ČVUT V PRAZE
FAKULTA STROJNÍ
ÚSTAV PŘÍSTROJOVÉ A ŘÍDICÍ TECHNIKY

ROZPOZNÁVÁNÍ OBRAZU V OKOLÍ ROBOTICKÉ RUKY PRO ÚČELY LOKALIZACE HLEDANÉHO PŘEDMĚTU

BAKALÁŘSKÁ PRÁCE

Autor: Jan Lorenc

Vedoucí: Ing. Zdeněk Novák, PhD.

2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lorenc** Jméno: **Jan** Osobní číslo: **482618**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídicí techniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rozpoznávání obrazu v okolí robotické ruky pro účely lokalizace hledaného předmětu

Název bakalářské práce anglicky:

Image recognition in proximity of a robotic arm for the purpose of locating the searched object

Pokyny pro vypracování:

1) Vypracujte rešerši na téma rozpoznávání obrazu. Analyzujte jednotlivé metody pro účely splnění podmínky lokalizace hledaného předmětu v okolí robotické ruky.
2) Zvolenou metodu naprogramujte a experimentálně ověřte.
3) Zhodnoťte dosažené výsledky a úspěšnost nalezení hledaného předmětu.

Seznam doporučené literatury:

[1] BRADSKI, Gary a KAEHLER, Adrian. Learning OpenCV: Computer Vision with the OpenCV Library. Sebastopol, CA : O'Reilly Media, Inc., 2008.
[2] HOWSE, Joseph a Joe MINICHINO. Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning. 3rd. edition. Birmingham: Packt Publishing, 2020. ISBN 9781789531619.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Zdeněk Novák, Ph.D., U12110.1

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.04.2021** Termín odevzdání bakalářské práce: **10.06.2021**

Platnost zadání bakalářské práce: _____

Ing. Zdeněk Novák, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 zákona č.121/2000Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

v Praze dne podpis autora

Abstrakt

Tématem bakalářské práce je využití strojového vidění k ovládání pohybů robotické ruky BCN3D Moveo. Teoretická část mapuje dostupné metody detekce objektů a jejich trasování. V rámci praktické části je v jazyce Python vytvořen program, s jehož pomocí jsou objekty rozpoznány, rozlišeny a sledovány, jsou určeny jejich souřadnice pro účely automatického uchopení a pomocí PyQt5 je vytvořeno grafické prostředí k ovládání nových funkcionalit. Po dokončení je systém otestován a je zhodnoceno jeho využití.

Klíčová slova: BCN3D Moveo, Python, PyQt5, GUI, robotická ruka, strojové vidění, detekce objektů, OpenCV, trasování, Eukleidovský tracker, referenční markery, ArUco

Abstract

The bachelor thesis studies possibilities of motion control of a BCN3D Moveo robotic arm via methods of computer vision. The theoretical section maps available methods of object detection and tracking. Within the practical section, a Python program is created to detect and track objects, as well as to determine those objects' coordinates for purposes of motion control. All forementioned functionalities are accessed through a GUI created using PyQt5 library. The finished software is tested, and its usability is evaluated.

Key Words: BCN3D Moveo, Python, PyQt5, GUI, robotic arm, computer vision, object detection, OpenCV, object tracking, Euclidian tracker, fiducial marker sets, ArUco

Poděkování

Děkuji svému vedoucímu Ing. Zdeňku Novákovi, PhD. za konzultace a rady při vypracovávání rešeršní části, stejně tak jako za pomoc a asistenci při části praktické. Dále děkuji Ing. Matouši Cejnekovi, PhD. za mnohé tipy a připomínky k praktické části.

Obsah

Seznam zkratek

BRIEF	Binary Robust Independent Elementary Features
BSD	Berkeley Software Distribution
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CSRT	Discriminative Correlation Filter with Channel and Spatial Reliability
CV	Computer Vision
DCF	Discriminative Correlation Filter
FAST	Features from Accelerated Segment Test
FMS	Fiducial Marker Sets
FPS	Frames per Second
GOTURN	Generic Object Tracking Using Regression Networks
GUI	Graphic User Interface
HoG	Histogram of Oriented Gradients
HSL	Hue, Saturation, Lightness
IDE	Integrated Development Environment
KCF	Kernalized Correlation Filters
LoG	Laplacian of Gaussian
mAP	Mean Average Precision
MIL	Multiple Instance Learning
MOSSE	Minimum Output Sum of Squared Error
ORB	Oriented FAST and Rotated BRIEF
PC	Personal Computer
RGB	Red, Green, Blue
RoI	Region of Interest
RPN	Region Proposal Networks
RTL	Runtime Library
SIFT	Scale-Invariant Feature Transform
SSD	Single Shot Detector
STL	formát souboru pro CAD
SURF	Speeded-Up Robust Features
TLD	Tracking, learning, detection
YOLO	You Only Look Once

Seznam obrázků

Seznam rovnic

Seznam tabulek

1 Úvod

V České republice v roce 2017 podle údajů ČSÚ (1) disponovalo robotickými výrobními linkami 53,4 % podniků s alespoň 250 zaměstnanci. Podle zprávy skupiny Deloitte (2) se obzvláště např. automobilový průmysl se 75,3 % začíná blížit limitu toho, co automatizovat lze. Ve snaze zvýšit svou produktivitu tak tyto podniky mohou směřovat ke zdokonalování stávajících automatických výrobních linek spíše než k dokupování dalších robotů. Strojové vidění (CV) bezpochyby patří mezi funkce, jimiž lze výrobu zefektivnit, a to bez větších investic.

Vzhledem k nedostatku pracovníků na trhu práce a k pravděpodobnému zvětšování tohoto nedostatku v důsledku demografického vývoje v ČR (3) je snahou podniků využít svých lidských zdrojů co nejefektivněji. Z tohoto důvodu jsou rutinní činnosti přenechávány robotům, což kromě úspory zaměstnanců a jejich zdraví přináší i další výhody. Robot práci provede se 100 % přesností, je zlepšena stabilita a snížena chybovost. Robot může být v provozu 24 hodin denně, čímž roste výrobní kapacita. Robot může být naprogramován pro různé úkony. Díky strojovému vidění přestal být zrak doménou člověka, a ten tak mohl být dále nahrazován. (2)

Krása strojového vidění spočívá v jeho všestrannosti. Častou aplikací jsou kontrolní činnosti, např. kontrola kvality výrobku, kontrola balení atp. Zde – obzvláště pak při hledání mikroskopických vad – kamery přesahují schopnosti lidského oka. CV systémy jsou schopny provádět nepřetržitou inspekci průmyslového procesu a umožňují předcházet defektům. Kromě dohledu nad stroji můžou CV systémy vykonávat i dohled nad zaměstnanci a zvyšovat tím bezpečnost provozu. (4)

Úkolem této závěrečné práce bude navrhnout systém strojového vidění k rozpoznání objektů za účelem jejich uchopení robotickou rukou. K dispozici je robotická ruka s pěti stupni volnosti BCN3D Moveo a fungujícím systémem ovládání pohybů. Tento systém bude rozšířen o možnosti strojového vidění a výsledek bude experimentálně otestován.

1.1 BCN3D

Jak je uvedeno na oficiálních stránkách BCN3D Technologies (5), robotická ruka BCN3D Moveo vznikla ve spolupráci firmy BCN3D Technologies a katalánského ministerstva školství. Byla navržena jako open-source projekt s cílem zpřístupnit víceosé robotické paže studentům strojírenských VŠ v rámci tréninku dříve, než se setkají s roboty průmyslovými. Výsledkem této snahy je robotická ruka s pěti stupni volnosti, jejíž neelektrické komponenty lze vyrobit na 3D tiskárně. Ruka byla navržena pro ovládání deskou Arduino, tato deska je podobně jako samotná ruka levná a uživatelsky přívětivá.

Veškerá dokumentace je dostupná skrze Github. K dispozici jsou CAD modely jednotlivých součástí i celé sestavy. Všechny potřebné díly – vytisknutelné i dokupované – jsou vypsány v kusovníku.

Uživatel má možnost modely měnit na míru svým požadavkům. Nemá-li uživatel potřebu konstrukci měnit, lze využít dostupných STL souborů pro 3D tisk. Přiložený je i modifikovaný firmware Marlin k ovládání robota.

Zadávací pracoviště bakalářské práce má ve svém vlastnictví již sestaveného robota BCN3D Moveo a to díky absolventské práci Patrika Zacha (6). Srdcem řídicí jednotky je deska Arduino Mega 2560, ta ovládá plošnou desku Shield RAMPS 1.4, sloužící k řízení v kartézském souřadnicovém systému, a Stepper drivery DRV8825, které ovládají použité krokové motory.

1.2 Návrh

Díky Zachově bakalářské práci (6) existuje systém s grafickým uživatelským prostředím, pomocí něhož lze manuálně řídit pohyby sestaveného robota. Tato práce bude mít za úkol rozšířit systém o další funkcionality.

Prvním rozšířením robotické ruky bude kamera a systém strojového vidění. Od systému strojového vidění bude požadováno detekovat předměty nacházející se na pracovní ploše.

Druhým úkolem programu bude schopnost rozlišit mezi detekovanými objekty. To je důležité kvůli určení souřadnic uchopovaného objektu.

Třetím rozšířením bude systém, kterým budou automaticky řízeny pohyby robotické ruky. Ta detekovaný předmět uchopí a přemístí jej na předem zvolené místo. Právě v této části bude využito v úvodu zmíněné bakalářské práce kolegy Zacha a jím vytvořeného systému implementujícího algoritmy k řešení přímé a inverzní kinematiky.

Výsledný program bude schopný identifikovat zvolený předmět, přesunout koncovou část robotické ruky do vhodné polohy, uchopit předmět a přemístit jej na předem stanovené místo. Úkolem obsluhy bude pouze stanovit, který předmět má být uchopen, a kam má být přesunut. Důraz bude kladen především na rychlost celého procesu, ale také na přesnost a maximální nezávislost fungování systému na typu a umístění sledovaného předmětu a světelných podmínkách.

2 Rešerše

2.1 Kamera

Použita bude kamera Creative Live! Cam Sync HD. Použitý snímač umožňuje natáčení videa o rozlišení HD (1280 x 720) při rychlosti snímání 30 FPS. Ohnisková vzdálenost je pevně nastavena na f/2,4. Kamera s počítačem komunikuje prostřednictvím výstupu USB2.0.

2.2 Dostupné softwarové prostředky

2.2.1 Python

Volba programovacího jazyka je prvním a zásadním krokem při tvorbě jakékoli aplikace. Programátoři se rádi dohadují, který jazyk je nejlepší – určujícím hlediskem volby jazyka je funkce, kterou bude mít. V případě strojového vidění je nejčastěji zmiňován jeden – Python (7).

Jak je uvedeno v Učebnici jazyka Python od Švece (8), Python byl vyvinut Guidem van Rossumem v roce 1991. Při jeho tvorbě kladl tento milovník britských Monty Pythonů důraz na jednoduchost a čitelnost. Python má několik důležitých vlastností, kterými jsou tyto požadavky naplněny: objektově orientovaný – Jazyk využívá objektů, malých, samostatných jednotek, které lze opětovně používat ve více aplikacích; rozšiřitelný – Se znalostí jazyka C lze přidávat nové funkce a moduly jazyka; interpretovaný – Program napsaný v Pythonu lze interaktivně spouštět a testovat bez nutnosti jej překládat do strojového kódu (kompilovat); vysokoúrovňový – Jazyk se vyznačuje větší mírou abstrakce, díky čemuž vypadá lidštěji; dynamicky typovaný – U proměnných nemusí být specifikován datový typ, jelikož typová kontrola probíhá za běhu programu.

2.2.2 PyCharm

PyCharm je integrované vývojové prostředí (IDE) určené primárně pro práci s jazykem Python. Byl vyvinut firmou JetBrains se sídlem v Praze.

Nabízí asistenci při psaní kódu prostřednictvím funkcí autocomplete a quick-fix, zvýrazňováním chyb a automatickou opravou. Umožňuje jednoduchou navigaci mezi soubory a orientaci v kódu. PyCharm je schopen identifikovat úseky kódu vhodné k refaktorování – redukci komplexnost a zvýšení čitelnost kódu bez změny funkcionality programu. Jeho součástí je debugger, nástroj umožňující sledovat chod programu za účelem opravy chyb. Community verze je dostupná zdarma. (9)

2.2.3 Knihovny

OpenCV

OpenCV je dnes možná nejrozšířenější knihovna pro práci se strojovým viděním. Za jejím vývojem stojí Gary Bradski. Stěžejními zdroji informací jsou dokumentace OpenCV (10) a Bradskiho učebnice (11).

OpenCV je rozsáhlá multiplatformní open-source knihovna vytvořená v jazyce C++. Jedná se o soubor algoritmů umožňujících využití strojového vidění a učení i v reálném čase. Jeho součástí je přibližně pět set funkcí. OpenCV je v rámci BSD licence dostupná zdarma k výzkumným i komerčním účelům. Díky dostupnosti této knihovny existuje mnoho pro účely této práce relevantních internetových návodů.

Knihovna je tvořena mnoha nezávislými moduly navrženými tak, aby každý plnil pouze jednu konkrétní funkcionality. Modulů existuje velké množství a každá aplikace si vystačí jen s některými z nich. Jejich výčet a obsah společně s detailním popisem principu použitých algoritmů lze vyčíst z rozsáhlé veřejně přístupné dokumentace dostupné na webových stránkách OpenCV. Účelům této práce budou dostačující následující moduly a jim příslušející funkce, třídy a datové a výčtové typy.

core – Jedná se o základní stavební blok celé knihovny. Definuje základní datové struktury jako skalár nebo bod. Důležitým dodatkem je matice Mat používaná k ukládání obrázků.

imgproc – Tento modul umožňuje upravit obrázek do zjednodušené a snáze čitelné podoby (převod barev, transformace geometrie, rozostření).

video – Modul video zajišťuje odhad pohybu a sledování objektů nebo rozpoznání pozadí.

features2d – Features2D hledá a definuje charakteristické rysy pro daný objekt.

NumPy

Knhihovna NumPy byla vytvořena Travisem Oliphantem v jazycích C a Python roku 2006. Původně používaný CPython měl problémy s rychlostí provádění matematických operací. Úkolem NumPy bylo využít toho, že je Python interpretovaný, a tedy vhodný jako matematický programovací jazyk.

