy zpracovávají dat samostatně před tím, než jsou spojeny do společné druhé skryté vrstvy (`dense2`). Všechny tyto skryté vrstvy obsahují 100 neuronů a síti umožňují dostatečné komplexní zpracování dat. Pro výstup naší nerunové sítě jsme určili jeden neuron.

```
# vstupni data
input_img_1 = layers.Input(shape = (70, 70, 3))
input_img_2 = layers.Input(shape = (70, 70, 3))
input_sqr_1 = layers.Input(shape = (2,))
input_sqr_2 = layers.Input(shape = (2,))
# prevedeni matice obrazu na vektor
flatten_img_1 = layers.Flatten()(input_img_1)
flatten_img_2 = layers.Flatten()(input_img_2)
# spojeni vstupu do jedne vstupni vrstvy A
input_merged_1 = layers.Concatenate()([flatten_img_1, input_sqr_1])
# skryta vrstva vstupni vrstvy A
dense1_img_1 = layers.Dense(100, activation = 'relu')(
    input_merged_1)
# spojeni vstupu do jedne vstupni vrstvy B
input_merged_2 = layers.Concatenate()([flatten_img_2, input_sqr_2])
# skryta vrstva vstupni vrstvy B
dense1_img_2 = layers.Dense(100, activation = 'relu')(
    input_merged_2)
# spojeni prvnich skrytych vrstev
concatenated = layers.Concatenate()([dense1_img_1, dense1_img_2])
# druha skryt vrstva
dense2 = layers.Dense(100, activation = 'relu')(concatenated)
# vystupni vrstva
output_layer = layers.Dense(1)(dense2)
```

Kód 3.2: Architektura neuronové sítě v prostředí Python, která odpovídá topologii na obrázku 3.1. První krokem je stanovení formátu vstupních dat pomocí `shape`. Dále převedení obrazové matice na vektor pomocí `layer.Flatten()` a spojení s vektorem polohy výřezu využitím `layers.Concatenate()`. Nakonec samotné definování vrstev `layers.Dense()`.

Při trénování neuronové sítě dochází k vývoji takzvaného modelu, který je možné ukládat a používat k predikci polohy v našem experimentu. Model jsme vytvořili (kód 3.3) na základě topologie naší sítě, kde jsme definovali vstupní a výstupní vrstvy.

```
model = models.Model(
  inputs = [input_img_1, input_img_2, # vstupni vrstvy
  input_sqr_1, input_sqr_2],
  outputs = output_layer)              # vystupni vrstva
```

Kód 3.3: Vytvoření modelu typu `models.Model` a definování vstupních vrstev (`inputs`) a výstupní vrstvy (`ouputs`).

Pro náš model jsme vybrali optimalizátor typu Adam, který je velmi užívaný při trénovaní. Jako ztrátovou funkci jsme zvolili průměrnou kvadratickou chybu, která se hodí pro predikci hodnot. K hodnocení výkonnosti modelu jsme stanovili metriku, která měří přesnost při predikci polohy bodu (kód 3.4).

```
model.compile(
  optimizer = 'adam',              # volba algoritmu
  loss = 'mean_squared_error',     # volba ztratove funkce
  metrics = ['accuracy'])          # volba hodnoceni
```

Kód 3.4: Kompilace modelu, při nastavení optimalizačního algoritmu (`optimizer`), ztrátové funkce (`loss`) a hodnotícího parametru výkonnosti sítě (`metrics`).

Po konfiguraci všech potřebných funkcí jsme definovali potřebná data společně s parametry pro trénování sítě (kód 3.5).

```
model.fit(
  # treninkova data
  [img_1_train, img_2_train, sqr_1_train, sqr_2_train],
  # cilova data
  coords_train,
  # parametry treninku
  epochs = 500, batch_size = 32,
  # validacni data
  validation_data = ([img_1_test, img_2_test, sqr_1_test,
  sqr_2_test],
  coords_test))
```

Kód 3.5: Definování trénovacího datasetu, správných souřadnic a testovacího datasetu (`validation_data`) pro model.

K predikci polohy sledovaného bodu jsme napsali funkci (kód 3.6), kde jsme načetli náš model a data, pomocí kterých chceme predikovat polohu.

```
def predict_coordinate(model, img_1, img_2, sqr_1, sqr_2):
    prediction = model.predict([img_1, img_2, sqr_1, sqr_2])
    return prediction
```

Kód 3.6: Funkce pro predikci polohy na základě zvoleného modelu a vstupních dat. Funkce vrací predikované souřadnice polohy sledovaného bodu.

Pro zajištění vysoké přesnosti a efektivity neuronové sítě při určování polohy bodu, jsme experimentovali s různými architekturami a nastaveními sítě. To nám umožnilo zjistit, jaké parametry mají příznivý, či škodlivý vliv pro náš experiment. V této kapitole si ukážeme, jak jsme postupovali při optimalizaci naší topologie. Kapitola je rozdělena na tři hlavní sekce, které postupně reprezentují kartézský souřadnicový systém. Postupovali jsem od jednoduché osy x až po prostor xyz, ve kterém jsme porovnali výsledky s jiným způsobem určení polohy.

3.1 Osa x

Abychom lépe porozuměli fungování neuronové sítě a sestavování architektury, začali jsme experimentovat se sítí zaměřenou pouze na osu x. Cílem tohoto postupu bylo zjistit, zda nám síť dokáže vracet predikovaná souřadnice bodu v mm. Tento krok byl klíčový k porozumění, jak síť se vstupními daty pracuje a zda dokáže určit souřadnice sledovaného bodu.

Počet neuronů v architektuře při testování je uveden v tabulce 3.1. Program, který jsme použili nalezneme v příloze 09.

Tabulka 3.1: Nastavení architektury sítě pro osu x.

	osa x
Počet neuronů v první skryté vrstvě	100
Počet neuronů v druhé skryté vrstvě	100
Počet neuronů ve výstupní vrstvě	1

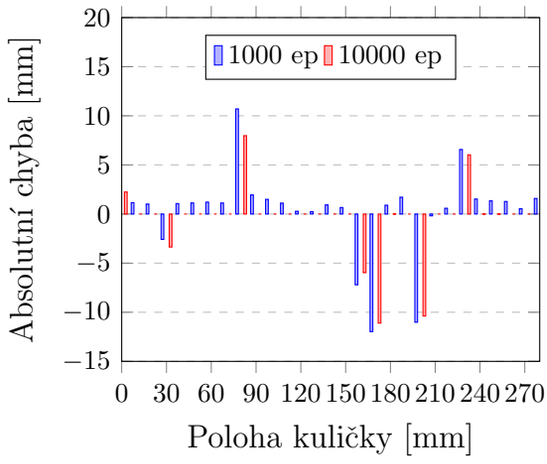
Pro první test jsme nasbírali 29 obrazových dat, která odpovídají poloze bodu v ose x od 0 až 280 mm při rozpětí 10 mm. Tato data jsme následně pomocí příkazu `train_test_split()` rozdělili na dva datasety: trénovací a testovací. Trénovací dataset obsahoval 80 % (přibližně 23 výřezů) z celkového počtu dat, zbylých 20 % (přibližně 6 výřezů) byly vyhrazeny pro testovací dataset. K předzpracování dat jsme použili program (příloha 07), který vytváří výřez kolem středu sledovaného bodu.

Trénovaní modelu jsem provedli pro dva počty epoch 1 000 a 10 000, tím jsme získali dva modely, kterými jsme následně predikovali polohu na dvou případech:

- Data pro predikci byla stejná, jako data na kterých se síť učila.
- Data při kterých byla poloha bodu odlišná od trénovacích dat. Polohu bodu v ose x jsme zvolili od 5 až 275 mm při rozpětí 10 mm.

K predikování souřadnice, jsme vytvořili program (příloha 10), který implementuje funkci `model.predict()` (kód 3.6). Po provedení predikce jsme výsledky zaznamenali a zjistili jsme, že síť nám vrací hodnoty v milimetrech. Toto zjištění bylo pro nás klíčové, jelikož eliminuje jakékoliv potřebné úpravy formátu vstupních dat. Následně jsme provedli výpočet absolutní chyby mezi skutečnou a předkovanou souřadnicí sledovaného bodu. To nám umožnilo posoudit přesnost naší sítě.

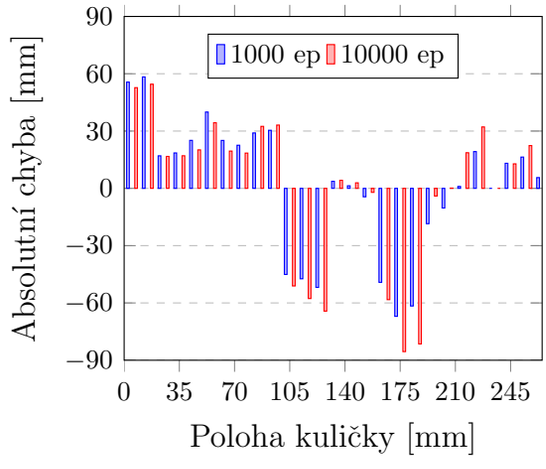
Absolutní chyba při prvním případu, kde data byla stejná jako trénovací, je prezentována v grafu (obr. 3.2). V tomto grafu jsou srovnány výsledky dvou modelů, které byly trénovány při 1 000 a 10 000 epochách. Volbou vysokého čísla v případě druhého modelu jsme chtěli zjistit, jak na to síť bude reagovat. A zda se budou výrazně měnit výsledky oproti nižšímu počtu epoch.



Obrázek 3.2: Graf absolutní chyby (predikovaná - skutečná) polohy bodu v ose x, při 1 000 a 10 000 epochách modelu. Na čtvercové síti od 0 mm do 270 mm při rastru 10 mm.

Predikované souřadnice se obvykle pohybovaly při absolutní chybě 0,1 až 2 mm, ale byly zaznamenány i chyby nabírající hodnoty až 10 mm do kladných i záporných hodnot. Tyto větší chyby zřetelně vidíme v grafu, kde se výrazně odlišují od ostatních hodnot. Předpokládáme, že tyto chyby mohou být způsobené rozdělením dat do trénovacího a testovacího datasetu. Kde data, která nebyla v trénovacím datasetu vykazují zvýšené chyby. Tento jev je patrný i v případě modelu, který prošel 10 000 epochami. Přestože v absolutní chyba se v tomto případě klesna na 0,001 mm. Zdá se, že větší počet epoch má vliv na přesnost neuronové sítě.

Stejným postupem jsme provedli test v druhém případě. Kde data nabyla stejná s trénovacími a lišila se od nich v poloze bodu. V tomto případě byl sledovaný bod posunut od 5 mm po 10 mm do 265 mm. Tímto jsme pozorovali, jak si model dokáže poradit s různorodými daty. Absolutní chyby jsou prezentovány na grafu (obr. 3.3).



Obrázek 3.3: Graf absolutní chyby (predikovaná - skutečná) polohy bodu v ose x, při 1 000 a 10 000 epochách modelu. Na čtvercové síti od 5 mm do 265 mm při rastru 10 mm.

Zde jsem zaznamenali výrazný nárůst chyb a zvýšení počtu záporných hodnot. Předpokládáme, že tento jev souvisí s nedostatečným počtem trénovacích dat pro neuronovou síť. V důsledku toho není síť schopna efektivně vytvořit funkci pro určení polohy.

Tato analýza výsledku naznačuje, že pro zlepšení přesnosti sítě budeme potřebovat rozšířit naše data. Tím docílíme lepšího základu pro trénování a zajištění lepší adaptace neuronové sítě na různorodá dat.

