VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

NAVÁDĚNÍ POMOCÍ KAMERY
POSITION CONTROL WITH CAMERA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Dominik Ficek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Miloslav Richter, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Dominik Ficek ***ID:*** 203219
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Navádění pomocí kamery

POKYNY PRO VYPRACOVÁNÍ:

1) Nastudujte problematiku zpracování obrazu a 3D rekonstrukce.
2) Navrhněte metody pro ovládání kurzoru monitoru pomocí rukou. První metoda bude používat kamery umístěné v prostoru (u monitoru) a značek na ruce. Druhá metoda bude používat kamery umístěné na ruce a značky v oblasti monitoru.
3) Navrhněte matematický aparát a realizujte algoritmy pro detekci a stanovení pozice.
4) Realizujte měřicí pracoviště a nasnímejte zkušební data. Vyzkoušejte na nich navržené algoritmy.
5) Zhodnoťte dosažené výsledky. Srovnejte jednotlivé metody.

DOPORUČENÁ LITERATURA:

Gonzalez R.C., Woods R.E.: Digital Image Processing, 4th edition, Pearson, 2017, ISBN 978-0133356724

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. Miloslav Richter, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Práce se zabývá stanovením pozice kamery vzhledem ke stanovenému systému souřadnic. Tento souřadnicový systém je dán polohou definovaných značek v prostoru. Jakožto zpětná vazba pro kontrolu správnosti stanovení pozice je zvoleno navádění kurzoru na monitoru. Cílem práce je navrhnout metody navádění kurzoru pomocí kamery pracující v reálném čase. V práci je popsána základní teorie zpracování obrazu, využití umělé inteligence v oblasti počítačového vidění a problematiky 3D rekonstrukce. Dále je uveden návrh dvou metod pro ovládání kurzoru pomocí jedné kamery. První metoda definuje pevný souřadnicový systém a pohybem kamery je realizováno navádění kurzoru. Druhá metoda pracuje s pohyblivým souřadnicovým systémem a pevně umístěnou kamerou. V práci je dále popsán proces realizace navržených metod, jejich zhodnocení a porovnání.

KLÍČOVÁ SLOVA

počítačové vidění, zpracování obrazu, konvoluční neuronové sítě, VGG19, objektový detektor, YOLOv3, perspektiva z n bodů, 3D rekonstrukce

ABSTRACT

Thesis focuses on camera's pose estimation in set world coordinate system. This coordinate system is defined by position of predefined marks. Cursor control is selected as a pose estimation feedback. Aim of this thesis is designing real time cursor control with camera methods. In theoretical part of this thesis is dedicated to explanation of basic theory of image processing, artificial intelligence in computer vision and 3D reconstruction. Following theoretical chapter is a chapter dedicated to the design of two position control with camera methods. First method defines fixed coordinate system and cursor is controlled by camera's movement. Second method utilizes fixed camera and movable coordinate system. Further chapters are dedicated to realization of designed methods, their evaluation and comparison.

KEYWORDS

computer vision, image processing, convolutional neural networks, VGG19, object detector, YOLOv3, perspective from n points, 3D reconstruction

BIBLIOGRAFICKÁ CITACE

FICEK, Dominik. *Navádění pomocí kamery.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/126972. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Miloslav Richter.

PROHLÁŠENÍ AUTORA O PŮVODNOSTI DÍLA

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucího závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: ……

Podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce panu Ing. Miloslavu Richterovi Ph.D. za odbornou pomoc, konzultace, ochotu, návrhy k práci a pohotovou komunikaci.

Obsah

Seznam obrázků

Seznam tabulek

1 ÚVOD

Práce řeší problematiku sledování polohy kamery v prostoru. Pro řešení je třeba stanovit pozici kamery v souřadnicovém systému, který je definovaný pomocí referenčních značek. Jako zpětná vazba pro kontrolu správnosti stanovení pozice je zvoleno ovládání kurzoru počítače.

Úloha může být rozdělena na tři hlavní části. První částí je detekce předem definovaných značek ve snímku kamery. Následně je ze znalosti polohy těchto značek ve 3D prostoru a polohy značek na snímku spočítána poloha kamery v prostoru. Poslední částí je úprava polohy kurzoru na monitoru.

Řešením práce je takzvaný *„proof of concept“*. Práce dokazuje, že je možné v reálném čase sledovat a specifikovat pohyb kamery vzhledem k definovanému souřadnicovému systému za použití definovaných bodů v prostoru. Aplikací těchto metod může být například koncept rozšířené reality.

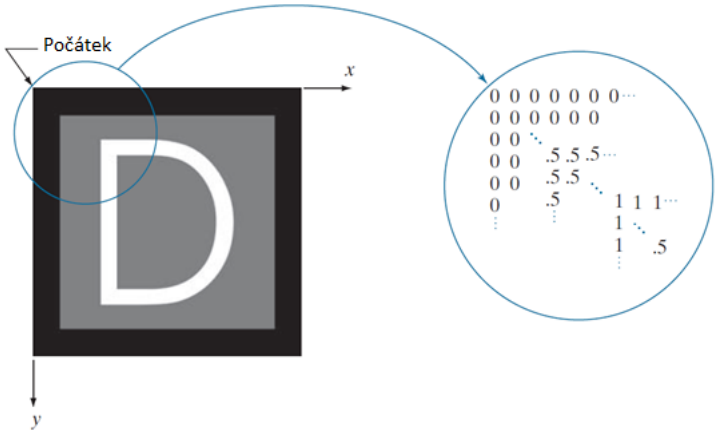
Text práce se skládá ze čtyř hlavních kapitol. V druhé kapitole je nastíněna problematika zpracování obrazu, základní teorie geometrických transformací, základní metodika kalibrace kamery, základy hlubokého učení a fungování konvolučních neuronových sítí a také problematika stanovení pozice kamery. Třetí kapitola pojednává o návrhu metod, pomocí kterých je stanovena pozice kamery. První metoda využívá značky pevně umístěné na monitoru a je zkoumána pozice pohybující se kamery v prostoru. Druhá metoda pracuje s pevně umístěnou kamerou na monitoru a pohybujícím se souřadnicovým systémem. Čtvrtá kapitola seznamuje s nástroji použitými k řešení práce a popisuje zvolený postup a metody použité pro řešení práce. Pátá kapitola diskutuje výsledky návrhu, výsledky provedených měření, porovnání navržených metod a možná vylepšení implementace.

2 DETEKCE V OBRAZE A STANOVENÍ POZICE

V této kapitole jsou shrnuty základy teorie zpracování obrazu, strojového učení, hlubokého učení, 3D transformací a kalibrace kamery.

2.1 Digitální obraz

Digitálnímu obrazu *f(x, y)* rozumíme jako vzorkované a kvantované spojité funkci *f(s, t)*. Diskrétní funkce *f(x, y)* je tedy matice o rozměru R×S, kde R je počet řádků matice a S je počet sloupců matice. Každý prvek této matice je nazýván *pixel*, neboli element obrazu (*picture element*) a hodnota *f(x, y)* pro každý pixel reprezentuje jeho jasovou hodnotu. [25]

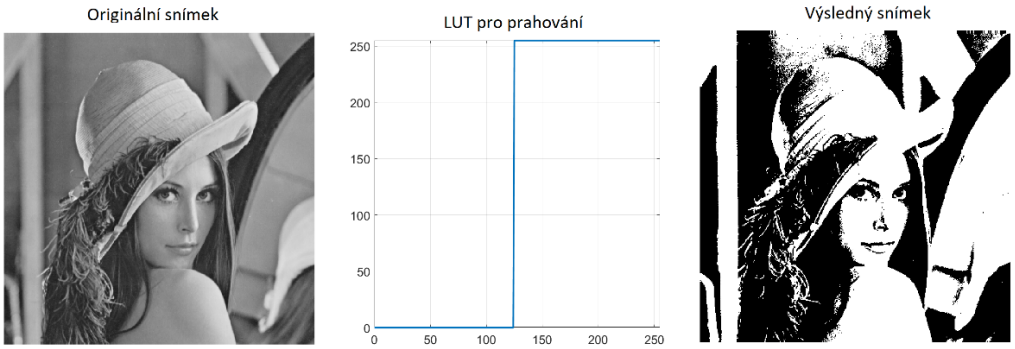


Obrázek 1 - Reprezentace obrazu pomocí pixelů [25]

Z obrázku 1 lze vypozorovat, že počátek matice je definován v levém vrchním rohu matice. Jedná se o konvenci, která je založená na principu zobrazování obrazu na displejích (např. televize nebo monitory), kde je obraz zobrazován postupně od levého vrchního rohu po řádcích. Pokud tedy indexujeme matici pomocí kartézských souřadnic *x* a *y*, tak se jedná o ekvivalent indexace matice pomocí řádků a sloupců, tedy $f(x, y) \Leftrightarrow f(s, r)$ kde *r* a s jsou indexy řádku a sloupce, respektivě. [25]

Na obrázku 1 lze také vidět číselné reprezentace jasu v obraze. Obecně platí, že čím je pixel světlejší, tím má větší hodnotu, lze tedy říct, že pixel o hodnotě nula odpovídá černé barvě a pixel o maximální hodnotě odpovídá bílé barvě. Pro správnou reprezentaci jasu je nutno určit maximální hodnotu jasu a krok jasových hodnot. Nejčastější reprezentace jasu je pomocí osmibitové hodnoty, tedy jasové hodnoty jsou v rozsahu $f(x, y) \in \langle 0; 255 \rangle$ s krokem jedna. Na obrázku 1 je uveden příklad normalizovaných jasových hodnot na rozsah $f(x, y) \in \langle 0; 1 \rangle$, jedná se také o velmi využívaný způsob reprezentace jasových hodnot (využití např. ve strojovém učení). [25]

Známe-li jasové úrovně všech pixelů v obraze, můžeme sestavit graf četnosti jasových hodnot v obraze. Takový graf se nazývá histogram. Operace s histogramem se nazývají jasové transformace a dovolují operace jako například ekvalizace histogramu, která transformuje obraz, tak aby bylo využito co nejvíce jasových úrovní (využití v úpravách fotografií). Důležitou jasovou transformací je také takzvané prahování, která způsobuje převod obrazu na binární, tedy obraz, který obsahuje pouze černé nebo bílé pixely. Tato transformace funguje na principu použití takzvané lookup tabulky (LUT), která definuje změnu jasových hodnot pro jakoukoliv jasovou úroveň v obraze, pro osmibitový obraz je tedy definovaný obor hodnot a definiční obor LUT jako $H(f) = D(f) = \langle 0; 255 \rangle$. Použití LUT pro prahování je ilustrováno na obrázku 2. [25]



Obrázek 2 - Prahování obrazu za pomocí LUT

Je nutno dodat, že prahování není jediný způsob použití LUT, jako další příklady jejich využití můžeme uvést zmenšení a zvětšení kontrastu obrazu, inverze obrazu nebo gamma korekce.

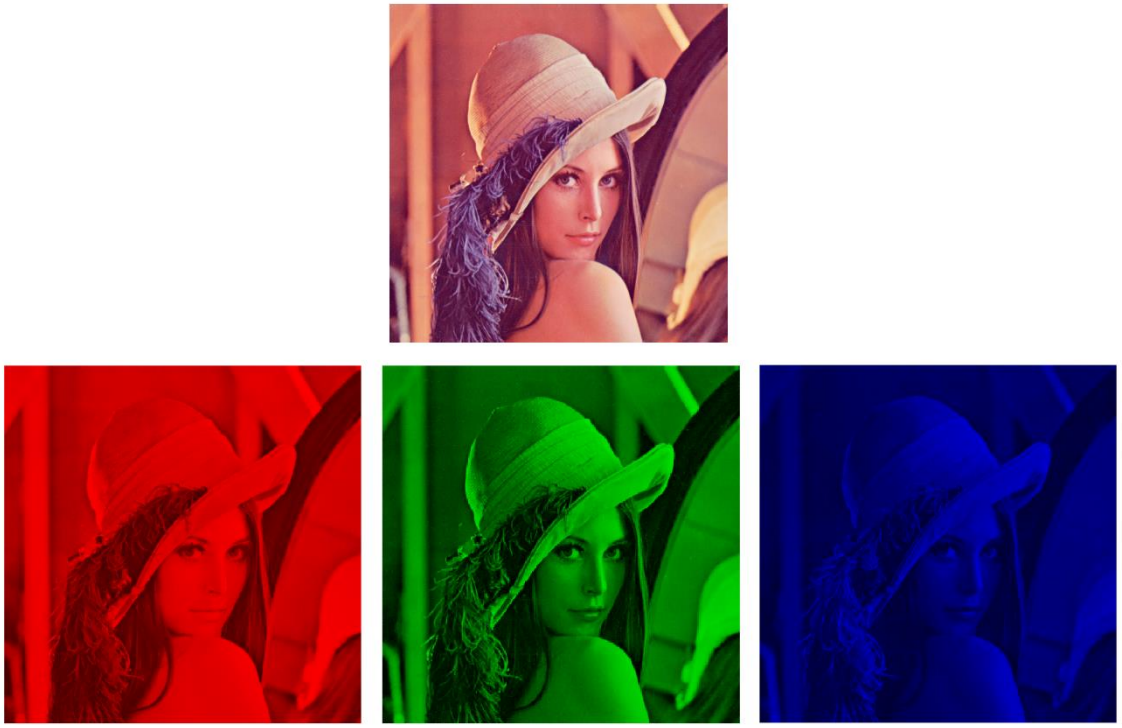
2.1.1 Reprezentace barev v obraze

V předchozí kapitole byly uvedeny techniky pracující pouze s obrazem šedotónným (často nepřesně označován jako černobílý obraz), pro reprezentaci barevných snímků je třeba přidat další dimenzi do matice reprezentující obraz. Matice R×S se tedy mění na tensor R×S×K kde K je počet kanálů obrazu. Pro reprezentaci barev jsou definovány různé barevné modely. Tyto modely byly vytvořeny pro využití v různých aplikacích (např. zobrazování na displeji nebo barevný tisk). Barevných modelů je v dnešní době mnoho, budou tedy specifikovány pouze modely použité v rámci řešení práce.

Model Red Green Blue

Zkráceně RGB model, je velmi využívaný barevný model sloužící hlavně pro zobrazování barevných dat na displeji (televize, monitory atd.). Jednotlivé

složky reprezentují obdobně jako šedotónný obraz jasové úrovně jednotlivých složek červené, zelené a modré barvy. RGB obraz je tedy reprezentován jako tensor R×S×3, kde každému pixelu přísluší tři jasové hodnoty. [25]



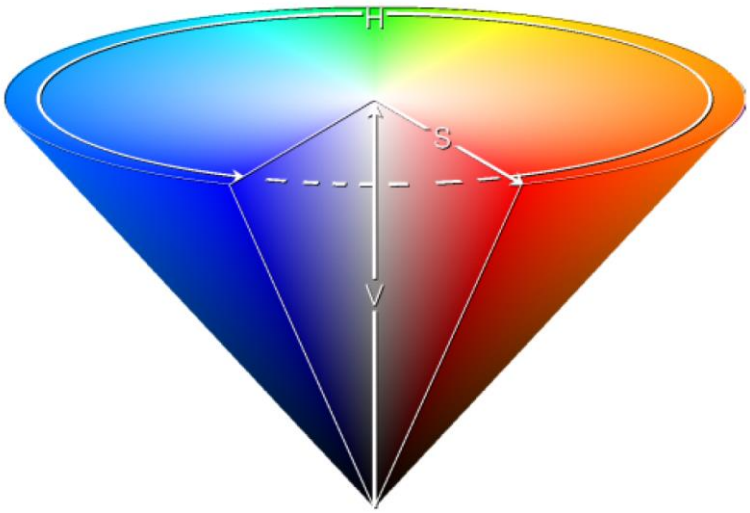
Obrázek 3 - Reprezentace obrazu pomocí RGB modelu

Model Hue Saturation Value

Zkráceně HSV model, je výsledkem nelineární transformace RGB modelu. Problém RGB modelu je takový, že není příliš intuitivní pro člověka a HSV model se tento problém snaží vyřešit. Model je složen ze tří složek:

- Hue – definuje samotnou barvu
- Saturation – Saturace barvy neboli její sytost
- Value – Jasová hodnota barvy

Ilustrace HSV modelu je vyobrazena na obrázku 4. Výhoda modelu leží v tom, že barva je definována pouze jednou hodnotou (hue) a tím se model stává intuitivní pro člověka. [26]



Obrázek 4 - Ilustrace HSV barevného modelu [26]

Přepočet z RGB modelu do HSV modelu je dán rovnicemi 2.1, 2.2 a 2.3. [26]

$$H = \begin{cases} 0 & pokud\ \max(R,G,B) = \min(R,G,B) \\ 60° \cdot \dfrac{G-B}{R-\min(RGB)} & pokud\ \max(R,G,B) = R \\ 60° \cdot \dfrac{G-B}{G-\min(RGB)} + 120° & pokud\ \max(R,G,B) = G \\ 60° \cdot \dfrac{G-B}{B-\min(RGB)} + 240° & pokud\ \max(R,G,B) = B \end{cases} \tag{2.1}$$

$$S = \begin{cases} 0 & pokud \max(R,G,B) = 0 \\ \dfrac{\max(R,G,B) - \min(R,G,B)}{\max(R,G,B)} & pokud \max(R,G,B) \neq 0 \end{cases} \tag{2.2}$$

$$V = \max(R,G,B) \tag{2.3}$$

2.2 Konvoluce a její využití ve zpracování obrazu

Prostorová filtrace je pro digitální obraz realizována pomocí 2D konvoluce (2.4). Jedná se o lokální jasové transformace, tedy jasová hodnota pixelu po transformaci je dána lokálním okolím pixelu.

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} f(x+i, y+j) \cdot h(i,j) \tag{2.4}$$

Kde *g(x, y)* je výsledný signál po konvoluci, *f(x, y)* je originální signál, *h(x, y)* je konvoluční filtr (také nazývaný jako kernel) o velikosti $m \times n$. Předpokládáme, že platí $m = 2a + 1$ a $n = 2b + 1$, kde a a b jsou nezáporná celá čísla. Z tohoto předpokladu vyplývá, že pracujeme s filtry o liché velikosti v obou osách. [25]

Pro operaci konvoluce platí komutativní (2.5), asociativní (2.6) a distribuční (2.7) zákony. Hlavní aplikace těchto zákonů je aplikace asociativního zákona, který dovoluje při použití více konvolučních filtrů aplikovat konvoluci filtrů vzájemně a až poté aplikovat výsledný konvoluční filtr na obraz. Vzhledem k faktu, že konvoluce je výpočetně drahá, je tato vlastnost výpočetně výhodná. [25]

$$f(x,y) * h(x,y) = h(x,y) * f(x,y) \tag{2.5}$$
$$f(x,y) * [g(x,y) * h(x,y)] = [f(x,y) * g(x,y)] * h(x,y) \tag{2.6}$$
$$f(x,y) * [g(x,y) + h(x,y)] = f(x,y) * g(x,y) + f(x,y) * h(x,y) \tag{2.7}$$

Mějme konvoluční kernel *w(x, y)*. Pokud můžeme tento kernel rozdělit na dva vektorové kernely *c(x)* a *r(y)*, tak aby platilo $c \cdot r = w$, mluvíme o takzvaném filtru separabilním. Dále je dokázáno, že vynásobení těchto vektorů dává stejný výsledek jako jejich konvoluce. Platí tedy $c \cdot r = c * r = w$ a platí tedy možnost uplatnění asociativního zákona $g(x,y) = f(x,y) * w(x,y) = [f(x,y) * c(x)] * r(y)$. Vyplývá tedy, že použitím separativních kernelů je výpočetně výhodné oproti použití neseparativních filtrů, proto je většina v dnešní době používaných filtrů separativní. [25]

2.2.1 Konvoluční kernely používané pro filtraci

V této podkapitole jsou popsány základní filtry typů dolní a horní propust.

Filtry typu dolní propust

Filtry dolní propusti jsou také označovány jako vyhlazovací filtry. Cílem použití těchto filtrů je odfiltrovat vysoké frekvence, tedy odstranit šum, zjednodušit scénu na snímku a odstranit nepodstatné detaily na snímku. Tyto filtry jsou často používány pro předzpracování snímků. [25]

Nejjednodušším filtrem typu dolní propust je filtr aplikující průměrování (v angličtině *averaging*). Ukázka průměrovacího filtru je uvedena ve vztahu 2.8. Nevýhodou tohoto filtru je, že příliš poškozuje důležité části scény na snímku (specificky hrany objektů) a také nemá kruhový charakter, což způsobuje závislost výstupu na orientaci objektů na snímku, proto vzniká nerovnoměrné rozostření. [25]

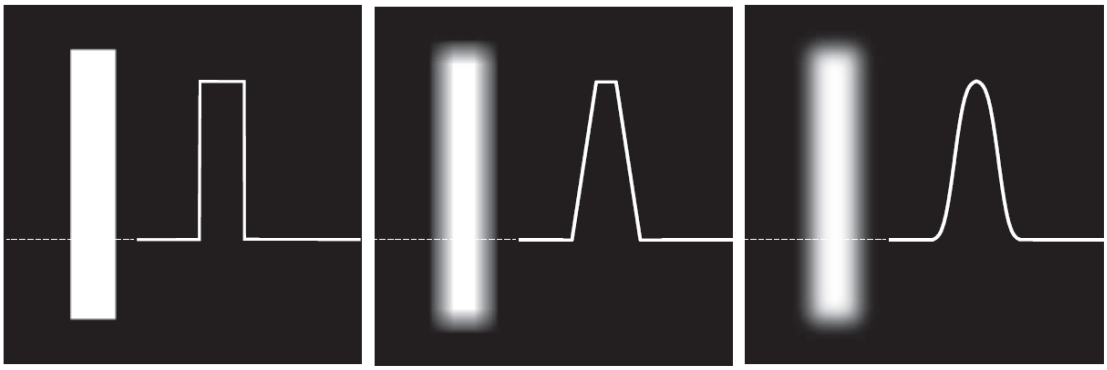
$$avg(x,y) = \frac{1}{M \cdot N} \tag{2.8}$$

Kde *M* a *N* je počet řádků a sloupců kernelu.

Nejpopulárnějším filtrem typu dolní propust je kernel s Gaussovým rozložením (vztah 2.9), jedná se o separabilním kernel kruhového charakteru se zvýšenou váhou na lokální okolí zkoumaného pixelu. Z těchto vlastností vyplývá, že oproti průměrovacímu filtru má menší vliv na rozostření hran a rozostření není závislé na orientaci objektů. [25] Porovnání filtru využívající průměrování a Gaussova filtru je znázorněno na obrázku 5, kde lze vypozorovat závislost filtrace průměrovacího filtru na orientaci hran.

$$gauss(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.9}$$

Kde σ je směrodatná odchylka rozložení.