Jak lze vyčíst z dokumentace k NumPy dostupné na webu (12), knihovna NumPy vznikla za účelem rozšíření Pythonu o možnost provádět pokročilé matematické operace. Přidává možnost vytvářet n-rozměrná pole, jeho součástí je řada matematických funkcí a Fourierovy transformace. Schopnost vytvářet velká n-rozměrná pole je obzvláště užitečná pro knihovnu OpenCV, jelikož digitální obrázek je defacto také jen pole čísel.

NumPy je dostupná zdarma skrze BSD licenci.

PyQt5

Knihovna PyQt5 je vedle tkinter jedním z často používaných prostředků k tvorbě grafických uživatelských rozhraní (GUI). Důvodem volby PyQt5 je návaznost na existující GUI vytvořené kolegou Zachem.

Qt je sadou multiplatformních knihoven vytvořených v jazyce C++, PyQt5 potom tvoří vazbu mezi Qt v5 a Pythonem. Tato vazba umožňuje využití Pythonu k tvorbě mobilních i desktopových aplikací napsaných v podporovaných programovacích jazycích. I díky tomu nachází Qt uplatnění v mnoha odvětvích včetně vytváření GUI.

Existuje několik softwarových nástrojů, pomocí nichž lze aplikaci v PyQt5 vytvářet metodou Drag and drop. Za zmínku stojí program Qt Designer, který je dostupný zdarma. (13)

2.3 Algoritmy pro rozpoznání obrazu

Algoritmus je návod, resp. postup, vedoucí k vyřešení stanoveného problému. Výše zmíněné knihovny a software poskytují prostředky a rámec pro implementaci algoritmů. Pro každou požadovanou funkcionalitu bude popsáno několik různých algoritmů, z nichž bude vybrán ten nejvhodnější. Při hledání vhodných algoritmů je potřeba klást obzvláštní důraz na rychlost zpracování obrazu kvůli použití v reálném čase. Cílem je najít vhodný kompromis mezi rychlostí a přesností/spolehlivostí.

2.3.1 Detekce kontur

K jednoznačnému popisu polohy objektu na snímku je řada informací na snímku nadbytečná. Dostatečně efektivity při rozpoznání a detekci objektu lze dosáhnout pouze analýzou jeho kontury. Kontura je spojitá křivka kopírující viditelnou hranici objektu. Díky tomu, že hledaný objekt bývá od svého okolí odlišen svou barvou nebo jejím odstínem, lze konturu matematicky popsat jako oblast s největším barevným gradientem.

Aby detekce kontur probíhala co nejefektivněji a bez fluktuací, je vhodné z obrázku odfiltrovat co nejvíce zbytečných informací. V ideálním stavu je výsledkem použití filtrů taková maska, na níž je vidět pouze detekovaný objekt a zbytek je skrytý. Filtrů existuje celá řada, každý z nich eliminuje určitou informaci, která by mohl vést k šumu (např. barevný filtr (14), Sobel filtr (10), prahový filtr/threshold (10), Gaussian blur (10)…). Filtrů může být použito několik, vše závisí na konkrétní aplikaci. Výsledná maska by měla mít podobu binárního obrázku (bílá/černá), kde detekované objekty jsou jednou barvou a zbytek (pozadí) druhou.

Jeli docíleno toho, že je kontura na obrázku dostatečně zřetelná, je aplikována funkce, jejímž úkolem je nalézt v obrázku uzavřené kontury a uložit je do matice. Takovou funkcí je např.

cv2.findContours() z knihovny OpenCV. S konturami lze v této podobě provádět další operace, pomocí nichž lze nálezy dále klasifikovat. (10)

2.3.2 Detekce pomocí subtrakce pozadí

Pozadím se v řeči strojového vidění myslí ta část obrazu, která je sice zaznamenána, ale není hlavním předmětem zájmu. Popředí je chápáno naopak jako soubor sledovaných objektů. Úkolem algoritmů subtrakce pozadí je vytvořit masku, jejíž aplikací na vstupní snímek získáme nový snímek, kde jsou vidět pouze objekty popředí a zbytek je skrytý.

Nastane-li situace, že je podoba pozadí známá ještě před spuštěním programu, lze masku popředí vytvořit pouhým odečtením uloženého snímku pozadí od aktuálního snímku z kamery. Ačkoli tato varianta zní teoreticky jednoduše, v praxi do takovéto situace vstupují nepříznivé vnější vlivy, např. přesné nastavení kamery, nebo světelné podmínky, včetně stínu sledovaných objektů.

V praktických aplikacích nebývá podoba pozadí známá předem. Při separaci pozadí od popředí se v takovém případě uvažuje, že pozadí zůstává statické, kdežto objekty popředí se pohybují. Předmětem srovnávání jsou potom sousledné snímky zaznamenané v určitých časových intervalech. Předpokladem pro správné fungování algoritmu je, že se objekty pohybují a kamera je statická.

Na tomto principu funguje algoritmus MOG, při jehož implementaci lze s výhodou využít knihovny OpenCV. V dokumentaci (10) je podrobně vysvětleno fungování příslušné funkce *cv2.BackgroundSubtractorMOG()*.

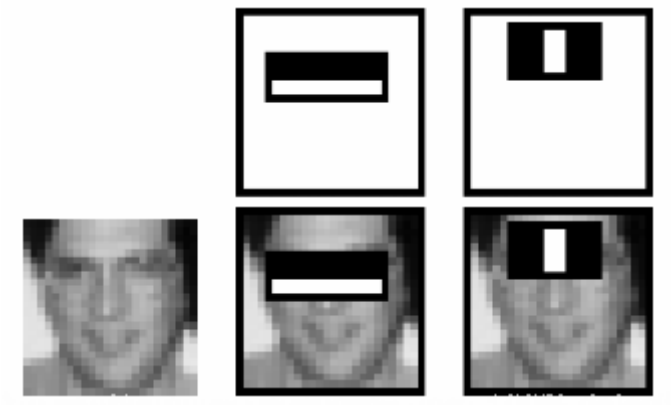
2.3.3 Detekce charakteristických rysů

Charakteristické rysy jsou unikátní pro každý objekt umožňují přesnou klasifikaci nalezeného objektu (např. zdali se jedná o vozidlo nebo strom). V tomto směru jde o krok dopředu oproti detektorům popsaným výše, které pouze odlišují objekt od pozadí. Algoritmů pro detekci pomocí charakteristických rysů existuje mnoho, ale důraz bude kladen na ty, jež jsou implementovány funkcemi dostupnými v OpenCV. Všechny tyto algoritmy jsou popsány v příslušných výzkumných pracích a v dokumentaci ke knihovně OpenCV (10).

Cascade Classifier

Metoda Cascade Classifier byla poprvé představena roku 2001 výzkumem „Rapid Object Detection using a Boosted Cascade of Simple Features“ od Paula Violy a Michaela Jonese (15). Využívá strojového učení, jehož prostřednictvím jsou z rozsáhlých sad učebních obrázků – tzv. pozitivů a negativů – extrahovány údaje o charakteristických rysech. Použitím většího množství charakteristických rysů je následně možné rozpoznat příslušné objekty na jiných než učebních obrázcích.

Rysy jsou vyhledávány pomocí kernelů – obdélníkových matic rozdělených na sektory. Kernely jsou umisťovány na všechna místa na obrázku a pixely odpovídající různým sektorům kernelu jsou od sebe odečítány. Tímto způsobem jsou nalezeny různé oblasti na obrázku a rozdíly mezi nimi. Např. na obrázku níže kernel pokrývá zvlášť oči a oblast pod očima a je zjištěno, že očím odpovídají tmavší pixely než jejich okolí. Vzhledem k počtu možných velikostí a pozic kernelů je nalezeno obrovské množství rysů (pro obrázek 24x24 více než 160 000). Valná většina těchto rysů je nevypovídající a jejich hledání na vstupním obrázku by vzhledem k velkému potřebnému výpočetnímu výkonu vedlo k výraznému zpomalení detekce. Obzvláště pro účely detekce v reálném čase je počet rysů zredukován pomocí Adaboost.



Obrázek 1: Princip Cascade Classifier. (10)

Adaboost má za úkol roztřídit rysy podle přesnosti, s kterou určují, jedná-li se o pozitiv nebo negativ. Pro výslednou detekci jsou vybrány pouze rysy, které byly schopny negativy od pozitivů rozlišit s přesností přesahující stanovenou hranici. Dle Violy a Jonese stačí 200 rysů k dosažení 95% přesnosti.

Vybrané rysy jsou nakonec rozděleny do skupin – kaskád. Obrázek je klasifikován těmito kaskádami postupně ve stádiích – pokud obrázek neprojde stádiem jedna, nepokračuje do stádia dvě – a tím je detekce dále urychlena.

Výhodou této metody je jednoduchá implementace – samotná knihovna OpenCV disponuje funkcemi pro čtení i vytváření „vyučených“ modelů z pozitivů a negativů.

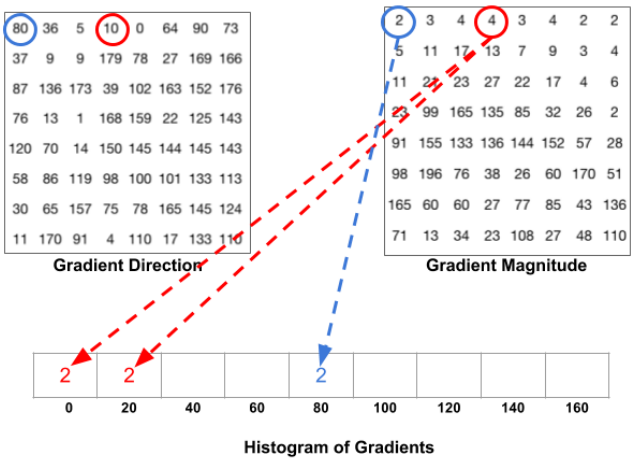
Histogram of Oriented Gradients (HoG)

Dalším způsobem, jak popsat a extrahovat charakteristické rysy objektu z obrázku, jsou histogramy. Pomocí histogramu lze popsat rysy jednoho unikátního hledaného objektu. Tuto metodu navrhli Dalal a Triggs ve své práci (16).

Prvním krokem při vytváření HoG je výpočet velikosti a směru gradientu pro každý pixel obrázku. Histogramy jsou vytvářeny pro jednotlivé buňky obrázku o rozměrech 8x8 pixelů. Velikosti gradientů jsou rozřazeny do devíti polí odpovídajících určitému intervalu hodnoty směru. Histogram si lze

představit jako sloupcový graf znázorňující distribuci intenzit jednotlivých barev (RGB) pixelů v dané části snímku. Kvůli eliminaci vlivu rozdílného nasvícení různých částí obrázku jsou histogramy vytvořeny i pro bloky tvořené 2x2 buňkami, tedy rozměr 16x16 pixelů.

Systém detekce má k dispozici vícero histogramů, které jsou přiřazovány k různým částem snímku společně s pravděpodobnostmi jejich správného přiřazení. Tyto pravděpodobnosti jsou uloženy do mapy a oblasti mapy s vyššími pravděpodobnostmi odpovídají částem obrázku, kde se vyskytuje hledaný objekt.



Obrázek 2: Schéma vzniku histogramu. (39)

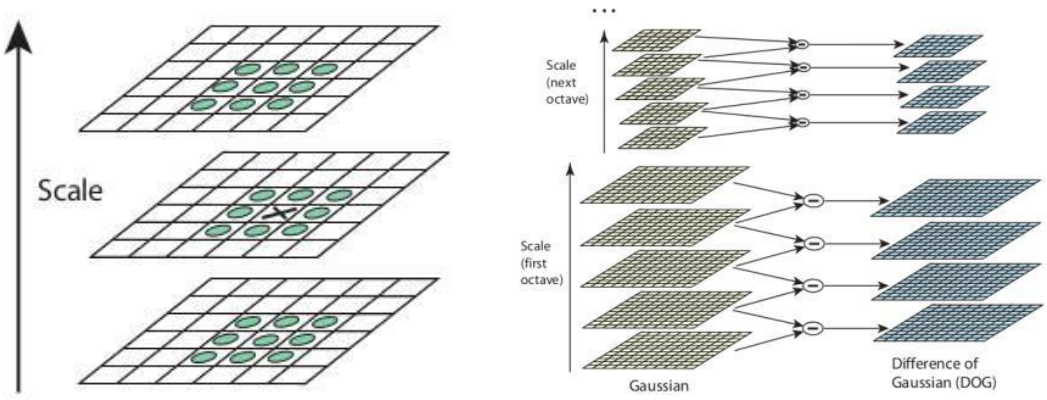
Knihovna OpenCV obsahuje funkci *cv2.HOGDescriptor()*, která umožňuje snadnou implementaci detekce pomocí histogramů. Existují dva algoritmy detekce, které využívají HoG – Meanshift a Camshift.

Scale-invariant Feature Transform (SIFT)

Dalším, kdo se věnoval hledání metody detekce charakteristických rysů, byl Lowe ve své práci „Distinctive Image Features from Scale-Invariant Keypoints“ (17). Metoda SIFT se vyznačuje značnou variabilitou a nezávislostí na natočení a velikosti (přiblížení) objektu. Nezávislost na natočení vychází z Harrisova detektoru rohů, na němž je SIFT postaven, a nezávislost na velikosti je přidána SIFTem.

Masky detekující rohy sledují intenzitu barev v oblasti masky a jejich změnu při přesunu masky – podle toho je prvek vyhodnocen buď jako roh, nebo jako hrana. Oblast obsažená maskou závisí na velikosti masky a velikosti obrázku a při použití příliš malé masky na příliš velký obrázek může být roh mylně identifikován jako hrana. SIFT umožňuje měnit velikost masky podle okamžité potřeby prostřednictvím parametru σ filtru LoG (Laplacian of Gaussian) pro odstranění šumu – větším hodnotám σ odpovídá větší maska a naopak. Odstranění šumu pomocí LoG je výpočetně náročné, a tak je používán rozdíl gaussiánů jako aproximace LoG. Proces nalezení rozdílů gaussiánů je zopakován pro několik oktáv v Gaussově pyramidě. Rozdíly gaussiánů jsou následně prohledány a jsou nalezena lokální maxima, jejichž hodnota je vyšší než mezní hodnota. Pro tyto body je sestaven deskriptor tvořený gradienty

v okolí těchto bodů. Deskriptor popisuje charakteristické rysy objektu a je srovnáván s obdobným deskriptorem vytvořeným pro zkoumaný obrázek, a tak jsou hledány dané objekty.