3.2 Rovina xy

Po úspěšném zjištění, že nerunová síť dokáže s určitou přesností predikovat souřadnice, jsme se rozšířili naši práci do roviny xy. Pro tento účel jsme nasbírali nová data, která představují rovinu o rozměrech 270x270 mm. K zpracování dat jsme opět využili náš program (příloha 07), kterým jsme zpracovali 784 obrazových dat z jedné kamery. Pro rozdělení dat do trénovacího a testovacího datasetu jsme využili funkci `model.predict()` (kód 3.6), přičemž jsme zvolili poměr rozdělení 8:2.

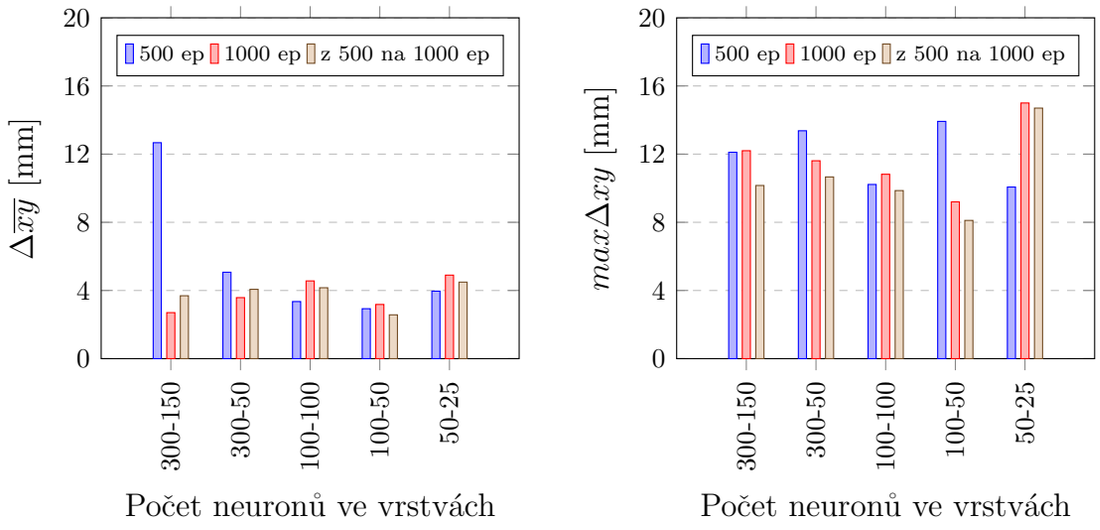
K otestovaní přesnosti našeho modelu bylo vybráno 27 obrazových dat, která se lišila od trénovacích. Tato testovací data reprezentovala uhlopříčku naší roviny. Zvolili jsme tento přístup pro rychlé provedení testů modelu a ověření přesnosti napříč rovinou. Nový program pro trénování sítě nalezneme v příloze 11 a k testování modelu v příloze 12.

3.2.1 Porovnání aktivačních funkcí

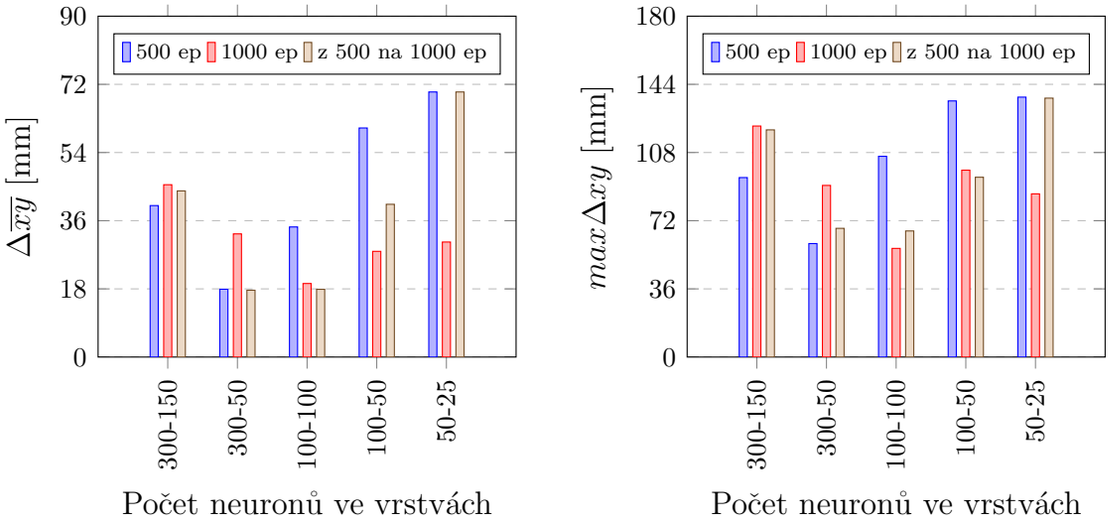
Naše první testy jsme zaměřili na volbu aktivační funkce. Pro experimenty jsme vybrali dvě nejpoužívanější funkce: ReLU a sigmoid. O nichž jsme se zmiňovali v první kapitole. Během testování jsme porovnávali průměrnou absolutní a maximální chybu mezi skutečnou a predikovanou polohou bodu. Testy jsem provedli na pěti různých topologiích sítě, které se lišili počtem neuronů ve skrytých vrstvách. Modely jsme trénovali při 500 a 1 000 epochách. Zkusili jsme i doučení již natrénovaného modelu z 500 na 1 000 epoch. Chtěli jsme zjistit chování aktivačních funkcí při změnách topologie a následně je porovnat mezi s sebou.

Z našich výsledků vyplývá, že funkce ReLU dosahuje nejvyšší přesnosti u topologie sítě $100 - 50$. Jak můžeme vidět na grafu (obr. 3.4), k zlepšení predikce došlo při doučení modelu. Tento pozitivní trend je zvlášť patrný u maximální absolutní chyby, kde došlo ke snížení o 5.8 mm. Konečná chyba modelu $max\Delta xy = 8{,}11$ mm, zatímco $\Delta\overline{xy}$ dosahuje hodnoty 2,57 mm.

Aktivační funkce při našich testech vykazuje výrazně vyšší hodnoty, které přesahují až 100 mm. Na grafu (obr. 3.5) vidíme určité změny topologie vedou ke zlepšení predikce. Nejnižší hodnota průměrné absolutní chyby $\Delta\overline{xy}$, dosahuje 17,64 mm při $100 - 100$. Maximální absolutní chyba $max\Delta xy$ dosáhla jedné z nižších hodnot a to 57,3 mm.



Obrázek 3.4: Graf vlivu aktivační funkce ReLU při různých topologií sítě, sledování průměrné absolutní chyby $\Delta\overline{xy}$ a maximální absolutní chyby $max\Delta xy$ predikce polohy. Po 500, 1 000 a z 500 na 1 000 epoch.



Obrázek 3.5: Graf vlivu aktivační funkce sigmoid při různých topologií sítě, sledování průměrné absolutní chyby $\Delta\overline{xy}$ a maximální absolutní chyby $max\Delta xy$ predikce polohy. Po 500, 1 000 a z 500 na 1 000 epoch.

K dalším experimentům pro zjištění ideální architektury sítě jsme z těchto testů zvolili následovně: funkci ReLU při topologii $300-150$, $100-50$ a sigmoid při $300-150$, $300-50$.

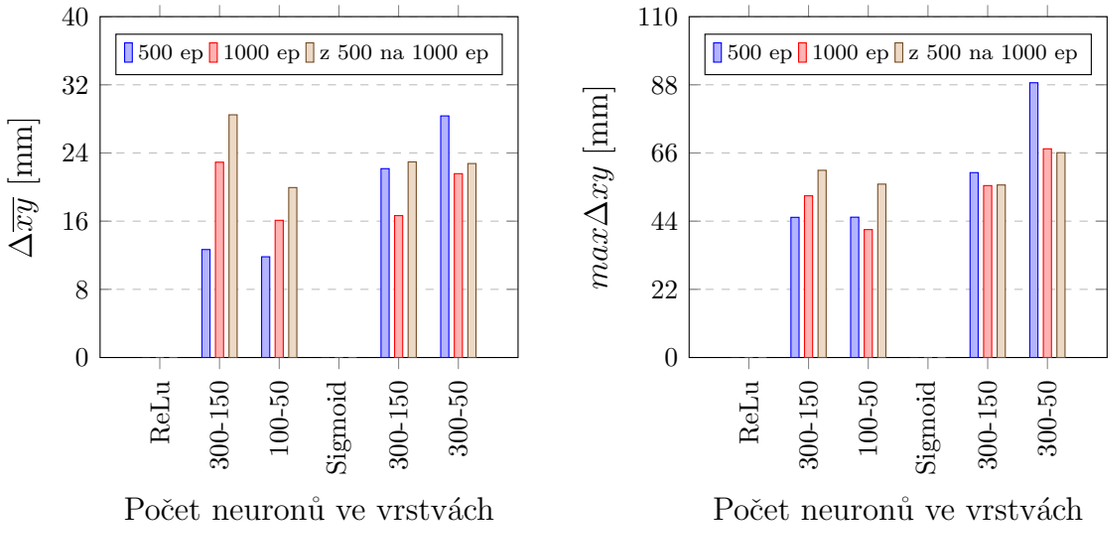
3.2.2 Vliv regulace

V našem dalším experimentu jsme implementovali regulaci vrstev. Použili jsme regulace dropout, L2 o nichž jsme se zmiňovali v kapitole 1. Pro první test jsme zvolili často používané hodnoty těchto regulací, tedy dropout rate = 0,5 a regulaci L2 = 0,001. Způsob přidání regulace do architektury sítě můžeme vidět na ukázce kódu 3.7, kde jsou přidané hodnoty, které jsme použili. V příloze 13 nalezneme program neuronové sítě, které implementuje regulace.

```
dense1_img1 = layers.Dense(num_neuron_first,
  activation = 'relu',
  kernel_regularizer = regularizers.l2(0.001))(input_merged1)
dropout1_img1 = layers.Dropout(0.5)(dense1_img1)
```

Kód 3.7: Přidání regulace dropout (`layers.Dropout()`) a L2 (`regularizers.l2()`) do vrstvy neuronové sítě.

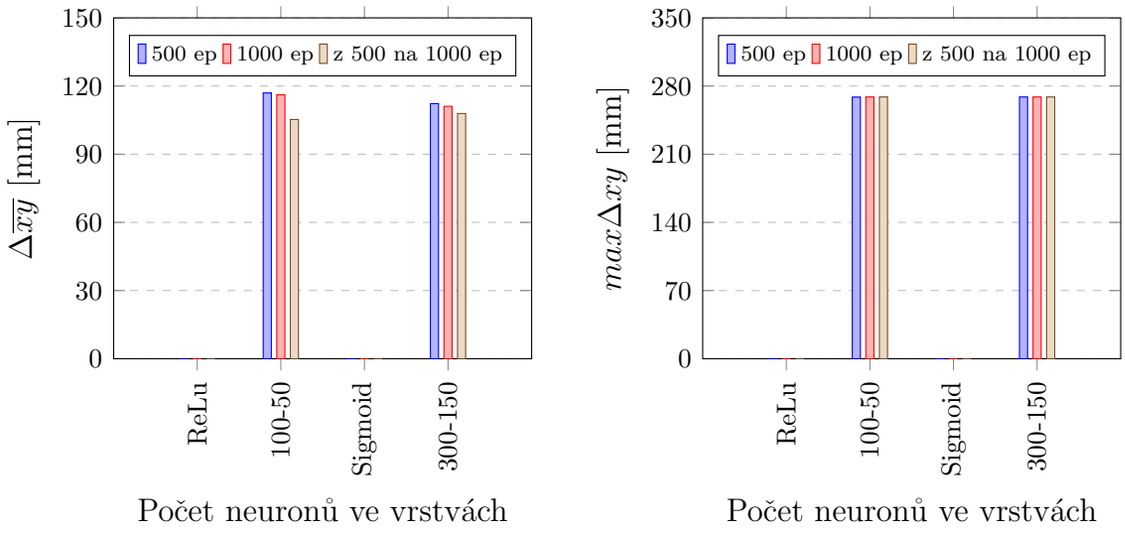
Na grafu uvedeném na obrázku 3.6 je patrné, že došlo ke snížení průměrné absolutní chyby $\Delta\overline{xy}$ v případě aktivační funkce sigmoid. Tento pokles je zaznamenám při topologii sítě $300-150$, kde se chyba z původních 45,5 mm snížila 16,65 mm při 1 000 epochách. Naopak u funkce ReLU jsme zaznamenali výrazné zhoršení přesnosti predikce. Kde $300-150$ vzrostla z hodnot pod 4 mm na více než 10 mm. Zpozorovali jsme trend snižující přesnost s navýšením počtu epoch, což poukazuje na potenciální přetrénování modelu.