Obrázek 5 - Porovnání filtrace pomocí průměrování a použití Gaussova filtru [25]

Alternativou pro Gaussův filtr je filtr aplikující medián. Jedná se o takzvaný nelineární filtr, jeho popis je dán nelineární funkcí medián. Výhodou tohoto filtru je, že nezpůsobuje rozostření hran objektů na snímku a zároveň úspěšně filtruje šum. Jeho nevýhodou je odstraňování tenkých objektů ze snímku a zvýšená výpočetní náročnost. [25]

Filtry typu horní propust

Z popisu filtrů typu dolní propusti jde vidět, že použitím průměrování dojde k rozmazávání obrazu. Protože průměrování lze analogicky přirovnat k integraci, je tedy jasné že filtry typu horní propust jsou popsány prostorovou derivací. V rámci filtrace nízkých frekvencí definujeme první a druhou derivaci obrazu. První derivaci definujeme pomocí následujících pravidel:

1. Výsledná hodnota musí být nulová v prostorech s konstantní intenzitou.
2. Výsledná hodnota musí být nenulová na skokovém přechodu mezi intenzitami nebo na krajích rampového přechodu mezi intenzitami.

3. Výsledná hodnota musí být nenulová v prostorech rampového přechodu mezi intenzitami.

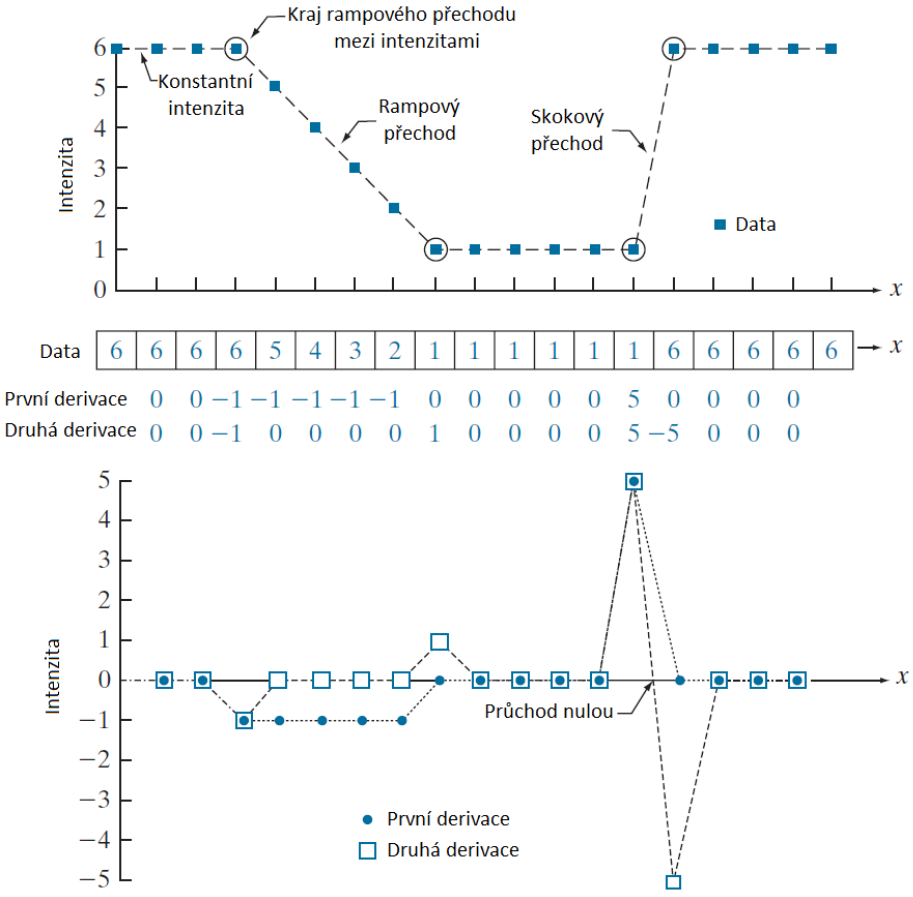
Druhou derivaci definujeme také pomocí pravidel:

1. Výsledná hodnota musí být nulová v prostorech s konstantní intenzitou.
2. Výsledná hodnota musí být nenulová na skokovém přechodu mezi intenzitami nebo na krajích rampového přechodu mezi intenzitami.
3. Výsledná hodnota musí být nulová v prostorech rampového přechodu mezi intenzitami.

Můžeme tedy definovat vztahy pro první (2.10) a druhou (2.11) derivaci. Tyto definice jsou ověřeny na obrázku 6. Z definic a z obrázku jde vydedukovat, že se jedná o funkce fungující jako detektory hran. [25]

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \tag{2.10}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2 \cdot f(x) \tag{2.11}$$



Obrázek 6 - Aplikace první a druhé derivace na diskrétní funkci [25]

První derivace v oboru zpracování obrazu je definována pomocí amplitudy gradientu. Gradient snímku je definován vztahem 2.13, amplituda tohoto gradientu je definována vztahem 2.14. [25]

$$\nabla f = grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.13}$$

$$M(x,y) = \|\nabla f\| = mag(\nabla f) = \sqrt{{g_x}^2 + {g_y}^2} \tag{2.14}$$

Konvoluční kernel, který implementuje operaci gradientu podél os x a y je realizován pomocí operátoru Prewittové (2.15) (2.16). [25]

$$P_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.15}$$

$$P_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.16}$$

Velmi populárním filtrem je Sobelův filtr (2.17) (2.18). Sobelův filtr na rozdíl od filtru Prewittové aplikuje jemné vyhlazení obrazu, a tím redukuje množství chybně detekovaných hran vlivem šumu. [25]

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.17}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.18}$$

Obdobně jako první derivace je definovaná druhá derivace pomocí gradientu. Nejjednodušší implementace takového filtru je Laplaceův operátor, který je definovaný vztahem 2.19. Samotný konvoluční filtr je definovaný vztahem 2.20. Modifikací tohoto filtru je Laplaceův operátor zohledňující diagonální hrany (2.21). [25]

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.19}$$

$$L_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.20}$$

$$L_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.21}$$

Obrázek 7 - Porovnání aplikace filtru horní propusti první a druhé derivace

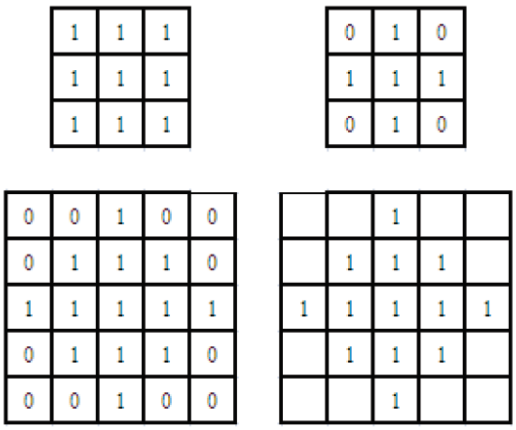
2.3 Morfologické transformace

Morfologie v kontextu zpracování obrazu znamená popis tvaru a struktury objektů v obraze. Morfologické operace tedy pracují s relativním rozložením pixelů namísto operace s hodnotami jednotlivých pixelů. Primárním cílem těchto operací je odstranit nedokonalosti ve strukturách a tvarech objektů. Morfologické operace jsou definované primárně pro binární obrazy, ale jsou užitečné také pro šedotónný obraz. [38]

2.3.1 Strukturální element

Morfologické operace využívají takzvaný strukturální element. Jedná se o matici standartně malých rozměrů. Tento strukturální element interaguje s obrazem za cílem upravit tvary objektů v obraze. Strukturální element je tedy matice složená z jedniček a nul v určitém tvaru. Standartní tvar strukturálního elementu je čtvercová matice o lichém počtu prvků v obou osách. Definujeme také počátek elementu, který nemusí být ve středu strukturálního elementu. Příklady reprezentace strukturálního elementu jsou vyobrazeny na obrázku 8. [38]

Interakce strukturálního elementu s obrazem je taková, že strukturální element je postupně přesouván na všechny pixely snímku a na každé své poloze provede porovnání svých hodnot a hodnot snímku. Výsledek této operace pak záleží na dané operaci, která je prováděna. [38]



Obrázek 8 - Strukturální elementy

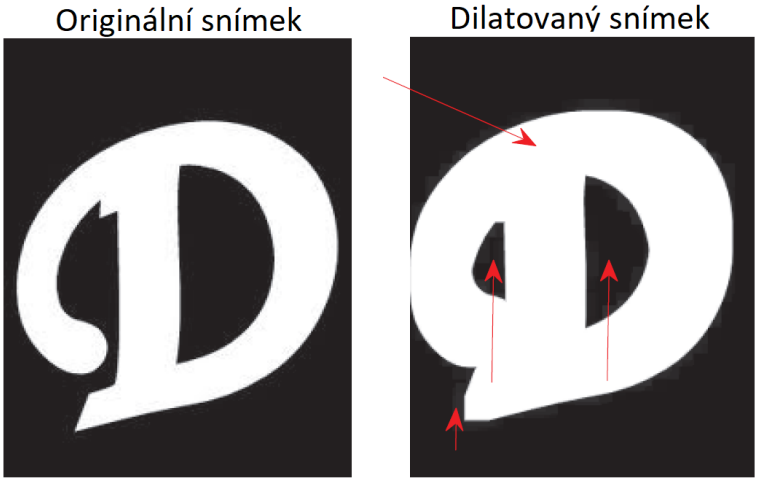
2.3.2 Morfologické operace

V této kapitole jsou popsány základní morfologické operace dilatace, eroze a z nich odvozené operace otevření a uzavření.

Dilatace

Operace dilatace způsobuje růst objektů v obraze. Rozsah tohoto růstu je dán strukturálním elementem. Dilatace obrazu A strukturálním elementem B je označována jako $A \oplus B$. Pokud je logický součin počátku strukturálního elementu a porovnávaného pixelu roven jedné, tak je lokální okolí stejné velikosti jako je velikost strukturálního elementu rovno logickému součtu strukturálního elementu a lokálního okolí studovaného pixelu. [38]

Tento proces je nejčastěji využíván pro vyplňování děr v objektech, avšak vzhledem k tomu, že dilatace rozšiřuje objekty, můžeme přirovnat tuto operaci k aplikaci filtru typu dolní propust. Aplikace dilatace je ilustrována na obrázku 9. [38]

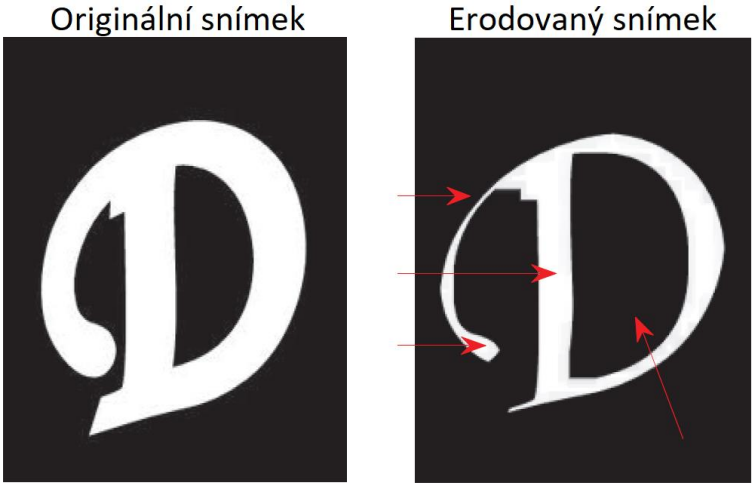


Obrázek 9 - Dilatace obrázku [38]

Eroze

Operace eroze naopak od dilatace způsobuje snižování velikostí objektů. Rozsah této ztráty je dán opět strukturálním elementem. Eroze obrazu A strukturálním elementem B je označována jako $A \ominus B$. Operace eroze nastaví hodnotu pixelů na nulu, pro které platí, že logický součin jedniček strukturálního elementu a jejich protějšků v obraze roven nule alespoň v jednom případě. [38]

Operace odstraní struktury, které jsou velikostně menší, než je strukturální element. Můžeme tedy říct, že dochází k odstranění zašuměných „spojení" objektů. [38] Ilustrace efektu eroze je vyobrazena na obrázku 10.



Obrázek 10 - Eroze obrázku [38]

Otevření a uzavření

Operace otevření je kombinací dilatace a eroze (2.22), obdobně je také definována operace uzavření (2.23). Z definice lze odvodit, že operace otevření redukuje šum v obraze a zároveň minimalizuje rozostření objektů, zatímco operace uzavření spojuje blízké objekty, tedy zaceluje díry v objektech. [25]

$$A \circ B = (A \ominus B) \oplus B \tag{2.22}$$

$$A \bullet B = (A \oplus B) \ominus B \tag{2.23}$$

2.4 Rozpoznávání pomocí hlubokého učení

Problematika klasifikace obrazu nebo detekce objektů a následné klasifikace v obraze je v moderní době řešena aplikací umělé inteligence. Umělé inteligenci rozumíme jako statistickému modelu, který na základě vstupu a vnitřních

proměnných určí hodnotu výstupu. Tento model je reprezentován pomocí umělých neuronových sítí. [19]

2.4.1 Základní pojmy a rozdělení strojového učení

V této podkapitole jsou popsány základní způsoby rozdělení strojového učení, a to z hlediska výstupní hodnoty a z hlediska přístupu k učení.

Učení s učitelem, učení bez učitele a zpětnovazební učení

Učení lze chápat jako proces získávání schopností nebo vědomostí. Samotné učení můžeme rozdělit do tří kategorií [29]:

1. Učení s učitelem, kde předpokládáme přístup ke správným odpovědím na vstupní otázky.
2. Učení bez učitele reprezentuje učení, kde je samostatně modelem ve vstupních datech vyhledáván vzor, pomocí kterého model rozhoduje o výstupu.
3. Zpětnovazební učení aplikující učení typu *„pokus omyl“* (v angličtině *trial and error*) ve svém prostředí.

V rámci řešení práce je využito učení s učitelem, proto v následujících kapitolách je věnován obsah právě této problematice.

Úloha regrese a úloha klasifikace

Rozsah aplikací strojového učení je jednoznačně velmi široký, pro řešení těchto problémů jsou vyvinuty dvě hlavní šablony [30]:

1. Úloha regrese má cíl odhadnout výstupní hodnotu $y \in \mathbb{R}$ na základě vstupu x. Pro příklad může být uvedena úloha předvídání hodnoty akcií na burze na základě různých statistických informací o dané společnosti.
2. Klasifikace reprezentuje úlohu odhadu výstupní hodnoty $y \in \{0, \dots, n-1\}$ na základě vstupu x. Oproti regresi je výstupem diskrétní hodnota, která odpovídá popisu vstupu, třídíme tedy vstupní data do kategorií. Jako příklad lze uvést např. klasifikace dokumentů z hlediska jazyka psaného textu, kde výstupní hodnoty můžou být např.:

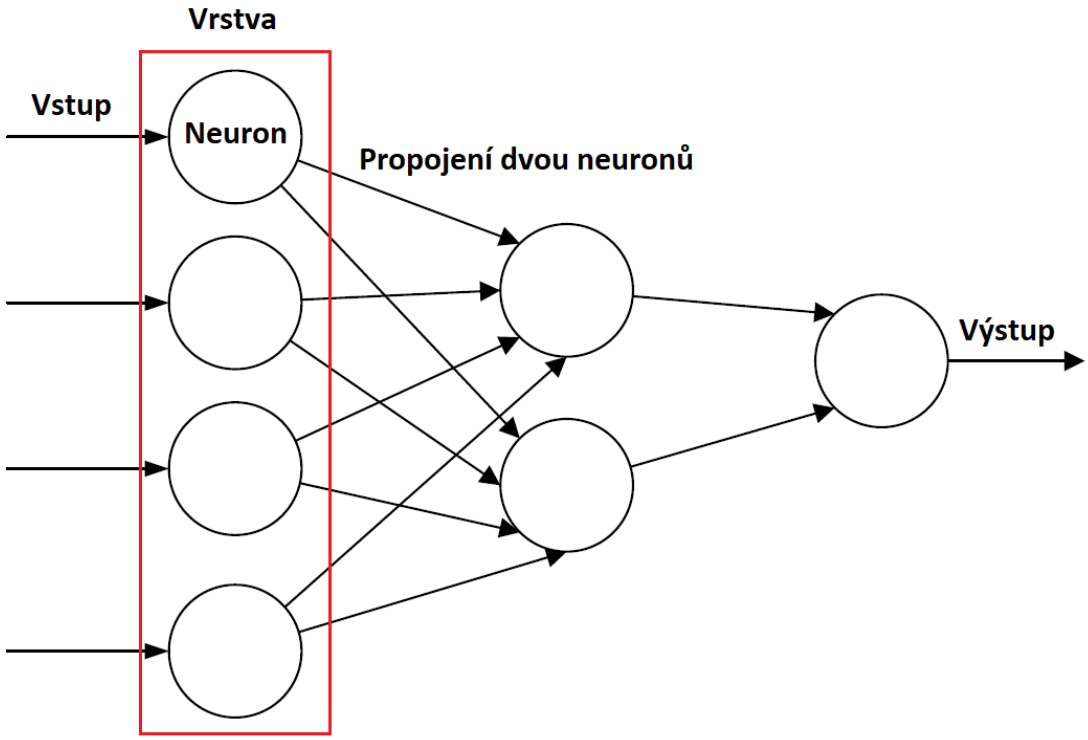
$$y \in \{Angličtina, Němčina, Francouzština, Španělština\}$$

2.4.2 Umělá neuronová síť

Umělé neuronové sítě (v angličtině *artificial neural network*, zkráceně ANN) byly vytvořeny za cílem modelovat lidský mozek a stále se jedná o důležitý prvek inspirace, v dnešní době se však už nejedná o *„návod“* jako tomu bylo v minulosti.

Tento ústup výzkumu neurovědy v oboru umělé inteligence je přisuzován nedostatku informací o funkčnosti lidského mozku. [19]

Umělá neuronová síť je soustava vnitřně propojených neuronů, které se inkrementálně učí vybírat ze vstupních dat podstatné lineární i nelineární trendy, tak aby umožňovaly spolehlivé odhady i za nových podmínek nebo za přítomnosti šumu či neúplných dat. Ve své podstatě se jedná o paralelní model, který lze rozložit na sériově zapojené bloky (vrstvy). [23]



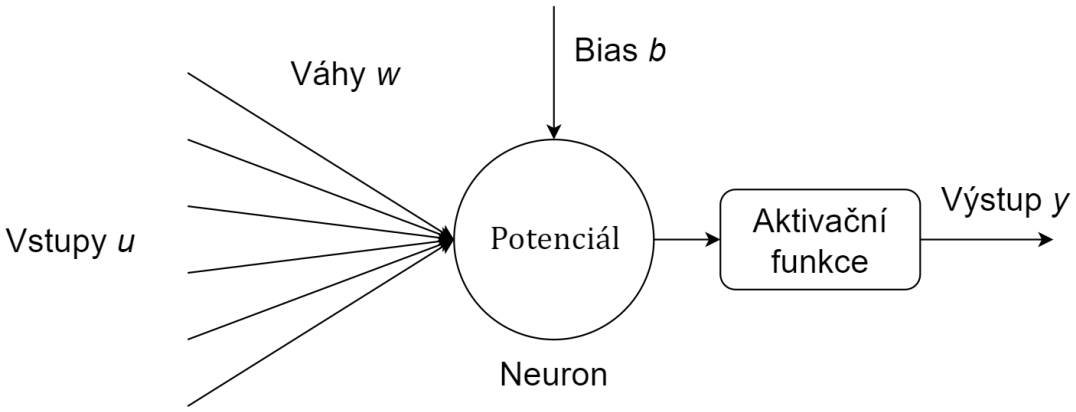
Obrázek 11 - Příklad jednoduché neuronové sítě [19]

Neuron je tedy jeden prvek neuronové sítě. Platí pro něj, že jeho vstup je vektor n hodnot a každý vstupní prvek má jinou váhu. Váha neuronu vyjadřuje jeho důležitost pro určení výstupu neuronové sítě. Hodnoty jednotlivých vah jsou dány učícím procesem neuronové sítě a jejich hodnoty společně se samotnou architekturou definují výstup neuronové sítě pro všechny vstupy. Vnitřní potenciál neuronu je dán vztahem (2.24) a tento potenciál je dále poslán jako výstup přes aktivační funkci do dalšího neuronu. [23]

$$\varphi = \sum_{i=0}^{n-1} u_i \cdot w_i + b \tag{2.24}$$

Kde φ je potenciál neuronu, n je počet vstupů do neuronu, u je vektor vstupů do neuronu, w je vektor vah pro jednotlivé vstupy do neuronu a b je ovlivnění

neuronu (v angličtině *bias*), tato hodnota je stejná pro všechny vstupy do stejného neuronu a přidává neuronu *„trénovatelnou“* hodnotu, tedy přidává síti flexibilitu.



Obrázek 12 - Neuron, jeho vstupy a výstupy

2.4.3 Učení neuronové sítě

Proces učení neuronové sítě je založen na postupném ukazováním předpřipravených dat z datasetu neuronové síti. Síť si poté během každé iterace upravuje své vnitřní proměnné (váhy a bias jednotlivých neuronů), tak aby dokázala obecně mapovat vstupní data k jejich správnému označení.

Prvním krokem procesu učení neuronové sítě je nastavení vah všech neuronů v síti na náhodnou hodnotu ve standardizovaném rozsahu (např. v rozsahu $\langle -0.3, 0.3 \rangle$). Následuje proces zpracování sady vstupních dat, kde každá vrstva neuronové sítě aplikuje na data svou transformaci. Výsledný odhad sítě je porovnán se správnou odpovědí a je vypracována takzvaná ztráta (v angličtině *loss*). Jedná se o míru *„chybnosti“* sítě, ideálně se snažíme, aby její hodnota v průběhu učení konvergovala k nule. Procesem zpětného výpočtu je vypočtena hodnota ztráty pro každý vnitřní parametr sítě. Ztrátová funkce (v angličtině *loss function*) je pak závislost hodnoty ztráty na vnitřních parametrech sítě. Pomocí ztrátové funkce je poté vypočítána hodnota gradientu pro každou hodnotu ztráty v síti. Výpočtem nových vnitřních proměnných sítě se snažíme o pokles gradientu ztrátových funkcí, tak aby dosáhly svého globálního minima (např. velmi známá metoda výpočtu vnitřních proměnných *Stochastic Gradient Descend SGD* [28]). Tyto změny gradientů v síti v závislosti na čase nebo spíše v závislosti na expozici vstupních dat můžeme označit jako takzvaný tok gradientů. Po ukončení procesu expozice vstupních dat síti, je síť definována svými vnitřními proměnnými a proces učení je ukončen. [27]

2.4.4 Vrstvy neuronových sítí

Umělé neuronové sítě lze rozložit na jednotlivé vrstvy, mezi kterými se přenášejí data. Jako vrstvu považujeme vstup do sítě, bloky sloužící pro zpracování vstupních dat a výstup ze sítě. Vzhledem k různorodosti využití umělé inteligence existuje velké množství bloků sloužící pro zpracování dat, proto jsou v rámci této práce popsány pouze bloky použité v rámci řešení práce.

Vstupní a výstupní vrstvy

Vstupní a výstupní bloky jsou také označovány jako viditelné bloky, protože jsou to jediné bloky jejichž vnitřní proměnné nejsou skryty [19]. Jako příklad můžeme uvést klasifikaci obrazu. Na vstupu neuronové sítě je přiveden snímek z kamery a výstupem ze sítě je pravděpodobnostní rozdělení pro všechny kategorie.