Obrázek 3: Schéma SIFT. (10)

Speeded-Up Robust Features (SURF)

Jak lze vyčíst z práce „Speeded-Up Robust Features (SURF)“ od Baye a spol. (18), metoda SURF je podobná metodě SIFT způsobem, jakým vytváří deskriptor z lokálních maxim aproximace LoG. Omezením SIFTu je časová náročnost výpočtu rozdílu gaussiánů na několika snímcích různých velikostí. SURF zavádí další zjednodušení ve formě aproximace rozdílu gaussiánů prostřednictvím pouze jednoho filtru. Díky tomuto vylepšení lze SURF použít v reálném čase.

Filtr využívaný SURFem, tzv. Box Filter, umožňuje docílit stejného rozostření obrazu (Gaussian blur), jakého je dosaženo SIFTem využívaným rozdílem gaussiánů. Filtr má tvar čtverce a jeho aplikace je velmi rychlá při použití integrálního obrazu (každý bod je součtem všech bodů nalevo nahoru od tohoto bodu na původním obrázku). Zájmové body jsou hledány pomocí čtvercových Hessiánských matic – body leží v místech, kde jsou determinanty těchto matic největší.

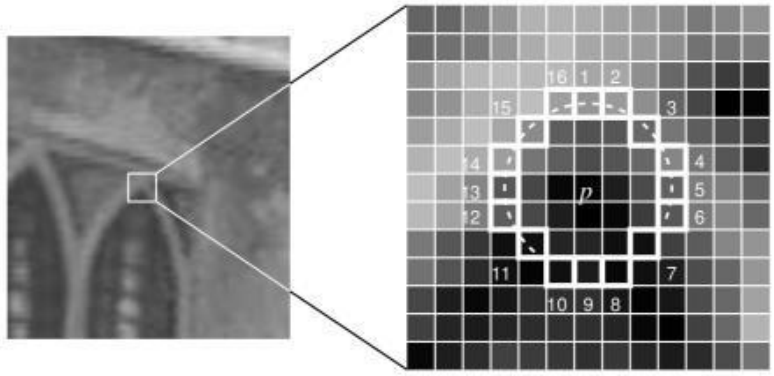
Výsledkem těchto modifikací oproti SIFTu je třikrát vyšší rychlost beze změny kvality detekce.

Features from Accelerated Segment Test (FAST)

Samotný název napovídá, že metoda FAST vede k ještě výraznějšímu zrychlení detekce charakteristických rysů. Algoritmus FAST byl představen prací „Machine-Learning for High-Speed Corner Detection“ od Rostena a Drummonda (19). Potřeba rychlejšího detektoru vycházela zejména z použití na malých robotech s omezeným výpočetním výkonem. Součástí je i algoritmus strojového učení pro vytváření rozhodovacího stromu na základě proměnné K_p, která označuje, jestli bod je nebo není roh.

Proces označování rohu pomocí FAST má 4 fáze: 1) Je vybrán pixel p a definována jeho intenzita I_p. 2) Pro danou intenzitu I_p je vybrána odpovídající hraniční hodnota t. 3) V okolí pixelu p je vybrána kružnice tvořená 16 pixely. 4) Intenzity pixelů kružnice jsou srovnány – jsou-li větší než I_p+t, bod je

označen jako světlý, jsou-li menší než I_p-t, bod je označen jako tmavý. Alespoň 12 z těchto 16 bodů musí patřit do jedné ze skupin světlé/tmavé, aby byl bod p označen jako roh. Existuje i vysokorychlostní varianta algoritmu FAST, která nejprve porovnává protilehlé body (na obr. 7 označené 1, 9, 5, 13) a k bodům v jejich příslušných okolích přistoupí jen tehdy, pokud jsou ony protilehlé pixely vyhodnoceny jako světlé/tmavé.



Obrázek 4: Schéma FAST. (10)

Metoda detekce rohů FAST je velmi rychlá, ale také silně citlivá na šum v obrázku. Zásadní nevýhodou je pak nemožnost identifikovat rysy při jejich natočení o více než ±15°.

Binary Robust Independent Elementary Features (BRIEF)

Metody SIFT a SURF ukládají informace o charakteristických rysech do vektorů se 128, resp. 64 prvky. Vektory obsahují proměnné typu float, a tak zabírají 512, resp. 256 bytů. Obzvláště vektory popisující velké množství rysů tak vyžadují velkou paměť ke svému zpracování. Tato limitace je adresována metodou BRIEF, popsanou stejnojmennou prací od Calondera a spol. (20).

Před porovnáváním je potřeba převést vektor deskriptoru na řetězec binárních čísel. BRIEF umožňuje přímé vytvoření tohoto řetězce z obrázku bez mezikroku v podobě vektoru. V okolí zájmových bodů vybere dvojice pixelů p a q a srovná jejich intenzity I(p) a I(q). Této dvojici označované $n_d(x,y)$ je podle výsledku srovnání intenzit přiřazena hodnota 1 nebo 0; obdobným procesem projdou všechny páry n_d a z jejich výsledků je sestaven řetězec. OpenCV podporuje řetězce o velikostech 128, 256 a 512 bitů (v bytech 16, 32 a 64). Takový deskriptor může být přímo srovnán s deskriptorem posuzovaného snímku pomocí Hammingovy vzdálenosti. Předností BRIEF je další navýšení rychlosti související s menší náročností na paměť.

Oriented FAST and Rotated BRIEF (ORB)

Fungování algoritmu ORB je popsáno v práci od Rubleeho a spol. (21). Z pohledu uživatele OpenCV je největším přínosem ORB její volná dostupnost, narozdíl od patentovaných metod SIFT a SURF.

Jedná se o fúzi metod FAST, jejímž úkolem je identifikace zájmových bodů, a BRIEF pro vytváření deskriptoru. Limitujícím faktorem je neschopnost metody FAST stanovit a metody BRIEF rozlišit natočení rysu, a tak ORB přichází s řešením: V blízkém okolí rohu je určena poloha místa s největší intenzitou a mezi polohou tohoto místa a polohou rohu je vytvořen vektor udávající směr. Úhel natočení θ tohoto vektoru poslouží k vytvoření transformované matice S_θ z matice S, která obsahuje informace o poloze charakteristických rysů.

Autoři ORBu tvrdí, že rychlostí ORB předčí SURF i SIFT a jeho deskriptor podává lepší výsledky než SURF. Objektivně jeho hlavní výhoda tkví v bezplatné dostupnosti a dobré optimalizaci pro vestavěné systémy s nízkými výpočetními výkony.

Feature Matching (10)

Výše popsané metody umožňují popsat charakteristické rysy, ale neumožňují je nalézt na zkoumaných obrázcích. Metody SURF a SIFT využívají Eukleidovské vzdálenosti, zatímco BRIEF a ORB využívají Hammingovy vzdálenosti při srovnávání rysů na obrázku s rysy deskriptoru. Pro účel vyhledávání deskriptory popsaných rysů existují samostatné algoritmy:

Brute-Force Matcher – Tento algoritmus vybere jeden rys deskriptoru a porovná jej se všemi rysy na zkoumaném obrázku. Jednomu rysu na obrázku může příslušet více rysů deskriptoru, potom je zvolen buď ten s největší blízkostí, nebo ty, jejichž podobnost přesahuje uživatelem stanovenou hraniční hodnotu.

FLANN Matcher – FLANN se vyplatí použít pro velké sady dat (velké obrázky). Ihned po spuštění rysy nachází stejnou rychlostí jako Brute-Force Matcher, ale po nelezení několika rysů zredukuje prohledávanou oblast tím, že rysy hledá jen v blízkých okolích již nalezených rysů.

2.3.4 Detekce pomocí konvolučních neuronových sítí

Faster Region-based Convolutional Network (Faster R-CNN)

Informace pro tuto podkapitolu byly brány z práce „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“ od Shaoqinga a spol. (22)

Konvoluční neuronové sítě (CNN) se vyznačují strukturou tvořenou konvolučními vrstvami. Každá tato vrstva má za úkol najít v obrázku různé rysy charakteristické pro danou sadu dat (např. vozidel). CNN musí zpracovávat velké objemy dat, a to i při jejich redukci prostřednictvím tzv. Selective Search, a tak jsou nevhodné pro využití v reálném čase. Všechny dále popsané metody jsou založené na CNN, liší se však jejich přístup k redukci dat a zrychlení detekce.

Problém s rychlostí je do velké míry vyřešen pomocí Faster R-CNN, využívající algoritmu Region Proposal Networks (RPN). RPN najde ve vstupním obrázku oblasti, ve kterých je větší pravděpodobnost

výskytu hledané třídy (třídou je myšlen typ detekovaného objektu). Následně je pořízen druhý obrázek a odpovídající oblasti s větší pravděpodobností jsou prozkoumány pomocí CNN. Vstupní data jsou tedy zredukována a není potřeba tak velký výpočetní výkon. Faster R-CNN umožňuje rychlosti kolem 5 FPS (frames per second).

You Only Look Once (YOLO)

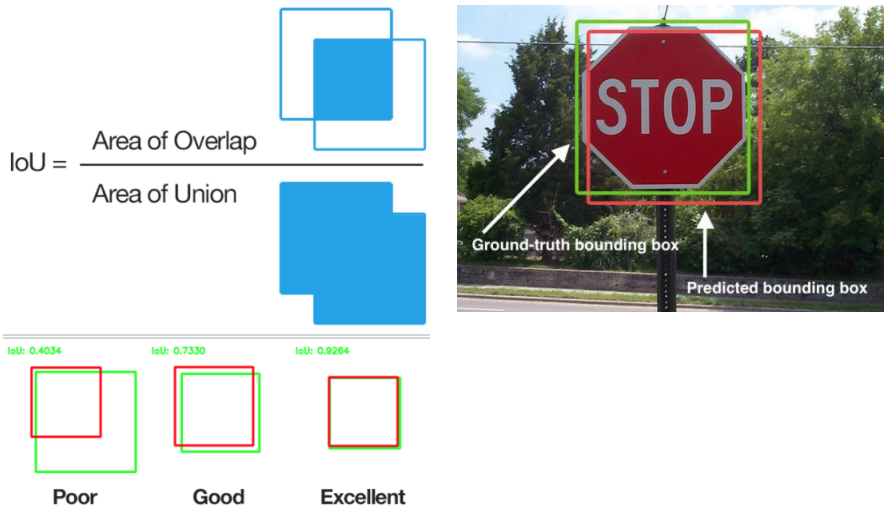
Jak poznamenávají Redmon a spol. ve své práci (23), metody R-CNN si práci zjednodušují tím, že zadané objekty hledají jen v těch místech obrázku, kde je větší pravděpodobnost jejich výskytu. YOLO oproti tomu posuzuje celý obrázek.

Obrázek je rozdělen mřížkou SxS na jednotlivá políčka. Pro každé políčko je pomocí CNN stanovena třída (o který objekt se jedná) a pravděpodobnost, že se skutečně jedná o danou třídu, a ne o objekt na pozadí. Je-li pravděpodobnost vyšší než stanovená hraniční hodnota, je příslušné políčko vybráno a použito k nalezení objektu.

Toto řešení má tři hlavní výhody: 1) YOLO dosahuje běžně rychlostí 45 FPS, jsou-li na vstupu obrázky s menším rozlišením, rychlost může být až 150 FPS. 2) YOLO vidí obrázek v celém kontextu, a tak hrozí menší riziko, že z pozadí vybere nesprávný objekt. 3) YOLO je robustnější vůči neočekávaným vstupům. Nevýhodou YOLO je neschopnost identifikovat malé objekty.

Single Shot MultiBox Detector (SSD)

O SSD se lze dočíst ze stejnojmenné práce od Liua a spol. (24). Metoda SSD funguje podobně jako YOLO, ale liší se v tom, jakým způsobem si rozdělí vstupní obrázek na políčka. Na začátku je vytvořena mapa políček, která nerozlišují mezi pozadím a hledanými objekty. Následnou snahou SSD je pomocí regrese upravit políčka tak, aby ohraničovala hledané objekty. SSD dosahuje rychlostí jako YOLO a přesnosti jako Faster R-CNN. Má ale problém s detekcí menších objektů.



Obrázek 5: Princip fungování metody SSD. (39)

Srovnání algoritmů využívajících konvoluční neuronové sítě

Srovnáním výše zmíněných metod se zabývají Huang a spol. (25). Všechny metody jsou velmi robustní a všestranně použitelné. Nelze prohlásit, která je nejlepší, nebo nejhorší – každá je vhodná pro jinou aplikaci. Porovnávány jsou parametry rychlosti (FPS), přesnosti (mAP, mean avarage precision), využití paměti a potřebného výpočetního výkonu. Použitím různých modelů pro popis objektů je dosahováno různých výsledků, což je také nutno brát v úvahu.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Obrázek 6: Výsledky srovnání vybraných metod pomocí modelu Pascal VOC2007 (24)

Tabulka 1: Srovnání metod detekce objektů využívajících CNN

METODA	SILNÉ STRÁNKY	SLABÉ STRÁNKY
Faster R-CNN	vysoká přesnost	nižší rychlosti (cca 5 FPS) velká náročnost na výpočetní výkon
YOLOv3	nejrychlejší metoda (až 150 FPS)	nižší přesnost
SSD	vysoká rychlost vysoká přesnost	obtížná detekce malých objektů

2.3.5 Detekce pomocí FMS

Poněkud odlišným přístupem k detekci objektů je použití tzv. Fiducial Marker Sets (FMS). Typů FMS existuje řada, společný je jejich princip. Jedná se o obrazce, které jsou díky svému distingovanému tvaru detekovány rychle a bez potřeby velkého výpočetního výkonu. FMS nacházejí využití v aplikacích rozšířené reality nebo lokalizace robotů.