Obrázek 3.6: Graf vlivu regulace dropout = 0,5 a L2 = 0,001 na $\Delta\overline{xy}$ a $max\Delta xy$ při 500, 1 000 a 500 na 1 000 epoch.

Během analýzy dat jsme zjistili potenciální důvod přeučení neuronové sítě. Všimli jsme si, že na vstupních datech je sledovaný bod vždy umístěn na středu výřezu. To nás vedlo k předpokladu, že síť spoléhá pouze na tuto uměle stabilní polohu. A nedochází k adekvátnímu zpracování obrazových dat sítí, které se nijak výrazně neliší. Abychom zabránili tomuto jevu, napsali jsme program (příloha 08), o kterém jsme se zmiňovali v kapitole 2. Tento program vytvoří výřez, ve kterém je bod náhodně umístěn. Dále jsme upravili metodu rozdělení dat za pomocí náhodného výřezu v poměru 3:1. Nově jsme získali tři obrazová data k jedné poloze bodu, tím jsme rozšířili počet trénovacích na 2 339 dat, počet testovacích na 778 dat.

Pro test při navýšení vstupních dat jsem zvolili dvě topologie z předchozího experimentu: $100-50$ při ReLU a $300-150$ při funkci sigmoid. Výsledky jsou zobrazeny v grafu na obrázku 3.7, které nám ukázaly, že maximální absolutní chyba se nemění a drží si hodnotu 268 mm. Na druhou stranu průměrná absolutní chyba vykazovala v závislosti na počtu epoch zlepšení přesnosti predikce. Avšak její hodnota vzrostla mnohonásobně, což jsme očekávali navýšením dat při zachování topologie.



Obrázek 3.7: Graf vlivu regulace dropout = 0,5 a L2 = 0,001 na $\Delta\overline{xy}$ a $max\Delta xy$ při navýšení trénovacích dat na celkový počet 2 339 dat. Po 500, 1 000 a z 500 na 1 000 epoch.

S nárůstem vstupních dat jsme zaznamenali opakovaný výskyt chyby týkající se velikosti výřezu. Důvodem byli kontury, které jsou základem pro vytvoření výřezu. Abychom chyby efektivně odstranili, napsali jsme program (příloha 14), který odhalí špatnou velikost a odstraní danou dvojici obrazových dat. Konzistentnost velikosti je důležitá, jelikož naše síť není tvořena pro různé velikosti vstupních dat.

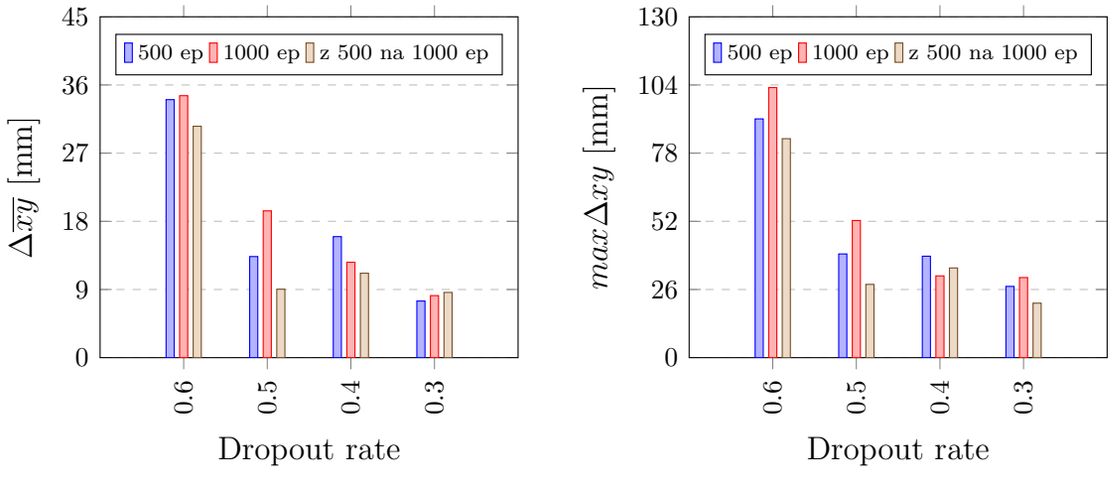
Během vývoje programu pro kontrolu velikosti jsme čelili otázce, jak zajistit, aby neuronová síť správně párovala obrazová data, která jsou důležitá pro přesnou predikci polohy. Vzhledem k nejistotě, jakým způsobem jsou data vzájemně párována, jsme vytvořili funkci pro párování pomocí ID. Toto unikátní ID je uloženo v názvu každého snímku a umožňuje nám správné párování fotek. Princip této funkce je zobrazen v ukázce kódu 3.8.

```
for filename1 in os.listdir(folder1):
  for filename2 in os.listdir(folder2):
    # parovnani podle ID: cam_1.1_x_y_z_
    if filename1.split('_')[1:5] == filename2.split('_')[1:5]:
      paired_test_images.append((os.path.join(folder1, filename1),
   os.path.join(folder2, filename2)))
                    break
```

Kód 3.8: Párování obrazových dat na základně unikátního ID uloženého v názvu snímku. Pomocí `.split('_')[1:5]` získáme ID: cam_1.0_x_y_z. Spárované obrazová data vložíme do seznamu `paired_test_images[]`.

Po implementování této funkce jsme přepsali program naší neuronové sítě (příloha 15). S tím jsme upravili i program pro testování modelu (příloha 16).

Provedli jsme experiment, který nám ukázal vliv velikosti regulace dropout. Architektura sítě byla $100-50$ s aktivační funkcí ReLU. Výsledek tohoto testu je zobrazen v grafu závislosti chyby na dropout rate při L2 = 0,01 (obr. 3.8).



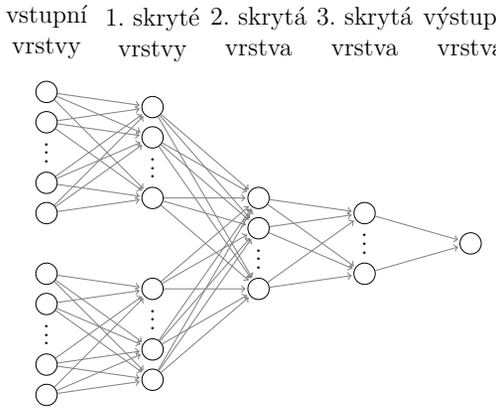
Obrázek 3.8: Graf vlivu dropout při L2 = 0,01 a aktivační funkce ReLU, při počtu 2 339 trénovacích dat. Po 500, 1 000 a z 500 na 1 000 epoch.

Z analyzovaných výsledků vylívá, že velikost regulace dropout má výrazný vliv na přesnost predikce polohy. Zjistili jsme, že nižší hodnota dropoutu vede k vyšší přesnosti neuronové sítě. Konkrétně rozdílem mezi hodnotami dropout 0,6 a 0,3 je 26,5 mm, což představuje zlepšení o 78 %. Přestože hodnota 0,3 vykazovala nejlepší výsledky, rozhodli jsme se použít hodnotu 0,4 pro naše další testy. Učinili jsem tak, abych zabránili možnému budoucímu přeučení modelu.

3.2.3 Přidání skryté vrstvy

Dalším způsobem zvýšení přesnosti naší neuronové sítě, který jsme zvolili, je přidání třetí skryté vrstvy. Abychom ověřili vliv počtu neuronů v jednotlivých vrstvách, provedli jsme řadu testů pro každou vrstvu. Hlavním kritériem hodnocení byla změna hodnoty průměrné absolutní chyby $\Delta\overline{xy}$ a maximální chyby $\max \Delta xy$.

Pro testy jsme nejprve upravili architekturu sítě (obr. 3.9) přidáním další skryté vrstvy. Přidáním této vrstvy jsme vytvořili nový program, který nalezneme v příloze 17. Program pro trénování modelu s novou topologií sítě nalezneme v příloze 18. Zde jsme se v naší práci přesunuli na výpočetní server katedry KMP. Při instalaci potřebných knihoven na server byly nainstalovány nové verze knihoven TensorFlow a Keras, které jsou uvedeny v tabulce 2.1.



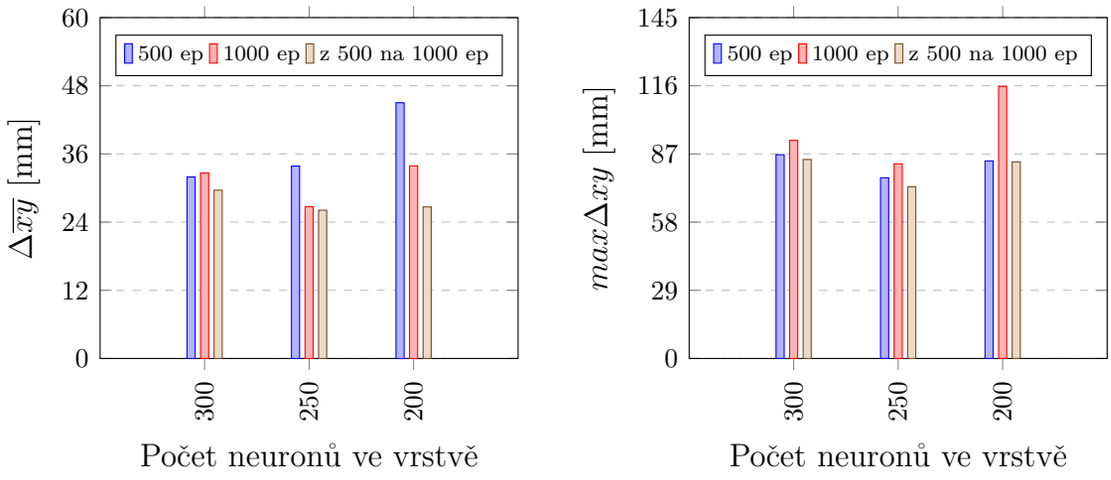
Obrázek 3.9: Nová architektura neuronové sítě s přidanou novou společnou skrytou vrstvou - 3. skrytá vrstva.

Provedli jsme řadu testů, při nichž jsme topologie sítě volili dle tabulky 3.2. Výsledky jsme zaznamenali a vynesli do grafů. Pro každou vrstvu jsme porovnali $\Delta\overline{xy}$ a $\max \Delta xy$ získaných hodnot.

Tabulka 3.2: Použitá topologie sítě při testech.