Obrázek 13 - Příklad klasifikace snímku

Plně propojené vrstvy

Mimo vstupní a výstupní bloky neuronové sítě jsou používané také takzvané bloky skryté, tedy bloky jejichž vnitřní proměnné jsou skryté. Základním skrytým blokem je plně propojený blok, který obsahuje *n* neuronů, které jsou propojené s každým neuronem v předchozí vrstvě. Tento blok je často používán například pro mapování zpracovaných dat v síti k výstupu. Jedná se o takzvaný *„brute force“* blok, jehož výpočetní náročnost je obecně vysoká a vysoce závislá na počtu vstupních neuronů. [20]

Konvoluční vrstvy

Konvoluční vrstva je jedna z nejdůležitějších vrstev neuronových sítí v aplikaci počítačového vidění. Tato vrstva umožňuje extrakci různých vlastností obrazu pomocí operace konvoluce. Konvoluční vrstva je složena z N konvolučních filtrů o velikosti M×M (často 3×3 nebo 5×5, ale také 1x1 nebo 7×7) a specifikovaným krokem (v angličtině *stride*). Krok definuje frekvenci výběru pixelů, pro které je konvoluce počítána. Pokud je krok roven jedné, je konvoluce počítána z každého pixelu na vstupu a výstupní obraz z bloku má stejnou dimenzionalitu jako jeho vstupní obraz, pokud je krok roven dvěma, je konvoluce počítána pouze pro každý druhý pixel a výstupní obraz z bloku je dvakrát menší v prvních dvou dimenzích, než obraz vstupní. Výstupem z konvoluční vrstvy je tedy M obrazů, které jsou označovány jako mapy vlastností (v angličtině *feature map*). [21]

Shromažďující vrstvy

Často navazující na konvoluční vrstvy jsou shromažďující vrstvy (v angličtině *pooling layers*). Tyto bloky mají za cíl zmenšit vstupní obraz. Každý vstupní obraz je rozdělen na mřížku s buňkou o velikosti M×M (často 2×2) a na každé buňce mřížky je aplikována matematická funkce. Shromažďující vrstvy jsou děleny podle matematické funkce, kterou aplikují, např. *MaxPooling* aplikuje vyhledávání maxima na buňku nebo *AveragePooling* aplikuje průměrování na buňku. Jak už bylo zmíněno, tyto bloky často navazují na konvoluční vrstvy, a to za cílem snížit velikost map vlastností, snížení následné výpočetní náročnosti a zvýraznění důležitých vlastností obrazu. [21]

Vypadávající vrstvy

Vypadávající vrstvy (v angličtině *dropout layer*) jsou aktivní pouze v procesu učení sítě a způsobují přepsání svého vstupu na nulu s pravděpodobností P, v případě že vstup nebyl přepsán, je vstup zvýšen podle rovnice 2.30. Smysl využití tohoto bloku je v potlačení závislosti výstupu na minimálním počtu neuronů. [22]

$$Výstup = Vstup \cdot \frac{1}{1-P} \tag{2.30}$$

Kde P je pravděpodobnost *„vypadnutí“* neuronu a $P \in \langle 0;\ 1)$.

Normalizační vrstvy

Normalizační vrstvy *(Batch Normalization)* provádí takovou transformaci na svůj vstup, aby její výstup měl průměrnou hodnotu nula a směrodatnou odchylku blízko

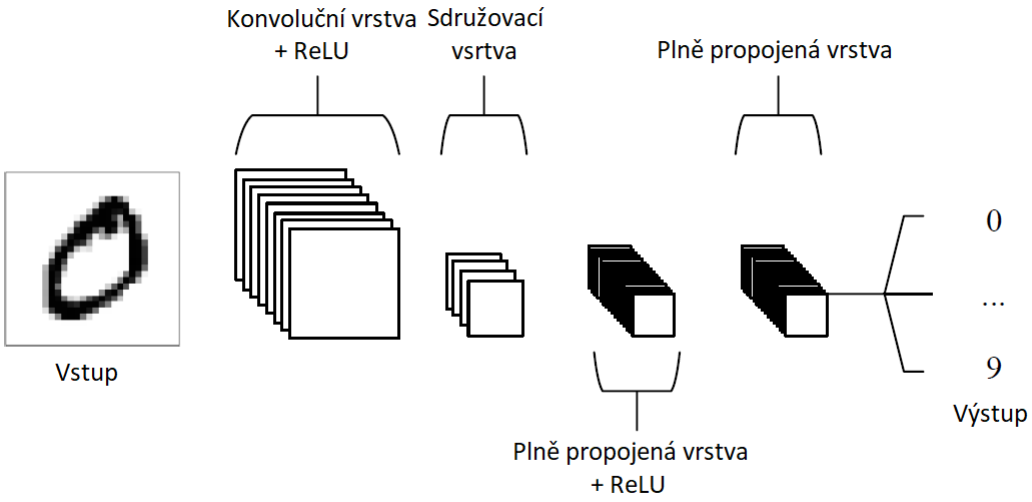
hodnotě jedna. Tato vrstva pomáhá dalším vrstvám učit se nezávisle na předchozích vrstvách a zrychluje proces učení. [23]

2.4.5 Konvoluční neuronová síť

Konvoluční neuronové sítě (CNN) jsou analogické tradičním umělým v tom smyslu, že jsou složeny z neuronů, které jsou optimalizovány procesem učení. Zásadním rozdílem mezi konvoluční neuronovou sítí a tradiční umělou neuronovou sítí spočívá ve využití CNN v oboru zpracování obrazu, speciálně v rozpoznávání a detekci objektů na scéně snímku. [21]

Tradiční architektura CNN spočívá v použití vrstev konvolučních, plně propojených a shromažďujících. Zjednodušená stavba architektury MNIST sloužící pro rozpoznávání ručně psaných číslic je vyobrazena na obrázku 18. Tuto architekturu můžeme rozebrat na čtyři klíčové části. [21]

1. Vstupní vrstva obsahuje pixelové hodnoty vstupního snímku
2. Konvoluční vrstva vypočte konvoluci vstupního obrazu a svých konvolučních kernelů, výstupem je mapa vlastností snímku.
3. Sdružovací vrstva aplikuje podvzorkování svého vstupu a tím redukuje počet parametrů sítě.
4. Plně propojené vrstvy se následně pokusí namapovat svá vstupní data na výstupní kategorie.



Obrázek 14 - Ukázka architektury CNN [21]

2.4.6 Aktivační funkce

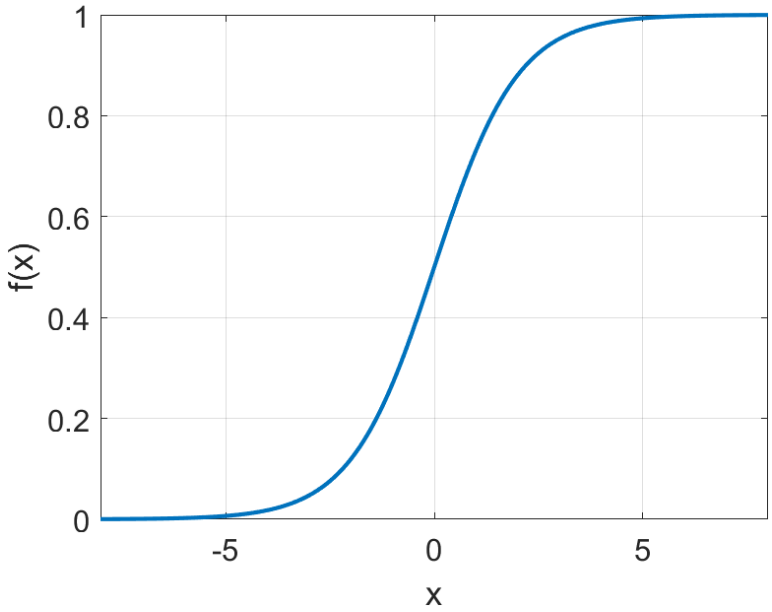
Aktivační funkce, již zmíněné v podkapitole 2.4.2, jsou nelineární transformace aplikované na potenciály neuronů, výsledek této transformace je následně poslán

do dalšího neuronu. Z rovnice 2.24 jde vidět, že neuronová síť je bez aktivačních funkcí pouze série lineárních rovnic, samotná relace mezi vstupem a správným výstupem je ale nelineární, aktivační funkce přináší do neuronových sítí nelinearitu, která pomáhá mapovat vstup ke správnému výstupu. [24] Mimo fakt, že tyto funkce pomáhají mapovat vstup na výstup, má každá funkce své výhody a nevýhody. Aktivačních funkcí je v dnešní době celá řada, proto bude v této kapitole věnována pozornost pouze určeným důležitým nebo moderním funkcím.

Sigmoid

Tato velmi známá a historicky velmi používaná aktivační funkce daná vztahem 2.25 je v dnešní době používána pro výstupní neurony pro binární rozhodnutí (např. možné využití v modelu který rozhoduje, jestli je na obrázku kočka nebo pes), její použití pro aktivaci v jiných vrstvách než výstupních, se nedoporučuje, protože funkce způsobuje pomalou konvergenci učení a saturaci gradientu při učení. Výhody této funkce jsou její jednoduchost a její jednoduchá derivace. [24]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.25}$$

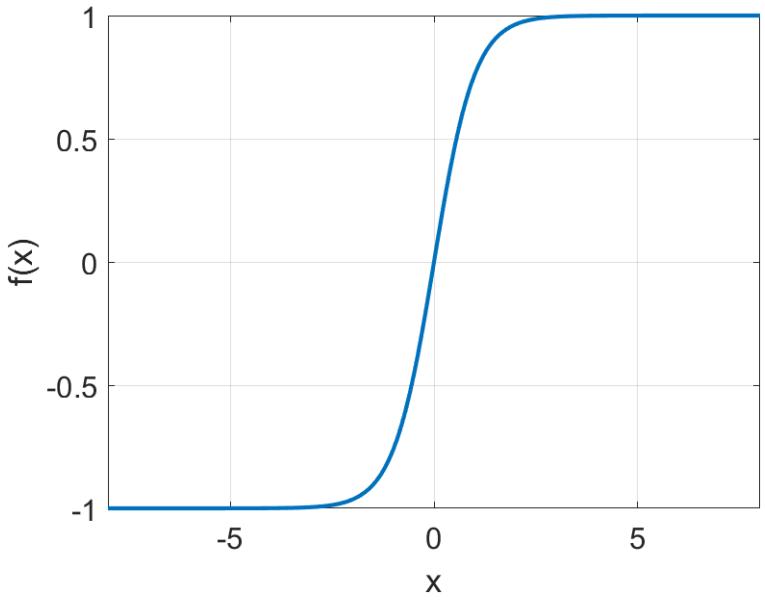


Obrázek 15 - Aktivační funkce Sigmoid

Hyperbolický tangens

Hyperbolický tangens (14) je preferovaná verze sigmoidu, protože je centrovaný kolem nuly a obsahuje silnější gradienty, tedy učení konverguje rychleji. Nevýhoda stále zůstává saturace gradientu při učení. [24]

$$f(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.26}$$



Obrázek 16 - Aktivační funkce hyperbolický tangens

Softmax

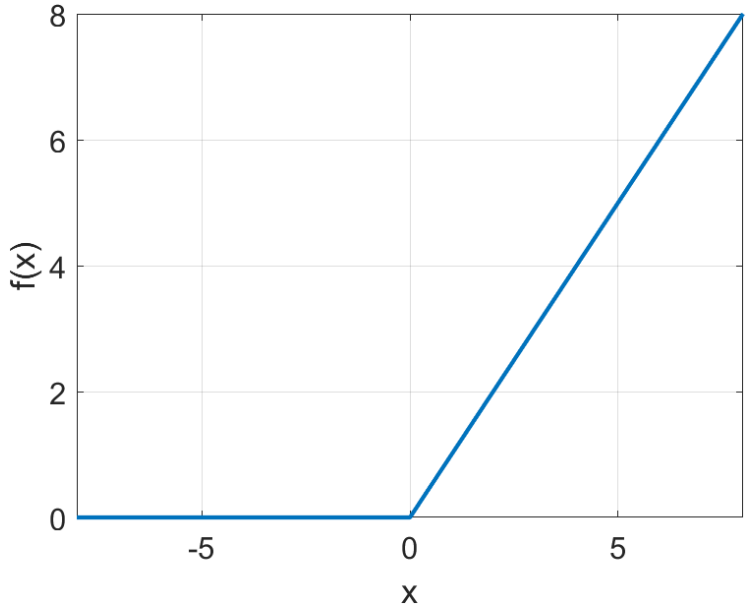
Softmax, definovaný podle vztahu (2.27), je aktivační funkce používaná v procesu klasifikace pro rozdělení pravděpodobností na výstupu neuronové sítě mezi všechny možné kategorie. V dnešní době využívaná aktivační funkce ve většině moderních klasifikátorech. [24]

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.27}$$

Rectified Linear Unit

Ve zkratce ReLU (15) je moderní aktivační funkce, která umožňuje optimální ladění procesu učení a rychlou konvergenci gradientu. V dnešní době se jedná o jednu z nejrozšířenějších aktivačních funkcí. Její výhodou je také minimální výpočetní zatížení. Nevýhodou je náchylnost k přeučení (v angličtině *overfitting*), tedy stavu, kdy se model začíná učit „nazpaměť" data, na kterých se učí a ztrácí schopnost pracovat v novém prostředí. Velkou nevýhodou je občasné „umírání" gradientů, tedy stav, kdy bude výstup z neuronu vždy nula, nezávisle na vstupu. [24]

$$f(x) = \max(x, 0) \tag{2.28}$$

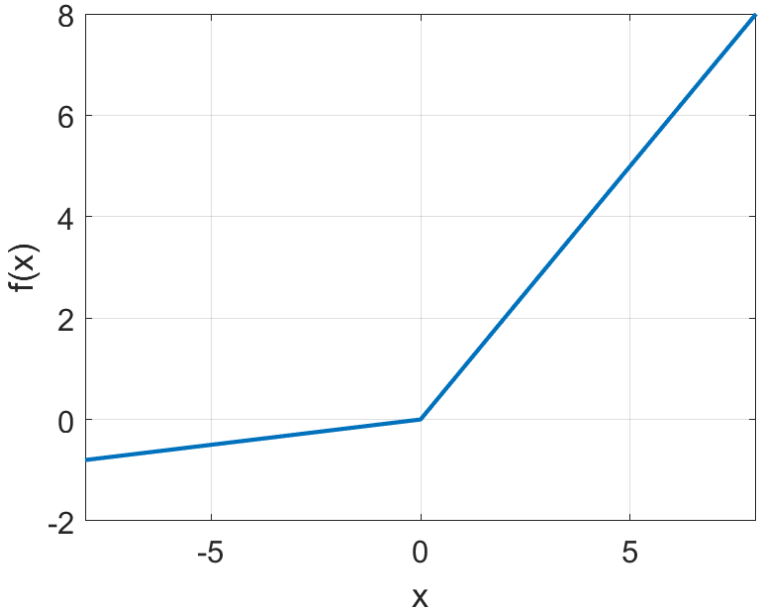


Obrázek 17 - ReLU aktivační funkce

Leaky Rectified Linear Unit

Modifikace ReLU aktivační funkce Leaky ReLU (16) byla vytvořena za cílem vyřešit hlavní problém ReLU funkce, a to umírání gradientů. Řešením je úpravou hodnot *f(x)* pro $x < 0$, viz. rovnice 2.29. Gradienty se mohou po výpadku znovu „oživit“.

$$f(x > 0) = x; \quad f(x \leq 0) = \alpha x \tag{2.29}$$



Obrázek 18 - Aktivační funkce Leaky ReLU pro α = 0.1

2.5 Geometrické transformace

Objekty jsou popsány svými souřadnicemi, které jsou vztaženy vzhledem ke svému souřadnicovému systému. Geometrické transformace mohou být aplikovány na jednotlivé souřadnice objektu, který tak mění svoji polohu. Transformujeme tedy bod *P*, který má kartézské souřadnice $[X, Y, Z]^T$, na bod *P'* o souřadnicích $[X', Y', Z']^T$. [1]

2.5.1 Translace

Translace nebo také posunutí bodu *P* je určena vektorem translace ve tvaru: $[X_t\ Y_t\ Z_t]^T$ nebo také maticí translace [1] v homogenním tvaru:

$$T(X_t, Y_t, Z_t) = \begin{bmatrix} 1 & 0 & 0 & X_t \\ 0 & 1 & 0 & Y_t \\ 0 & 0 & 1 & Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

Kde X_t je translace bodu podél osy *x*, Y_t je translace bodu podél osy *y* a Z_t je translace podél osy *z*.

2.5.2 Rotace

Rotace nebo také otáčení bodu *P* kolem soustavy souřadnic [1]. Rotace okolo os *x*, *y* a *z* jsou zapsány jako homogenní matice v tomto tvaru:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sin\gamma & 0 & 0 \\ sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.34}$$

Kde α, β a γ jsou rotace kolem os *x*, *y* a *z* respektivě.

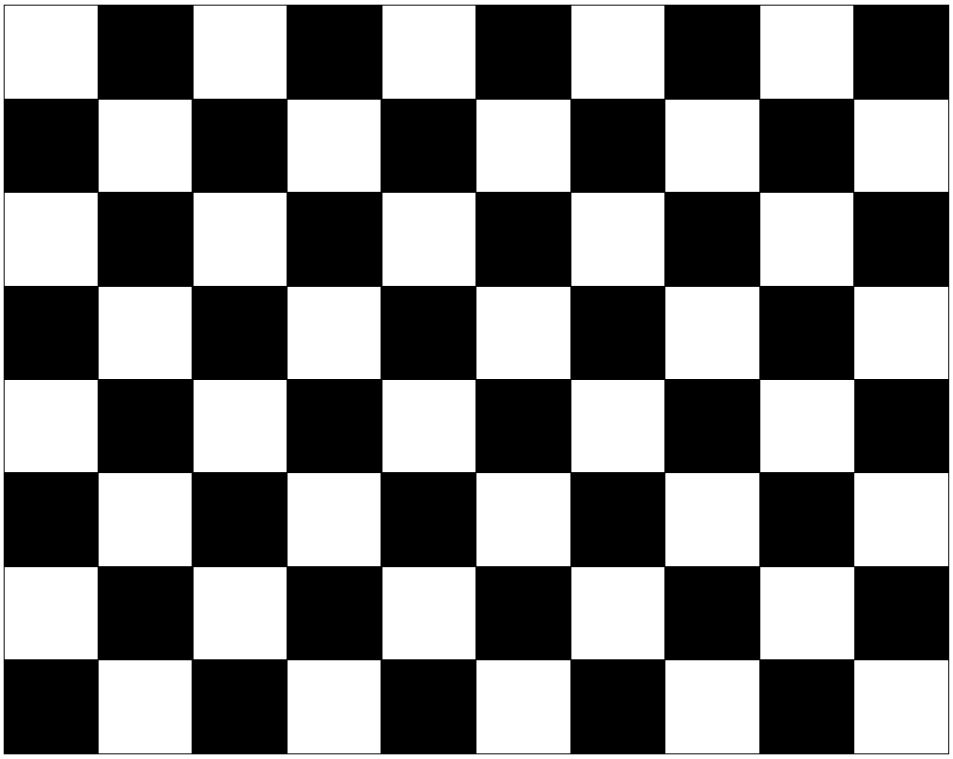
2.6 Parametry kamery

V počítačovém vidění jsou trojrozměrné geometrické informace o snímaných objektech získávány pomocí kamer. Vztah mezi trojrozměrnou polohou objektů ve světě a dvourozměrnou polohou na snímku je dán parametry kamery. Ve většině případů jsou tyto parametry zjištěny experimentem a následným výpočtem, tento proces je nazýván kalibrace kamery. [2]

2.6.1 Kalibrace kamery

Metod pomocí, kterých jsou kamery kalibrovány je mnoho, v následující části je popsána tradiční kalibrace použitím kalibrační aparatury. Mezi další metody kalibrace kamery patří např. metoda samostatné kalibrace nebo metoda založená na aktivním vidění. [2]

Tradiční metoda kalibrace kamery je založena na použití kalibrační aparatury. Tato aparatura obsahuje známou strukturu referenčních bodů, tedy i známou vazbu mezi těmito body ve 2D a 3D prostoru. Ze znalosti těchto referenčních vazeb je možno poté ze snímků struktury vypočítat pomocí optimalizačních algoritmů parametry kamery.[2]



Obrázek 19 - Standartně používanou strukturou pro kalibraci je šachovnice [3]

2.6.2 Vnitřní parametry kamery

Vnitřní parametry kamery popisují vnitřní optické vlastnosti kamery. Tyto parametry jsou vyjádřeny pomocí matice vnitřních parametrů kamery a vektoru koeficientů zkreslení. [16]

Matice vnitřních parametrů kamery

Tato matice [16] je vyjádřena jako:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.35}$$

Kde c_x a c_y vyjadřují optický střed kamery v pixelech, f_x a f_y vyjadřují ohniskovou vzdálenost kamery v pixelech a s vyjadřuje skosení pixelu, tedy jestli jsou osy x a y vzájemně kolmé (skos je roven nule, pokud jsou osy vzájemně kolmé).

Koeficienty zkreslení

Koeficienty zkreslení vyjadřují optické nedokonalosti čoček kamery. Vektor koeficientů zkreslení je v knihovně *OpenCV*, která je využita pro řešení práce [16] implementován jako:

$$d = [k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6] \tag{2.36}$$

Kde k_1 – k_6 jsou radiální koeficienty zkreslení a p_1, p_2 jsou tangenciální koeficienty zkreslení.

Radiální koeficienty zkreslení

Radiální zkreslení se projevuje, když se světelné paprsky ohýbají více na okrajích čočky. Vzniká tím zkreslení soudkovité nebo poduškovité. Toto zkreslení je vyjádřeno jako:

$$x' = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad [px] \tag{2.37}$$
$$y' = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad [px] \tag{2.38}$$

Kde x'/y' jsou zkreslené souřadnice pixelů, x/y jsou souřadnice pixelů po odstranění zkreslení, k_1 – k_3 jsou radiální koeficienty zkreslení a r je souřadnice pixelu daná rovnicí 2.39.

$$r^2 = x^2 + y^2 \quad [px] \tag{2.39}$$

Kde *x* a *y* jsou nezkreslené souřadnice pixelu.

Obrázek 20 - Radiální zkreslení. Zleva: bez zkreslení, soudkovité, poduškovité [16]

Tangenciální koeficienty zkreslení

Tangenciální zkreslení [17] se projevuje, když není čočka kamery paralelní s rovinou snímku. Tangenciální zkreslení je vyjádřeno jako:

$$x' = x + [2 \cdot p_1 \cdot x \cdot y \cdot (r^2 + 2 \cdot x^2)] \tag{2.40}$$
$$y' = y + [p_1 \cdot (r^2 + 2 \cdot y^2) \; + \; 2 \cdot p_2 \cdot x \cdot y] \tag{2.41}$$

Kde *x'* a *y'* jsou zkreslené souřadnice pixelů, *x* a *y* jsou souřadnice pixelů po odstranění zkreslení, p_1 a p_2 jsou tangenciální koeficienty zkreslení a *r* je souřadnice pixelu daná rovnicí 2.39.