Podoba vzoru se různí pro všechny typy FMS, některé jejich vlastnosti jsou ale vždy stejné: jejich obrys bývá černý kvůli zřetelnému odlišení od okolí, výplň je pak tvořena jednoduchými černobílými vzory. Černá a bílá jsou používány kvůli vysokému kontrastu. Algoritmy FMS využívají adaptivní filtry k odlišení bílých a černých oblastí markeru. Tyto dvě skutečnosti přispívají ke snazší identifikaci markeru i při proměnlivých světelných podmínkách. Marker mívá čtvercový nebo kruhový tvar. Vzor výplně markeru je unikátní pro každý objekt, díky čemuž od sebe lze konkrétní markery odlišit.

ArUco Markers

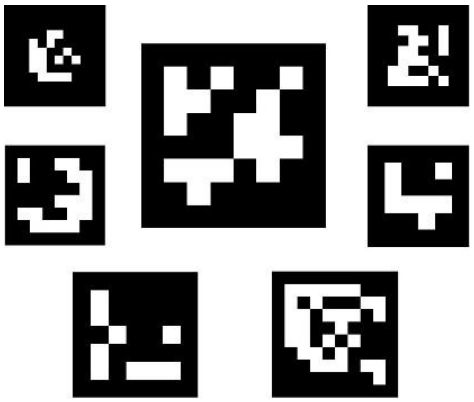
Díky rozsáhlosti knihovny OpenCV (10) lze výhod FMS využít bez nutnosti instalovat další knihovny – součástí OpenCV je sada markerů ArUco. Za jejím vznikem, popsaným v práci „Automatic generation and detection of highly reliable fiducial markers under occlusion“ (26), stojí tým z univerzity v Córdobě.

ArUco markery jsou tvořeny černým okrajem a vnitřkem s podobou binární matice. Jednolitý černý okraj napomáhá k rychlé detekci, zatímco vzor vnitřku je nositelem informace. Velikost markeru je dána počtem bitů. V praxi to znamená, že čím větší marker je použit, tím více existuje unikátních vzorů, které od sebe lze rozlišit. Největší obvykle používané markery jsou 64bitové s velikostí 8x8; při identifikaci markerů je aplikace omezena rozlišením kamery – příliš velké markery (ve smyslu velikosti matice) mohou být nečitelné. K chybám v identifikaci může rovněž vést použití příliš obsáhlé databáze markerů, a to v důsledku vzájemného zaměňování.

Při identifikaci jsou markery na snímku srovnávány s markery uloženými v databázi (slovníku). V průběhu srovnávání jsou zaznamenány všechny markery v zorném poli a je určena pravděpodobnost jejich příslušnosti ke konkrétním markerům v databázi. Marker z databáze s největší pravděpodobností příslušnosti je přiřazen markeru v zorném poli. Jednoznačnému určení napomáhá menší počet markerů v databázi, jejich co největší odlišnost a maximální počet přechodů (odlišné sousední bity). Na tyto požadavky je brán zřetel při vytváření databáze: Zprvu je určena počet a velikost markerů v databázi. Algoritmus si určí hraniční hodnotu τ a začne vytvářet nové markery. První marker je okamžitě vložen do databáze, ale pro ostatní je počítána Hammingova vzdálenost od markerů v databázi. Je-li vzdálenost větší než hraniční vzdálenost τ, je marker vložen do databáze. V momentě, kdy jsou Hammingovy vzdálenosti všech nových generovaných markerů menší než τ, je přijata menší hraniční vzdálenost τ-1 a cyklus se opakuje. Algoritmus provede minimální počet iterací potřebných k naplnění databáze požadovaným počtem markerů.

Jakmile je marker detekován, je určena jeho poloha, vyjádřená natočením os a polohou středu souřadnicového systému. Je-li jednomu z markerů přiřazena role referenčního souřadnicového systému, lze v tomto systému vyjádřit polohy ostatních objektů na pracovní ploše.

Algoritmus byl samotnými jeho tvůrci testován. Tvorba databáze s 10, 100 a 1000 markery velikosti 6x6 zabrala 8, 20 a 90 minut. Při použití databáze s 24 markery 6x6 trvalo 11,08 ms identifikovat všechny markery na jednom snímku o rozlišení 640x480, což odpovídá 90 FPS.



Obrázek 7: Příklady ArUco markerů (10)

2.4 Algoritmy pro sledování objektů

Je-li na video aplikována detekce objektů, jsou tyto objekty vyhledávány v každém snímku (framu) zvlášť. Nalezení objektu na jednom snímku nijak nezjednoduší jeho hledání ve snímku následujícím. Tento problém je adresován trasováním objektů – na rozdíl od detekce jsou trasovací algoritmy schopny zapamatovat si označený objekt a sledovat ho při jeho pohybu na videu. Použití detekce objektů se na videu projevuje fluktuací označeného detekovaného objektu, trasování je v tomto směru o poznání stabilnější, což je žádoucí při stanovování polohy objektu.

Knihovna OpenCV nabízí sedm základních metod, princip i použití všech sedmi je vysvětleno v dokumentaci (10). Dále metody popisují a srovnávají Mi a Yang v rámci své práce zkoumající využití trasovacích algoritmů na 360° video (27).

BOOSTING

BOOSTING je z metod dostupných v OpenCV nejstarší. Její fungování vychází z Adaboost, použitého při detekci obrazu pomocí Cascade Classifier. K sledování objektu jsou použity pozitivy a negativy – jako pozitiv je brán obdélníkový výřez obsahující sledovaný objekt, jako negativy jsou brány obdélníkové výřezy pozadí. Tímto způsobem jsou z jednoho snímku určeny charakteristické rysy sledovaného objektu a na dalším jsou tyto rysy hledány v přibližně stejné oblasti, ve které se objekt nacházel na snímku prvním.

MIL (Multiple Instance Learning)

Podobně jako BOOSTING i MIL pracuje s pozitivy a negativy. Pozitivy a negativy jsou seskupeny do balíčků. Balíček s negativy obsahuje obdélníkové výřezy pozadí stejně jako BOOSTING. Oproti tomu balíček s pozitivy obsahuje de facto pouze jeden pozitiv – výřez s trasovaným objektem – a ostatní jsou odhady generované v blízkém okolí výchozího výřezu. V balíčku s pozitivy je díky tomu alespoň jeden

výřez, který správně odhaduje polohu objektu na dalším snímku. Díky velkému počtu pozitivů program počítá s více variantami a je robustnější, např. vůči částečnému zakrytí objektu.

KCF (Kernelized Correlation Filters)

KCF pracuje s balíčky negativů a vygenerovaných pozitivů. Navíc využívá toho, že se pozitivy navzájem překrývají a oblasti těchto překrytí využívá k nalezení dalších charakteristických rysů. Díky tomu lze objekty v po sobě jdoucích snímcích nalézt s vyšší přesností i rychlostí. Tato metoda je při použití OpenCV doporučována.

TLD (Tracking, learning and detection)

Metoda trasování zvaná TLD z části využívá i metod detekce objektů. Základem je algoritmus trasující objekty při jejich pohybu napříč snímky. Detektor lokalizuje a kontroluje výběr trasovače a případně ho poopraví. Program zároveň sleduje činnost detektoru a jeho chyby a učí se, jak je nedělat v budoucnu. V důsledku oprav řízených detektorem výběr výrazněji kmitá. Zároveň ale detektor umožňuje sledování objektu i při jeho částečném zakrytí.

MEDIANFLOW

Trasovacím algoritmům se může stát, že při rychlé změně pohybu sledovaného objektu začnou sledovat nesprávnou část obrazu. MEDIANFLOW zavádí takzvanou Forward-Backward chybu, jak to popisují Kalal, Mikolaiczyk a Matas (28). Z po sobě jdoucích snímků je vytvořena dopředná trajektorie (Forward). Následně je vzat jeden bod sledovaného objektu z posledního snímku a je trasován jeho pohyb ve zpětném směru od posledního k prvnímu snímku (Backward). Mezi těmito dvěma trajektoriemi existuje odchylka. Metoda MEDIANFLOW se snaží tuto odchylku minimalizovat. Díky tomuto přístupu jsou chybná sledování detekována a je ohlášena ztráta objektu místo sledování nesprávného objektu.

GOTURN (Generic Object Tracking Using Regression Networks)

GOTURN využívá konvolučních neuronových sítí (CNN) používaných při rozpoznání objektů. Vstupem jsou dva sousledné snímky – n a n-1. K dispozici je model, kterým je trasovaný objekt nalezen na obou snímcích. Objekt je v obou snímcích označen rámečkem (bounding box) a rozdíl mezi rámečky je použit k odhadu polohy nového výřezu na snímku n+1. Při trasováním v další fázi jsou na vstupu použity snímky n a n+1, přičemž bounding box na snímku n+1 je zpřesněn pomocí CNN.

MOSSE (Minimum Output Sum of Squared Error)

MOSSE využívá adaptivních stabilních korelačních filtrů k označení trasovaného objektu na snímku. Z videa je vybráno několik úvodních snímků k inicializaci a na nich je trasovaný objekt označen.

Označený objekt je následně oříznut a použit k vytvoření filtru. Filtr je posléze porovnáván s dalšími snímky. V průběhu trasování je filtr schopen adaptovat se na změny vzhledu objektu. Díky tomu je velmi robustní vůči změně světelných podmínek, vzdálenosti nebo natočení objektu. (29)

CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability)

Metoda CSRT je založena na algoritmu DCF (Discriminative Correlation Filter), který rozšiřuje. Algoritmus DCF byl představen v práci (30) od Lukežiče a spol. Zavádí spolehlivost určení polohy a kanálu (ve smyslu digitálního obrazu, např. 3kanálový barevný prostor RGB). CSRT umožňuje měnit velikost výběru, a to právě díky možnosti určit spolehlivost polohy. Spolehlivost kanálu barvy určuje, který z filtrů kanálů má jakou váhu při trasování objektu (vychází z barvy objektu). Díky určení spolehlivosti jednotlivých filtrů lze využít jen relevantní filtry a tím proces zrychlit bez ztráty přesnosti.

Tabulka 2: Srovnání trasovacích metod

METODA	SILNÉ STRÁNKY	SLABÉ STRÁNKY
BOOSTING	slušná přesnost	nespolehlivé ohlašování chyb trasování
MIL	stabilita výběru vysoká přesnost	nespolehlivé ohlašování chyb trasování problém při delším zakrytí objektu
KCF	vysoká rychlost (cca 400 FPS) ohlašování chyb trasování	fixní velikost výběru nevzpamatuje se ze zakrytí objektu
TLD	dobře se vyrovnává se zakrytím a ztrátou objektu	nižší rychlost (cca 15 FPS) časté označování falešných pozitivů
MEDIANFLOW	ohlašování chyb trasování dobře pracuje s pomalým pohybem a malým zakrytím	nezvládá vyšší rychlosti pohybu objektu
GOTURN	dobře se vyrovnává se zakrytím a ztrátou objektu	dokáže trasovat jen objekty z tréninkových dat
MOSSE	velmi vysoká rychlost (2500 FPS)	nižší přesnost fixní velikost výběru
CSRT	robustnost vůči podmínkám vysoká přesnost	problém při delším zakrytí objektu

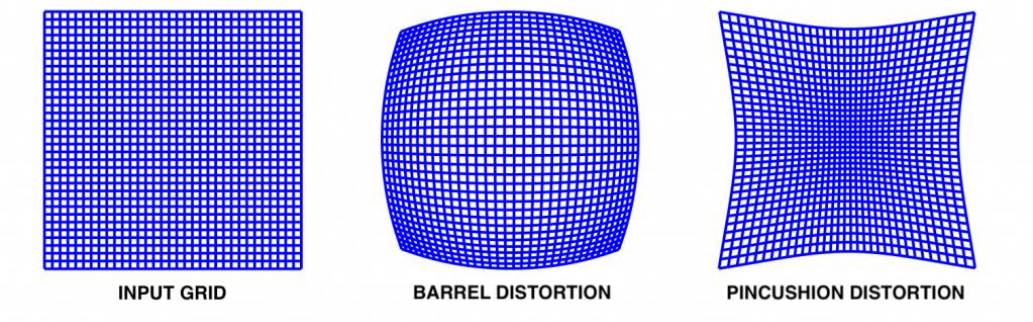
2.5 Algoritmy pro kalibraci obrazu

Algoritmy rozpoznání a trasování objektů obzvláště při aplikaci na videozáznam v reálném čase trpí chybovostí související s proměnlivým a nepřesným zobrazením skutečnosti. Navzdory přání programátora je obraz velmi variabilní – mění se světelné podmínky, nastavení úhlu a přiblížení kamery, navíc konstrukce kamery způsobuje zkreslení. Geometrické deformace obrazu nezpůsobují úbytek informací na obrázku, a je tedy možné obraz kalibrovat a eliminovat zkreslení.

O různých typech optického zkreslení a jejich matematických popisech se lze dočíst z knihy „Fundamentals of Optics“ od Jenkinse a Whitea (31). Způsoby odstranění těchto vad pak umožňuje OpenCV, princip odstraňování vad je popsán dokumentací k této knihovně (10).

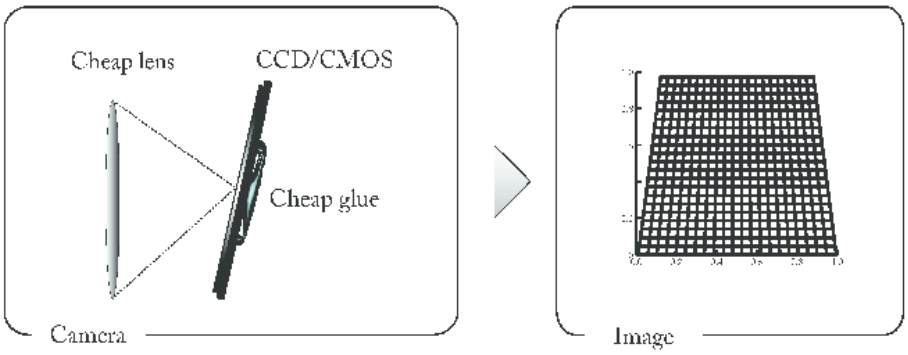
Rozlišuje se zkreslení (distorze) radiální a tangenciální.