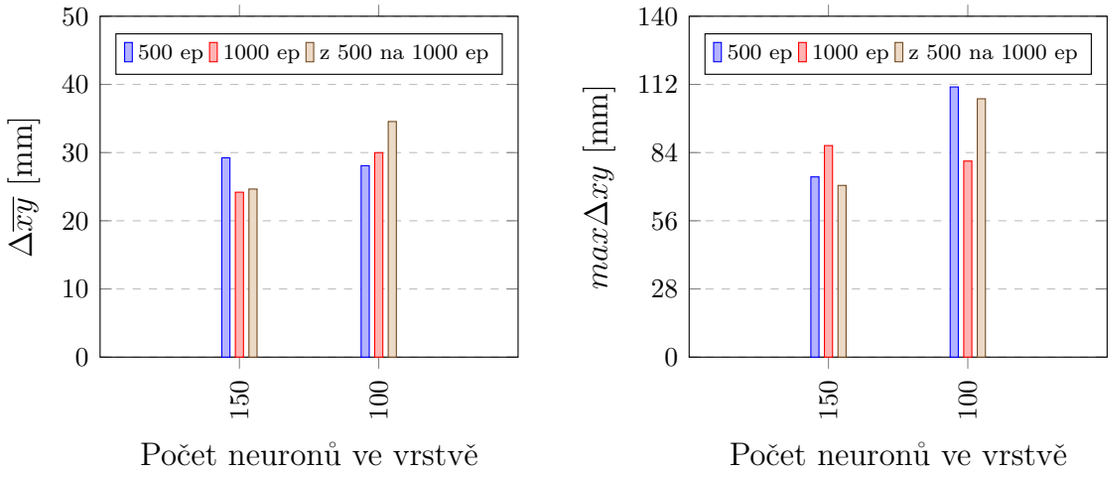
Topologie sítě			
test	1. vrstvy	2. vrstva	3. vrstva
1	-	200	100
2	400	-	100
3	400	200	-
Aktivační funkce		ReLU	
Dropout		0,4	
L2		0,01	

Analýzou testů první skryté vrstvy jsme zjistili, že má výrazný vliv na časovou náročnost trénování sítě. Vyšším počtem neuronů vzrůstala doba, po kterou probíhalo trénování. Nejlepší výsledky vykazoval počet 200 neuronů. Dochází zde k výraznému zlepšení při počtu epoch. Tento trend vidíme při doučení modelu z 500 na 1 000 epoch, kde dochází k zlepšení o 41 % v případě $\Delta\overline{xy}$ (obr. 3.10).



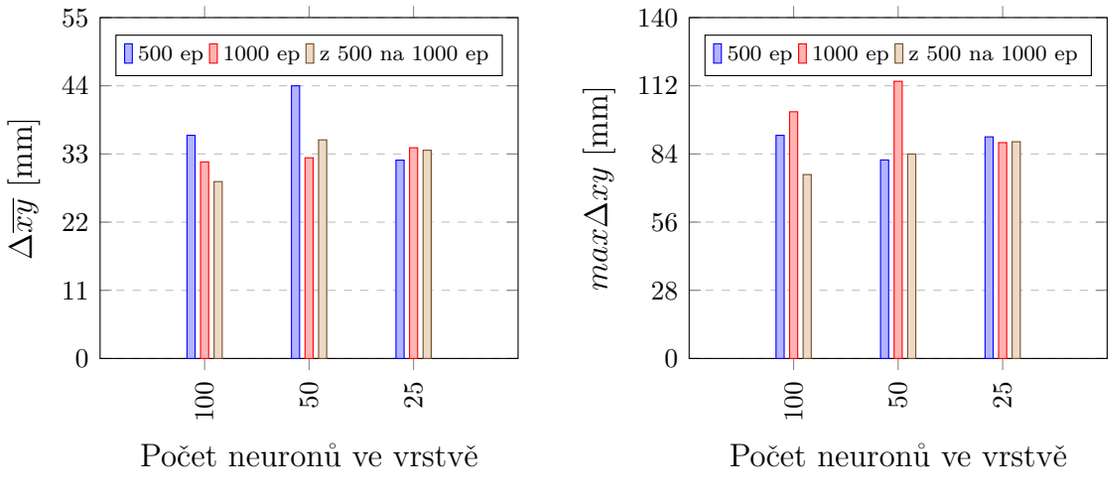
Obrázek 3.10: Graf vlivu změny počtu neuronů v první skryté vrstvě na přesnost predikce polohy při topologii: změna−200 − 100.

Změnou počtu neuronů v druhé skryté vrstvě jsme identifikovali trend, kdy snižování počtu neuronů vedlo ke zvýšení nepřesnosti predikce. Konkrétně při počtu 100 neuronů nedocházelo ke zlepšení výsledků. Naopak došlo k nárůstu průměrné absolutní chyby $\Delta\overline{xy}$. S rostoucím počtem epoch se chyba zvětšovala. Po 500 epochách byla chyba 28,08 mm, po 1000 epochách 29,99 mm a nárůst chyby při doučení modelu z 500 na 1000 epoch činil 34,56 mm. Při srovnávání výsledku s první vrstvou (obr. 3.10) jsme zaznamenali částečné zlepšení predikce.



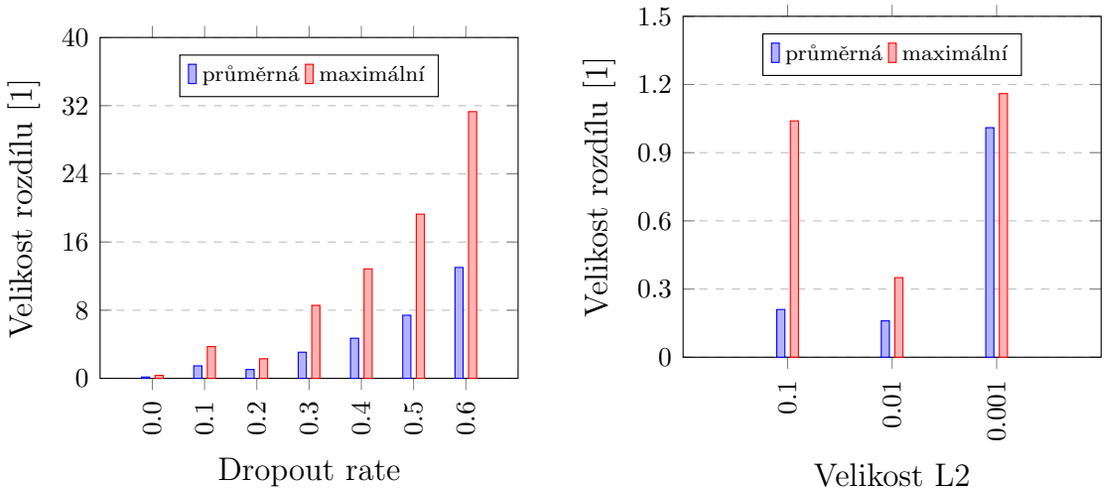
Obrázek 3.11: Graf vlivu změny počtu neuronů v druhé skryté vrstvě na přesnost predikce polohy při topologii: 400−změna−100.

Poslední sérii testů jsme zaměřili na změnu počtu neuronů ve třetí (poslední) skryté vrstvě (obr. 3.12). Zde se ukázalo, že počet 100 neuronů vedl ke snižování průměrné absolutní chyby $\Delta\overline{xy}$ s rostoucím počtem epoch. Konkrétně rozdíl $\Delta\overline{xy}$ při 500 epochách a doučení z 500 na 1000 epoch činil 7,47 mm. S tímto počtem jsme zaznamenali také snížení maximální absolutní chyby $max\Delta xy$, která se snížila o 25,79 mm.



Obrázek 3.12: Graf vlivu změny počtu neuronů v třetí skryté vrstvě na přesnost predikce polohy při topologii: 400 − 200−změna.

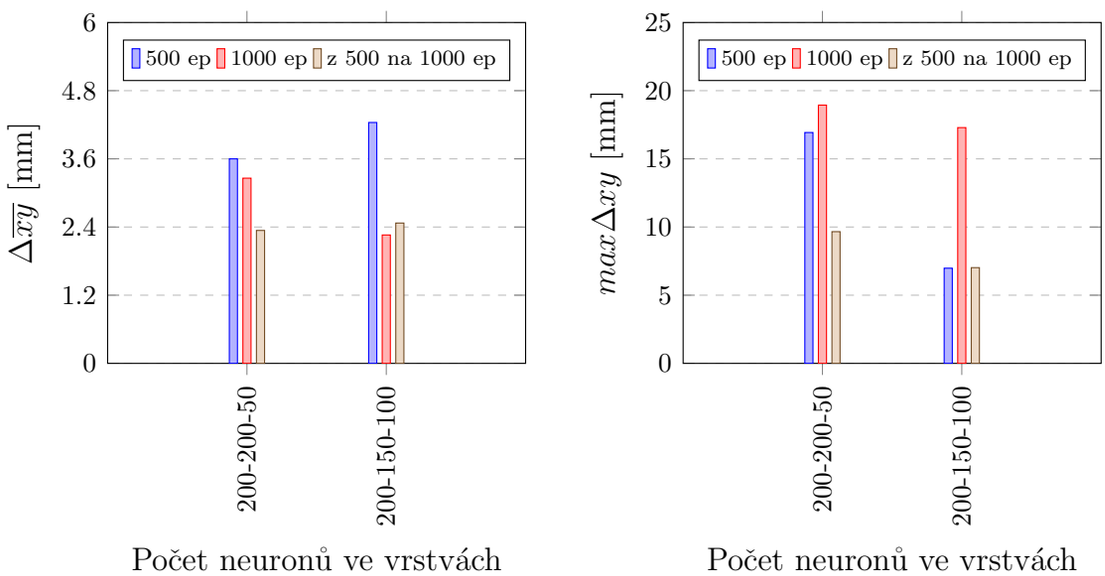
Při diskutování výsledků předešlých testů, které jsme zaměřili na změny počtu neuronů ve skrytých vrstvách, jsem došli k názoru, že chyby predikce jsou příliš velké. Abychom mohli tento způsob určování polohy v prostoru prakticky využívat v různých oblastech ve strojírenství, je potřeba, aby byly chyby redukovány na úroveň maximálně 2 mm. Z tohoto důvodu jsme se rozhodli vytvořit jednoduchou neuronovou síť, která simuluje sčítací funkci se dvěma vstupy. Tento model nám umožnil zaměřit se na testování a pochopení vlivu regulace dropout na výstupní přesnost sítě. Vytvořili jsme jednoduché datasety s náhodnými čísly pro trénování a testování modelu. Naše výsledky testů jsme uvedli do grafu (obr. 3.13), který porovnává velikost rozdílu výstupů mezi skutečným a predikovaným součtem vstupů.



Obrázek 3.13: Graf vlivu regulace dropout a L2 na přesnost výstupu modelu, který simuluje sčítací funkci se dvěma vstupy. Zobrazení velikosti rozdílu mezi skutečným a predikovaným součtem vstupů.

Z našeho výzkumu vyplývá, že dropout má zásadní vliv na přesnost výstupu z modelu. S rostoucí hodnotou dropout ratu dochází k vyššímu rozdílu mezi skutečnou a predikovanou hodnotou. Naopak při nižším dropout ratu jsme zpozorovali zlepšení výstupů, jak v případě průměrném rozdílu, tak maximálním rozdílu. Konkrétně při hodnotě 0,0 dropoutu byl průměrný rozdíl 0,16 a maximální 0,35. Na základě těchto výsledků jsme se rozhodli, že pro naši neuronovou síť k určování polohy bodu, regulaci dropout eliminovat, abychom dosáhli vyšší přesnosti. Dále jsme také otestovali vliv L2 regulace a zjistili jsme, že optimální velikost regulace je 0,01. Zde byl průměrný rozdíl roven 0,16 a maximální rozdíl roven 0,35 mezi mezi skutečnou a predikovanou hodnotou.