2.6.3 Vnější parametry kamery

Maticí vnějších parametrů [16] kamery se rozumí matice *[R|t]*, tedy matice vyjadřující polohu kamery ve světovém souřadnicovém systému. Matice je výsledkem spojením rotační matice v nehomogenním tvaru s translačním vektorem v nehomogenním tvaru (2.42).

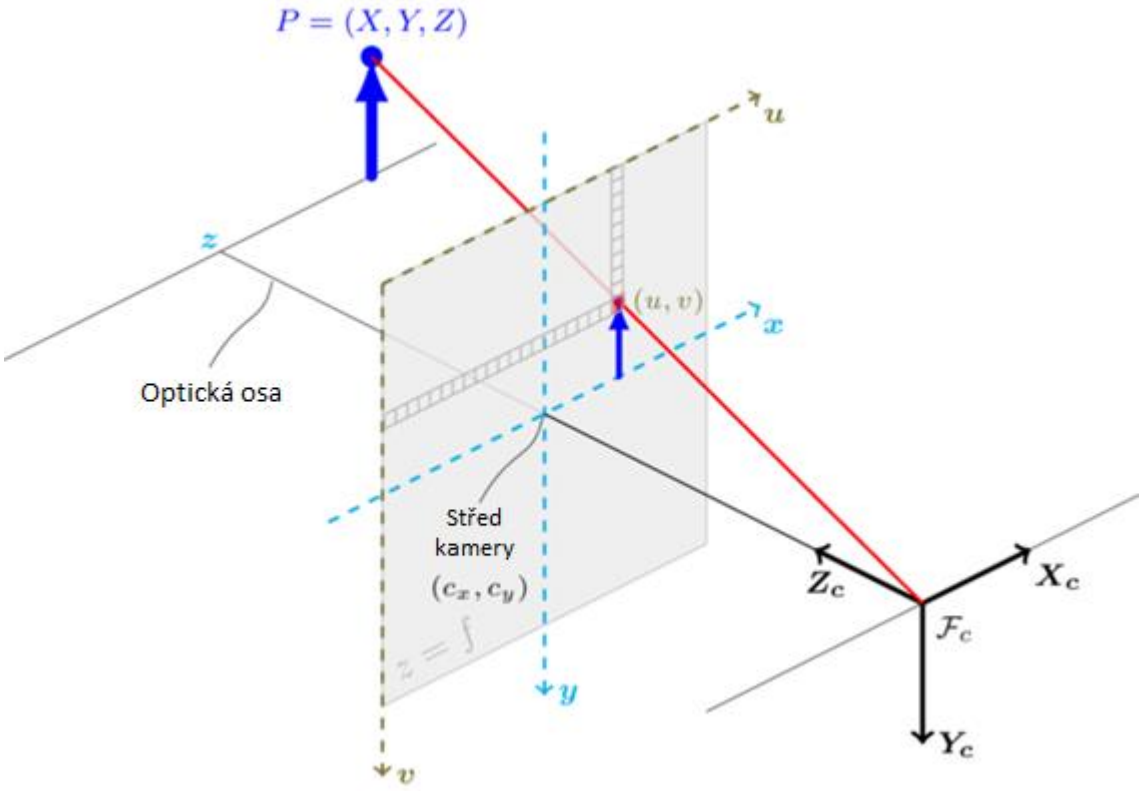
$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{2.42}$$

2.6.4 Model kamery

Ze znalosti parametrů kamery lze sestrojit *„pinhole“* model kamery [16]. Model vyjadřuje závislost mezi souřadnicemi ve 2D prostoru snímku a 3D souřadnicemi světa. Model je matematicky vyjádřen jako:

$$s \cdot p' = K \cdot [R|t] \cdot P' \tag{2.43}$$

Kde s (z anglického *scale*) vyjadřuje přiblížení nebo oddálení snímku, p' jsou 2D souřadnice snímku v homogenním tvaru $[u, v, 1]^T$, K je matice vnitřních parametrů kamery, $[R|t]$ je matice vnějších parametrů kamery a P' jsou 3D světové souřadnice v homogenním tvaru $[X_w, Y_w, Z_w, 1]^T$.



Obrázek 21 – *„pinhole“* model kamery [16]

2.7 Stanovení pozice kamery

V této kapitole je popsána problematika stanovení pozice kamery vůči určenému světovému souřadnicovému systému na základě znalosti o poloze klíčových bodů ve 2D prostoru snímku a ve 3D prostoru světových souřadnic. Tento problém se nazývá Perspektiva z N bodů (v angličtině *Perspective-n-point problem*, zkráceně *PnP*) [18].

Řešení problému vychází z *„pinhole“* modelu kamery, kde hledáme hodnoty matice vnějších parametrů kamery *[R|t]*. Předpokládá se, že kamera je kalibrovaná a je známa matice vnitřních parametrů kamery. Pro n známých bodů, tedy vzniká soustava n rovnic (2.44). Pozice kamery obsahuje 6 stupňů volnosti (tři složky rotace a tři složky translace), je tedy nutné pro řešení problému znát alespoň 3 páry souřadnic klíčových bodů, aby bylo dosaženo konečného počtu řešení. Řešení této soustavy rovnic je navrženo mnoho. Nejčastěji se jedná o řešení vyžadující $n > 3$ (např. řešení *EPnP* [14]), ale jsou také řešení pracující pouze s $n = 3$ (např. řešení *P3P* [13]). Obecně také platí, že pokud je počet řešení pro matici *[R|t]* větší než jedné, je nutno dalším zpracováním určit správné řešení. [18]

$$\begin{aligned} s \cdot p'_0 &= K \cdot [R|t] \cdot P'_0 \\ s \cdot p'_1 &= K \cdot [R|t] \cdot P'_1 \\ &\cdots \\ s \cdot p'_{n-1} &= K \cdot [R|t] \cdot P'_{n-1} \end{aligned} \tag{2.44}$$

Kde s vyjadřuje přiblížení nebo oddálení snímku, p' je vektor 2D souřadnic známých bodů ve snímku v homogenním tvaru, K je matice vnitřních parametrů kamery, *[R|t]* je hledaná matice vnějších parametrů kamery a P' je vektor 3D světových souřadnic známých bodů v homogenním tvaru.

3 NÁVRH METOD PRO OVLÁDÁNÍ KURZORU POMOCÍ RUKOU

V této kapitole je popsán návrh jednotlivých metod. Obě metody zajišťují ovládání kurzoru na monitoru, a to buď aktivním pohybem kamery nebo samotným souřadnicovým systémem ve formě aparatury se značkami.

3.1 Použité značky

Značky slouží jako záchytné body v prostoru. Řešení bylo navrženo pro dva druhy značek. Jako první použité značky byly zvoleny QR kódy a následně byly navrženy vlastní značky.

Velikost značek hraje roli při jejich detekci a má také vliv na přesnost určení pozice. Menší značka má praktické výhody v tom, že neomezuje příliš okolní prostor, avšak za cenu obtížnější detekce a nižší přesnosti určení pozice kamery. Nižší přesnost určení pozice kamery při použití rozměrově menších značek můžeme odvodit, tím že pokud budeme značky zmenšovat, tak každý roh značky bude mít při řešení PnP problému menší váhu, tím efektivně redukujeme počet bodů, které přispívají k výpočtu. Volba velikosti značek je tedy kompromisem mezi požadovanou přesností a praktickou použitelností značek.

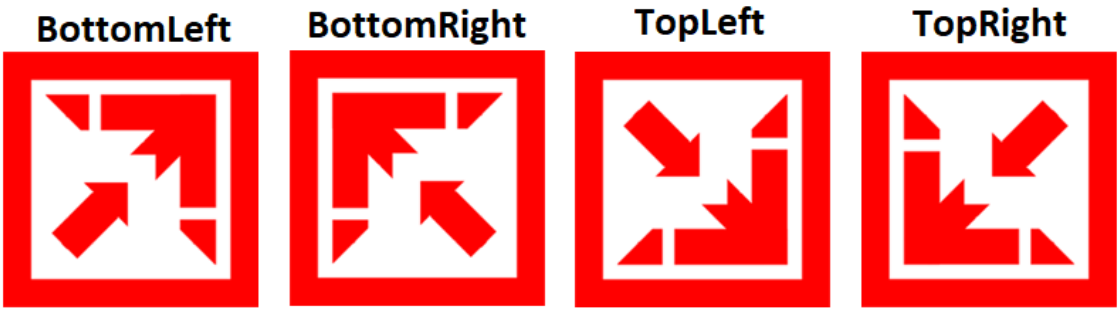
3.1.1 QR kódy

Pro jednoduchost použití byly zvoleny jako značky QR kódy, které v sobě mají zapsanou informaci o její poloze např. *„TopLeft“* pro značku nacházející se vlevo nahoře.

Obrázek 22 – Ukázka QR kódů [4]

3.1.2 Vlastní značky

Jakožto vlastní značky byly navrženy směrově univerzální ukazatele. Ukázka značek je vyobrazena na obrázku 23. Smyslem použití těchto směrově univerzálních značek je jednoduchost jejich použití v praxi.



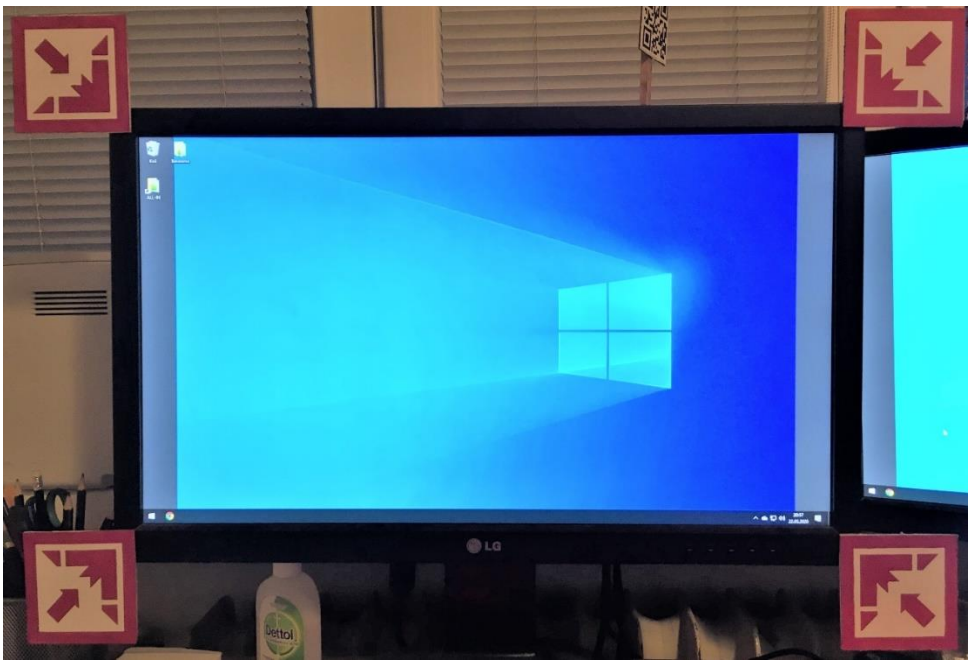
Obrázek 23 – Ukázka vlastních značek

3.2 Návrh metody A – značky umístěné na monitoru

První metoda využívá značky umístěné na monitoru a pohyblivou kameru v prostoru, která slouží jako naváděcí nástroj pro kurzor.

3.2.1 Návrh měřicího prostředí

Měřící prostředí bude složeno z monitoru, na kterém jsou umístěny čtyři značky, uspořádání prostředí je znázorněno na obrázku 24.

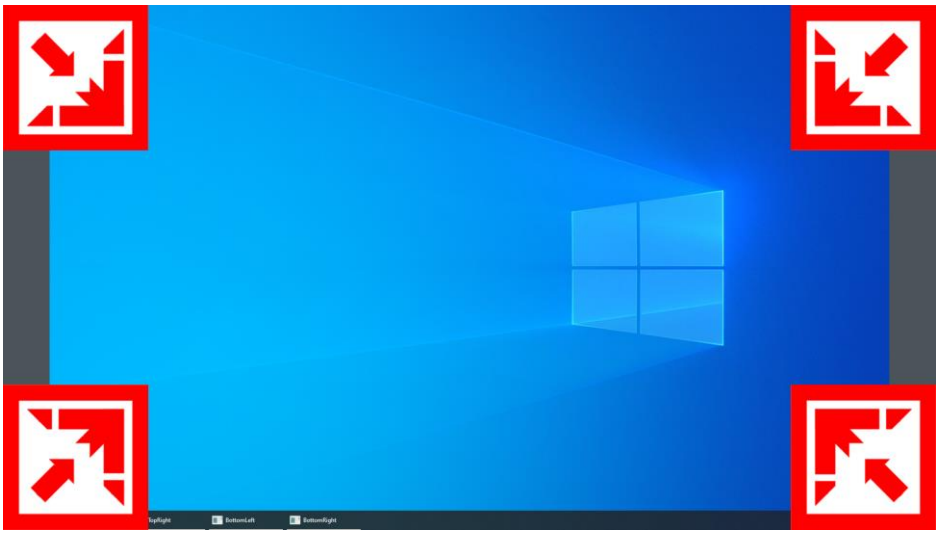


Obrázek 24 - Ukázka návrhu měřícího prostředí pro metodu A

Měřící prostředí je sestaveno tak, aby vnitřní obdélník tvořený značkami realizoval měřící plochu. Pokud uživatel ukáže kamerou na jeden z rohů tohoto obdélníka, kurzor by se měl přesunout do odpovídajícího rohu obrazovky.

Pro případy, kdy by nebylo možné fyzicky umístit značky na monitor byla připravena desktopová aplikace, která umístí značky na popředí obrazovky

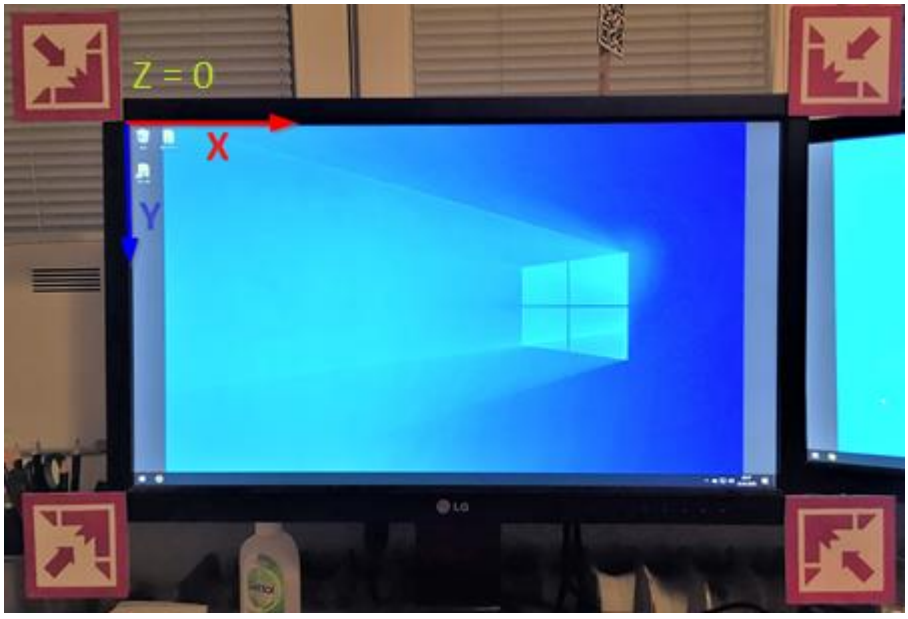
(obrázek 25). Uživatel tedy může za cenu snížení pracovní plochy na obrazovce ušetřit fyzické místo kolem obrazovky.



Obrázek 25 - Desktopová aplikace pro umístění značek na obrazovku

3.2.2 Souřadnicový systém

Pro orientaci v prostoru je nutno zavést světový koordinační systém. Systém je obdobou kartézského systému souřadnic a je zvolen, tak že počátek systému je v levém horním rohu vnitřního obdélníka tvořeného značkami. Hodnoty osy *x* stoupají směrem vpravo, hodnoty osy *y* stoupají směrem dolů a hodnoty osy *z* rostou směrem ke kameře. Uspořádání os je znázorněno na obrázku 6. Jako jednotky systému jsou zvoleny centimetry.



Obrázek 26 - Ukázka počátku a orientace os koordinačního systému u metody A

3.3 Návrh metody B – kamera umístěná na monitoru

Metoda využívá stacionárně umístěnou kameru na monitoru. Předpokládá se, že se kamera nachází na standartní poloze umístění web kamer – nad středem obrazovky. Pohybem značek, které jsou umístěné na pomocné aparatuře dojde k posunutí kurzoru. Ovládání kurzoru pomocí této aparatury je navrženo, tak že by se kurzor na obrazovce měl posunout na místo kam je aparaturou ukázáno.

3.3.1 Pomocná aparatura

Pomocná aparatura, která umožňuje pohyb souřadnicového systému v prostoru je vyobrazena na obrázku 27. Jedná se o dřevěnou konstrukci, na které jsou pevně připevněny značky. Aparatura ve své podstatě rozměrově tvoří kostru krychle o straně 30cm.

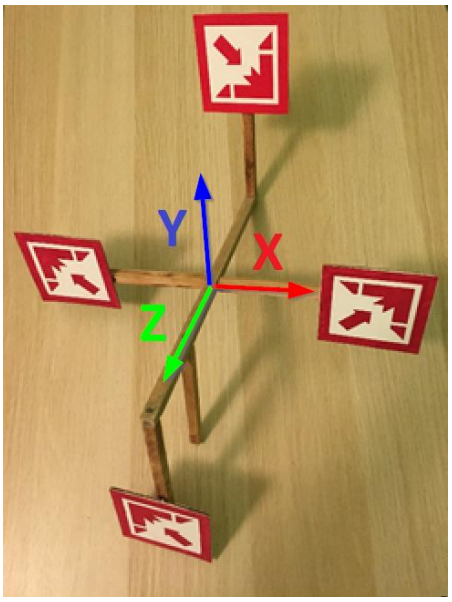
Aparatura byla navržena, aby značky byly rozmístěny ve všech třech souřadnicích (x, y a z), aby byla zvýšena přesnost určení pozice. Toto opatření je nutné z důvodu zmenšení plochy, na které jsou značky rozmístěny, oproti metodě A, kde jsou značky umístěny na monitoru. Zmenšení této plochy má za důsledek snížení přesnosti určení pozice, protože se body nacházejí blíže u sebe a chyba, určené polohy kamery způsobí menší chybu reprojekce bodů.

Obrázek 27 - Metoda B – Pomocná aparatura

3.3.2 Souřadnicový systém

Souřadnicový systém světa se odvíjí od pomocné aparatury pro přenos značek. Středem souřadnicových os je zvolen uprostřed aparatury, osa x stoupá směrem

vpravo, osa *y* stoupá směrem nahoru a osa *z* stoupá směrem ke kameře. Souřadnicový systém je vyobrazen na obrázku 28. Jako jednotky jsou zvoleny centimetry.



Obrázek 28 - Ukázka počátku a orientace os koordinačního systému u metody B

4 REALIZACE

V této kapitole jsou uvedeny nástroje použité buď pro vývoj nebo pro realizaci měřícího prostředí. Dále je zde rozebrán proces kalibrace kamery, proces detekce značek a proces stanovení pozice.

4.1 Použité nástroje

V této podkapitole jsou shrnuty nástroje použité pro realizaci.

4.1.1 Programovací jazyk

Jako programovací jazyk byl zvolen jazyk Python [5]. Jazyk byl zvolen z důvodu úspory času psaní kódu, která je jedna z jeho hlavních předností. Další možností byl jazyk C++, který nabízí rychlejší běh programu za cenu delší doby vývoje, což je z hlediska řešení práce nevýhodné z důvodu velkého množství samostudia potřebného pro porozumění problematice.

4.1.2 Programové vybavení

Jako vývojové prostředí bylo využito Microsoft Visual Studio Code. Microsoft Visual Studio Code je vysoce přizpůsobivé vývojové prostředí. Umožňuje instalaci pluginů, které umožňují podporu různých programovacích jazyků nebo upravují či přidávají funkcionalitu. Jedná se o editor vyvíjený firmou Microsoft pro operační systémy Microsoft Windows, Linux a macOS. [6]

Software použitý pro vytvoření 3D modelů určených pro 3D tisk je SOLIDWORKS. Jedná se o známý 3D CAD systém, který nabízí intuitivní a přehledné ovládání. [33]

4.1.3 Knihovny

Programovací jazyk Python podporuje instalací modulů, které slouží jako knihovny. Instalace těchto modulů je možná přes podpůrný program, který je instalován společně s Pythonem: *pip*. Instalace je provedena zadáním příkazu *pip install <nazev_baliku>* do systémové konzole. Moduly jsou standartně publikovány na webových stránkách. [7]

V této kapitole budou uvedeny nejdůležitější knihovny použité v rámci řešení práce.

OpenCV

Knihovna určená pro počítačové vidění, zpracování obrazu a strojové učení. Knihovna obsahuje více než 2500 optimalizovaných algoritmů, tyto algoritmy

mohou být použity např. pro detekci a rozpoznání obličeje, identifikaci objektů, sledování pohybu kamery a objektů. Knihovna je psaná pro jazyk C++, ale byla přenesena pro použití v jiných jazycích např. Python, MATLAB, Java a JavaScript. OpenCV je podporováno operačními systémy Microsoft Windows, Linux a macOS. [8]

PyAutoGUI

Tento modul je určený pro jazyk Python a zajišťuje ovládání kurzoru a klávesnice. Knihovna je podporována operačními systémy Microsoft Windows, Linux a macOS. [9]

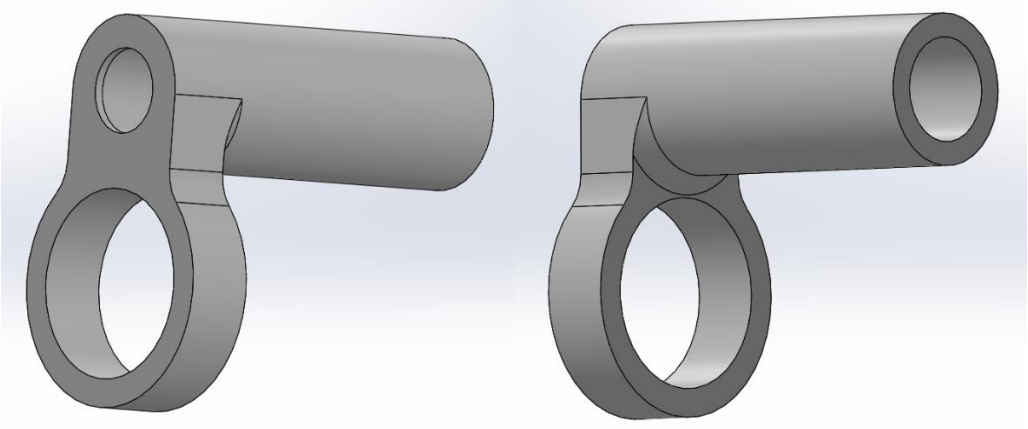
Tensorflow

Tensorflow [31] je open source platforma pro strojové učení a tensorové počty vyvíjena společností Google. Knihovna využívá Keras API [32]. Keras je taktéž knihovna určená pro strojové učení, Tensorflow tedy rozšiřuje možnosti použití Kerasu. Jedná se o jednu z předních knihoven využívaných ve světě pro strojové a hluboké učení.