Radiální distorze je způsobena tím, že se světlo láme různě v různých částech čočky. Radiální distorze má dva podtypy: 1) Soudkovitá distorze – V tomto případě dochází k tím menšímu zvětšení obrazu, čím větší je vzdálenost od optické osy. Tohoto efektu využívají kamery s efektem rybího oka za účelem zachycení nekonečné roviny na obrázku konečné velikosti. 2) Poduškovitá distorze – Poduškovitá distorze je opakem soudkovité – se zvětšující se vzdáleností od optické osy dochází k výraznějšímu zvětšení obrazu. To vede k zakřivení čar, které neprocházejí středem, směrem ke středu.



Obrázek 8: Radiální distorze. (39)

Tangenciální distorze vychází z perspektivy snímání. Perspektiva je pojem známý hlavně fotografům. V závislosti na vzdálenosti kamery od snímaného objektu se objekt jeví různě velký. Pokud čočka kamery objekt snímá pod úhlem, s měnící se vzdáleností se velikost různých částí objektu mění různě a objekt zdánlivě mění tvar.



Obrázek 9: Tangenciální distorze (40)

Konkrétní postupy k eliminaci zkreslení vysvětlují např. Tang a spol. v práci „A Precision Analysis of Camera Distortion Models“ (32) a Zhang v práci „A Flexible New Technique for Camera Calibration“ (33). Oba typy distorze lze aproximovat pomocí polynomických funkcí. Koeficienty jednotlivých členů se nazývají koeficienty distorze a definují míru a typ zkreslení. Vliv konstrukce kamery lze vyjádřit pomocí vnitřních koeficientů kamery, ty vycházejí z ohniskové vzdálenosti a polohy optického středu.

radiální	$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \dots)$	*(1)*
	$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \dots)$	*(2)*
tangenciální	$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$	*(3)*
	$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$	*(4)*
koeficienty distorze	$k_1, \dots, k_n, p_1, p_2$	
koeficienty kamery	f_x, f_y, c_x, c_y	

Pro potřeby výpočtu je potřeba zaznamenat velké množství bodů a znát jejich souřadnice na snímku i ve skutečnosti. Za tímto účelem se používají šachovnice nebo soustředné kružnice, které kamera snímá společně se snímanými objekty a v reálném čase provádí kalibraci.

Knihovna OpenCV nabízí funkci *cv.findChessBoardCorners()*, která je schopna nalézt rohy šachovnice. Koeficienty distorze lze díky znalosti souřadnic kalibračních bodů získat pomocí funkce *cv.calibrateCamera()*. Koeficienty distorze jsou společně se zkoumaným snímkem vstupními parametry funkcím *cv.undistort()* nebo *cv.remapping()* a tyto vytvářejí nezkreslenou variantu snímku.

2.6 Algoritmy pro určení polohy objektu pomocí 2D obrazu

Určování vzájemné polohy těles a stanovení pohybu z jejich příslušných souřadných systémů je problém z oboru kinematiky.

Robot BCN3D Moveo je ovládán prostřednictvím řídicího systému vytvořeného absolventem ČVUT Patrikem Zachem. Blíže se lze s řízením seznámit v Zachově bakalářské práci (6). V průběhu určování natočení jednotlivých kloubů jsou řešeny dva typy úloh – úloha přímé a úloha inverzní kinematiky. Přímá kinematika – hledání souřadnic koncového bodu při zadaných natočeních kloubů a délkách ramen – je popisována pomocí Denavit-Hartenbergovy transformace. Ta zavádí 4 parametry, popisující souřadnicový systém daného ramene. Souřadnice koncového bodu lze určit vynásobením transformačních matic těchto parametrů pro všechna ramena. Inverzní kinematika má naopak za úkol stanovit jednotlivá natočení ramen, jsou-li souřadnice koncového bodu známy. Použitím inverzní a přímé kinematiky (v tomto pořadí) lze přemístit konec robotické ruky do konkrétního bodu v prostoru.

Úkolem systému strojového vidění je stanovení souřadnic polohy předmětu k uchopení. Při použití jedné kamery lze určit pouze rovinné souřadnice, bude tedy předpokládáno, že z=0, takže souřadnice bodů budou mít tvar [x, y, z=0].

Metody transformace souřadnic objektu z absolutního souřadného systému do relativního lze řešit více metodami, ty budou rozebrány dále.

Analytická metoda

Jednodušší vztahy mezi tělesy lze spočítat analyticky. Tento způsob přichází v úvahu v situacích, kdy lze úlohu převést pouze do roviny. Vychází se většinou z triangulace a pravoúhlých trojúhelníků.

Transformační matice

Jedním ze způsobů určení polohy v několika souřadných systémech jsou transformační matice, jako je to popsáno ve skriptech Mechanika B od Valáška, Šiky a Baumy (34) . Základní pohyby (posuvy v osách, rotace) lze matematicky vyjádřit pomocí transformačních matic. Zachova práce (6) pracuje při výpočtu přímé kinematiky s Denavit-Hartenbergovou úmluvou. Ta pro vyjádření transformace používá 4 parametry – θ_n (rotace kolem osy z), d_n (translace v ose z), a_n (translace v ose x) a α_n (rotaci kolem osy x).

DH matice

$$T_n^{n-1} = R_z(\theta_n)T_z(d_n)T_x(a_n)R_x(\alpha_n) = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} & T_x \\ R_{yx} & R_{yy} & R_{yz} & T_y \\ R_{zx} & R_{zy} & R_{zz} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5)$$

Tato matice slouží k transformaci mezi souřadnými systémy dvou sousedních ramen. Jednotlivé členy matice označují úhly pootočení (označené R) a posuvy (označené T). Celkovou transformační matici lze získat jako součin dílčích transformací mezi sousedními rameny.

3 Strojové vidění pro BCN3D Moveo

3.1 Koncepční návrh

Na základě poznatků získaných v rešeršní části byl sestaven koncepční návrh. Nadřazenými problémem je volba konstrukce. Druhým dílem návrhu je software. Ten je tvořen několika částmi a výchozím předpokladem je, že jednotlivé části programu – kalibrace, detekce, určení souřadnic a grafické prostředí – jsou na sobě nezávislé, a lze tedy každou navrhnout a vytvořit víceméně samostatně.

Při volbě konstrukce existují dvě varianty – varianta s kamerou pevně umístěnou tak, aby snímala robota a jeho okolí, a varianta s kamerou upevněnou ke koncovému ramenu robota. Volba padla na druhou zmíněnou možnost, protože tím odpadá nutnost určovat polohu koncového snímače vzhledem ke snímaným objektům. Pracovní oblastí bude rovina v okolí robota.

Kalibrace robota bude provedena pomocí referenčních markerů ArUco. Jejich předností je snadná a rychlá identifikovatelnost na obraze. Samotní tvůrci ArUco uvádějí kalibraci robotů a určení jejich umístění v prostoru jako jedno z možných využití. S ohledem na složitost určení natočení jednotlivých ramen robota s pěti stupni volnosti bylo přistoupeno ke zjednodušení. Úkolem uživatele bude manuálně nastavit ramena robota tak, aby byl vzpřímený. Robot nedisponuje jinou zpětnou vazbou, a tak bude uživatel muset vzpřímenost odhadnout. Vychází se z předpokladu, že tímto způsobem zkalibrovaný robot bude oproti ideálnímu stavu vykazovat dostatečně malé odchylky při najíždění do zadaných poloh.

Detekce objektů v okolí robota bude provedena pomocí funkce *cv2.findContours()*, jež je součástí knihovny OpenCV. Tato funkce nalezne uzavřené obrazce na jednokanálovém snímku. Tento snímek lze získat pomocí filtrů, které jsou rovněž součástí OpenCV. Tento způsob detekce umožňuje nalezení objektů na kontrastním (např. bílém) pozadí. Na rozdíl od detekce pomocí neuronových sítí nebo charakteristických rysů nelze tímto způsobem nalézt konkrétní předměty (např. najdi nůžky). Výhodou je snadná implementace a nízká výpočetní náročnost.

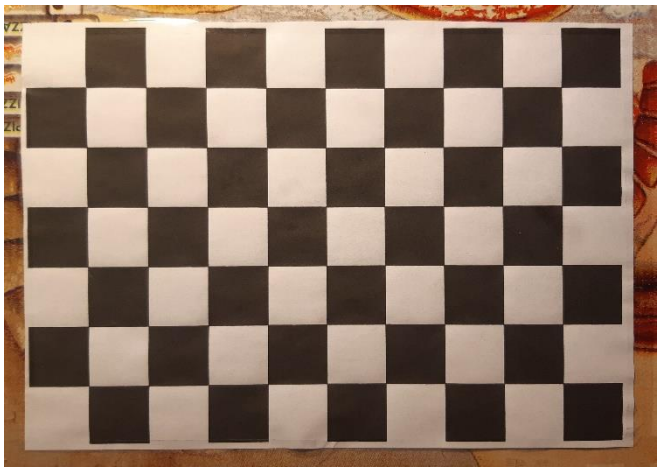
Souřadnice objektů na pracovní ploše budou určovány triangulací. Díky kalibraci lze určit parametry transformační matice mezi pevným souřadným systémem a souřadným systémem koncového efektoru. Z těchto parametrů je možné určit rozměry trojúhelníka tvořeného vrcholy objekt-koncový efektor-průmět koncového efektoru do pracovní plochy.

Grafické prostředí kolegy Zacha bylo vytvořeno programem QtDesigner, využívajícím knihovny PyQt5. Aby bylo možné mít přístup k funkcionalitám Zachova GUI, bude toto GUI pouze rozšířeno o systém strojového vidění a nebude vytvářeno nové.

3.2 Kalibrace kamery

Jak je zmíněno v sekci 2.5, každá kamera se vyznačuje specifickými optickými parametry. Tyto parametry, označované jako koeficienty distorze a koeficienty kamery, tvoří stejnojmenné matice. Správné určení těchto matic je kritické jak kvůli eliminaci zkreslení obrazu, tak kvůli správnému chodu některých funkcí, např těch z knihovny ArUco.

Kalibrace byla provedena pomocí samostatného skriptu *camera_calibration.py*. Kód ke kalibraci byl převzat z dokumentace OpenCV a byl doplněn o několik řádků tak, aby byl zaznamenán jen každý n-tý snímek a kalibrační matice byly po ukončení sběru dat uloženy do samostatných souborů ve složce *calibration*. Z nabízených kalibračních obrazců byla zvolena šachovnice o rozměrech 7x6 a délce strany jednoho čtverečku 26mm (taktéž součástí dokumentace, viz Obrázek 10), příslušnou funkcí v OpenCV definující podobu šachovnice je *cv2.drawChessboardCorners()* a kalibrační matice jsou po nasbírání dat spočteny funkcí *cv.calibrateCamera()*.



Obrázek 10: Šachovnice použitá při kalibraci kamery

3.3 Kalibrace robota

3.3.1 Výběr metody

Účelem kalibrace bude určení polohy, kam má být umístěn objekt po uchopení. Aby byla kalibrace urychlena, probíhá vůči zřetelně odlišenému obrazci – ArUco markeru. Detekce markerů je rychlá, přesná a poměrně robustní vůči světelným podmínkám. Lze použít více markerů k určení většího množství poloh.

3.3.2 Implementace ArUco

Marker je umístěn do okolí robota a po jeho nalezení je spočítána jeho poloha. Knihovna, jež umožňuje detekci ArUco, nabízí i řadu funkcí pomocí nichž lze přímo určit polohu koncového ramene robota s kamerou, což lze využít ke kontrole odchylky koncového efektoru. Veškeré funkcionality

kalibrace jsou součástí třídy (class) *calibrator*. ArUco disponuje mnoha funkcemi pracujícími s detekovanými markery. Tyto funkce se staly základem vlastních metod třídy *calibrator*. Ta je uložena v samostatném souboru *aruco.py*. Použitím tříd a metod plnících jednu konkrétní úlohu je zpřehledněna hlavní kostra programu v souboru *main.py*.

Metoda *__init__()* (viz Zdrojový kód 1) slouží k vytvoření instance třídy *calibrator*. Kromě jiných věcí, které umožňuje, jsou v ní uloženy neměnné parametry, které není potřeba načítat opakovaně, a proměnné, k nimž je potřeba mít přístup v rámci celé třídy. Na začátku jsou určeny parametry typu a velikosti hledaných markerů. V prostřední části jsou načteny kalibrační matice. V případě použití jiného rozlišení záznamu než 640x360 je potřeba přepočítat matici *self.camMtx* na nový rozměr. V koncové části jsou použité proměnné. V seznamech *self.markers* a *self.ids* jsou uloženy pixelové souřadnice nalezených markerů a jejich příslušná id. Seznam *self.pose_single* obsahuje rotační a translační vektor zvoleného markeru. Proměnná *self.angle_d* udává odchylku koncového ramene od osy robota.

Zdrojový kód 1: aruco.py, řádky 7-26

```
def __init__(self):
    self.dictionary = aruco.Dictionary_get(aruco.DICT_4X4_50)
    self.parameters = aruco.DetectorParameters_create()
    self.marker_size = 150          # for single marker detection

    # # camera calibration matrices (camera calibrated at 640x360)
    # self.camMtx0 = np.load("calibration/mtx.npy")
    # self.width = 640              # cap.get(cv.CAP_PROP_FRAME_WIDTH)
    # ratio = 640 / self.width
    # self.camMtx = np.true_divide(self.camMtx0, ratio)

    self.camMtx = np.load("calibration/mtx.npy")
    self.distCoeff = np.ndarray([0])      # np.load("calibration/dist.npy")
    self.camRvec = np.load("calibration/rvec.npy")
    self.camTvec = np.load("calibration/tvec.npy")

    self.markers = []
    self.ids = []
    self.pose_single = []
    self.angle_d = float
```

Metoda *detect_markers()* (viz Zdrojový kód 2) má za úkol nalezení zvoleného typu markerů. Výstupem funkce jsou seznamy s těmito markery a jejich id. Kvůli zrychlení procesu byla vytvořena i metoda *reference_marker()*. Tato funkce vybere ze seznamu nalezených markerů pouze takový, odpovídající zvolenému id (v případě, že by snímek zachytil více markerů). To přinese zjednodušení následujícím dvěma metodám *find_pose_single()* a *camera_angle()*, které tak nemusí svůj výpočet provádět pro všechny nalezené markery, ale jen pro jeden.