Pro další testy jsme zvolili topologii sítě na základě předchozích experimentů. Vybrali jsem topologii $200 - 200 - 50$ a $200 - 150 - 100$ s aktivační funkcí ReLU. Regulaci pro naši síť jsme zvolili pouze L2 s velikostí 0,01. Zjistili jsme, že při eliminaci dropout náš model vykazuje průměrnou absolutní chybu pod 4,5 mm a maximální absolutní chybu pod 20 mm. Z výsledků jsme zjistili, že naše volba topologií vykazuje zlepšení přesnosti predikce, zejména $200 - 150 - 100$. Kde nejnižší $\Delta\overline{xy}$ byla 2,26 mm a $max\Delta xy$ 6,98 mm při 1000 epochách.



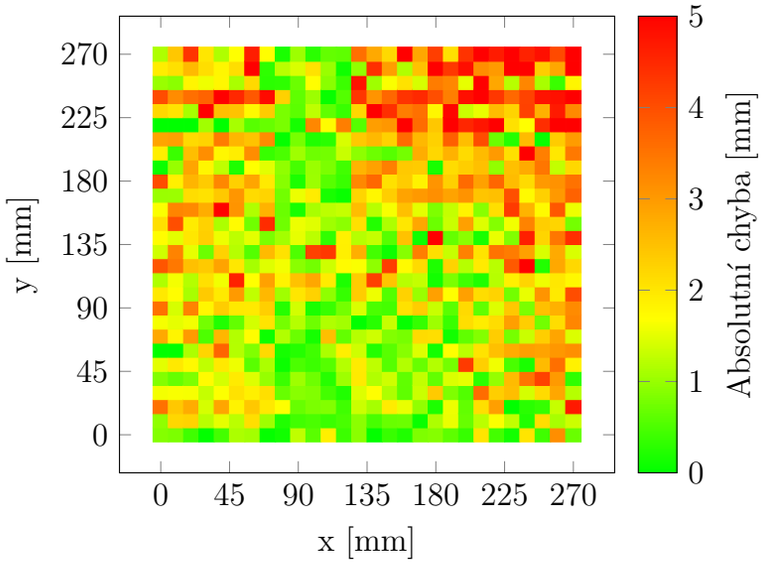
Obrázek 3.14: Graf dvou testů topologie při eliminaci regulace dropout, po 500, 1 000 a z 500 na 1 000 epoch.

Abychom získali podrobnější přehled o přesnosti naší neuronové sítě, provedli jsme analýzu chyb v celé rovině xy. Tento proces nám umožnil získat absolutní chybu mezi skutečnou a predikovanou polohou v celém rozsahu roviny. Tyto chyby jsme následně kvantifikovali a vyjádřili procentuálně v rozsahu po 1 mm při dvou topologiích sítě. Výsledky této analýzy jsme uvedli v tabulce 3.3. Z analýzy vyplývá, že většina větších chyb se vyskytuje převážně na ose y (hloubka). Zde jsme zaznamenali, že přesnost pod 1 mm nepřesahuje 50 %. Na ose x jsme naopak dosáhli lepší přesnosti, kde bylo více než 60 % chyb menších než 1 mm v rozsahu celé roviny.

Tabulka 3.3: Procentuální četnost absolutní chybu v osách x a y v rovině, při rozsahu od 0 mm nad 5 mm po 1 mm, o topologii sítě $200-200-50$ a $200-150-100$ po 1500 epochách.

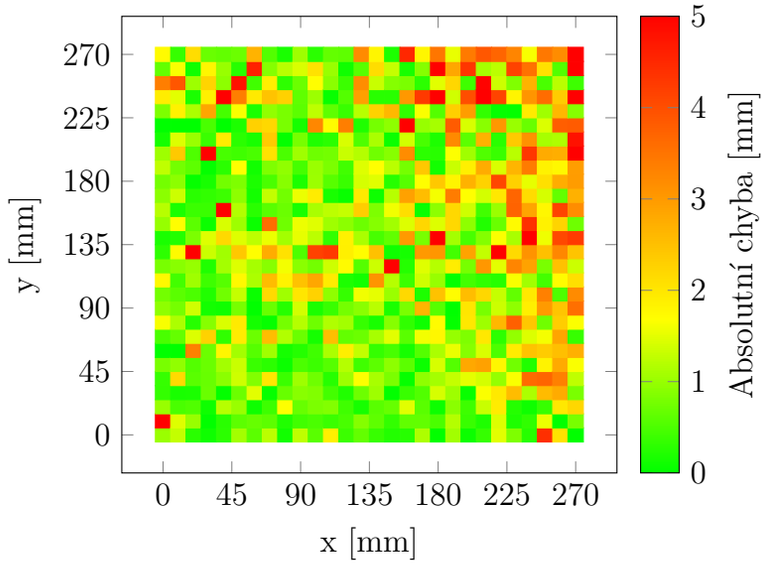
osa	pod 1 mm	1 až 2 mm	2 až 3 mm	3 až 4 mm	4 až 5 mm	nad 5 mm
četnost absolutní chyby při topologii $200-200-50$ [%]						
x	64,8	28	4,6	1,9	0,4	0,3
y	46,9	30,3	12,3	5,3	3,1	2,1
četnost absolutní chyby při topologii $200-150-100$ [%]						
x	74,3	19,8	4,4	0,6	0,5	0,4
y	27,4	30	24,1	9,1	7,2	3,1

Pro lepší pochopení přesnosti neuronové sítě v ose y jsme vytvořili grafy (obr. 3.16, 3.15), které zobrazují absolutní chybu v rozsahu celé roviny. Z analýzy grafů jsme zjistili, že velikost chyby v ose y (hloubka) roste se vzdáleností sledovaného bodu od kamer, které jsou rovnoběžné s osou x.



Obrázek 3.15: Graf absolutní chyby osy y v rovině při modelu sítě $200-150-100$, testovací data v rozsahu celé roviny xy 270x270 mm při 1500 epochách.

Při použití topologie $200-150-100$ (obr. 3.15) jsme zaznamenali vyšší výskyt chyb v ose y. Tyto chyby byly nejčastější u souřadnic nejvzdálenějších od kamer. S touto topologií dosahovaly chyby menší než 1 mm tedy 27,38 % z celkového počtu 729 souřadnic. Naproti tomu s topologií $200-200-50$ (obr. 3.16) se nám podařilo zvýšit podíl chyb menších než 1 mm na 46,92 %. Vzhledem k těmto výsledkům jsme se rozhodli tuto topologii využít pro naše další experimenty, které jsme rozšířili do trojrozměrného prostoru xyz.



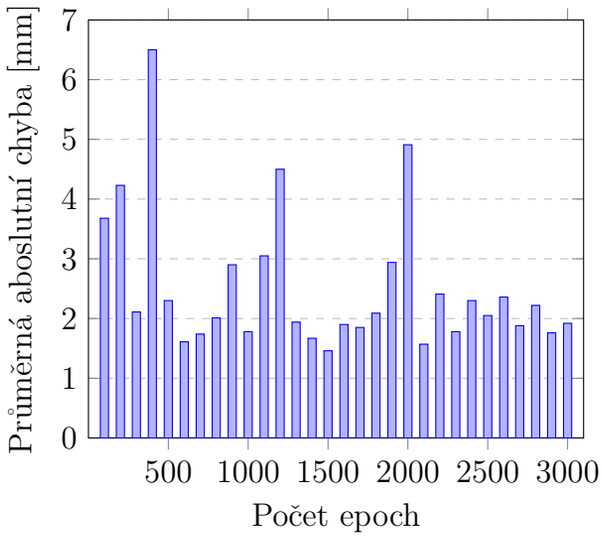
Obrázek 3.16: Graf absolutní chyby osy y v rovině při modelu sítě $200 - 200 - 50$, testovací data v rozsahu celé roviny xy 270x270 mm při 1500 epochách.

3.3 Prostor xyz

Pro možnost testování neuronové sítě v prostoru jsme nejprve shromáždili nová data, která zahrnovala celý pracovní prostor o velikosti 300x300x300 mm. Celkový počet nově získaných dat činil 29 610 snímků z jedné kamery, v příloze 19 je uvedena ukázka těchto snímků. Celý dataset je dostupný na katedře mechaniky, pružnosti a pevnosti. Tato data jsme zpracovali a vytvořili dva datasety pro prostor xyz. Trénovací dataset obsahuje 80 472 dat a testovací dataset 26 823 dat.

K trénování neuronové sítě jsme napsali dva nové programy (příloha 20 a 21). Během práce s velkým objemem dat, jsme narazili na problém. Ten byl spojen s nesrovnalostmi v počtu dat v jednotlivých datasetech. Při trénování síť očekává stejný počet vstupů jako výstupů. To nám vyhazovalo chybu a nebyli jsme schopni sít trénovat. Zjistili jsme, že složka s testovacími daty z kamery 1 obsahovala méně obrazových dat než složka druhé kamery. Chyba byla způsobena volbou HSV rozsahu pro nalezení kontur. Tento jev způsobovaly různé světelné podmínky na snímcích. Abychom efektivně zamezili této chybě, vytvořili jsme program (příloha 22), který srovná obrazová data pomocí ID a vymaže data, která nemají dvojici. Díky tomuto řešení jsme byli schopni efektivně trénovat síť s přesně spárovanými daty z obou kamer.

Z testů, které jsme provedli v rovině xy, jsme vybrali neuronovou síť s finální topologií $200 - 200 - 50$. Abychom zjistili optimální počet trénovacích cyklů (epoch), provedli jsme rozsáhlý test. Nejprve jsme natrénovali model při 100 epochách s nově získanými daty. Následně jsme implementovali smyčku pro další trénink modelu, kde jsme postupně zvyšovali počet epoch až na finálních 3000. Modely jsme testovali pomocí 28 623 dat z celého zkoumaného prostoru. Pro posouzení výkonnosti modelu v závislosti na počtu epoch jsme vyhodnocovali celkovou absolutní chybu a výsledky jsme zaznamenali do grafu (obr. 3.17).



Obrázek 3.17: Graf vlivu počtu epoch na celkovou absolutní chybu v prostoru xyz modelu s topologii $200 - 200 - 50$, od 100 do 3000 po 100 epochách při počtu 28 623 dat.

Na základě analýzy výkonnosti modelu při různých počtech epoch jsme pro naše další experimenty vybrali dvě specifické hodnoty: 600 a 1500 epoch. Při testování modelu po 600 epochách jsme zaznamenali celkovou absolutní chybu 1,61 mm. Zatímco po 1500 epochách chyba klesla na 1,46 mm. Abychom získali podrobnější přehled o přesnosti predikce souřadnic modelu v různých osách, provedli jsme další analýzu absolutních chyb ve všech osách (x, y, z). Tyto chyby jsme následně kvantifikovali a vyjádřili procentuálně do tabulky 3.4, která poskytuje přehledný souhrn četnosti chyb v každé ose.

Tabulka 3.4: Procentuální četnost absolutní chybu v osách x, y a z v prostoru, při rozsahu od 0 mm nad 5 mm po 1 mm, o topologii sítě $200 - 200 - 50$ po 600 a 1500 epochách.