4.1.4 Použité kamery

V rámci řešení práce byly použity dvě kamery.

První zvolenou kamerou je endoskop. Díky své malé velikosti j tato kamera vhodná pro využití při použití metody A. Pro zjednodušení manipulace s kamerou byl navržen a vytištěn model na 3D tiskárně (obrázek 29). Tento model umožňuje upevnění kamery na prstu a tím zjednodušuje manipulaci s kamerou. Na obrázku 30 je vyobrazen použitý endoskop a jeho uchycení ve vytištěném modelu.



Obrázek 29 - Model pro usnadnění manipulace s endoskopem

Obrázek 30 - Endoskop a vytištěný úchyt s endoskopem

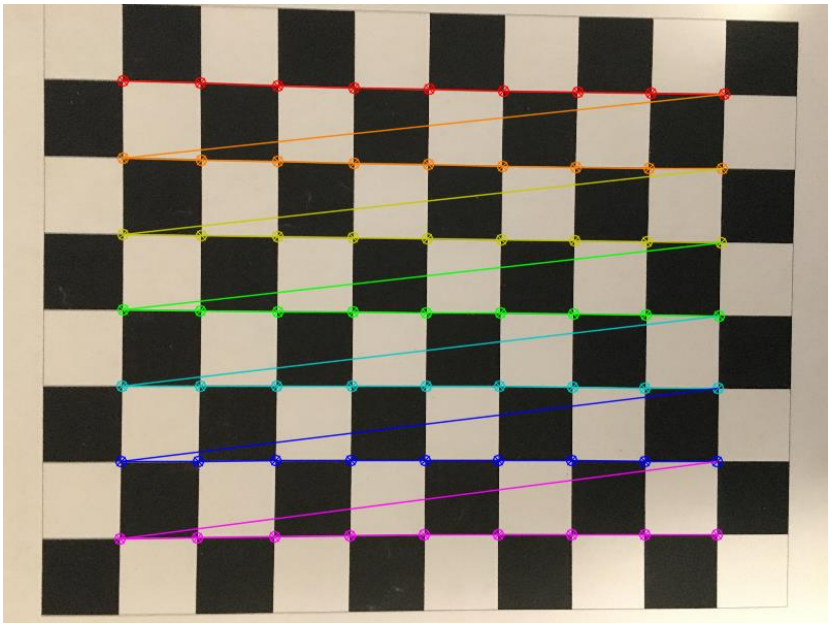
Druhou zvolenou kamerou je webkamera. Konkrétně se jedná o webkameru Creative Labs WebCam Live! Ultra [34].

Obrázek 31 - Webkamera Creative Labs Live! Ultra

4.2 Kalibrace kamery

Pro určení vnitřních parametrů kamer a koeficientů zkreslení kamer bylo nutno kamery kalibrovat. Kalibrace byla provedena na šachovnici vyobrazené na obrázku 19. Bylo tedy jednotlivými kamerami pořízeny video nahrávky šachovnice, a to tak aby se na různých snímcích nahrávky projevily všechny eulerovské úhly rotace a aby šachovnice na snímku zabírala co největší plochu.

Proces kalibrace se skládá z detekce klíčových bodů šachovnic ve všech snímcích a následné kalkulace. Klíčové body jsou na šachovnici takové body, kde se setkávají rohy dvou černých čtverců viz. obrázek 32.



Obrázek 32 - Klíčové body snímku šachovnice

Detekci šachovnicových rohů zajišťuje funkce knihovny OpenCV *findChessboardCorners()* [11] s těmito argumenty:

```
cv2.findChessboardCorners(
     image,          # snímek šachovnice
     patternSize[,   # počet vyhledávaných šachovnicových rohů
                     # v každém řádku a sloupci
     corners[,       # výstupní pole, kde se zapíší souřadnice
                     # nalezených bodů
     flags           # speciální nastavení funkce
]]) →
     retval,         # kontrolní výstup funkce
     corners         # souřadnice nalezených bodů
```

Jedním z argumentů funkce je také *flags*, tedy nastavení pro úpravu funkcionality funkce [11]. Tato nastavení mohou být:

- *CALIB_CB_ADAPTIVE_THRESH* – Konvertuje snímek do černé a bíle barvy, místo toho, aby byla pro rozlišování černých čtverců šachovnice použita vypočítaná průměrná hodnota jasu snímku.

- *CALIB_CB_NORMALIZE_IMAGE* – Před zpracováním snímku provede operaci ekvalizace histogramu pomocí funkce *equaizeHist()* [15], která normalizuje jas snímku a zvýší jeho kontrast.
- *CALIB_CB_FILTER_QUADS* – Umožňuje použití extra kritérií pro vyfiltrování špatně nalezených bodů.
- *CALIB_CB_FAST_CHECK* – Umožní provedení rychlé kontroly před vyhledáváním šachovnicových rohů, zdali je na snímku šachovnice. Může zrychlit proces kalibrace, pokud vstupní video neobsahuje pouze kalibrační strukturu.

V rámci řešení práce nebylo nutno využít nikteré ze zmíněných nastavení.

Po dokončení vyhledávání bodů ve všech snímcích šachovnice je proveden výpočet matice vnitřních parametrů kamery a koeficientů zkreslení kamery. Tento výpočet je realizován funkcí knihovny OpenCV *calibrateCamera()* [11] s těmito argumenty:

```
cv2.calibrateCamera(
     objectPoints,        # pozice bodů v reálných souřadnicích
                          # na šachovnici
     imagePoints,         # souřadnice bodů na snímcích
     imageSize[,          # velikost snímků v pixelech
     cameraMatrix[,       # vstupní matice vnitřních parametrů
                          # kamery
     distCoeffs[,         # vstupní koeficienty zkreslení
                          # kamery
     rvecs[,              # vektory rotace mezi snímky
     tvecs[,              # vektory translace mezi snímky
     flags[,              # speciální nastavení funkce
     criteria             # kritéria pro ukončení výpočtu při
                          # použití iterativního optimalizačního
                          # algoritmu
]]]]]]) →
     retval,              # kontrolní výstup funkce
     cameraMatrix,        # výstupní matice vnitřních parametrů
     distCoeffs,          # výstupní koeficienty zkreslení
     rvecs,               # vektory rotace mezi snímky
     tvecs                # vektory translace mezi snímky
```

Jedním z argumentů funkce je také *flags*, tedy nastavení pro úpravu funkcionality funkce [11]. Tato nastavení mohou být:

- *CV_CALIB_USE_INTRINSIC_GUESS* – Umožňuje použít počáteční odhad matice vnitřních parametrů ze vstupního argumentu *cameraMatrix.*
- *CV_CALIB_FIX_PRINCIPAL_POINT* – Optický střed kamery zůstane nastaven na střed obrazu nebo pokud je použit odhad matice vnitřních parametrů kamery, tak zůstane optický střed kamery nezměněn.
- *CV_CALIB_FIX_ASPECT_RATIO* – Při použití odhadu matice vnitřních parametrů kamery zůstane poměr složek ohniskové vzdálenosti f_x a f_y nezměněn. Pokud není použit odhad matice vnitřních parametrů kamery, tak správné hodnoty ohniskové vzdálenosti nejsou zapisovány do matice, je pouze zachován jejich poměr.

- *CV_CALIB_ZERO_TANGENT_DIST* – Koeficienty tangentního zkreslení p_1 a p_2 jsou nastaveny a zůstanou nulové.
- *CV_CALIB_FIX_K1, …, CV_CALIB_FIX_K6* – Adekvátní koeficient zkreslení (k_1 – k_6) je nastaven na nulu. Pokud je použit odhad matice vnitřních parametrů kamery, je koeficient nastaven na hodnotu adekvátního koeficientu zkreslení ve vstupním parametru *distCoeffs.*
- *CV_CALIB_RATIONAL_MODEL* – Povolí výpočet koeficientů zkreslení k_4, k_5 a k_6. Pokud toto nastavení není povoleno, jsou vypočteny pouze koeficienty k_1, k_2, p_1, p_2 a k_3.

V rámci řešení práce nebylo nutno využít nikteré ze zmíněných nastavení.

Argumentem funkce jsou také terminační kritéria *criteria*. Tato kritéria slouží pro ukončení výpočtu v krajních situacích, kdy dojde k dosažení příliš vysokých iterací při kalkulaci a je třeba program zastavit.

Po provedení kalibrace kamer byly získány matice vnitřních parametrů (kapitola 2.6.2 Vnitřní parametry kamery) K_A a koeficienty zkreslení d_A pro endoskop a matice vnitřních parametrů K_B a koeficienty zkreslení d_B pro webkameru.

$$K_A = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 813.05 & 0 & 294.25 \\ 0 & 814.14 & 269.16 \\ 0 & 0 & 1 \end{bmatrix}$$

$$d_A = [k_1, k_2, p_1, p_2, k_3] = [0.22377 \ -1.18527 \ -0.00033 \ -0.00238 \ 0.20183]$$

$$K_B = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 527.34 & 0 & 334.06 \\ 0 & 528.35 & 219.15 \\ 0 & 0 & 1 \end{bmatrix}$$

$$d_B = [k_1, k_2, p_1, p_2, k_3] = [-0.33414 \ 0.22943 \ 0.00036 \ 0.00084 \ -0.14249]$$

4.3 Detekce a dekódování QR kódů

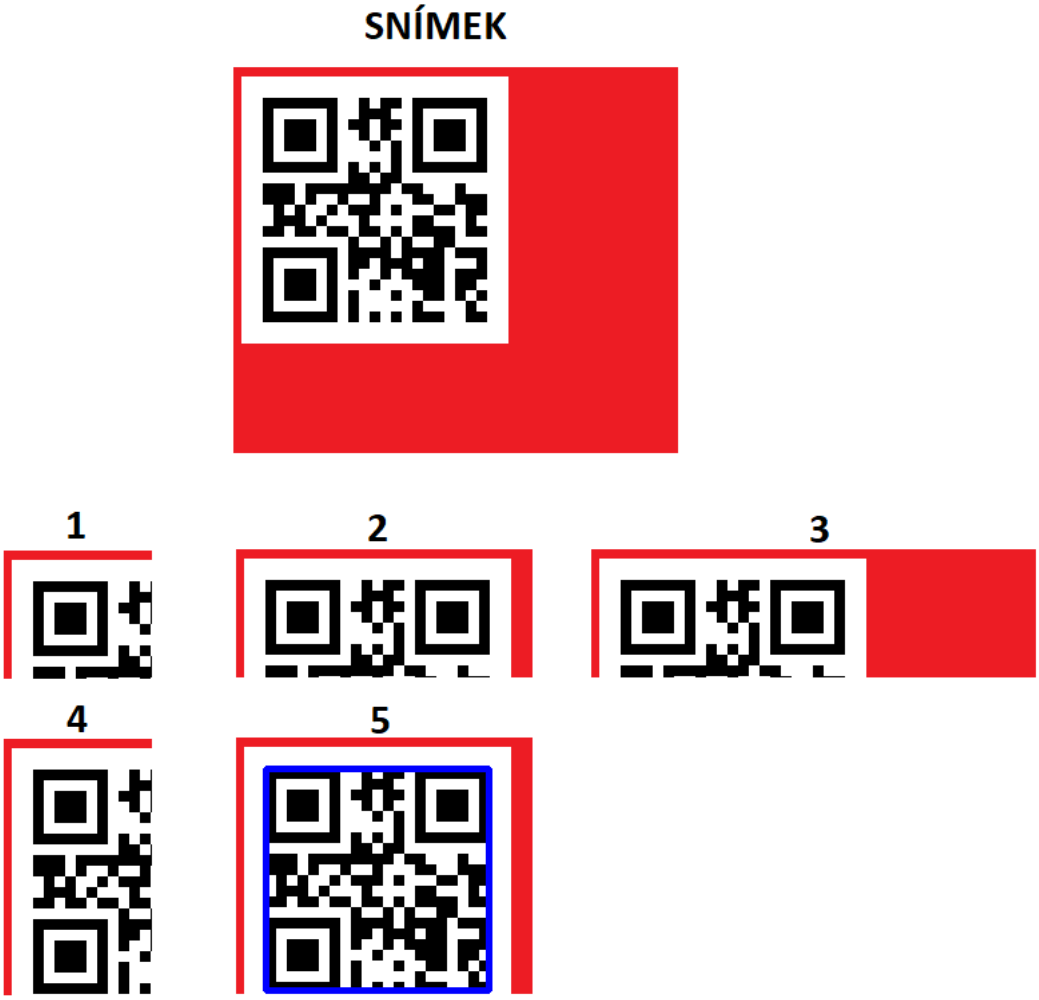
Pro stanovení pozice kamery ze snímku je třeba zjistit polohu definovaných značek ve snímku. V této podkapitole je popsána metoda detekce a klasifikace QR kódů ve snímcích.

Rozpoznávání QR kódů ve snímku je realizováno pomocí třídy knihovny OpenCV *QRCodeDetector* a její metody *detectAndDecode()*. Metoda se ve snímku pokusí vyhledat QR kód a vrátí informace o pozici značky ve snímku a také dekóduje informaci zašifrovanou v kódu. Metoda funguje pouze pokud je na snímku maximálně jeden QR kód, proto je nutno všechny QR kódy postupně detekovat.

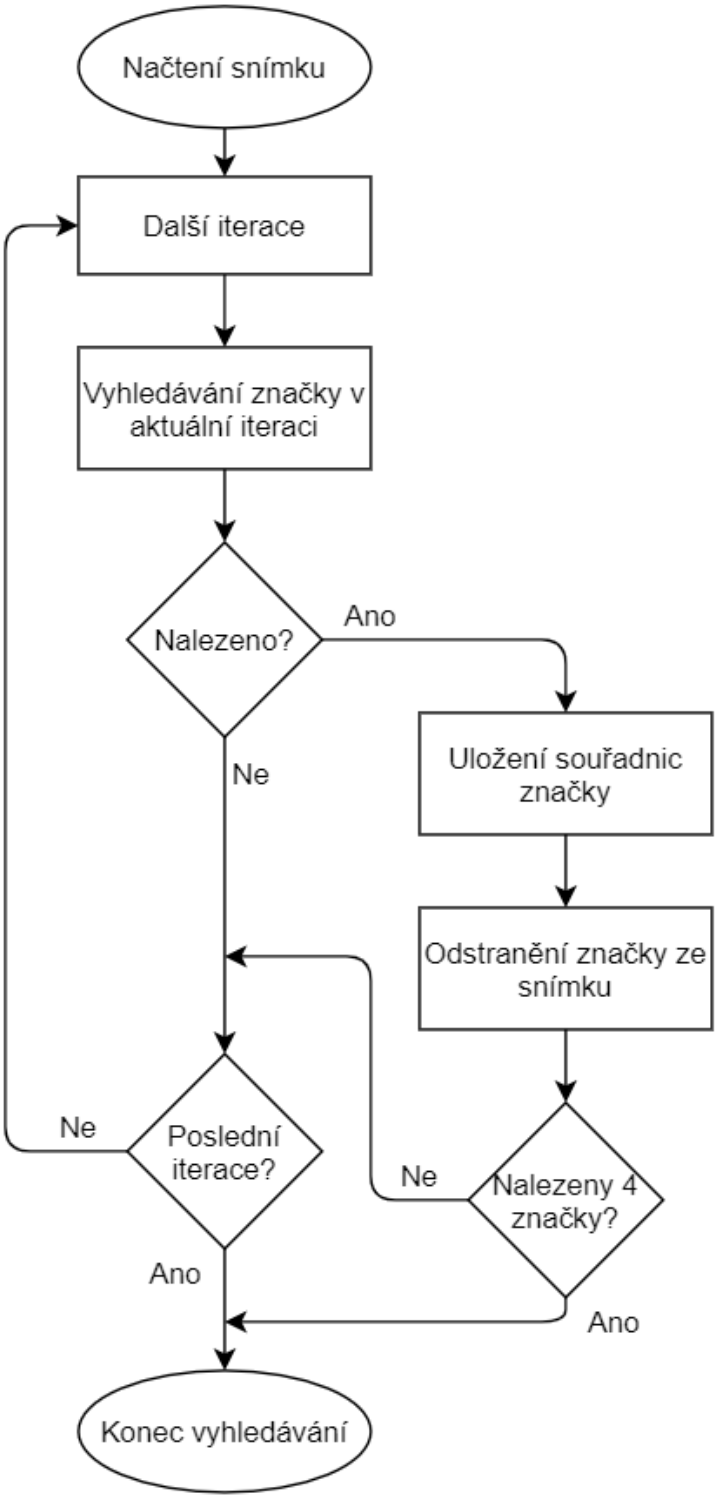
```
cv2.QRCodeDetector.detectAndDecode(
     img[,                # vstupní snímek
     points[,             # výstupní souřadnice ohraničení QR
                          # kódu
     straight_qrcode      # výstupní opravený a binarizovaný
                          # obrázek QR kódu
]]) →
     retval,              # dekodovany text
     points,              # výstupní souřadnice orhaničení QR
                          # kódu
     straight_qrcode      # výstupní opravený a binarizovaný
                          # obrázek QR kódu
```

Výstupem této metody je argument *points*, který slouží jako 2D souřadnice klíčových bodů, pomocí kterých je počítána pozice kamery a také argument *retval*, který nese informaci o pozici detekované značky v prostoru.

Vyhledávání značky je ilustračně znázorněno na obrázku 33. Na obrázku je nahoře zobrazen zdrojový snímek, dole jsou poté zobrazené jednotlivé iterace vyhledávání. Iterace probíhá po horizontálních pruzích a po každém dosažení pruhu do konce snímku, dojde k rozšíření vyhledávacího segmentu do výšky a vyhledává se v novém, nyní vyšším pruhu. Cílem tohoto vyhledávání je postupně detekovat všechny značky na snímku. Nalezené značky je po detekci nutno ze snímku odstranit, aby bylo možno nalézt další značky ve snímku. Algoritmus vyhledávání značek ve snímcích formou vývojového diagramu je vyobrazen na obrázku 34. V prvních čtyřech iteracích není ve vyhledávajících segmentech značka kompletní, značka tedy nebyla nalezena. V páté iteraci je ve vyhledávacím segmentu značka kompletní a může být detekována.



Obrázek 33 – Znázornění vyhledávání značky ve snímku v 5 iteracích



Obrázek 34 – Algoritmus iterativního vyhledávání značek ve snímku

4.4 Detekce a klasifikace vlastních značek

Obdobně jako při rozpoznávání QR kódů je nutno i při rozpoznávání vlastních značek provést detekci a následnou klasifikaci. V této podkapitole jsou popsány metody využité pro detekci značek ve snímcích a klasifikaci detekovaných značek.

4.4.1 Detekce vlastních značek

Detekce vlastních značek je vyřešena využitím barevných vlastností značek. Konkrétně se jedná o detekci červených čtverců o specifikované minimální velikosti. Pro odfiltrování nechtěných barevných složek od červené barvy je realizováno pomocí převodu snímku do HSV formátu a odfiltrování všech barev, které nepatří do rozsahů $HUE \in \langle 170°; 180° \rangle \cup \langle 0°; 10° \rangle$ $SAT \in \langle 70; 255 \rangle$ $VAL \in \langle 70; 255 \rangle$, kde platí obory hodnot dle *OpenCV:* $HUE \in \langle 0°; 180° \rangle$ $SAT \in \langle 0; 255 \rangle$ $VAL \in \langle 0; 255 \rangle$. Po této filtraci následuje aplikace morfologické transformace otevření a uzavření, čímž je redukován detekovaný šum a jsou zaceleny defekty v objektech na snímku. Následně je třeba vyhledat ve snímku čtverce. Tato operace je realizována pomocí funkce knihovny *OpenCV findContours()* [35]. Funkce vyhledává kontury v binárním obraze pomocí algoritmu strukturální analýzy sledováním hran [36].

```
cv2.findContours(
     image,          # zdrojový binární obraz
     mode,           # volba metody extrakce
     method[,        # volba metody aproximace kontur
     contours[,      # detekované kontury
     hierarchy[,     # výstupní vektor popisující topologii
                     # obrazu
     offset          # hodnota o kterou jsou všechny detekované
                     # kontury posunuty
]]]) →
     contours,       # detekované kontury
     hierarchy       # výstupní vektor popisující topologii
                     # obrazu
```

Argument *mode* definuje způsob extrakce kontur z obrazu a rozdělení detekcí podle jednotlivých hierarchií. Možná nastavení jsou:

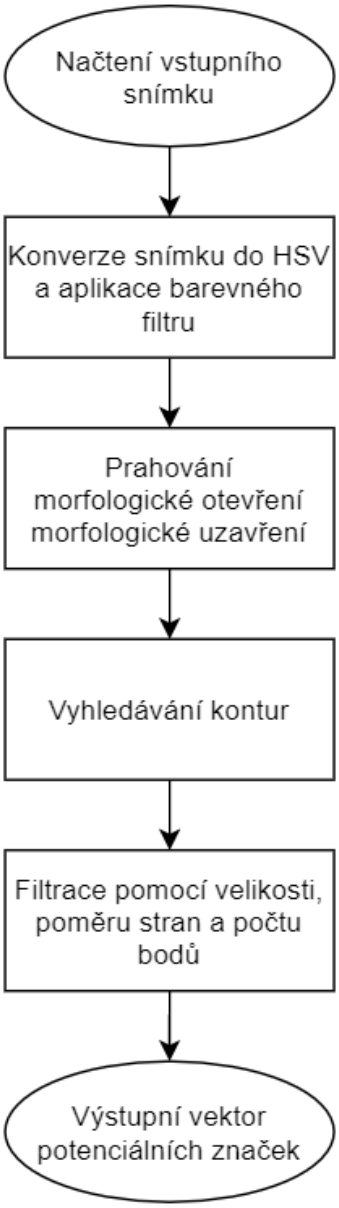
- *CV_RETR_EXTERNAL* – Jsou extrahovány pouze extrémní vnější kontury.
- *CV_RETR_LIST* – Extrahuje všechny kontury a nenastaví nijak hierarchii.

- *CV_RETR_CCOMP* – Extrahuje všechny kontury a organizuje je do dvou úrovní hierarchie. V první úrovni jsou vnější objekty a v druhé úrovni jsou díry v těchto objektech.
- *CV_RETR_TREE* – Extrahuje všechny kontury a sestrojí hierarchický strom.