Zdrojový kód 2: aruco.py, řádky 28-52

```
def detect_markers(self, frame):
    markers, ids, _ = aruco.detectMarkers(frame, self.dictionary,
        parameters=self.parameters)
    detected = len(markers)

    if ids is None:
        ids = []

    for i in range(detected):
        x = int(markers[i][0][0][0])
        y = int(markers[i][0][0][1])
        cv.putText(frame, str(ids[i]), (x, y), cv.FONT_HERSHEY_PLAIN, 1,
                (0, 0, 255), 2)

    self.markers = markers
    self.ids = ids

def reference_marker(self, reference_id):
    ids = self.ids
    markers = self.markers
    success = False
    reference_marker = []
    for id in ids:
        if id[0] == reference_id:
            reference_marker = markers[reference_id]
            success = True
    return success, reference_marker
```

Metoda *find_pose_single()* (viz Zdrojový kód 3) si bere polohy rohů nalezeného markeru a pomocí funkce *aruco.estimatePoseSingleMarkers()* spočítá translační vektor a rotační vektor, obsahující Eulerovy úhly pro transformaci mezi souřadným systémem markeru a kamery. Metoda *camera_angle()* si bere rotační vektor a řešením Rodriguesovy rotační rovnice (35) určí rotační matici. Ta je pak vynásobena jednotkovým vektorem a úhel mezi kamerou a osou robota je určen jako odchylka mezi původním a transformovaným vektorem.

Zdrojový kód 3: aruco.py, řádky 54-72

```
def find_pose_single(self, frame, markers):
    rvecs, tvecs, _ = aruco.estimatePoseSingleMarkers(markers,
        self.marker_size, self.camMtx, self.distCoeff)
    for rvec, tvec in zip(rvecs, tvecs):
        aruco.drawAxis(frame, self.camMtx, self.distCoeff, rvec, tvec, 60)
    self.pose_single = [rvecs, tvecs]

def camera_angle(self):
    rvecs, _ = self.pose_single
    rvecs = np.array(rvecs[0][0])

    rot, _ = cv.Rodrigues(rvecs)      # solve rodrigues
    k = np.array([0, 0, 1])           # unit vector in z
    k_rot = rot.dot(k)

    dot_product = np.dot(k, k_rot)
    angle_r = math.pi - np.arccos(dot_product)
    angle_d = np.degrees(angle_r)
    self.angle_d = angle_d
```

3.4 Detekce objektů

3.4.1 Výběr metody

Ačkoli se kapitola 2.3 algoritmům pro rozpoznání obrazu a jejich výhodám a nevýhodám věnuje velmi obšírně, bylo na jejím základě obtížné rozhodnout, která z metod je nejvhodnější. Vyzkoušet každou ze zmíněných metod by bylo časově obtížné, a tak byl vyzkoušen jeden zástupce z každé ze čtyř skupin detekčních algoritmů. Na společném základě vlastního ozkoušení a poznatků z odborných zdrojů citovaných v rešerši byla sestavena tabulka výhod a nevýhod algoritmů.

Tabulka 3: Výběr detekčního algoritmu

ALGORITMUS (pokusná metoda)	VÝHODY	NEVÝHODY
kontury (cv2.detectContours())	rychlost jednoduchost (lze napsat na několika řádcích) robustnost možnost použití různých filtrů	neschopnost identifikovat typ objektu (např. míček) značná citlivost na světelné podmínky
subtrakce pozadí (MOG)	Pozadím je rozuměna ta část obrazu, která zůstává nehybná. Pro zvolenou aplikaci nevhodná metoda! Uvedena pro úplnost.	
charakteristické rysy (Cascade Classifier)	možnost detekce typu objektu snadná tvorba modelu snadná implementace v OpenCV	riziko vytvoření zavádějícího modelu na základě zkreslených dat častá chybná detekce (chybějící detekce, false positive) potřeba vysokého rozlišení
konvoluční neuronové sítě (YOLO)	vysoká kvalita detekce možnost detekce typu objektu dostupnost rozsáhlých modelů	rychlost výpočetní náročnost (zahřívání CPU) obtížná tvorba vlastních modelů

Koncepční návrh nepočítá s detekcí konkrétních typů objektů, a tak jsou přednosti detekce pomocí charakteristických rysů a konvolučních neuronových sítí nadbytečné. Objekty proto budou detekovány přes své kontury pomocí funkce *cv2.detectContours()*. Při vhodném nastavení parametrů použitých filtrů a zachování nepříliš proměnlivých okolních světelných podmínek je tento způsob velmi efektivní a rychlý.

3.4.2 Implementace

Detekce objektů je realizována metodami uloženými do třídy *detector* a samostatného skriptu *object_detection.py*.

Instance třídy *detector* v sobě nese parametry použitého filtru, rozsah velikosti detekované kontury v pixelech, body definující nepravidelný n-úhelník kolem detekovaných kontur a parametry příslušných elips opsaných těmto n-úhelníkům. Třída *detector* obsahuje defacto jen dvě metody – *detect_contours()* a *method_n()*.

Metoda *detect_contours()* (viz Zdrojový kód 4) využívá několika funkcí z knihovny OpenCV. Vstupem této funkce je snímek z kamery, výstupem pak dva seznamy – vrcholy detekovaných n-úhelníků (*approxes*) a parametry opsaných elips (*ellipses*). Snímek je nejprve pomocí další metody třídy *detector* zpracován do podoby masky (binární snímek) tak, aby bíle zůstaly označeny pokud možno pouze předměty, které chceme detekovat, a černě zbytek, nebo naopak. Detekce kontur probíhá na každém snímku zvlášť. Z toho důvodu jsou při každé iteraci vytvořeny prázdné seznamy *approxes* a *ellipses*. Kontury jsou na masce nalezeny pomocí funkce *cv2.findContours()*. Nalezené kontury jsou uloženy do seznamu *contours*. Skrz tento seznam je následně iterováno a jsou vybrány pouze kontury jejichž velikost v pixelech je v zadaném intervalu – tímto způsobem jsou eliminovány nadbytečné detekce. Schválené kontury jsou kvůli úspoře paměti aproximovány n-úhelníky a v tomto tvaru ukládány do zmíněného seznamu *approxes*. Parametry odpovídajících opsaných elips jsou uloženy v seznamu *ellipses*.

Zdrojový kód 4: object_detection.py, řádky 11-28

```
def detect_contours(self, frame):
    processed = self.method_1(frame, self.parameters)
    area_min, area_max = self.area
    approxes = []
    ellipses = []
    contours, hierarchy = cv.findContours(processed, cv.RETR_TREE,
        cv.CHAIN_APPROX_SIMPLE)
    for cnt in contours:
        area = cv.contourArea(cnt)
        if (area > area_min) and (area < area_max):
            peri = cv.arcLength(cnt, True)
            approx = cv.approxPolyDP(cnt, 0.02*peri, True)
            (x, y), (MA, ma), angle = cv.fitEllipse(cnt)

            approxes.append([approx])
            ellipses.append([(x, y), (MA, ma), angle])
        else:
            pass
    self.approxes = approxes
    self.ellipses = ellipses
```

Hlavním faktorem ovlivňujícím kvalitu detekce je způsob filtrování nadbytečných informací ze snímku – tvorba masky. Snímek z kamery zaznamenává nadbytečné množství informací. Řada informací o snímku, které lidský mozek bere jako jednoznačnou reprezentaci hledaného objektu, představuje pro detektor šum bránící rozpoznání. Metod tvorby masky s různými parametry byla vyzkoušena řada a úspěšnější z pokusů byly zachovány jako metody v rámci třídy *detector*. GUI umožňuje využití dvou metod tvorby filtru – filtr bílé barvy a prahový filtr.

Prahový filtr (viz Zdrojový kód 5) – Maska je vytvořena dvojnásobnou aplikací prahové funkce *cv2.threshold()*. Tato funkce srovná všechny pixely s předem zvolenou hraniční hodnotou a v závislosti na tom, jestli je hodnota pixelu nad, nebo pod touto hraniční hodnotou, je pixelu přiřazena jedna ze dvou krajních hodnot. Metoda je nastavena tak, aby bylo možné parametry měnit v průběhu chodu programu. Poprvé je prahová funkce aplikována na barevný snímek, čímž jsou odstraněny odstíny jednotlivých barev (v barevném prostoru RGB). Následně je snímek převeden do odstínů šedé a je na něj aplikován Gaussian Blur, čímž je odstraněn šum. Nakonec je podruhé použita prahová funkce a jejím výstupem je v tomto případě už výsledná maska, tedy pouze dvoubarevný (černý a bílý) snímek.

Zdrojový kód 5: object_detection.py, řádky 32-38

```
def method_1(self, frame, t):
    _, threshold_color = cv.threshold(frame, t[0], t[1], cv.THRESH_BINARY)
    gray = cv.cvtColor(threshold_color, cv.COLOR_BGR2GRAY)
    blur = cv.GaussianBlur(gray, (5, 5), 5)
    _, threshold_bw = cv.threshold(blur, t[2], t[3], cv.THRESH_BINARY)
    return threshold_bw
```

Filtr bílé barvy (viz Zdrojový kód 6) – Snímek je z převeden z barevného prostoru RGB do HSL. Maska je vytvářena na základě dvou parametrů – horního a dolního limitu hodnoty světlosti (L, lightness). Funkce OpenCV *cv2.inRange()* vytvoří masku, kde jsou označeny pixely, jejichž hodnota světlosti se vejde do zvoleného intervalu.

Zdrojový kód 6: object_detection.py, řádky 59-74

```
def method_4(self, frame):
    t = self.parameters
    blur = cv.GaussianBlur(frame, (5, 5), 5)
    hls = cv.cvtColor(blur, cv.COLOR_BGR2HLS)

    h_min = 0
    h_max = 179
    l_min = t[0]
    l_max = t[1]
    s_min = 0
    s_max = 255

    lower = np.array([h_min, l_min, s_min])
    upper = np.array([h_max, l_max, s_max])
    mask = cv.inRange(hls, lower, upper)
    return mask
```

3.5 Trasování objektu

3.5.1 Výběr metody

Metod trasování je knihovnou OpenCV nabídnuta řada a detailněji jsou probrány v kapitole 2.4. Všechny tyto metody pracují s charakteristickými rysy, stanovenými na jednom snímku a vyhledávanými na následujícím snímku. Ačkoli řada z těchto metod dosahuje vysoké rychlosti a přesnosti, při použití společně se zvoleným detektorem se objevily neočekávané problémy. Jelikož pokaždé, když je detekován nový objekt, musí být tyto trackery znovu inicializovány, je tím prvním problémem pokles rychlosti. I při sebelepší detekci dochází příležitostně k detekci falešných pozitivů. Tracker není schopen rozlišit, jeli detekce správná, či nikoli, a tak dochází k opakované inicializaci a hromadí se trasované RoI. Vysoký počet trasovaných RoI má na rychlost trasování také negativní vliv, navíc obrazovka plná čtverečků označujících neexistující objekty nevytváří dojem přesnosti a spolehlivosti.

Z výše zmíněných důvodů bylo přistoupeno k vytvoření vlastního trackeru na principu Eukleidovských vzdáleností. Ve smyslu trackerů dostupných v OpenCV se nejedná o plnohodnotný tracker – při vyhledávání objektu na nových snímcích není využíváno podoby trasovaného objektu z předchozích snímků. Fungování Eukleidovského trackeru je navázáno na detektor – trasování vychází z toho, že vzdálenost geometrických středů odpovídajících si detekovaných kontur na po sobě jdoucích snímcích nepřekročí stanovenou hodnotu. Z toho důvodu není tento tracker schopen odolávat zakrytí objektu nebo jeho krátkodobému zmizení ze záběru. Vlastnosti Eukleidovského trackeru jsou pro tuto

aplikaci dostačující – tracker je rychlý, výpočetně nenáročný a snadno implementovatelný. Při implementaci byl využit návod z portálu pyimagesearch. (35)

3.5.2 Implementace

Eukleidovský tracker byl implementován v rámci třídy *tracker* (viz Zdrojový kód 7), uložené ve stejnojmenném skriptu *tracker.py*.

Každá instance třídy *tracker* má v paměti uloženy dvě proměnné – slovník (ve smyslu programovacích jazyků) *ellipses_ids* a integer *ids*. Ve slovníku *ellipses_ids* jsou evidovány detekované elipsy (resp. jejich parametry) podle jim přiřazených id. Celočíselná proměnná *ids* potom představuje id, které bude přiřazeno nově detekované elipse.

Fungování trackeru zajišťuje defacto jediná metoda a to metoda *update()*. Vstupem této metody je seznam se všemi nově detekovanými elipsami (rozuměj objekty). Metoda *update()* je tedy (stejně jako metoda *detector.detect_contours()*) volána při načtení každého snímku. Na začátku je vytvořen prázdný slovník *new_ellipses_ids* do něhož jsou ukládány elipsy se svými id (ať už nové, nebo již detekované). Následně jsou iteračně načítány středy nově detekovaných elips a jsou spočteny Eukleidovské vzdálenosti každého takového středu od středů elips detekovaných na předchozím snímku. Jeli některá z těchto vzdáleností menší než zvolená mezní vzdálenost (zde bylo zvoleno 25px), je tato elipsa považována za již nalezenou a ve slovníku *ellipses_ids* dojde k aktualizaci jejích parametrů beze změny id. Pakliže není v blízkosti nalezena žádná elipsa, je detekce považována za novou a je jí přiřazeno nové id s hodnotou proměnné *ids*. Proměnná *ids* musí být následně zvýšena o 1. Výstupem metody *update()* je tedy slovník *ellipses_ids* s všemi detekovanými elipsami a jejich id, podle nichž můžou být voleny konkrétní elipsy.

Součástí třídy *tracker* je i metoda *draw_ellipse()*. Tato metoda je použita k vynesení detekovaných elips na snímek. Společně s křivkou elipsy je v její blízkosti také její id a úhel, který svírá hlavní osa elipsy se svislicí.