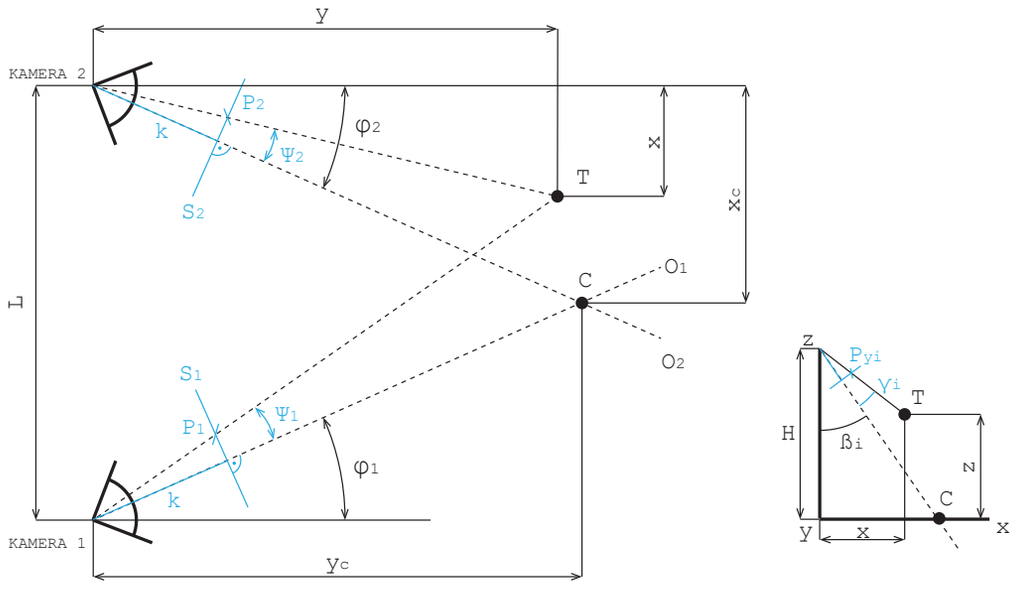
osa	pod 1 mm	1 až 2 mm	2 až 3 mm	3 až 4 mm	4 až 5 mm	nad 5 mm
četnost absolutní chyby při topologii $200 - 200 - 50$ po 600 epochách [%]						
x	41,5	30,1	16	6,87	2,5	3
y	12,4	14,1	15,8	3	13,7	28,1
z	50,4	32,7	12	3,3	0,8	0,8
četnost absolutní chyby při topologii $200 - 200 - 50$ po 1500 epochách [%]						
x	44,3	31,2	15,3	5,4	1,6	2,2
y	32,3	26,7	18,4	10,3	5,3	7
z	50,8	32,3	12,9	2,9	0,6	0,5

Model neuronové sítě, natrénovaný na 600 epoch, vykazuje nižší přesnost než model s 1500 epochami. Zejména na ose y. Zde byla četnost chyb pod 1 mm nízká. Z testů jsme zjistili, že 1500 epoch je optimální pro trénování. Získali jsme výrazně lepší přesnost zejména v ose y. Zde četnost chyb pod 1 mm dosahuje 32,28 %. Současně došlo ke snížení četnosti chyb nad 5 mm z 28,09 % u 600 epoch na 6,98 % u 1500 epoch. Na základě těchto výsledků jsme se rozhodli pro náš finální experiment použít model neuronové sítě s 1500 epochami.

3.3.1 Porovnání s triangulací

Pro finální experiment jsme zvolili záznam videa trajektorie bodu, který posloužil jako zdroj vstupních dat pro neuronovou síť a triangulační metodu. Pro synchronizované video ze dvou kamer jsme napsali nový program (příloha 23). Video jsme natočili ve formátu MP4 s frekvencí snímání 25 fps (příloha 24). Další nový program (příloha 25), postavený na principu z první kapitoly, jsme použili k převedení videa na správný formát dat pro neuronovou síť. K testování modelu jsme upravili existující program (příloha 16) a to tak, že jsme změnili metodu extrakce informací z názvu snímku (příloha 26).

K predikci polohy bodu pomocí triangulace jsme vycházeli ze schématu zobrazeného na obrázku 3.18. Z tohoto schématu jsme odvodili matematické vztahy 3.1, 3.2, a 3.3 pro výpočet x, y a z souřadnic bodu. Na základě těchto vztahů jsme vytvořili program (příloha 27), který vypočítá souřadnice bodu z polohových dat středu bodu na snímku.



Obrázek 3.18: Schéma pro výpočet polohy sledovaného bodu pomocí triangulace s počátkem souřadnicového systému pod levou kamerou (kamera 2).

x_c, y_c - souřadnice optického středu C [mm], x, y, z - souřadnice bodu T [mm], O_i - optická osa kamery, φ_i - úhel natočení kamer v rovině xy [rad], Ψ_i - úhel záběru bodu v ose x [rad], β_i - úhel sklonu kamery v rovině xz [rad], γ_i - úhel záběru bodu v ose z [rad], P_i - poloha středu bodu na snímku v ose x [px], S_i - označení senzoru kamery, P_{y_i} - poloha středu bodu na snímku v ose y [px] ,k - konstantní vzdálenost senzoru kamery [px], H - výška kamery od stolu [mm], L - rozteč mezi kamerami [mm]

Odvozené matematické vztahy pro výpočet souřadnic snímaného bodu využitím triangulační metody z dat získaných pomocí videa:

$$x = \tan\left(\varphi_1 - \Psi_1\right) \cdot y, \tag{3.1}$$

$$y = \frac{L}{\tan\left(\varphi_1 - \Psi_1\right) + \tan\left(\varphi_2 + \Psi_2\right)}, \tag{3.2}$$

$$z = H - \frac{x}{\tan\left(P_{y_1} + \gamma_1\right)}. \tag{3.3}$$

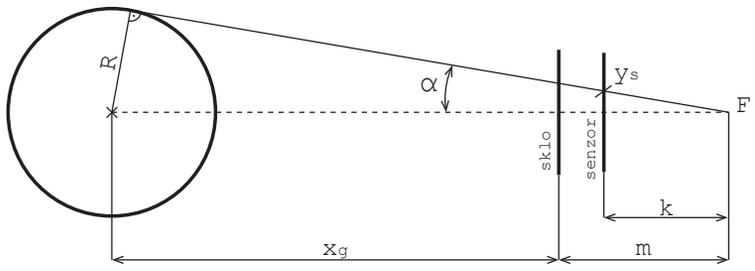
Potřebné úhly záběru bodu na snímku pro výpočet souřadnic:

$$\Psi_i = \frac{P_m - P_i}{k}, \tag{3.4}$$

$$\gamma_i = \arctan\left(\frac{P_{m_y} - P_{y_i}}{k}\right). \tag{3.5}$$

Kde značí: P_m Polohu středu snímače v ose x. $[px]$
P_{m_y} Polohu středu snímače v ose y. $[px]$

K získání úhlů záběru Ψ_i, γ_i (3.4, 3.5) bylo nezbytné určit konstantní vzdálenost k mezi snímačem a ohniskem kamery. Tuto klíčovou konstantu jsme si vyjádřili pomocí vztahů 3.6 a 3.7, které jsme odvodili ze schématu zobrazeného na obrázku 3.19.



Obrázek 3.19: Schéma pro výpočet konstantní vzdálenosti k senzoru od ohniska kamery F, odvozeno pomocí jednoho pohledu kamery na sledovaný bod.

x_{g_i} - vzdálenost středu bodu od skla kamery [mm], y_{s_i} - poloha okrajového pixelu bodu na snímku [px], k - konstantní vzdálenost senzoru od ohniska kamery [px], m - vzdálenost skla od ohniska kamery [mm], α_i - úhel záběru bodu [rad], F - ohnisko kamery, R - poloměr bodu [mm]

$$\frac{R}{x_{g_i} + m} = \sin \alpha_i. \tag{3.6}$$

$$\frac{y_{s_i}}{k} = \tan \alpha_i. \tag{3.7}$$

Pro výpočet k a m jsme si ze vztahu 3.7 vyjádřili α_i a dosadili do 3.6. Takto jsme získali rovnici 3.8 o dvou neznámých. K výpočtu jsme potřebovali dvě rovnice, které jsme získali odměřením dvou poloh sledovaného bodu (x_{g_i}, y_{s_i}) a dosadili do rovnic:

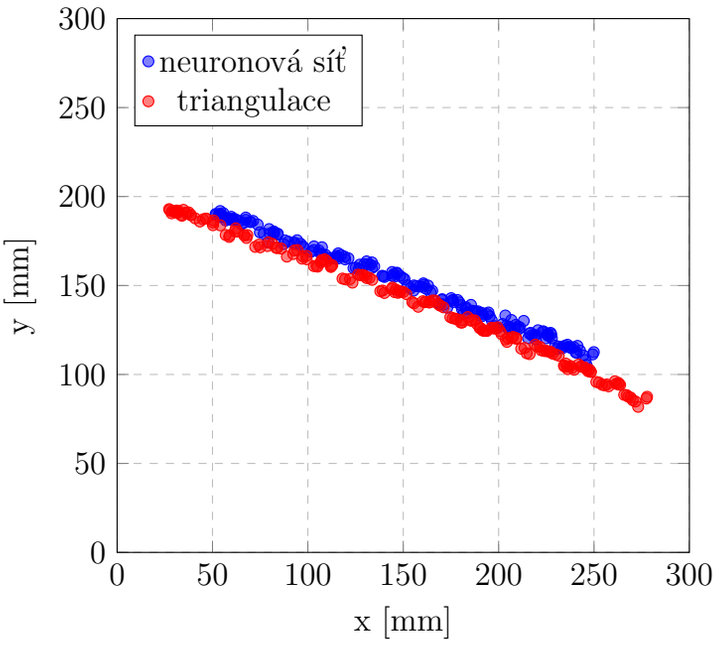
$$\frac{R}{x_{g_i} + m} = \sin\left[\arctan\left(\frac{y_{s_i}}{k}\right)\right]. \tag{3.8}$$

Souřadnice získané triangulací jsme převedli do souřadnicového systému, který jsme definovali pro náš pracovní prostor. Tento proces zahrnoval jednoduchý přepočet, pro který jsme si nejprve odměřili vzdálenosti mezi počátky jednotlivých souřadnicových systémů. Přepočet jsme zahrnuli do programu (příloha 25), aby výsledné souřadnice byly vypočteny a prezentovány přímo v souřadnicovém systému našeho pracovního prostoru.

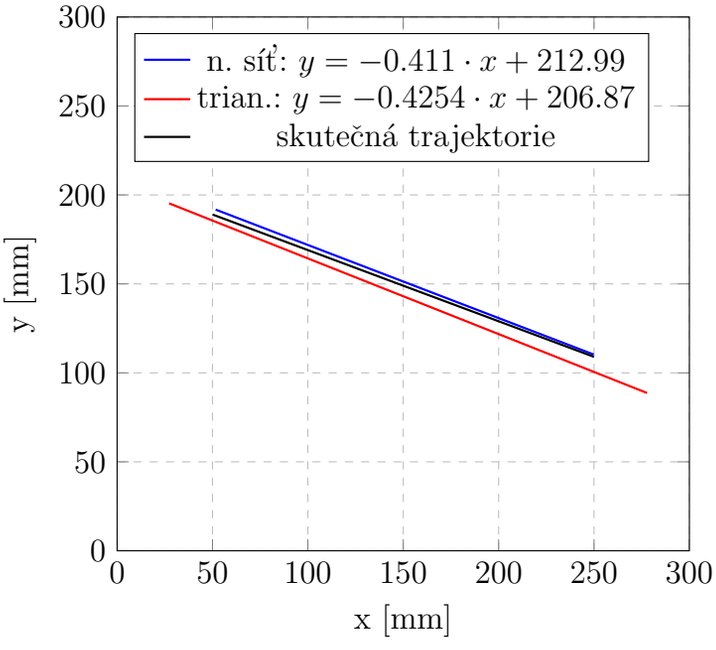
3.3.2 Výsledek experimentu

Pro porovnání dvou metod predikce polohy bodu jsme natočili video, ve kterém sledovaný bod opisuje trajektorii předem stanové přímky. Video jsme natočili při 25 fps o délce 9 sekund. Zpracováním videa, při dodržení stejného HSV rozsahu, jsme získali 234 vstupních dat, ze kterých jsme použili pouze 184 dat. Tento počet nám stanovila zvolená trajektorie přímky. Vstupní data jsme použili, jak pro model neuronové sítě, tak pro triangulaci. Takto jsme zajistili stejnorodost vstupních dat. Jediný rozdíl mezi metodami byl v způsobu zápisu hodnot do názvu snímku: pro model neuronové sítě jsme zaznamenávali polohu výřezu, zatímco pro triangulaci polohu středu bodu na snímku. Pomocí dvou různých programů (příloha 24, 25) jsme následně získali souřadnice sledovaného bodu pomocí obou metod. Tento přístup nám umožnil srovnání výkonnosti a přesnosti obou přístupů při predikci polohy bodu.