Argument *method* definuje metody aproximace kontur. Její možná nastavení jsou:

- *CV_CHAIN_APPROX_NONE* – Funkce vrátí všechny detekované body kontur. Všechny uložené vzájemně následující body jsou horizontální, vertikální nebo diagonální sousedi.
- *CV_CHAIN_APPROX_SIMPLE* – Aplikuje jednoduchou kompresi, všechny horizontální, vertikální nebo diagonální segmenty jsou redukovány pouze na jejich konečné body.
- *CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS* – Aplikuje jeden z aproximačních algoritmů navrhovaných v publikaci TEH C.H., CHIN Roland T. *On the detection of dominant points on digital curve*. [37]

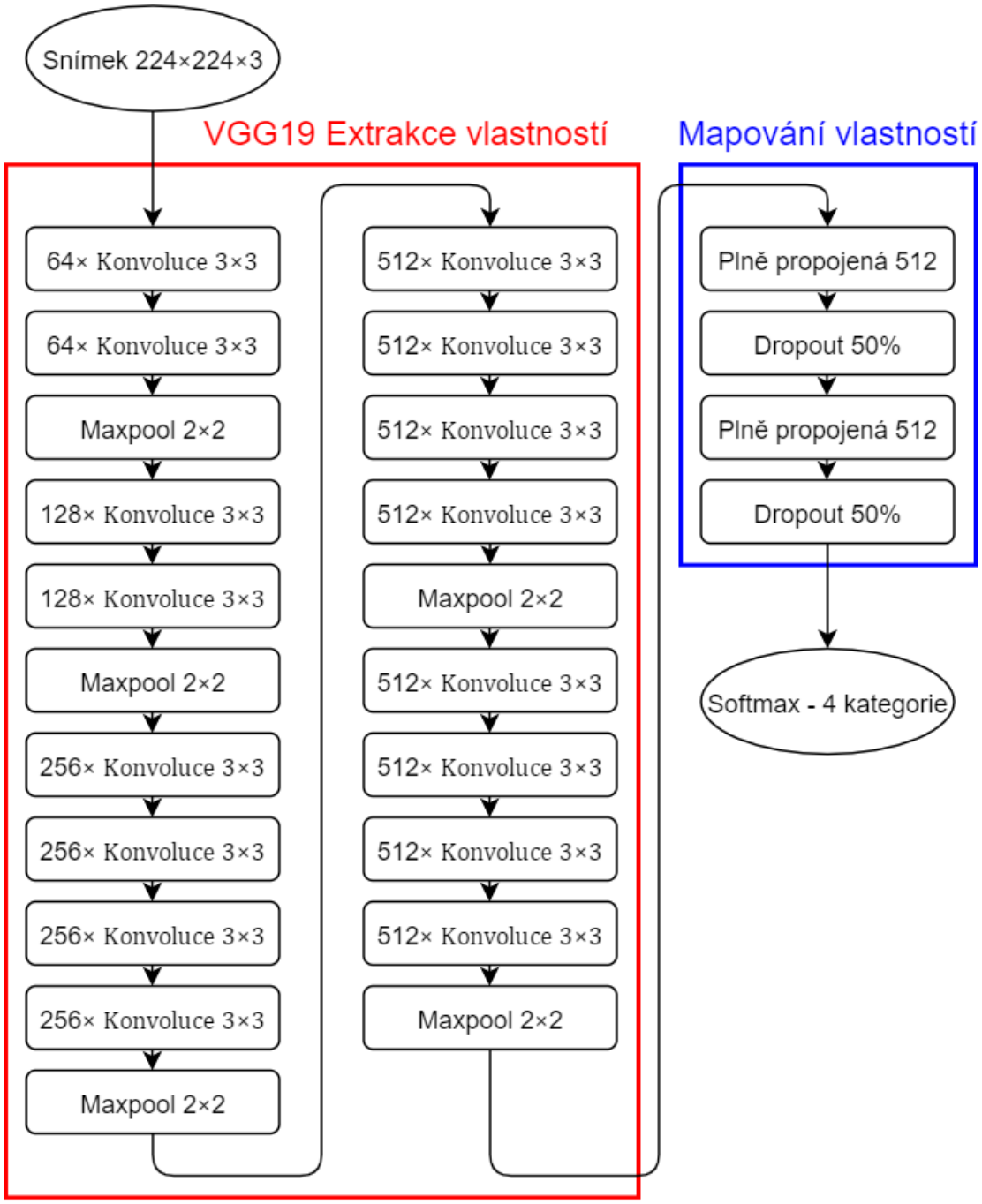
Pro detekci kontur byla zvolena extrakce *CV_RETS_EXTERNAL* pro extrakci vnějších kontur a metodu aproximace *CV_CHAIN_APPROX_SIMPLE* pro zjednodušení práce s výstupním vektorem funkce. Výstupem z této funkce je tedy vektor detekovaných kontur. Z definice čtverce můžeme filtrovat v tomto vektoru, a to tak že jsou ponechány kontury, které jsou definovány čtyřmi krajními body a s poměrem stran $AR = \frac{\check{s}\acute{i}\check{r}ka}{v\acute{y}\check{s}ka} \in \langle 0.8, 1.2 \rangle$. Po této selekci je výsledkem vektor kontur, které mohou být potenciálně označeny jako značky a detekce je dokončena. Blokový diagram tohoto algoritmu je vyobrazen na obrázku 35.



Obrázek 35 - Algoritmus detekce červených čtverců

4.4.2 Klasifikace vlastních značek

Klasifikace vlastních značek je realizována použitím konvoluční neuronové sítě. Použitá architektura extraktoru vlastností je VGG19 architektura [10]. Architektura klasifikátoru je vyobrazena na obrázku 36.



Obrázek 36 – Klasifikátor vlastních značek

Pro učení sítě bylo využito techniky *„Transfer learning“*. Jedná se o využití vah předem naučené sítě. V tomto konkrétním případě jsou načteny váhy VGG19 sítě pro klasifikace ImageNet [4] databáze. Tímto krokem byl rapidně snížen čas potřebný k natrénování sítě, protože není třeba trénovat extrakci vlastností, ale pouze mapování těchto vlastností na výstup.

Pro vylepšení schopnosti klasifikace sítě byla využita technika augmentace dat. Jedná se o úpravu dat, na kterých je síť učena. Mezi standartní úpravy paří

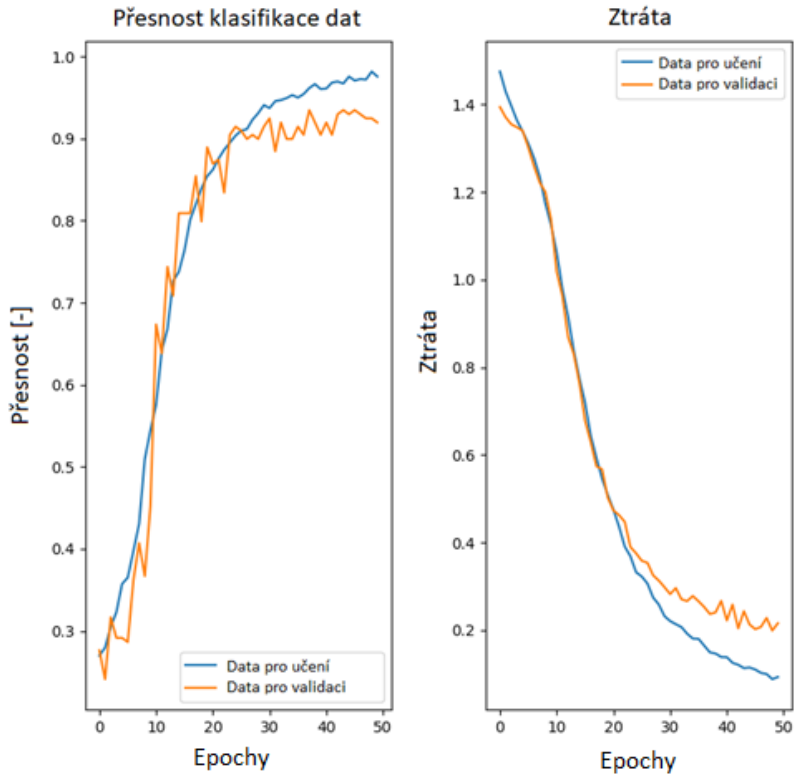
např. náhodná translace nebo rotace snímku, náhodné úpravy jasových hodnot v celém snímku, náhodné přiblížení nebo oddálení snímku. Pro tento případ byla využita augmentace pomocí jasových úprav. Využití této augmentace tedy umožňuje více zobecnit jas při a učení a klasifikaci.

Učení proběhlo v rámci 50 epoch, tedy učení „prošlo“ datasetem 50×. Počet snímků v datasetu a jejich rozdělení mezi možné kategorie je vypsán v tabulce 1. V tabulce jsou vypsán i počet validačních snímků. Jedná se o data, na kterých se síť neučí, ale slouží pro kontrolu, jestli nedochází k přeučení. Na konci každé epochy je síť na těchto datech otestována a je možno z výsledků těchto testů diagnostikovat případné problémy. Grafy závislosti úspěšnosti klasifikace dat na epochách a závislosti úrovni ztrát sítě na epochách jsou vyobrazeny na obrázku 38.

Tabulka 1 - Počet snímků datasetu klasifikátoru vlastních značek

Počet snímků v datasetu	Značka *TopLeft*	Značka *TopRight*	Značka *BottomLeft*	Značka *BottomRight*	Celkem
Učení	376	366	482	414	1638
Validace	56	44	54	45	199

Obrázek 37 - Ukázka dat z datasetu pro značku *TopLeft*



Obrázek 38 - Proces učení VGG19 klasifikátoru

4.5 Využití objektového detektoru pro detekci i klasifikaci značek

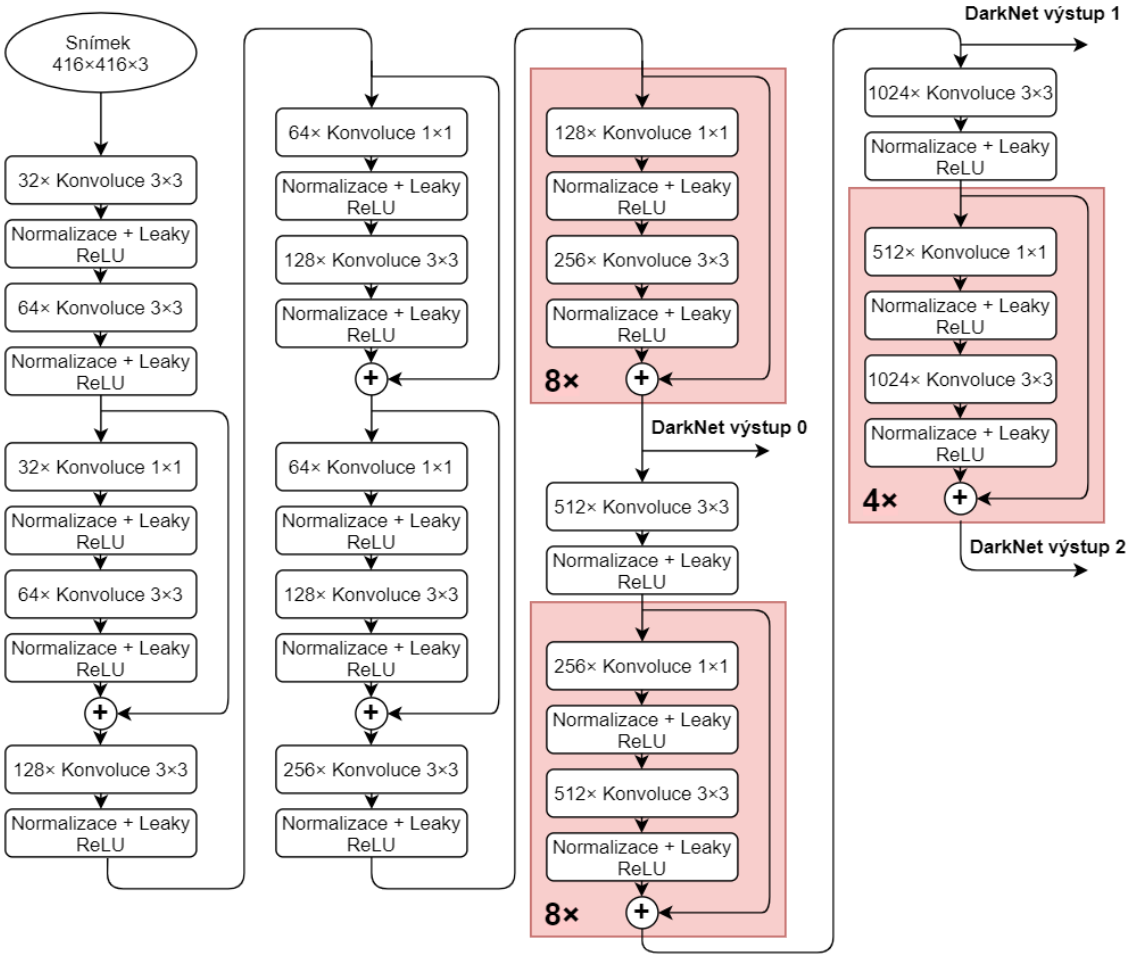
Pokud uvažujeme, že v každém snímku proběhnou alespoň 4 klasifikace, dojdeme k závěru, že použití tradičního klasifikátoru není příliš vhodné pro využití v reálném čas z hlediska výpočetní náročnosti. Řešením může být použití neuronové sítě fungující jako objektový detektor. Architektura *„You Only Look Once“* (zkráceně *YOLO*) [39][40][41] umožňuje při pouze jednom pohledu na snímek detekovat i klasifikovat objekty na snímku, předpokládáme tedy snížení požadavku na výpočetní výkon.

Princip této sítě je rozdělení snímku na tři mřížky o velikosti buněk $S \times S$, $R \times R$ a $T \times T$. Uvažujme tedy vstupní snímek o velikosti 416×416, který se rozdělí na tři mřížky o velikosti buněk 13×13, 26×26 a 52×52 pixelů. Každá z těchto buněk obsahuje takzvané *„anchor boxes“*. Jedná se o rozšíření buněk, které umožňuje detekci více objektů v jedné buňce. Každá z těchto *„anchor boxů“* má definovaný tvar a na základě podobnosti predikovaného objektu s tímto boxem je konkrétnímu boxu objekt přiřazen. Počet těchto *„anchor boxů“* v každé buňce definujeme jako B. YOLO poté provede B× predikci pro každou buňku mřížek. Výsledkem predikce jsou tedy

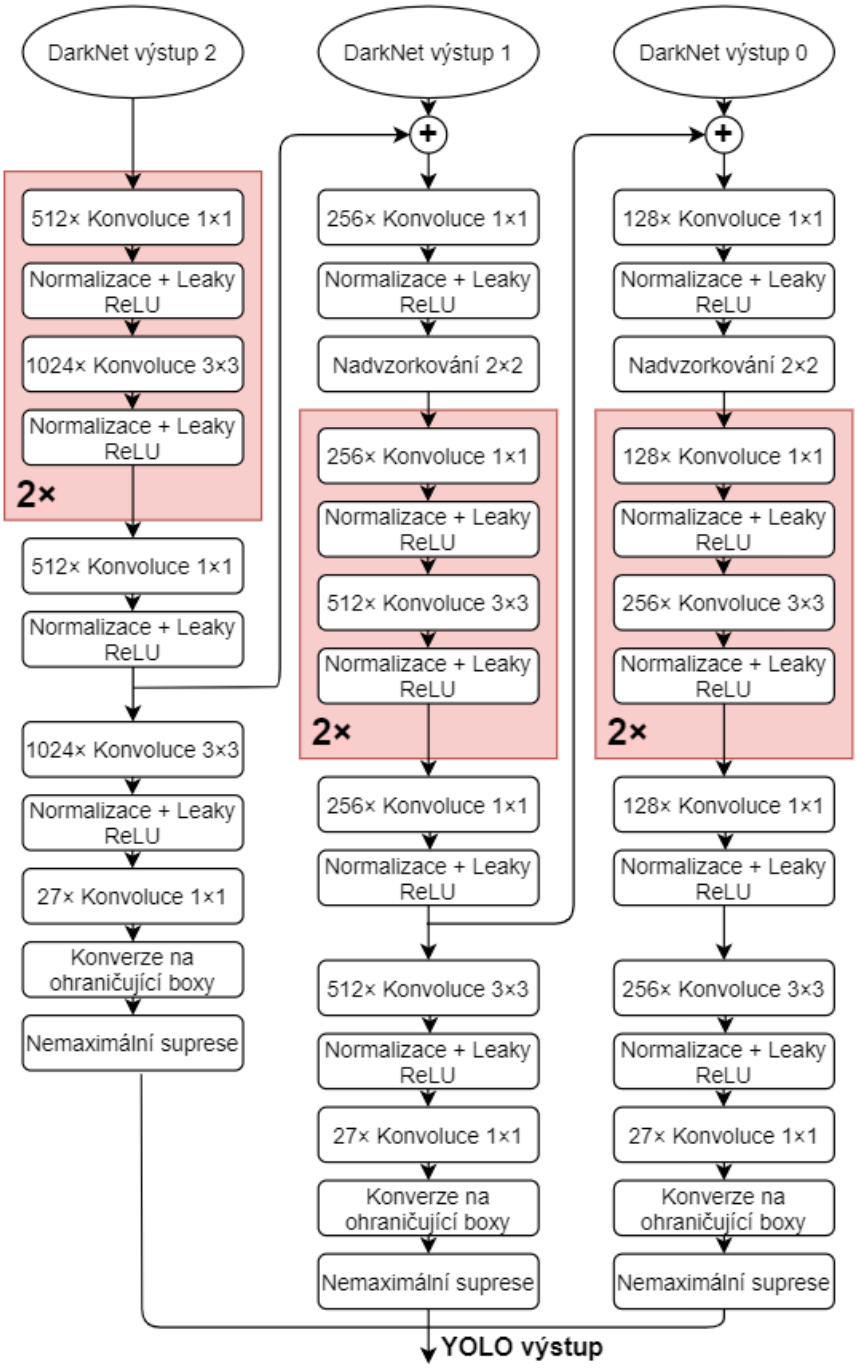
3 3D tensory o velikostech $[13, 13, B \cdot (5 + \mathrm{C})], [26, 26, B \cdot (5 + \mathrm{C})], [52, 52, B \cdot (5 + \mathrm{C})]$, kde C je počet tříd, mezi které objekty klasifikujeme. Konstanta 5 ve výpočtu koresponduje s informacemi o poloze a velikosti detekovaných ohraničujících boxů (v angličtině *bounding box*) detekovaných objektů. Jedná se o hodnoty středu boxu, šířku a výšku boxu a jistota predikce těchto hodnot. Můžeme tedy mluvit o vektoru $[x_{střed}, y_{střed}, šířka, výška, jistota]$. Třetí dimenze výsledných 3D tensorů je implementována pro hodnoty $B = 3$ a $C = 4$. Pokud tedy aplikujeme tento princip na snímek tak zjistíme, že pro každý objekt na snímku bylo detekováno více ohraničujících boxů. Tento problém je vyřešen využitím nemaximální suprese (v angličtině *non-maximum suppression*). Jedná se o techniku potlačení predikcí s menší jistotou predikce, než je definovaná konstanta *NMS* (v rámci řešení práce bylo definováno $NMS = 0.5$). Vzhledem k tomu, že samotná aplikace nemaximální suprese není dostatečná k odfiltrování duplikovaných detekcí, definujeme vztah pro takzvané *„Intersection Over Union"* (zkráceně IOU) (4.1). Jedná se o definici překrytí dvou 2D ploch na snímku. Můžeme tedy odstranit všechny detekce, které mají velké *IOU* s detekcí s největší jistotou detekce.

$$IOU = \frac{Plocha\ překrytí}{Sdružení\ ploch} \tag{4.1}$$

Samotnou architekturu lze rozdělit na dvě hlavní části. První část je DarkNet-53 konvoluční neuronová síť, jedná se o extraktor vlastností snímků, a druhou částí je predikce polohy a velikosti ohraničujících boxů a jejich klasifikace, tuto část označujeme jako *YOLO* sekci. Architektura *DarkNet-53* sítě je vyobrazena na obrázku 39. Z obrázku jde vidět, že síť má tři výstupy, jedná se o mapy o dimenzích 52×52, 26×26 a 13×13. Na tyto výstupy navazuje *YOLO* sekce, která je vyobrazena na obrázku 40.



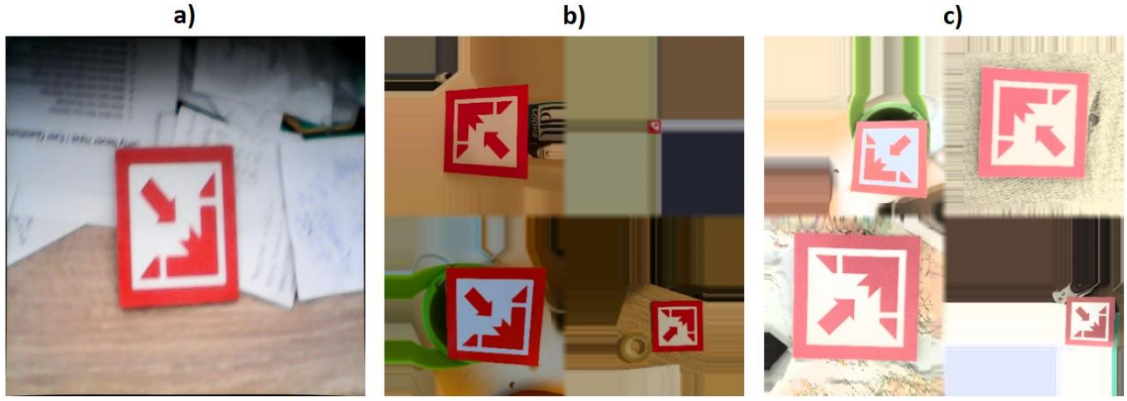
Obrázek 39 - DarkNet-53 architektura



Obrázek 40 - *YOLO* architektura

Pro učení sítě bylo použito 390 snímků pro učení a 55 snímků pro validaci. Na snímcích byly provedeny augmentace typu náhodné úpravy jasu, náhodné aplikace translace a náhodného přibližování a oddalování. Ukázka snímků z datasetu je vyobrazena na obrázku 41. Učení sítě dále požaduje vytvořit ke každému snímku v datasetu soubor obsahující informace o polohách značek na snímku. Ukázka formátu tohoto souboru pro snímek *a* z obrázku 41 je uvedena zde:

```
{
    "filename": "TL_7990_yolo2020-04-21_16_10_29_137030.png",
    "object": [
        {
            "bndbox": {
                "xmax": 0.6514423076923077,
                "xmin": 0.30528846153846156,
                "ymax": 0.7644230769230769,
                "ymin": 0.3004807692307692
            },
            "class_num": 2
        }
    ],
    "path":
"c:\\Users\\dfice\\Desktop\\Bakalarka\\Progs\\learning\\yolov3\\train\\TL_7990_yolo2020-04-21_16_10_29_137030.png",
    "shape": [
        416,
        416,
        3
    ],
    "source_id": "TL_7990_yolo2020-04-21_16_10_29_137030.png"
}
```