Zdrojový kód 7: tracker.py, řádky 4-40

```
class tracker:
    def __init__(self):
        self.ellipses_ids = {}
        self.ids = 0

    def update(self, detections):
        new_ellipses_ids = {}
        for ellipse in detections:
            (x, y), (_, _), _ = ellipse
            c = np.array([x, y])

            already_detected = False
            for id, ellipse0 in self.ellipses_ids.items():
                (x0, y0), (_, _), _ = ellipse0
                c0 = np.array([x0, y0])
                distance = np.linalg.norm(c-c0)

                if distance <= 25:
                    already_detected = True
                    break

            if already_detected is True:
                new_ellipses_ids[id] = ellipse

            elif already_detected is False:
                new_ellipses_ids[self.ids] = ellipse
                self.ids += 1

        self.ellipses_ids = new_ellipses_ids.copy()

    def draw_ellipse(self, frame):
        ellipses = self.ellipses_ids
        for key in ellipses:
            (x, y), (MA, ma), angle = ellipses[key]
            cv.ellipse(frame, (int(x), int(y)), (int(MA) // 2, int(ma) //
               2), angle, 0, 360, (0, 0, 0), thickness=2)
            cv.putText(frame, str(key), (int(x), int(y - 20)),
               cv.FONT_HERSHEY_COMPLEX, 0.7, (0, 0, 0), 2)
            cv.putText(frame, str(int(angle)), (int(x), int(y + 20)),
               cv.FONT_HERSHEY_COMPLEX, 0.7, (0, 0, 0), 2)
```

3.6 Souřadnice objektu

Použití jedné kamery vylučuje určení hloubky obrazu. Při stanovování polohy objektu je potřeba provést určité zjednodušení tak, aby některá ze souřadnic polohy objektu byla známa. V tomto případě touto souřadnicí bude z=0. Hledané objekty budou smět být položeny pouze v jedné rovině, na níž sedí i samotný robot. Díky tomuto přístupu je potřeba dopočítat už jen 2 zbývající souřadnice objekty x a y. K tomu je využito triangulace.

Prvním krokem pro provedení triangulace je vytvoření pomyslného trojúhelníka s některými známými parametry. Toho je docíleno zarovnáním koncového efektoru robota (resp. kamery) s daným objektem. Úkolem metody s všeříkajícím názvem *align_frame_with_object()* (viz Zdrojový kód 8) je určit, kterým směrem se robot musí přesunout tak, aby střed snímku korespondoval se středem objektu. Při

takovémto uspořádání prochází optická osa kamery, jejíž odchylka od osy robota je známá, hledaným objektem. Tím je získán kýžený trojúhelník, tvořený vrcholy objekt-koncový efektor robota-průmět koncového efektoru do roviny z=0.

Zdrojový kód 8: main.py, řádky 776-788

```
def align_frame_with_object(self, object_center):
    frame = self.frame
    frame_size = np.array([len(frame[0]), len(frame)])
    frame_center = np.divide(frame_size, 2)
    fc_x = frame_center[0]
    fc_y = frame_center[1]
    cv.circle(frame, (int(fc_x), int(fc_y)), 2, (0, 0, 255), 4)

    o_x = object_center[0]
    o_y = object_center[1]
    cv.circle(frame, (int(o_x), int(o_y)), 2, (0, 255, 0), 4)

    self.direction_vector = np.subtract(object_center, frame_center)
```

V momentě, kdy je kamera zarovnána k objektu, je zavolána metoda *frame_center_to_world_coordinates()* (viz Zdrojový kód 9). Na začátku je aplikována metoda *forwardkinematics()* kvůli určení aktuálních souřadnic koncového efektoru. V dalším kroku se spočítají souřadnice x a y objektu ležícího v rovině z=0 uprostřed snímku. Pomocí souřadnice z koncového efektoru a jeho odchylky od osy robota je spočítána vzdálenost objektu od osy robota. S využitím údaje o úhlu pootočení robota kolem své osy od nulové polohy (ve skriptu *main.py base_spinbox_2.value()*) je spočtena poloha objektu v kartézských souřadnicích (jedná se defacto o převod z polárních do kartézských souřadnic). Metoda byla vytvořena tak, aby jí bylo možné určit souřadnice jak objektu, tak ArUco markeru. Poloha ArUco Markeru bude určovat pozici, kam má být objekt umístěn. Podle zvolené aplikace (ArUco nebo objekt) jsou souřadnice uloženy do seznamu *target* nebo *object* a vypsány v GUI

Zdrojový kód 9: main.py, řádky 823-857

```
def frame_center_to_world_coordinates(self, calibration = True):
    self.forwardkinematics()
    tx, ty, tz = self.T
    distance_from_robot = tz * math.tan(math.radians(self.view_angle))
    base_angle = self.base_spinbox_2.value()

    x = +round(tx + distance_from_robot *
        math.cos(math.radians(base_angle)))
    y = -round(ty + distance_from_robot *
        math.sin(math.radians(base_angle)))
    z = 0
    if calibration == True:
        self.target = [x, y, z]
        self.cv_x_target_spinbox.setValue(x)
        self.cv_y_target_spinbox.setValue(y)
        self.cv_z_target_spinbox.setValue(z)
    elif calibration == False:
        self.object = [x, y, z]
        self.cv_x_obj_label.setText(str(x))
        self.cv_x_obj_label.setFont(QtGui.QFont("Ms Shell Dlg 2", 10,
               weight=QtGui.QFont.Bold))
        self.cv_x_obj_label.setStyleSheet("QLabel { color : black }")
        self.cv_y_obj_label.setText(str(y))
        self.cv_y_obj_label.setFont(QtGui.QFont("Ms Shell Dlg 2", 10,
               weight=QtGui.QFont.Bold))
        self.cv_y_obj_label.setStyleSheet("QLabel { color : black }")
        self.cv_z_obj_label.setText(str(z))
        self.cv_z_obj_label.setFont(QtGui.QFont("Ms Shell Dlg 2", 10,
               weight=QtGui.QFont.Bold))
        self.cv_z_obj_label.setStyleSheet("QLabel { color : black }")
```

3.7 Řízení pohybů

Systém kolegy Zacha poskytuje příslušné funkce zajišťující pohyb jednotlivých ramen prostřednictvím firmwaru Marlin a G-kódu. Doplněk se strojovým viděním počítá vstupy pro tyto funkce, aby byl robot umístěn do požadované polohy.

Jelikož jsou pohyby ramen iniciovány okamžitě při změně nastavení úhlu (viz 3.8), řídí program robota zásahy do polí s těmito úhly. Výhodami tohoto přístupu je vzájemný soulad mezi manuálně a automaticky nastavenými úhly a možnost nastavení krajních poloh ramen prostým stanovením maximální, resp. minimální hodnoty vepsatelné do pole.

Směrový vektor získaný metodou *align_frame_with_object()* (viz 3.6) je zpracován metodami *move_accordingly()* a *move_how()* k určení, kterým směrem a jak rychle se má robot přesunout. Metoda *move_accordingly()* využívá metod *move_horizontally()* a *move_vertically()*, jejichž zásahy jsou již přímo měněny hodnoty úhlů ve spinboxech příslušných ramen a tím je pohyb iniciován. Všechny výše zmíněné metody jsou k nahlédnutí níže (Zdrojový kód 10).

Zdrojový kód 10: main.py, řádky 747-763, 790-799, 809-817

```
def speed(self, part, speed):
    spinbox = part + "_spinbox"
    result = getattr(self, spinbox)
    result.setValue(speed)

def step(self, part,  step):
    spinbox = part + "_spinbox_2"
    result = getattr(self, spinbox)
    old = result.value()
    new = old + step
    result.setValue(new)

def move_horizontally(self, step, speed):
    self.setG(type="01")
    part = "base"
    self.speed(part=part, speed=speed)
    self.step(part=part, step=step)

def move_how(self, direction):
    dv_x, dv_y = self.direction_vector
    quick = 50
    if direction == "horizontal":
        if dv_x > quick or dv_x < -quick:
            speed = 360
            step = 0.5
        else:
            speed = 72
            step = 0.1

def move_accordingly(self):
    if self.moveswitch is True:
        lim = self.limit_value
        dv_x, dv_y = self.direction_vector

        if dv_x > lim or dv_x < -lim:
            step, speed = self.move_how("horizontal")
            step = -step if dv_x > 0 else step
            self.move_horizontally(step, speed)
```

3.8 GUI

K tvorbě grafického prostředí bylo využito knihovny PyQt5 a editoru QtDesigner. GUI k systému strojového vidění bylo vytvořeno ve stejném duchu jako GUI kolegy Zacha. Aby bylo možné mít při spuštění programu přístup jak k manuálnímu ovládání, tak k systému strojového vidění, byl do existujícího GUI přidán widget umožňující rozdělení okna na záložky. Do jedné záložky byla pak vložena Zachova část programu s některými úpravami (záložka KINEMATICS) a do druhé grafické prostředí k systému strojového vidění (záložka COMPUTER VISION).

S ohledem na změnu způsobu ovládání pohybů, která byla potřebná, má-li být robot řízen automaticky, bylo původní GUI (záložka KINEMATICS) upraveno. Odstraněna byla tlačítka OK, jimiž se spouštěly pohyby ramen do nastavených poloh. Nově se každé rameno začne přesouvat okamžitě, když dojde ke změně hodnoty v políčku nastavení úhlu. Přidáno bylo okno PRESET POSITIONS s dvěma

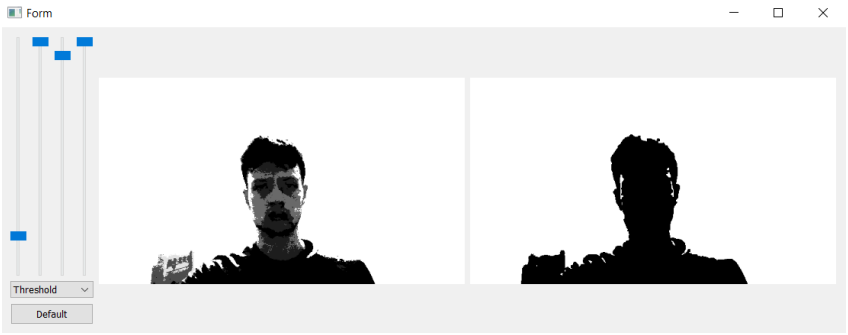
přednastavenými pozicemi robota (HOME a LOOKOUT) a jednou pozicí (TARGET), která se mění podle souřadnic určených kalibrací markery ArUco. Za tlačítky HOME a LOOKOUT se skrývají přesná natočení ramen. Tlačítkem TARGET je volána metoda pro výpočet inverzní kinematiky, která natočení spočte z kartézských souřadnic, v nichž je určena poloha markeru. Aby program nesahal mimo obrazovku a mezi okny nevznikla mezera, bylo umístění některých oken upraveno.

Záložka COMPUTER VISION (viz Obrázek 12) nese podobné designové rysy jako záložka KINEMATICS. Je použit stejný font, barva je místo modré červená. Tato záložka je rozdělena do oken:

CAMERA – V tomto okně má uživatel možnost vybrat port, ze kterého bude načítána kamera. Příslušná roletka obsahuje id od 0 do 5, připojení tolika kamer je ale nepravděpodobné. O (ne)načtení kamery ze zvoleného portu informuje nápis (NOT) LOADED. Jeli kamera nalezena, lze tlačítkem ON/OFF ve spodní části okna spustit nebo vypnout snímání.

CALIBRATION – Okno CALIBRATION obsahuje ovládací prvky, jimiž je řízena kalibrace kamery pomocí ArUco (viz kapitola 3.3). Před spuštěním kalibrace je zvoleno id markeru, podle kterého se robot má zkalibrovat. Roletka ArUco ID nabízí id od 0 do 6. Další id je potřeba přidat zásahem do skriptu *mainwindow.py*. Po zvolení id uživatel vybere, mají-li markery v okolí robota být vyhledávány automaticky, nebo poloautomaticky. V prvním případě robot systematicky prohledá oblast kolem sebe. V druhém případě markery do záběru kamery dostane manuálním ovládáním robota uživatel. V obou případech se s markerem po jeho zachycení zarovná automaticky. Kalibrace je spuštěna po stisknutí tlačítka Initialize. Kalibrace běží, dokud marker není nalezen. Detekce není možná, dokud není provedena kalibrace.

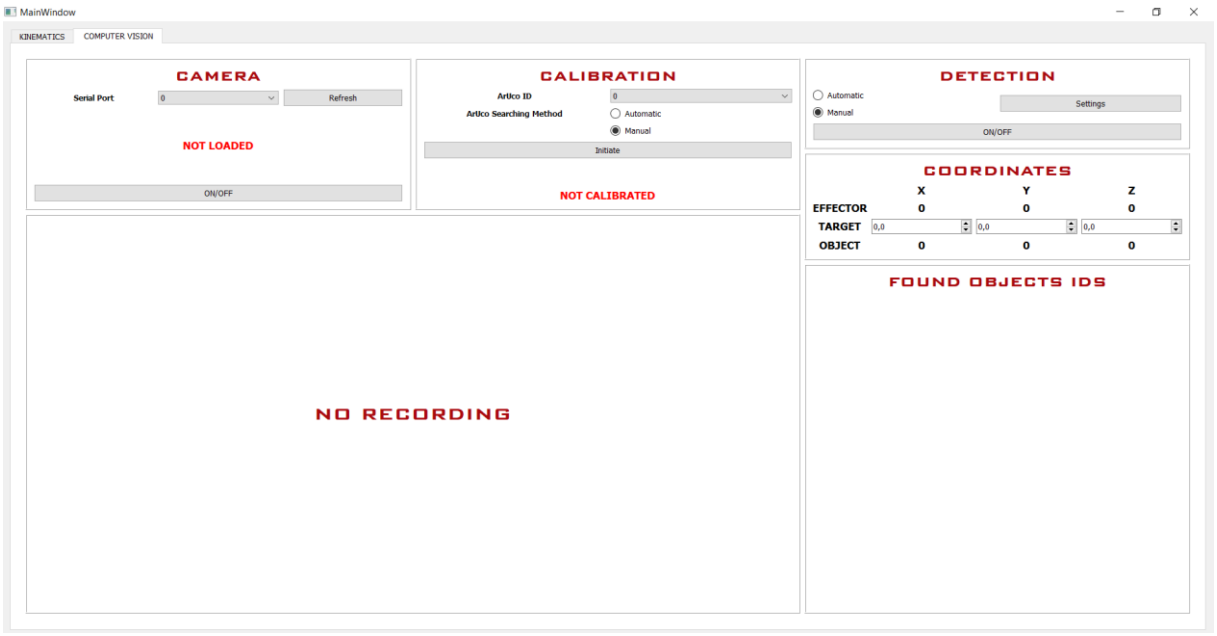
DETECTION – Obdobně jako v případě kalibrace, i pro účely detekce je možné zvolit mezi automatickým a poloautomatickým vyhledáváním objektů. Jelikož objektů může být identifikováno více, robot na základě záznamu z kamery neprovádí okamžité automatické zarovnání k nalezenému objektu. Detekované objekty jsou v záznamu označeny elipsou a svým id. V pravém dolním rohu GUI v okně FOUND OBJETCS IDS se dynamicky vytvářejí tlačítka s id nalezených objektů. Parametry filtru pro tvorbu masky lze měnit v novém okně objevivším se po stisknutí tlačítka Settings podle v témže okně viditelných náhledů.