Získané souřadnice jsme vynesli do grafu (obr. 3.20), který umožňuje vizuální porovnání predikce polohy sledovaného bodu neuronovou sítí a triangulací. Prvním pozorováním jsme si všimli, že triangulace nám vrátila souřadnice ve větším rozsahu než model při stejném počtu dat. Při analýze výsledků jsme zaznamenali specifický trend v rozdílech mezi souřadnicemi získanými oběma metodami. Konkrétně jsme na ose x identifikovali, že v rozmezí mezi 140 mm a 150 mm se rozdíl souřadnic snížil na méně než 1 mm. Mimo toto rozmezí však rozdíl narostl a dosáhl až 25 mm na začátku a konci trajektorie. Rozdíly na ose y byly obecně nižší s průměrnou hodnotou 9,6 mm. Z další provedené analýzy jsme zjistili, že triangulace vracela vyšší chyby v souřadnicích, když byl bod blíže kamerám (na ose x). Vzdalujíc se od kamer se výsledky triangulace blížily hodnotám z modelu, přičemž rozdíl se stabilizoval okolo 2,5 mm. Tento poznatek naznačuje, že přesnost triangulace může být citlivá na specifickou geometrickou konfiguraci a vzdálenost od zdroje dat.



Obrázek 3.20: Graf porovnání predikcí trajektorie sledovaného bodu, zaznamenané na video při 25fps a zachování stejného zpracování 186 dat, použité metody predikce: neuronová síť a triangulace.



Obrázek 3.21: Graf porovnání skutečné trajektorie a trajektorií získaných lineární regresí souřadnic z triangulační metody a z neuronové sítě. Trajektorie zaznamenaná na video při 25fps a zachování stejného zpracování 186 dat.

K ověření správnosti predikce obou metod jsme získaná data zpracovali pomocí lineární regrese a výsledky jsme znázornili v grafu (viz obr. 3.21). Také jsme vyznačili skutečnou trajektorii bodu po přímce. Z grafu je patrné, že trajektorie získaná triangulací je delší než skutečná trajektorie, což jsme již dříve pozorovali na grafu s vyznačenými body (obr. 3.20). Tento graf dále potvrzuje nepřesnost triangulace na ose y, zejména když je bod blíže kamerám (na ose x). Rozdíl mezi predikovanou polohou a skutečnou dosahoval průměrné hodnoty 8,61 mm.

Model neuronové sítě vykazuje výrazně lepší přesnost predikce polohy. Ačkoliv zde dochází k nárůstu chyby v závislosti na vzdálenosti od kamer, tato chyba dosahovala průměrného rozdílu 2,74 mm od skutečné trajektorie. To odpovídá zlepšení o 68 % oproti triangulace. Přesnost modelu na ose x téměř odpovídá skutečné trajektorií bodu. Tato analýza ukazuje, že při použití neuronové sítě dosahujeme vyšší přesnosti predikce polohy bodu.

3.3.3 Výpočet okamžité rychlosti

Pro demonstraci výpočtu okamžité rychlosti jsme použili video, které jsme natočili k porovnání metod. Použili jsme předpoklad, že nedochází ke ztrátě snímků během nahrávání. Ten nám umožňuje jednoduše vypočítat časový interval mezi jednotlivými polohami, známe-li celkovou délku videa a počet snímků. Pro výpočet okamžité rychlosti jsme použili metodu, která se opírá o derivaci polynomu (3.9). Tento polynom jsme proložili skrze získané body trajektorie v prostoru, využívající kvadratické lineární regrese metodou nejmenších čtverců. Vzorce, které jsem použili pro výpočet okamžité rychlosti v ose z (pro osu x a y jsme postupovali analyticky):

$$z_t = A \cdot t^2 + B \cdot t + C. \tag{3.9}$$

Kde značí:	z_t	Funkce popisující polohu v ose z za čas.	$[m]$
	A, B, C	Koeficienty kvadratické lineární regrese.	
	t	Čas.	$[s]$

Pro regulaci koeficientů regrese, jsme nejprve vypočítali chybu Δz_n mezi skutečnou polohou z_n a polohou získanou pomocí regrese:

$$\Delta z_n = z_n - A \cdot t_n^2 + B \cdot t_n + C. \tag{3.10}$$

Následně jsme vypočítali sumu chyb $\sum \Delta z$ zvolených po sobě jdoucích bodů:

$$\sum \Delta z = \sum_{i=0}^{n} \Delta z_{(n+i)}^2. \tag{3.11}$$

Jednotlivými derivacemi $\sum \Delta z$ jsme dosáhli optimalizace koeficientů regrese

$$\frac{\mathrm{d}\sum \Delta z}{\mathrm{d}A} = 0, \quad \frac{\mathrm{d}\sum \Delta z}{\mathrm{d}B} = 0, \quad \frac{\mathrm{d}\sum \Delta z}{\mathrm{d}C} = 0, \tag{3.12}$$

což zredukovalo odchylky a zvýšilo přesnost výsledků. Získali jsme novou funkci polynomu s optimalizovanými koeficienty:

$$f(t_n, t_{(n+1)}, ..., z_n, z_{(n+1),...}). \tag{3.13}$$

Provedli jsme derivaci funkce časem, ve kterém jsme zjišťovali okamžitou rychlost v ose z:

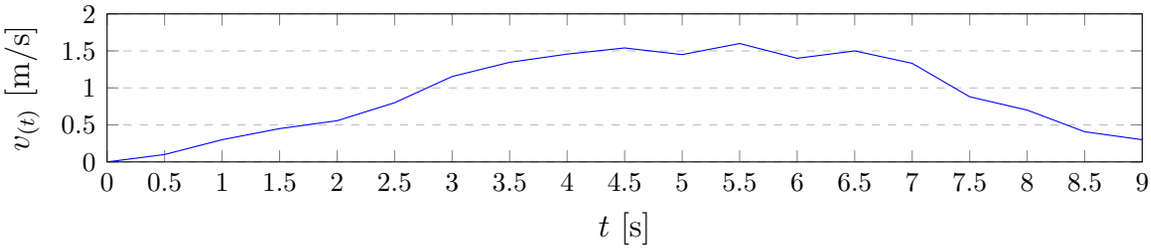
$$\frac{\mathrm{d}f}{\mathrm{d}t} = v_{z_{(t)}}. \tag{3.14}$$

Velikost vektoru okamžité rychlosti $v_{(t)}$ jsme získali pomocí vztahu:

$$v_{(t)} = \sqrt{v_{x_{(t)}}^2 + v_{y_{(t)}}^2 + v_{z_{(t)}}^2}. \tag{3.15}$$

Celý výpočet jsme realizovali skrze program (příloha 28), který načítá pět po sobě jdoucích bodů. Proloží nimi polynom a následně vypočítá vektor okamžité rychlosti $v_{(t)}$ pro střední bod výběru. Tento přístup umožňuje přesné a efektivní určení rychlosti bodu v konkrétních časových okamžicích během její trajektorie. Při výpočtu vektoru zrychlení bychom postupovali analyticky.

Pro ukázku jsme vypočítali okamžitou rychlost pro bod o souřadnicích x = 171,36 mm, y = 139,35 mm a z = 151,91 mm v čase t = 4,133 s. V tomto čase měl bod okamžitou rychlost $v_{(t)} = 0{,}7$ m/s (3.15). Průběh okamžité rychlosti v průběhu celého videa jsme vynesli do grafu (obr. 3.22).



Obrázek 3.22: Graf okamžité rychlosti bodu z 9 s záznamu videa trajektorie bodu při 25 fps, polohy bodu získány pomocí neuronové sítě.

3.3.4 Výkon neuronové sítě

Pro testování výkonnosti naší neuronové sítě s cílem zjistit její schopnost určovat polohu v reálném čase, jsme použili dva typy hardwaru: osobní notebook a výpočetní server katedry KMP. Test jsme provedli pro 30 obrazových dat (snímků):

Intel® Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz:	30 snímků za 271 ms
Intel® Xeon(R) Gold 6140 CPU @ 2.30GHz × 72:	30 snímků za 142 ms

Ve výsledcích vidíme, že určování polohy v reálném čase nebude problém ani při používání osobního notebooku. U použití servu máme více výpočetního výkonu, který lze využít pro výpočet rychlosti, či vykreslování polohy do grafu v reálném čase. To by mohlo být v určitých oblastech aplikace výhodné.

Závěr

V rámci této bakalářské práce jsme se věnovali vývoji neuronové sítě pro trasování předem připraveného geometrického bodu v prostoru s využitím kamerového sytému. Hlavním cílem bylo otestovat schopnost neuronové sítě predikovat souřadnice bodu.

Z výsledku naší práce vyplývá, že kvalita a množství trénovacích dat jsou klíčové pro úspěšné učení neuronové sítě. Během testování jsme zjistili, že zvýšení počtu dat umožnilo síti se efektivně učit. Formát těchto dat měl také výrazný vliv na učení. Ukázalo se, že naše volba zpracování vstupních dat, která snížila jejich velikost vytvořením výřezu kolem snímaného bodu, měla na neuronovou síť pozitivní vliv. Dokázali jsme tak snížit náročnost vstupní vrstvy snížením počtu vstupních neuronů.

Naše volba architektury sítě se ukázala být vhodnou při práci s daty ze dvou kamer. Jelikož síť nejprve zpracovávala obrazová data s polohou výřezu a následně tyto informace využívala pro určení polohy bodu. Naše testy ukázaly, že pro regresní úlohy není vhodné používat sigmoidní aktivační funkci a dropout regulaci.

Práce také prokázala, že neuronová síť může nabídnout vyšší přesnost a flexibilitu ve srovnání s triangulační metodou. Jelikož síť není závislá na ideálních podmínkách snímání a může se adaptovat na měnící se prostředí. Toto bylo potvrzeno v našem posledním experimentu, kde neuronová síť dosahovala lepší přesnosti v určení trajektorie bodu na základě stejných dat.

Nicméně i přes úspěchy tohoto výzkumu existuje několik oblastí, které vyžadují další zkoumání. Pro praktické využití této metody je nezbytné dosáhnout vyšší přesnosti, neboť naše síť vykázala přesnost pod 1 mm ve 44,34 % případů na ose x, na ose y pouze 32,28 % a na ose z 50,75 %. U praktické aplikace bychom měli dosáhnout minimálně 80 % přesnosti ve všech třech osách. K zlepšení bychom mohli zvětšit výřez ze snímku, neboť se zdá, že síť klade menší důraz na obrazová data a spoléhá na polohu výřezu. Další možností je změna algoritmu pro učení, kde jsme v experimentu použili základní metodu zpětného šíření chyb.

Při získávání dat pro učení sítě byl vytvořen rozsáhlý dataset obsahující 26 910 snímků kontrastního bodu z jedné kamery, kde je poloha uložena přímo v názvu každého snímku. Tento dataset může být využit pro další výzkumné projekty zaměřené na trasování bodu v prostoru pomocí neuronoví sítě.