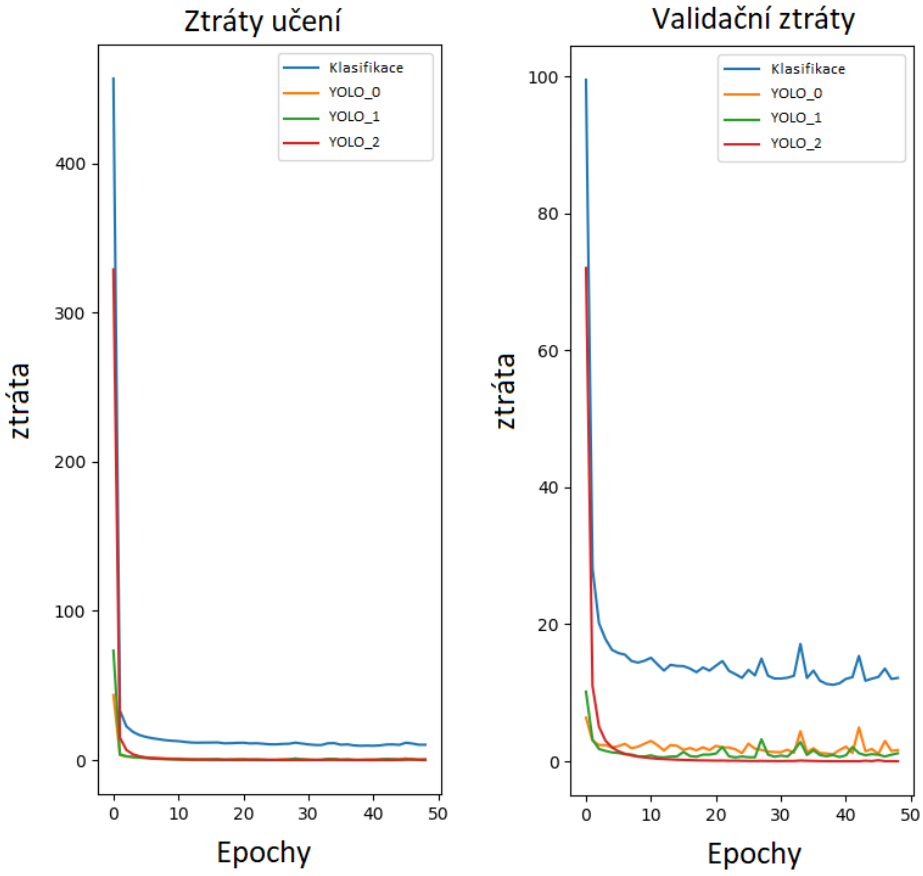



Obrázek 41 - Ukázka snímků z datasetu

Jak už bylo zmíněno, tento soubor musí obsahovat informace o všech objektech na snímku, pro tento případ informace o značkách na snímku. Takové soubory jsou označovány jako anotační soubory. Soubor tedy obsahuje informace:

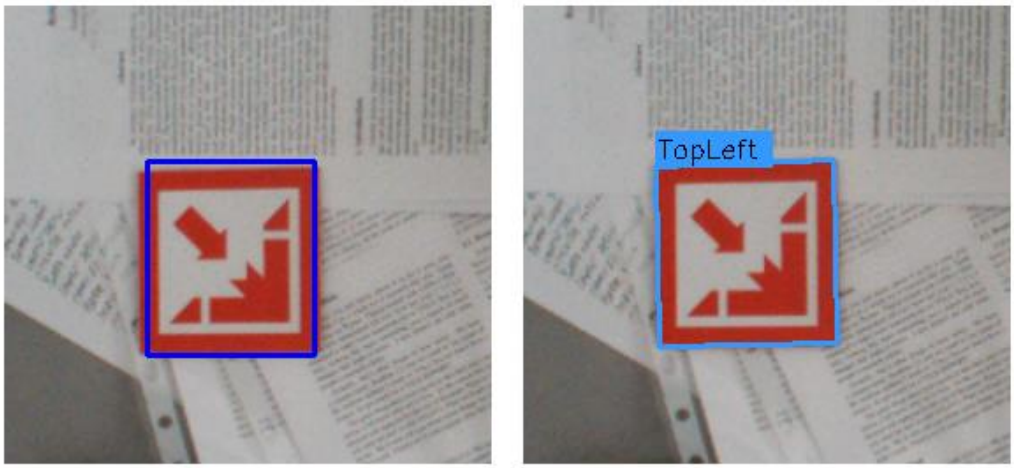
- *filename*: název souboru snímku
- *object*: obsahuje seznam dvojic informací, tedy *2×n* kde *n* je počet značek na snímku.
    - *bndbox*: ohraničení značky, obsahuje souřadnice dvou bodů obdélníka, který značku ohraničuje. Souřadnicový systém snímku je normalizován na rozsah $\langle 0, 1 \rangle$.
    - *class_num*: číselné označení kategorie značky
- *path*: systémová cesta k souboru
- *source_id*: ID snímku, pro tento případ je použito stejné ID jako je název souboru snímku.

Grafy závislosti ztrát sítě na epochách jsou vyobrazeny na obrázku 42. V grafech jsou uvedeny čtyři ztráty, první, základní ztráta referuje ke ztrátě při klasifikaci, tedy stejná ztráta jako např. ztráta sítě VGG19. Další ztráty *YOLO_0*, *YOLO_1* a *YOLO_2* jsou ztráty predikce poloh detekovaných objektů pro tři definované mřížky.



Obrázek 42 - Ztráty při učení YOLO sítě

Při použití této sítě však nastává problém při použití detekovaných bodů v algoritmu stanovení pozice, protože výstupní odhad ohraničení značek ze sítě nesplňuje požadavky na přesnost určení těchto bodů. Je tedy nutno detekované oblasti zpřesnit. Toho můžeme dosáhnou použitím algoritmu vyhledávání značek popsaném v podkapitole 4.3.2. Použijeme tedy odhadovanou oblast sítě jako referenci a o 10% oblast zvětšíme na všechny strany, poté použijeme v této oblasti algoritmus vyhledávající značky. Nalezené souřadnice nyní odpovídají požadavkům na přesnost.



Obrázek 43 - Porovnání detekované oblasti značky YOLO sítí (vlevo) a zpřesněné oblasti pomocí algoritmu využívající vyhledávání barev (vpravo)

4.6 Výpočet pozice kamery v souřadnicovém systému

Ze znalosti polohy značek na snímku a znalosti polohy těchto značek ve 3D prostoru je možno v tomto prostoru určit polohu kamery.

Výpočet pozice kamery je realizován pomocí funkce knihovny OpenCV *solvePnP* [11], která vyřeší již nastíněný problém *Perspektiva z N bodů* a vypočítá polohu středu kamery ve složkách translačního a rotačního vektoru.

```
cv2.solvePnP(
     objectPoints,  # 3D souřadnice známých bodů
     imagePoints,   # 2D souřadnice známých bodů
     cameraMatrix,  # matice vnitřních parametru kamery
     distCoeffs[,   # koeficienty zkreslení kamery
     rvec[,         # odhad rotačního vektoru
     tvec[,         # odhad translačního vektoru
     useExtrinsicGuess[, # použití odhadu – true/false
     flags          # metoda výpočtu
]]]]) →
     retval,   # kontrolní výstup funkce
     rvec,     # rotační vektor
     tvec      # translační vektor
```

Funkce může pro zvýšení přesnosti použít vnější odhad rotace a translace. Tento odhad je v programu využíván pro každý výpočet, který není prvním. Jeden z argumentů funkce je také *flags*, tedy jaký algoritmus má funkce použít pro své řešení [11]. Jako metody výpočtu je možno použít:

- *CV_ITERATIVE* – Metoda založená na Levenberg-Marquardtově optimalizaci [12]. Funkce vyhledává takovou pozici, aby co nejvíce zredukovala chybu zpětného promítnutí, která je rovna sumě kvadrátů vzdáleností mezi zadanými 3D souřadnicemi a vypočtenými 3D souřadnicemi.
- CV_P3P – Metoda založená na studii X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang *Complete Solution Classification for the Perspective-Three-Point Problem* [13]. Metoda vyžaduje právě 3 body ve 2D a 3D prostorech.
- CV_EPNP – Metoda založená na studii F.Moreno-Noguer, V.Lepetit and P.Fua *EPnP: Efficient Perspective-n-Point Camera Pose Estimation* [14].

Pro řešení této práce byla využita metoda *CV_ITERATIVE,* jejíž nevýhoda je její výpočetní náročnost, ale poskytuje přesné výsledky. Nevýhoda z hlediska výpočetní náročnosti této metody je však zanedbatelná oproti výpočetní náročnosti detekce značek.

4.7 Převod pozice kamery na pozici kurzoru na obrazovce

Známou polohu kamery je nutno přepočítat na pozici kurzoru na obrazovce. Funkce *solvePnP* popsaná v podkapitole 4.6 vrací translační vektor reprezentující polohu

středu kamery vzhledem ke středu souřadnicového systému a rotační vektor reprezentující rotační složku polohy kamery v prostoru.

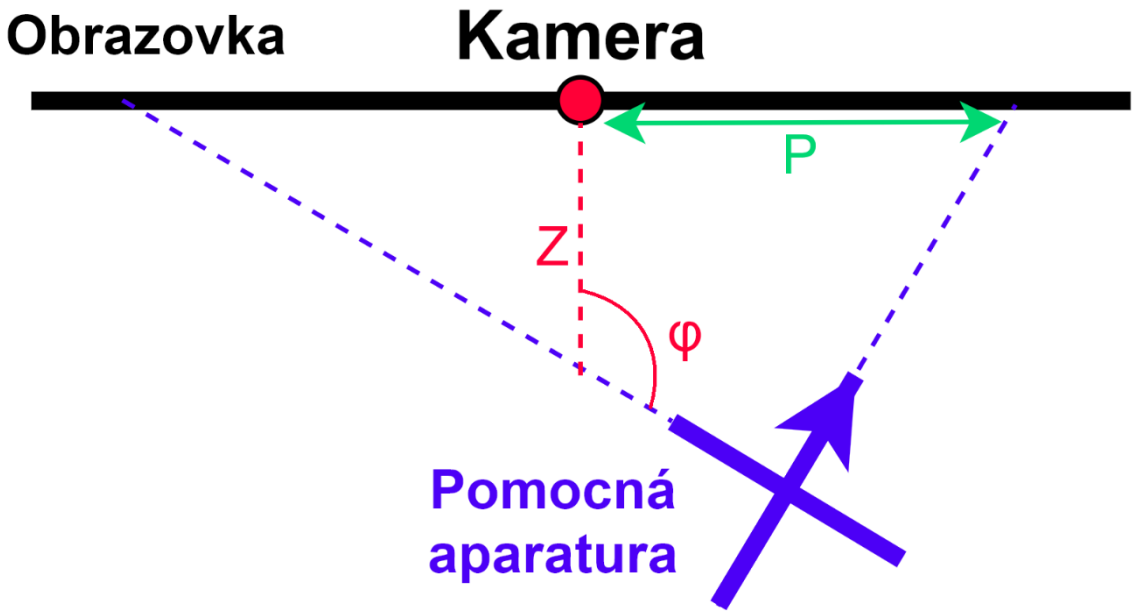
Pro metodu A, kde je souřadnicový systém umístěn na monitoru vyjadřuje samotná translace pozici, kde kamera ukazuje, tedy pozici, kde by se měl přesunout kurzor. Za předpokladu, že jsou značky umístěny na rozích monitoru, můžeme získat polohu kurzoru těmito vztahy:

$$k_x = \frac{x}{\Delta X} \cdot R_x \quad [px] \tag{4.2}$$

$$k_y = \frac{y}{\Delta Y} \cdot R_y \quad [px] \tag{4.3}$$

Kde k_x a k_y jsou pozice kurzoru na obrazovce v pixelech, x a y jsou složky translačního vektoru středu kamery, ΔX a ΔY jsou vzdálenosti mezi značkami, R_x a R_y jsou složky rozlišení obrazovky.

Pro metodu B je nutno uvažovat rotaci, výsledná pozice je závislá na tom, jak je otočený souřadnicový systém. Je tedy nutno zjistit na který bod obrazovky je aparaturou ukazováno. Na obrázku 44 je vyobrazena situace, ze kterého lze odvodit souřadnice pozice, na které je aparaturou ukazováno.



Obrázek 44 - Pozice bodu, na který je ukazováno

Lze tedy odvodit že poloha bodu, na který je ukazováno je:

$$P_x = z \cdot \sin\varphi_y \cdot \sin(90° - \varphi_y) \quad [cm] \tag{4.4}$$

$$P_y = z \cdot \sin\varphi_x \cdot \sin(90° - \varphi_x) \quad [cm] \tag{4.5}$$

Kde P_x a P_y jsou 3D souřadnice bodu, na který je ukazováno, z je složka translace osy z a φ_x a φ_y jsou rotace kolem os x a y.

Nyní je možno provést normalizaci polohy a z fyzické velikosti obrazovky a rozlišení obrazovky je možno spočítat polohu kurzoru:

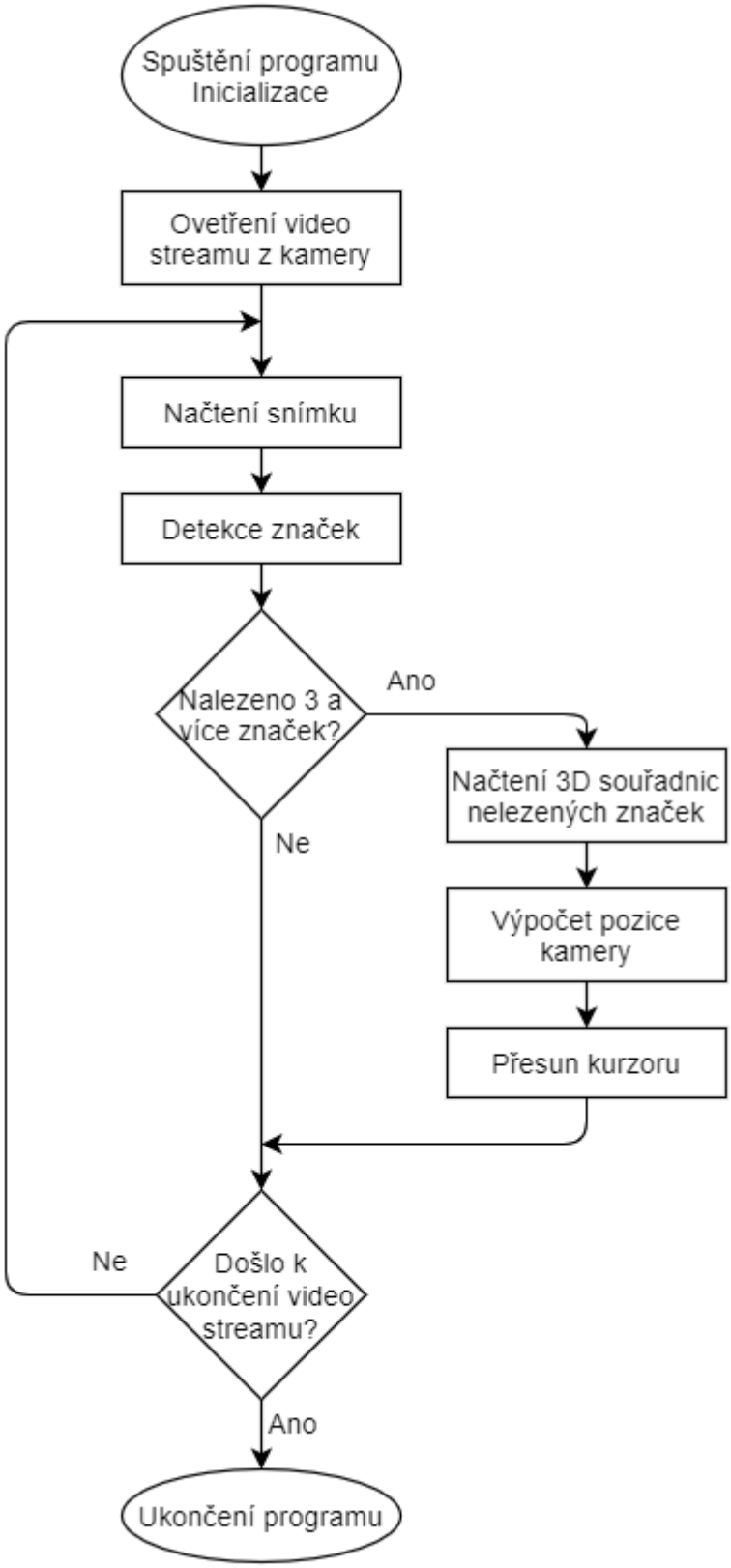
$$k_x = \left(\frac{P_x}{V_x} + 0.5\right) \cdot R_x \quad [px] \tag{4.6}$$

$$k_y = \left(\frac{P_y}{V_y} + 0.5\right) \cdot R_y \quad [px] \tag{4.7}$$

Kde k_x a k_y jsou souřadnice kurzoru na obrazovce, P_x a P_y jsou souřadnice bodu na obrazovce, na který je ukazováno, R_x a R_y jsou složky rozlišení obrazovky a V_x a V_y jsou složky fyzické velikosti monitoru v centimetrech.

4.8 Výsledný algoritmus

Výsledný algoritmus se skládá z několika částí. Tento algoritmus je znázorněn pomocí vývojového diagramu na obrázku 45. Program po spuštění provede svou inicializaci a otevře video stream z kamery. Následně se načte snímek a proběhne detekce značek na snímku. Pokud byly detekovány alespoň tři značky (pro správnost detekce pozice je potřeba znát pozici alespoň tří značek), načtou se informace o 3D souřadnicích těchto značek v prostoru (referenční souřadnice značek) a vypočte se pozice kamery v referenčních 3D souřadnicích. Ze známé polohy kamery se poté vypočítá pozice kurzoru a provede se přesun. Dále se načte další snímek a proces se opakuje, dokud není ukončen video stream z kamery.



Obrázek 45 - Algoritmus programu

5 VÝSLEDEK REALIZACE

V této kapitole jsou diskutovány výsledky realizace, jsou provedena měření zaměřující se na výpočetní náročnost algoritmů a na přesnost algoritmů.

5.1 Vyhodnocení výpočetní náročnosti algoritmů

Jedním z cílů návrhu metod pro stanovení pozice kamery v prostoru je použití v reálném čase, proto bylo provedeno měření časové náročnossti algoritmů. Měření každého z algoritmů proběhlo pro 500 snímků. Specifikace testovací výpočetní jednotky jsou vypsány v tabulce 2 a výsledky měření jsou vypsány v tabulce 3.

Tabulka 2 - Specifikace výpočetního výkonu na testovacím počítači

Specifikace výpočetního výkonu na testovacím počítači	
Procesorová jednotka	Intel Core i5-4460S 2.9 GHz
Paměť RAM	16 GB DDR3 1600 MHz
Grafická jednotka	NVIDIA GeForce GTX 960 2GB DDR5

Tabulka 3 - Výpočetní náročnost navržených algoritmů

Výpočetní náročnost algoritmů – statistická data z 500 zpracovaných snímků	
Měřené algoritmy	Výpočetní náročnost pro jeden snímek
QR kódy – Metoda A	426.11 ± 25.91 ms
QR kódy – Metoda B	1267.68 ± 34.13 ms
Vyhledávání pomocí barev + VGG19 klasifikace	206.33 ± 13.23 ms
YOLO detekce + zpřesnění detekovaných bodů	138.80 ± 6.82 ms
Výpočet pozice kamery – Metoda A	0.41 ± 0.30 ms
Výpočet pozice kamery – Metoda B	0.29 ± 0.46 ms
Konverze pozice kamery na přesun kurzoru – Metoda A	1.19 ± 0.50 ms
Konverze pozice kamery na přesun kurzoru – Metoda B	1.11 ± 0.36 ms

Z tabulky 3 můžeme pozorovat velkou výpočetní náročnost iterativních metod pro výhledávání QR kódů v obraze, obvzvláště pro metodu B, tento nárust výpočetní náročnosti je způsoben nižším iteračním krokem vyhledávání u metody B, protože se předpokládá, že jsou značky umístěny blíže u sebe než u metody A. Můžeme tedy konstatovat, že tyto metody nejsou vhodné pro použití v reálném čase. Metoda využívající klasifikátor VGG19 už může být považována jako metoda fungující v reálném čase, avšak její výpočetní náročnost je závislá na prostředí, ve kterém byl pořízen snímek. Na scéně, kde se nachází větší množství červených objektů, bude každý z nich klasifikován, tedy roste výpočetní náročnost algoritmu.

Nejrychlejší metodou detekce značek na snímcích je metoda využívající objektový detektor *YOLO*. Výpočetní náročnost tohoto algoritmu není závislá na scéně na snímku, tedy můžeme při využití čekat podobné hodnoty jako jsou hodnoty naměřené. Jedná se tedy o optimální řešení z hlediska výpočetní náročnosti.

Z měření výpočetní náročnosti algoritmů výpočtu pozice kamery a konverze této pozice na polohu kurzoru vyplývá, že jsou tyto hodnoty zanedbatelné oproti výpočetním náročnostem algoritmů pro detekci a klasifikaci značek ve snímcích.

5.2 Vyhodnocení přesnosti stanovení pozice

Pro vyhodnocení přenosti stanovení pozice kamery byl proveden experiment. Experiment spočíval v postupném míření kamerou (metoda A) nebo pomocnou aparaturou (metoda B) na definovanou mřížku zobrazenou na obrazovce. Přesunem kurzoru byla poté vyhodnocena chyba stanovení pozice kurzoru, která je úměrná chybě stanovení pozice kamery. Výpočet chyby byl pojat jako euklidovská vzálenost vypočtené pozice kurzoru od očekávané hodnoty. Tato vzdálenost je daná vztahem 5.1.

$$\Delta = \sqrt{\Delta_x^2 + \Delta_y^2}\ [px] \tag{5.1}$$

Kde Δ_x je chyba v *x-ové* souřadnici v pixelech a Δ_y je chyba v *y-ové* souřadnici v pixelech.

Podmínky měření jsou vypsány v tabulce 4. Naměřené chyby jsou vyčísleny v tabulce 5 pro metodu A a v tabulce 6 pro metodu B. Vizualizace těchto hodnot je vyobrazena na obrázcích 46 a 47.

Tabulka 4 - Podmínky měření přesnosti stanovení pozice

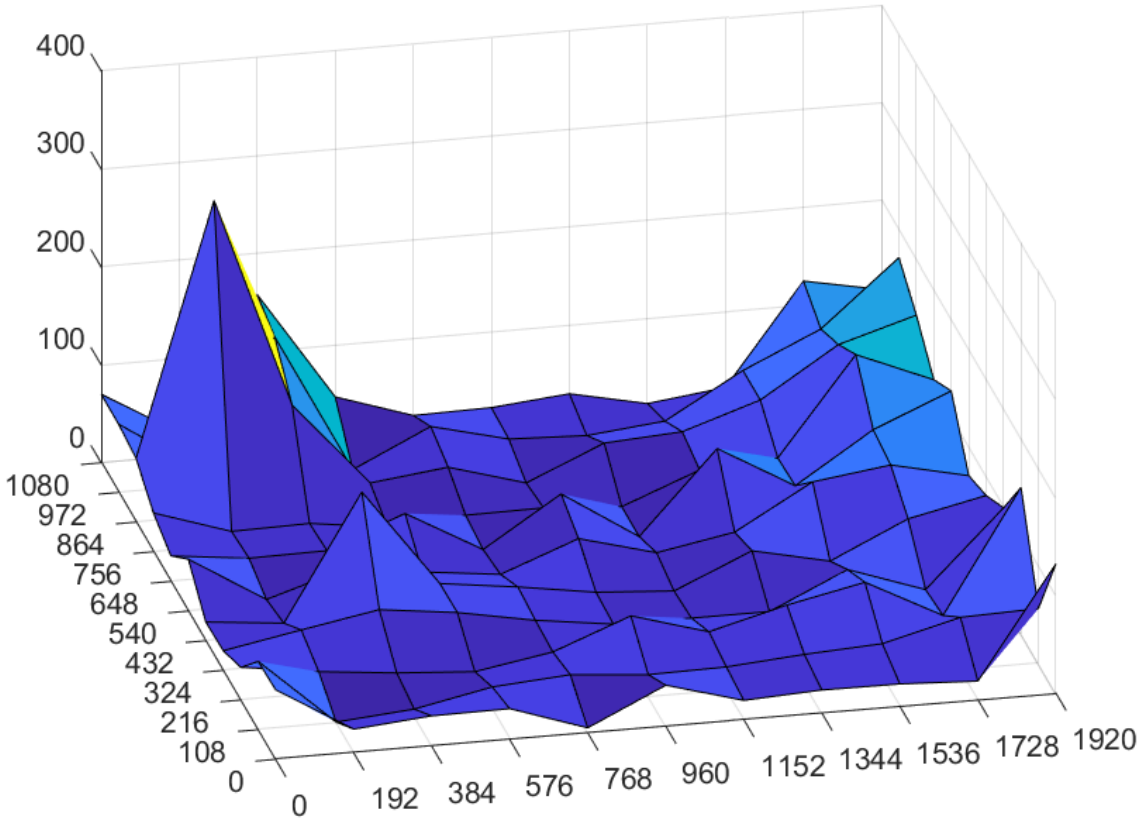
Podmínky měření přesnosti stanovení pozice	
Šířka použitých značek	7.8 cm
Výška použitých značek	7.8 cm
Vzdálenost kamery od souřadnicového systému	70 cm
Velikost pixelu měřícího monitoru	0.235 mm

Tabulka 5 - Chyba stanovení polohy kurzoru – Metoda A

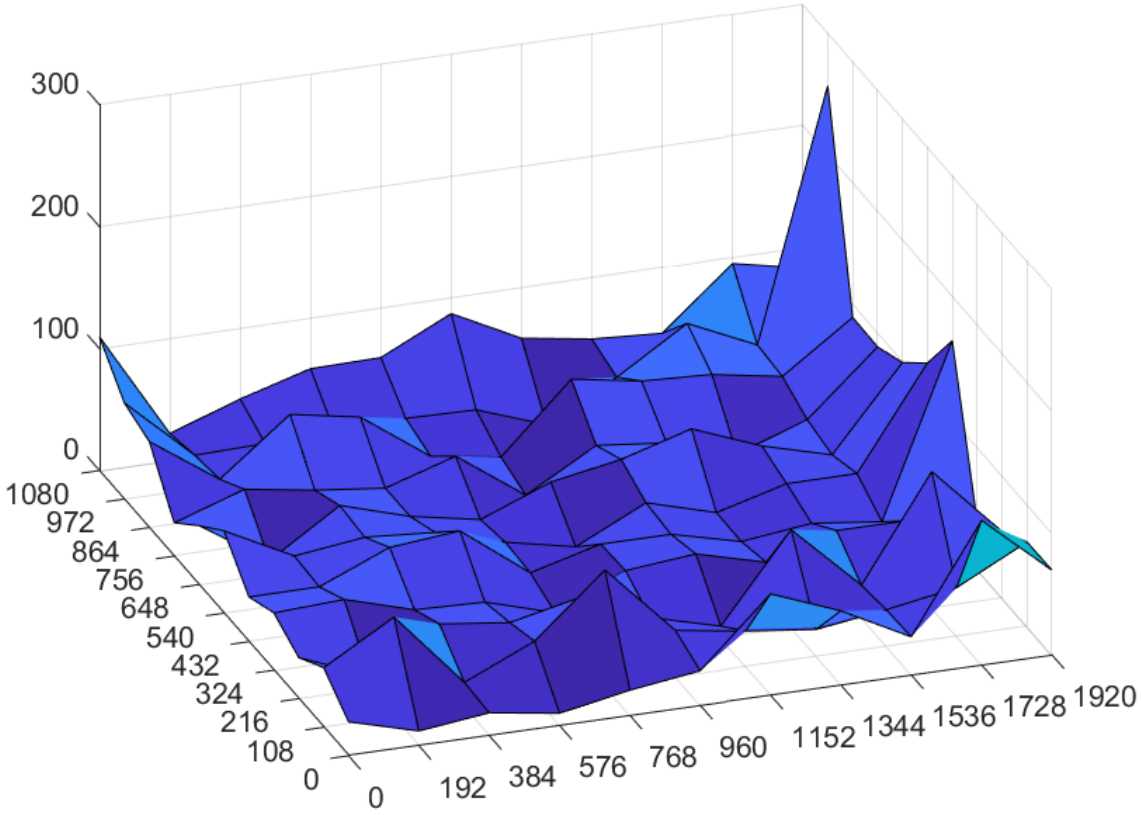
Metoda A – chyba stanovení polohy kurzoru												
Δ [px]		x [px]										
		0	192	384	576	768	960	1152	1344	1536	1728	1920
y [px]	0	71.06	24.04	30.84	31.45	5.09	41.82	19.64	23.44	22.85	18.97	131.94
	108	69.78	1.00	7.52	23.36	29.07	22.36	23.50	29.55	33.71	53.85	56.04
	216	33.42	22.20	14.16	8.69	22.04	52.09	29.13	49.64	66.09	25.46	149.40
	324	20.10	34.53	48.49	38.78	24.09	13.42	12.77	12.72	48.79	21.59	98.31
	432	19.10	11.18	138.34	38.08	27.83	15.69	10.22	44.47	27.20	63.60	80.62
	540	52.95	5.10	29.24	19.59	10.45	40.15	19.02	33.22	77.81	88.26	70.61
	648	26.00	16.97	17.91	49.12	5.54	55.49	5.19	88.30	43.61	94.02	126.92
	756	39.60	14.04	15.40	4.92	10.26	10.98	5.46	36.15	41.65	140.42	107.94
	864	65.19	321.75	106.11	19.75	29.02	6.17	40.12	44.94	70.53	120.34	143.67
	972	69.77	18.44	145.45	3.09	40.27	20.74	17.51	25.53	70.42	106.00	172.20
	1080	70.06	24.02	158.21	46.98	21.28	22.52	29.90	13.14	22.71	124.68	109.25

Tabulka 6 - Chyba stanovení pozice kurzoru – Metoda B

Metoda B – chyba stanovení polohy kurzoru												
Δ [px]		x [px]										
		0	192	384	576	768	960	1152	1344	1536	1728	1920
y [px]	0	27.80	12.04	18.68	10.00	20.62	28.28	83.10	60.84	31.26	118.51	69.43
	108	48.85	82.07	20.25	45.79	95.88	38.05	29.15	22.20	30.08	40.00	69.12
	216	33.73	10.63	44.78	37.59	50.00	25.08	9.85	80.65	26.91	111.72	54.33
	324	47.01	49.74	32.98	31.02	14.04	40.31	45.65	30.53	48.38	48.01	38.95
	432	36.69	36.72	60.75	66.03	23.43	38.60	38.33	24.17	30.81	23.85	163.98
	540	67.48	38.64	45.69	41.40	25.00	13.04	40.82	24.52	39.96	39.96	122.25
	648	48.30	11.40	48.33	31.14	20.25	42.72	53.01	69.81	47.51	31.91	99.37
	756	27.29	46.52	37.48	30.89	48.88	8.54	41.04	34.93	17.46	37.36	88.77
	864	70.58	31.32	76.28	65.80	25.32	16.40	72.03	62.24	59.30	49.41	89.55
	972	78.77	27.73	29.00	28.44	33.12	31.78	11.40	41.88	77.80	51.87	256.22
	1080	109.64	22.14	42.15	58.69	59.54	87.32	58.82	49.93	46.32	95.21	83.22



Obrázek 46 - Chyba stanovení pozice kurzoru – Metoda A



Obrázek 47 - Chyba stanovení pozice kurzoru – Metoda B

Z naměřených dat lze vypozorovat, že největší chyba vzniká na krajích displeje. Tato skutečnost je způsobena nutností otáčet kamerou nebo souřadnicovým systémem, tak že kamera snímá značky pod ostrým úhlem, čímž vzniká chyba chybným stanovením rohů značky a tím vzniká chyba výpočtu stanovení pozice. Průměrná chyba po celé ploše obrazu je 48.58 px pro metodu A a 47.89 px pro metodu B. Průměrná chyba pro plochu displeje vně jeho kraje je 40.04 px pro metodu A a 41.11 px pro metodu B.

5.3 Porovnání navržených metod

Z měření v podkapitole 5.2 můžeme vyčíst, že přesnost stanovení pozice kamery je přibližně stejná pro metodu A a pro metodu B. Během samotného procesu měření byl však zaznamenám výskyt chybných hodnot stanovení pozice pro metodu B, bylo tedy nutno měření pro některé hodnoty opakovat za cílem tyto chybná data odfiltrovat. Oproti měření přesnosti stanovení pozice pro metodu A, kde nedocházelo téměř k žádnému výskytu chybných hodnot se jedná o nevýhodu metody. Tento šum metody B je způsoben nepřesnostmi ve výpočtu vzdálenosti značek od kamery, které způsobují chybné stanovení pozice kurzoru.

Další nevýhodou metody B je samotná aparatura, která zajišťuje pohyb souřadnicového systému v prostoru. Jedná se o poměrně velkou konstrukci, která není pro člověka pohodlná ani praktická pro používání, oproti metodě A, která umožňuje umístění kamery na prst.

5.4 Možná vylepšení realizace

Uvedená realizace řešení práce má své nevýhody a nedokonalosti. V této podkapitole jsou uvedeny možné modifikace, které by měly přispět k lepšímu sledování značek na snímcích nebo praktické modifikace usnadňující použití navržených metod.

Při rozostření snímků při rychlejším pohybu kamery dochází k přerušení detekce značek ve snímcích, tento jev můžeme pozorovat hlavně při použití *YOLO* detektoru. Jedná se o nedostatek způsobený pravděpodobně nedostatečně rozsáhlým datasetem. Doporučená velikost datasetu pro učení *YOLO* detektoru je 1000 snímků pro každou rozpoznávanou kategorii. Tento nedostatek by se také dal vyřešit např. použitím Kalmanova filtru, který umožňuje předpovídat polohu sledovaných objektů na snímku v případech, kdy jejich poloha nemůže být změřena.

Desktopová aplikace popsaná v podkapitole 3.2.1 byla vytvořena jako základní overlay zobrazující značky na monitoru, které však blokují klíčové oblasti umožňující ovládání ostatních aplikací na počítači. Aplikace by tedy mohla být např. upravena tak aby skryla značku na kterou bude ukázáno kurzorem, tím pádem

jsou stále k dispozici tři značky (podmínka stanovení pozice) a je odkryta část obrazovky, se kterou se uživatel snaží interagovat.

Jednou z nevýhod návrhu metody B je nutnost použití pomocné aparatury. Velikost této aparatury byla zvolena dle požadavků na přesnost stanovení pozice kamery. Tento problém by bylo možné eliminovat použitím dvou kamer, bylo by tedy méně požadavků na konstrukci aparatury pro přesné stanovení pozice. Výsledná aparatura by tedy mohla být mnohem menší a značek na aparatuře by mohlo být méně.

6 ZÁVĚR

V druhé kapitole byla popsána základní teorie zpracování obrazu, která byla využita v rámci řešení práce. Následuje rozbor základů hlubokého učení pomocí konvolučních neuronových sítí, které jsou v řešení práce využívány. Dále jsou uvedeny základy teorie geometrických transformací v 3D prostoru a je nastíněn problém vypočtení pozice kamery ze znalosti polohy klíčových bodů na snímku a jejich ekvivalentů ve 3D prostoru.

Další kapitola se zabývá návrhem metod pro navádění pomocí kamery. První metoda (označováno jako Metoda A) využívá značky umístěné na monitoru a pohybem kamery v prostoru je naváděn kurzor. Ovládání kurzoru na monitoru je zpětná vazba, podle které jednoduše určíme správnost stanovení pozice kamery v prostoru. Pro jednoduchost připravení prostředí metody A byla vytvořena desktopová aplikace, která zobrazí značky přímo na obrazovce monitoru. Druhá metoda (Metoda B) využívá pevně umístěnou kameru na monitoru a pohyb kurzoru je realizován pohybem souřadnicového systému v prostoru. Pohyb souřadnicového systému v prostoru je realizován pomocí pomocné aparatury.

Čtvrtá kapitola je věnována realizaci navržených metod. Jsou navrženy tři způsoby detekce značek na snímcích. První způsob využívá jako značky QR kódy, detekce těchto značek je realizována pomocí iterace přes snímek. Pro následující dvě metody jsou použity značky vlastního návrhu. První metoda pracující s těmito značkami využívá filtraci pomocí barev jako detekční algoritmus a pro klasifikaci nalezených značek je využita konvoluční neuronová síť založená na architektuře *VGG19*. Poslední metodou, která realizuje detekci i klasifikaci značek je metoda pracující s objektovým detektorem *YOLO*. Jedná se o neuronovou síť, která přistupuje k detekci značek jako k úloze regrese a tyto detekce klasifikuje. Ze znalosti polohy značek na snímcích a informaci o jejich poloze ve 3D prostoru je možno vyřešit *PnP* problém, následně jsou v kapitole uvedeny způsoby přepočtu stanovené polohy na polohu kurzoru na obrazovce.

Poslední kapitola se zabývá měřením výpočetních požadavků navržených algoritmů a přesnosti stanovení pozice metod A a B. Na základě těchto měření je proveden posudek a metody jsou mezi sebou porovnány. Následně je věnována podkapitola diskuzi o možných vylepšení realizace a způsob řešení nedostatků.

Literatura

[1] ŽÁRA Jiří. *Moderní počítačová grafika*. 2. přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[2] Qi WANG, Li FU, Zhenzhong LIU. *Review on Camera Calibration*, 2010 [online]. [cit. 2019-12-27]. Dostupné z: https://www.researchgate.net/publication/224151450_Review_on_Camera_Calibration

[3] *7x9 Checkerboard for camera calibration* [online]. [cit. 2019-12-27] Dostupné z: https://www.mrpt.org/downloads/camera-calibration-checker-board_9x7.pdf

[4] ImageNet [online]. [cit. 2020-05-21] Dostupné z: http://www.image-net.org/

[5] *Python software foundation. Python Language Reference, version 3.6* [online]. [cit. 2019-12-30] Dostupné z: https://www.python.org

[6] *Microsoft Visual Studio Code documentation* [online]. [cit. 2019-12-30] Dostupné z: https://code.visualstudio.com/docs

[7] *The Python package index (PyPI)* [online]. [cit. 2019-12-31] Dostupné z: https://pypi.org/

[8] *OpenCV* [online]. [cit. 2019-12-31] Dostupné z: https://opencv.org

[9] *PyAutoGUI documentation* [online]. [cit. 2019-12-31] Dostupné z: https://pyautogui.readthedocs.io/en/latest/index.html

[10] SIMONYAN Karen, ZISSERMAN Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition,* 2015 [online]. [cit. 2020-05-21]. Dostupné z: https://arxiv.org/abs/1409.1556

[11] OpenCV – Camera calibration and 3D reconstruction [online]. [cit. 2019-12-31] Dostupné z: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[12] LOUKARIS, Manolis. *A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar*, 2004 [online]. [cit. 2019-12-31]. Dostupné z: https://www.researchgate.net/publication/239328019_A_Brief_Description_of_the_Levenberg-Marquardt_Algorithm_Implemened_by_levmar

[13] Gao XIAO-SHAN, Hou XIAO-RONG, Tang JIANLIANG, Cheng HANG-FEI. *Complete Solution Classification for the Perspective-Three-Point Problem*, 2003 [online]. [cit. 2019-12-31]. Dostupné z: https://www.researchgate.net/publication/3193582_Complete_Solution_Classification_for_the_Perspective-Three-Point_Problem

[14] LEPETIT Vincent, MORENO-NOGUER Francesc, FUA Pascal. *EPnP: An accurate O(n) solution to the PnP problem*, 2009 [online]. [cit. 2019-12-31]. Dostupné z: https://www.researchgate.net/publication/49458221_EPnP_An_accurate_On_solution_to_the_PnP_problem
[15] OpenCV – Histograms [online]. [cit. 2020-01-01] Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html
[16] OpenCV Documentation: *Camera Calibration and 3D Reconstruction* [online]. [cit. 2020-01-03] Dostupné z: https://docs.opencv.org/master/d9/d0c/group__calib3d.html
[17] MathWorks: *Camera Calibration* [online]. [cit. 2020-01-03] Dostupné z: https://www.mathworks.com/help/vision/ug/camera-calibration.html
[18] Lu XIAO. *A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation*, 2018 [online]. [cit. 2020-01-03]. Dostupné z: https://www.researchgate.net/publication/328036802_A_Review_of_Solutions_for_Perspective-n-Point_Problem_in_Camera_Pose_Estimation
[19] GOODFELLOW Ian, BENGIO Yoshua, COURVILLE, Aaron. *Deep learning*. MIT Press, 2016. ISBN 9780262035613.
[20] BASHA S. H. Shabbeer, DUBEY Shiv Ram, PULABAIGARI Viswanath, MUKHERJEE Snehasis. *Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification*, 2019 [online]. [cit. 2020-05-14]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S0925231219313803
[21] O'SHEA Keiron Teil, NASH Ryan. *An Introduction to Convolutional Neural Networks*, 2015 [online]. [cit. 2020-05-14]. Dostupné z: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks
[22] Keras – Dropout layer [online]. [cit. 2020-05-14] Dostupné z: https://keras.io/api/layers/regularization_layers/dropout/
[22] Keras – BatchNormalization layer [online]. [cit. 2020-05-14] Dostupné z: https://keras.io/api/layers/normalization_layers/batch_normalization/
[23] SAMARASINGHE Sandhya. *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. Auerbach Publications, 2006. ISBN 9780849333750.
[24] NWANKPA Chigozie, IJOMAH Winifred, GACHAGAN Anthony, MARSHALL Stephen. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*, 2019 [online]. [cit. 2020-05-14]. Dostupné z: https://www.researchgate.net/publication/328826136_Activation_Functions_Comparison_of_trends_in_Practice_and_Research_for_Deep_Learning
[25] GONZALEZ Rafael C., WOODS Richard E. *Digital Image Processing (4th Edition)*. Pearson 2017. ISBN 9780133356724.

[26] Chang JUN-DONG, Yu SHYR-SHEN, Chen HONG-HAO, Tsai CHWEI-SHYONG. *HSV-based Color Texture Image Classification using Wavelet Transform and Motif Patterns*, 2010 [online]. [cit. 2020-05-15]. Dostupné z: https://www.researchgate.net/publication/228895050_HSV-based_Color_Texture_Image_Classification_using_Wavelet_Transform_and_Motif_Patterns

[27] ŠŤASTNÝ Jiří, ŠKORPIL Vladislav. *Neural networks learning methods comparison*, 2005 [online]. [cit. 2020-05-20]. Dostupné z: https://www.researchgate.net/publication/292636184_Neural_networks_learning_methods_comparison

[28] RUDER Sebastian. *An overview of gradient descent optimization algorithms*, 2016 [online]. [cit. 2020-05-20]. Dostupné z: https://arxiv.org/abs/1609.04747

[29] SATHYA R., ABRAHAM Annamma. *Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification*, 2013 [online]. [cit. 2020-05-20]. Dostupné z: https://www.researchgate.net/publication/273246843_Comparison_of_Supervised_and_Unsupervised_Learning_Algorithms_for_Pattern_Classification

[30] SMOLA Alex, VISHWANATHAN S.V.N. *Introduction to Machine Learning*. Cambridge University Press, 2008. ISBN 0 521 82583 0.

[31] *Tensorflow* [online]. [cit. 2020-02-20] Dostupné z: https://www.tensorflow.org/

[32] *Keras* [online]. [cit. 2020-02-20] Dostupné z: https://keras.io/

[33] *SOLIDWORKS* [online]. [cit. 2020-02-21] Dostupné z: https://www.solidworks.com/

[34] *Creative Labs WebCam Live! Ultra* [online]. [cit. 2020-02-21] Dostupné z: https://support.creative.com/Products/ProductDetails.aspx?ProdID=10451&prodName=WebCam+Live!+Ultra

[35] OpenCV – Structural Analysis and Shape Descriptors [online]. [cit. 2020-05-21] Dostupné z: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[36] SUZUKI Satoshi, BE Keiichi A. *Topological structural analysis of digitized binary images by border following*, 1985 [online]. [cit. 2020-05-21]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/0734189X85900167

[37] TEH C.H., CHIN Roland T. *On the detection of dominant points on digital curve,* 1989 [online]. [cit. 2020-05-21]. Dostupné z: https://www.researchgate.net/publication/3191687_On_the_detection_of_dominant_points_on_digital_curve

[38] SRISHA Ravi, KHAN Am. *Morphological Operations for Image Processing : Understanding and its Applications,* 2013 [online]. [cit. 2020-05-21]. Dostupné z: https://www.researchgate.net/publication/3191687_On_the_detection_of_dominant_points_on_digital_curve

[39] REDMON Joseph, DIVVALA Santosh, GIRSHICK Ross, FARHADI Ali. *You Only Look Once: Unified, Real-Time Object Detection,* 2015 [online]. [cit. 2020-05-22]. Dostupné z: https://arxiv.org/abs/1506.02640

[40] REDMON Joseph, FARHADI Ali. *YOLO9000: Better, Faster, Stronger,* 2016 [online]. [cit. 2020-05-22]. Dostupné z: https://arxiv.org/abs/1612.08242

[41] REDMON Joseph, FARHADI Ali. *YOLOv3: An Incremental Improvement,* 2018 [online]. [cit. 2020-05-22]. Dostupné z: https://arxiv.org/abs/1804.02767

Seznam symbolů, veličin a zkratek

px	Pixel
2D	Dvourozměrný
3D	Trojrozměrný
OpenCV	Open Source Computer Vision knihovna
ANN	Umělá neuronová síť
CNN	Konvoluční neuronová síť
LUT	Vyhledávací tabulka (Lookup table)
ReLU	Rectified Linear Unit
IOU	Intersection Over Union

Seznam příloh na CD

Název souboru/složky		Význam
./README.md	–	Návod pro použití a zprovoznění přiložených programů
./camera/calibration/webcam/	–	Kalibrační snímky webkamery
./camera/calibration/endoscope/	–	Kalibrační snímky endoskopu
./camera/webcam_params.json	–	Vnitřní parametry webkamery
./camera/endoscope_params.json	–	Vnitřní parametry endoskopu
./dataset/vgg19/	–	Snímky pro učení sítě VGG19
./dataset/yolov3/	–	Snímky pro učení sítě YOLO
./marks/	–	Použité značky v rámci řešení práce
./calibrate_camera.py	–	Skript pro kalibraci kamery
./construct_tf_records.py	–	Skript pro konverzi datasetu viz. návod
./cursor_control.py	–	Hlavní knihovna práce
./live_controls.py	–	Hlavní skript práce viz. návod
./load_darknet_weights.py	–	Skript pro konverzi vah viz. návod
./overlay.py	–	Desktopová aplikace zobrazující značky na monitoru
./utility.py	–	Pomocná knihovna práce
./vgg19.py	–	Skript pro učení sítě VGG19 viz. návod
./yolov3_model.py	–	Konstrukce modelu YOLO sítě
./yolov3_train.py	–	Skript pro učení sítě YOLO viz. návod