Můžeme tedy říci, že využití neuronové sítě pro trasování bodu v prostoru ukazuje slibný potenciál pro praktické uplatnění. Zvláště v oblastech, kde nelze zaručit konstantní světelné podmínky. Zde poskytuje významnou flexibilitu pro adaptaci na pracovní prostředí.

Literatura

[1] Fabian Mueller, Christian Deuerlein, and Michael Koch. 2019. Intuitive welding robot programming via motion capture and augmented reality. *IFAC-PapersOnLine*, 52(10):294–299.

[2] YuKang Liu and YuMing Zhang. 2014. Toward welding robot with human knowledge: A remotely-controlled approach. *IEEE Transactions on Automation Science and Engineering*, 12(2):769–774.

[3] Lucas Christoph Ebel, Patrick Zuther, Jochen Maass, and Shahram Sheikhi. 2020. Motion signal processing for a remote gas metal arc welding application. *Robotics*, 9(2):30.

[4] Gong Zhang, Yuhang Zhang, Shuaihua Tuo, Zhicheng Hou, Wenlin Yang, and Xu a další. 2021. A novel seam tracking technique with a four-step method and experimental investigation of robotic welding oriented to complex welding seam. *Sensors*, 21(9):3067.

[5] OpenAI. 2024, *ChatGPT 3.5*. Online. Dostupné z: https://chatgpt.com.

[6] Blocker Cameron J. Dehnui Zhang, Zhen Xu. 2021. Neural network based 3D tracking with a graphene transparent focal stack imaging system. *Nature Communications*, 12(1):2413.

[7] Dmitry Puzyrev, Kirsten Harth, Torsten Trittel, and Ralf Stannarius. 2020. Machine Learning for 3D Particle Tracking in Granular Gases. *Microgravity Science and Technology*, 32(5):897–906.

[8] Richard I. Hartley and Peter Sturm. 1997. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157. Online. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1077314297905476.

[9] Zivid, 2024. Zivid Stripe Vision Engine. Online. Dostupné z: https://www.zivid.com/3d-vision-technology-principles. [cit. 2024-05-14].

[10] I. Ahmad, Weiguo Zheng, Jiancong Luo, and Ming Liou. 2006. A fast adaptive motion estimation algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(3):420–438.

[11] Mohit Gupta, Amit Agrawal, Ashok Veeraraghavan, and Srinivasa G. Narasimhan. 2011. Structured light 3d scanning in the presence of global illumination. pages 713–720.

[12] Teledyne E2V, 2018. 3d imaging technology - time of flight. Online. Dostupné z: https://www.azom.com/article.aspx?ArticleID=16003. [cit. 2024-05-14].

[13] M. Hansard, S. Lee, O. Choi, and R.P. Horaud. 2012, *Time-of-Flight Cameras: Principles, Methods and Applications*. SpringerBriefs in Computer Science. Springer London.

[14] Ulla Wandinger. 2005. Introduction to lidar. In *Lidar: range-resolved optical remote sensing of the atmosphere*, pages 1–18. Springer.

[15] E Shreyas, Manav Hiren Sheth, and Mohana. 2021. 3d object detection and tracking methods using deep learning for computer vision applications. In *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 735–738.

[16] Jiří Šíma and Roman Neruda. 1996, *Teoretické otázky neuronovỳch sítí*. Matfyzpress. ISBN 80-85863-18-9.

[17] Eva Volná. 2008. Neuronové sítě 1. *Ostrava: Ostravská univerzita v Ostravě. Vydání: druhé*. ISBN 978-80-7464-329-3.

[18] Jianxin Wu. 2017. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495.

[19] Bin Ding, Huimin Qian, and Jun Zhou. 2018. Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)*, pages 1836–1841. IEEE.

[20] Anthony TC Goh. 1995. Back-propagation neural networks for modeling complex systems. *Artificial intelligence in engineering*, 9(3):143–151.

[21] Michael A. Nielsen. 2015, *Neural Networks and Deep Learning*. Online. Dostupné z: http://neuralnetworksanddeeplearning.com/chap2.html.

[22] Michael A. Nielsen. 2015, *Neural Networks and Deep Learning*. Online. Dostupné z: http://neuralnetworksanddeeplearning.com/chap1.html.

[23] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR.

[24] Simon S. Haykin. 2009, *Neural networks and learning machines*. Prentice Hall, New York, 3 vydaní edition. ISBN 978-0-13-147139-9.

[25] Chollet F. 2021, *Deep Learning with Python, Second Edition*. Manning. ISBN 978-80-247-3100-1.

[26] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[27] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, and Pauli Virtanen a další. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362. Online. Dostupné z: https://doi.org/10.1038/s41586-020-2649-2,.

[28] Wes McKinney et al. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

[29] Alex Clark. 2015, *Pillow (PIL Fork) Documentation*. readthedocs. Online. Dostupné z: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[30] Charlie Clark Eric Gazoni, 2023. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.1.2 documentation. Online. Dostupné z: https://openpyxl.readthedocs.io/en/stable/.

[31] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, and a další. Dean. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[32] Francois Chollet et al., 2015. Keras. Online. Dostupné z: https://github.com/fchollet/keras.

[33] Praveen. 2021, *How to find HSV range of an Object for Computer Vision applications?* Online. Dostupné z: https://medium.com/programming-fever/how-to-find-hsv-range-of-an-object-for-computer-vision-applications-254a8eb039fc.

Seznam obrázků

1.1 Schéma triangulační metody. 14
1.2 Výsledky studie pro sledování více bodových objektů. 15
1.3 Výsledky studie sledování částic v granulárních plynů. 16
1.4 Struktura formálního neuronu. 17
1.5 Příklad acyklické architektury. 18
1.6 Příklad architektury vícevrstvé neuronové sítě. 19
1.7 Graf aktivační funkce ReLU. 20
1.8 Graf aktivační funkce sigmoid. 20
1.9 Zpětné šíření chyby. 21
1.10 Ukázka principu regulace dropout. 22

2.1 Popis kamery se stativem. 24
2.2 Pracovní prostor. 25
2.3 Výstup z programu pro získávání dat. 26
2.4 Definice pracovního prostředí. 26
2.5 Volba barvy kontrastního bodu. 27
2.6 Popis držáku kontrastního bodu. 28
2.7 Ukázka sbíraných snímků. 30
2.8 Test velikosti výřezu. 32
2.9 Ukázka výřezů z kamer. 32
2.10 Princip vytvoření výřezu. 32
2.11 Ukázky náhodných výřezů sledovaného bodu z jednoho snímku. . . . 33

3.1 První architektura sítě. 34
3.2 Graf absolutní chyby v ose x od 0 mm po 10 mm. 39
3.3 Graf absolutní chyby v ose x od od 5 mm po 10 mm. 39
3.4 Graf vlivu aktivační funkci ReLU. 41
3.5 Graf vlivu aktivační funkci sigmoid. 41
3.6 Graf vlivu regulace dropout = 0,5 a L2 = 0,001. 42
3.7 Graf vlivu regulace při navýšení trénovacích dat. 43
3.8 Graf vlivu dropout při L2 = 0,01. 44
3.9 Nová architektura sítě. 45
3.10 Graf vlivu změny počtu neuronů v první skryté vrstvě. 46
3.11 Graf vlivu změny počtu neuronů v druhé skryté vrstvě. 46
3.12 Graf vlivu změny počtu neuronů v třetí skryté vrstvě. 47
3.13 Graf vlivu regulace v jednoduché neuronové síti. 47
3.14 Graf dvou testů topologie. 48
3.15 Graf absolutní chyby osy y v rovině při modelu sítě $200 - 150 - 100$. 49

3.16 Graf absolutní chyby osy y v rovině při modelu sítě $200 - 200 - 50$. . 50
3.17 Graf vlivu počtu epoch na absolutní chybu v prostoru. 51
3.18 Odvození triangulace. 52
3.19 Výpočet konstanty k. 53
3.20 Graf porovnání modelu a triangulace. 55
3.21 Graf porovnání lineární regrese modelu a triangulace. 55
3.22 Graf okamžité rychlosti. 57

Seznam tabulek

2.1 Použité knihovny a jejich verze. 23

3.1 Nastavení architektury sítě pro osu x. 38
3.2 Použitá topologie sítě při testech. 45
3.3 Procentuální četnost absolutní chybu v osách x a y v rovině. 49
3.4 Procentuální četnost absolutní chybu v osách x, y a z v prostoru. . . 51

Seznam kódů

2.1 Funkce pro vytvoření značky. 25
2.2 Nastavení potřebných proměnných. 29
2.3 Získání obrazu z kamer. 29
2.4 Snímání a ukládání dat. 29
2.5 Tvorba masky bodu. 31
2.6 Nalezení přibližného středu bodu. 31
2.7 Funkce pro vytvoření výřezu. 31
2.8 Funkce pro náhodný výřez. 33
3.1 Funkce pro extrahování vstupních dat. 35
3.2 Architektura neuronové sítě pro osu x. 36
3.3 Tvorba modelu. 36
3.4 Kompilace modelu. 37
3.5 Konfigurace modelu. 37
3.6 Funkce k predikci souřadnic. 37
3.7 Přidání regulace na vrstvu. 42
3.8 Párování vstupních dat. 44

Přílohy

Příloha 01:	Model držáku kamery:	drzak_kamery.zip	24
Příloha 02:	Program pro nastavení:	set_up_cameras.py	25
Příloha 03:	Čtvercová síť s rastrem 10 mm:	ctvercova_sit.pdf	26
Příloha 04:	Program pro HSV rozsah:	hsv_picker.py	27
Příloha 05:	Model držáku kontrastního bodu:	drzak_bodu.zip	27
Příloha 06:	Program pro snímání dat:	data_capture.py	28
Příloha 07:	Program pro výřez:	data_preprocessing.py	30
Příloha 08:	Program pro náhodný výřez:	ran_data_preprocessing.py	33
Příloha 09:	Neuronová sít osy x:	neural_network_x.py	35
Příloha 10:	Predikce v ose x:	neural_network_x_test.py	38
Příloha 11:	Neuronová síť roviny xy:	neural_network_xy.py	40
Příloha 12:	Predikce v rovině xy:	neural_network_x_test.py	40
Příloha 13:	Regulace sítě xy:	regulace_n_w_xy.py	42
Příloha 14:	Kontrola velikosti:	size_control.py	43
Příloha 15:	Upravená neuronová síť xy:	neural_network_xy_f.py	44
Příloha 16:	Upravená predikce v rovině xy:	neural_network_xy_f_test.py	44
Příloha 17:	Tři vrstvy v neuronové síti:	n_network_xy_final.py	45
Příloha 18:	Test tří vrstev v neuronové síti:	n_network_xy_final_test.py	45
Příloha 19:	Ukázka snímků datasetu:	ukazky_snimku.zip	50
Příloha 20:	Neuronová síť prostor xyz:	neural_network_xyz.py	50
Příloha 21:	Test sítě v prostoru xyz:	neural_network_xyz_test.py	50
Příloha 22:	Vyrovnání počtu dat ve složkách:	number_of_date.py	50
Příloha 23:	Nahrávání trajektorie bodu:	video_capture.py	52
Příloha 24:	Video pro finální experiment:	video_final.zip	52
Příloha 25:	Získání vstupních dat z videa:	data_preproces_video.py	52
Příloha 26:	Test modelu daty z videa:	model_video_test.py	52
Příloha 27:	Výpočet pomocí triangulace:	triangulace.py	52
Příloha 28:	Výpočet okamžité rychlosti:	total_velocity.py	57