Obrázek 11: Náhled okna Settings

COORDINATES – Tato sekce dává uživateli informaci o souřadnicích tří důležitých bodů v okolí robota – souřadnice efektoru (EFFECTOR), souřadnice vybraného objektu (OBJECT) a souřadnice, kam má být objekt po zvednutí umístěn (TARGET). Řádky EFFECTOR a OBJECT jsou pouze informativní a ukazují souřadnice spočtené programem. Do řádku TARGET lze zasahovat a měnit, kam má být objekt umístěn. Výchozí souřadnice TARGET odpovídají umístění markeru ArUco při kalibraci.

FOUND OBJECTS IDS – Při detekci jsou do a z tohoto okna vkládána a odebírána tlačítka podle toho, kolik objektů různých id je detekováno. Při zmáčknutí některého z tlačítek jsou souřadnice příslušného objektu poslány metodě *inversekinematics()*, která spočítá potřebná natočení ramen a vynese jejich hodnoty do příslušných spinboxů v záložce KINEMATICS. Změnou hodnot těchto spinboxů je okamžitě iniciován pohyb krokových motorů o daný úhel.



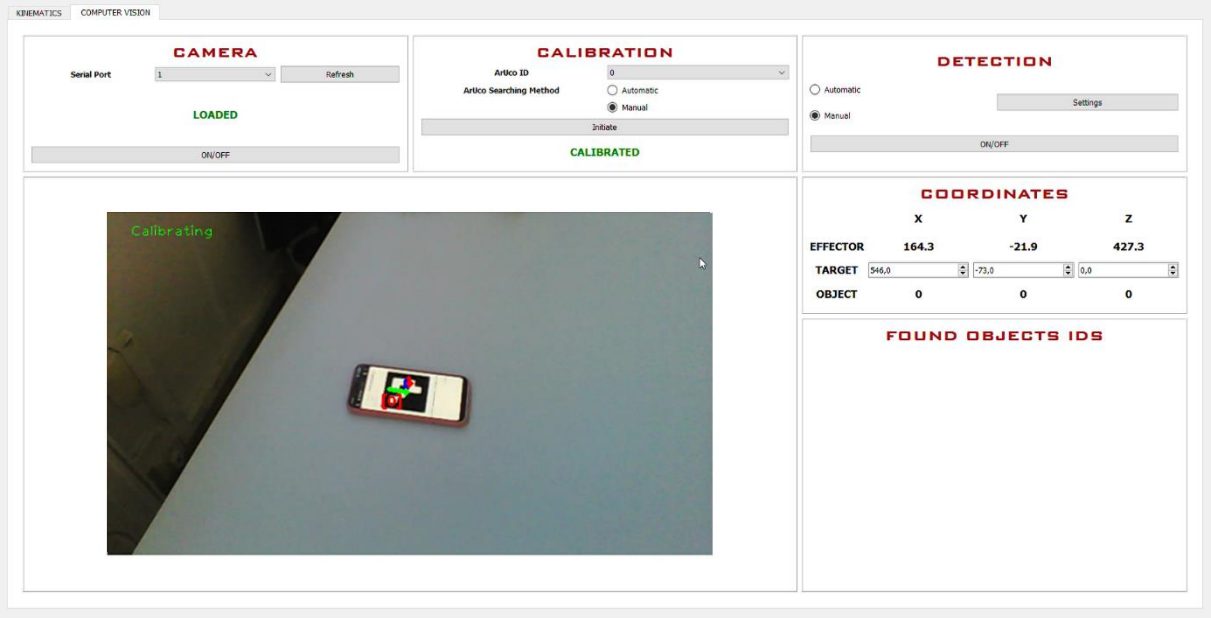
Obrázek 12: Náhled grafického prostředí pro strojové vidění

3.9 Ověření fungování finálního programu

Finální program byl vyzkoušen v praxi. Robot byl umístěn do prostředí s přirozeným světlem, pracovní plochou byl bílý stůl. Zvlášť byla testována detekce objektů a ArUco markerů. Byly otestovány automatické vyhledávání a schopnost robota zarovnat se k markeru nebo předmětu. Po zarovnání byla určena poloha příslušného objektu v souřadnicovém systému robota. Získané souřadnice byly překontrolovány hrubým přeměřením svinovacím metrem.

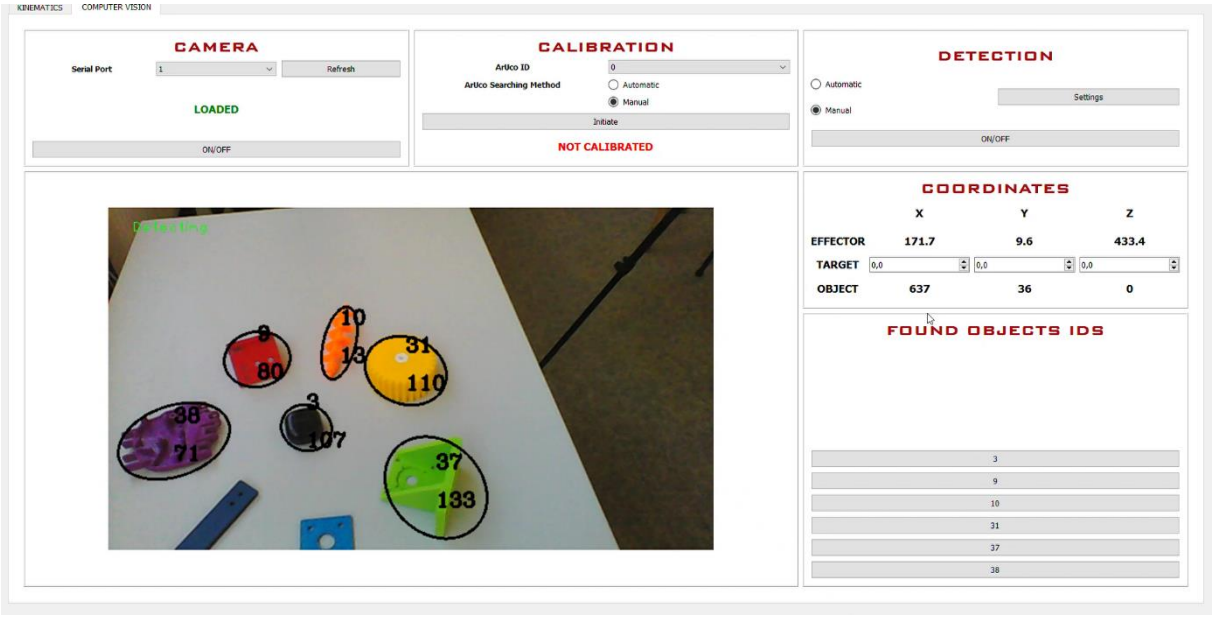
Marker byl zobrazen na displeji telefonu. K dispozici byl i vytištěný marker, ale ten nedosahoval dostatečné kontrastnosti, což v některých momentech vedlo k výpadkům detekce. Marker byl v průběhu zarovnávání robota přesouván, aby byla ověřena schopnost programu pružně reagovat.

Průběh testování byl zaznamenán (Příloha 3) – byl natáčen robot i obrazovka PC za chodu programu. Obrázek 13 ukazuje stav GUI poté, co došlo k zarovnání kamery a spočtení polohy. V tomto případě byla vypočtená poloha [546, -73, 0]. Vzdálenosti stanovené metrem se shodovaly s vypočtenými.



Obrázek 13: Snímek z testování detekce ArUco

Pro detekci objektů bylo vybráno několik objektů různých barev, aby byla otestována všestrannost programu. Souběžně se systémem detekce byl testován i tracker. I v tomto případě byl průběh zaznamenáván na videu (Příloha 4). Poloha žlutého objektu id 31 je [637, 36, 0], kdežto metrem změřená poloha byla [730, 40, 0]. Níže (Obrázek 15) je k vidění snímek GUI v průběhu detekce objektů po zarovnání se žlutým objektem.



Obrázek 14: Snímek z testování detekce objektů

4 Závěr

Na základě předchozí rešerše byl naprogramován systém automatického řízení robotické paže BCN3D Moveo. Všechny prvky umožňující řízení kinematiky paže – mechanické části, elektronické zapojení i softwarový rámec – byly vytvořeny v rámci předchozí bakalářské práce kolegy Patrika Zacha. Nadstavba využívá prvků strojového vidění k určování poloh objektů v okolí paže za účelem jejich uchopení. Systém strojového vidění byl zakomponován do GUI společně se systémem pro manuální ovládání paže pomocí přímé a inverzní kinematiky. Po dokončení byl systém otestován v praxi.

Jednou z využitých metod detekce byly markery ArUco. Zamýšlenou aplikací bylo určení polohy umístění předmětu po jeho uchopení a přemístění. Program je schopen spočítat a uložit polohu markeru poté, co je k němu robot zarovnán. Poloha markeru je počítána triangulací. GUI umožňuje volbu id konkrétního markeru a jeho automatického vyhledávání.

Objekty na pracovní ploše jsou detekovány pomocí funkcí knihovny OpenCV k detekci kontur. Do GUI byly vloženy dva algoritmy k tvorbě masky – filtr bílé barvy a prahový filtr. Parametry obou filtrů lze měnit v průběhu chodu programu. Kvůli odlišení objektů bylo zvoleno kontrastní bílé pozadí. Konturám nalezeným detektorem je opsána elipsa. Za polohu tělesa je brán střed elipsy, úhel pootočení hlavní osy elipsy vůči horizontále poslouží k určení úhlu natočení jednoho z ramen (wrist1) při uchopování předmětu.

K vzájemnému odlišení těles a jejich sledování za pohybu kamery na live záznamu program využívá Eukleidovského trackeru. Objekty (resp. parametry jejich elips) jsou uloženy společně s jim přidělenými id. GUI dynamicky vytváří tlačítka s těmito id, kliknutím na tlačítko je vybrán objekt k uchopení. Elipsy nalezených objektů společně s id a úhly hl. os jsou zobrazeny v záznamu z kamery.

Zaměřování sledovaných objektů probíhá iteračně. Program v reálném čase porovnává vzájemnou polohu středu snímku s polohou objektu (resp. středu jeho elipsy) a podle toho upravuje směr a rychlost pohybu a velikost kroku při každé iteraci. Jsou-li bod a objekt od sebe vzdálenější, jsou nastaveny větší rychlost i krok a naopak. Zarovnání je iniciováno automaticky – v případě kalibrace jejím spuštěním a nalezením markeru, v případě detekce volbou konkrétního objektu.

Všechny sledované předměty jsou při určování polohy redukovány do bodu. Po zarovnání robota k bodu program spočítá reálné souřadnice tohoto bodu triangulací. Parametry myšleného trojúhelníku jsou z-ová souřadnice efektoru a odchylka koncového ramene.

Výsledky testů: Detekce markerů probíhala svižně a při dobrém nasvícení nebo jejich dostatečně kontrastním provedení nedocházelo při jejich sledování k výpadkům. Detekce pomocí kontur vykazovala větší míru nepřesnosti. Jakákoli změna světelných podmínek (např. snímáním jinak nasvícené části pracovní plochy, změnou světla v průběhu dne) vedla buď k selhání při detekci, nebo naopak detekci neexistujících předmětů (např. stínu). Pokud se ale kamera příliš nepohybovala a nemusela se adaptovat

na různé nasvícení, při správné volbě parametrů fungovala spolehlivě. Byla schopna odlišit navzájem blízko položené předměty i při pohybu robota. Většina zkoušených barev byla detekovatelná. Problém přirozeně mohl nastat při detekci světlých odstínů barev, obzvláště růžové a žluté. Eukleidovský tracker fungoval naprosto spolehlivě.

Za spolehlivé lze označit i zaměřování robota vůči objektu. Robot se zarovnával přesně a poměrně rychle. V některých případech docházelo k přejezdu robota za úroveň objektu. Při dané volbě parametrů se nestalo, že by se robot kolem ustálené polohy rozkmital. Změnou určitých parametrů (rychlost posuvu, velikost kroku, provedení výpočtu na každém n-tém snímku) by bylo možné zarovnání zrychlit. V daných podmínkách byly parametry hledány metodou pokus-omyl, a tak se autor spokojil se zvolenými parametry. Určování souřadnic objektů/markerů vykazovalo určitou míru chybovosti. Např. poloha žlutého kolečka ve videu z přílohy Příloha 4 byla podle programu [637, 36, 0], kdežto po hrubém přeměření svinovacím metrem šlo spíše o [730, 40, 0]. Obdobná chyba (měření přibližně o 10% větší než výpočet) se objevovala pravidelně. Možným vysvětlením odchylek jsou nekorespondující úhly natočení ramen v reálu a „na papíře“ v GUI. Tento problém je obzvláště markantní při nenulovém natočení druhého ramene (shoulder), jehož motor nevyvíjí dostatečný moment. Z důvodu použití nevyhovujících motorů nebylo dále možné vyzkoušet, zdali by robot byl schopen uchopit a přemístit nalezený objekt.

Návrhy a připomínky: ArUco nabízí možnost určit polohu kamery vůči markeru spočtením transformačních matic. Práce počítá s tím, že veškeré detekované předměty leží v jedné rovině na pracovní ploše, a tak je poloha markeru počítána stejným způsobem jako poloha ostatních objektů. S využitím všech funkcí ArUco by bylo možné polohu umístění zvolit libovolně v prostoru, nejen v rovině z=0. Obecně lze říci, že nebylo využito plného potenciálu markerů ArUco. Do úvahy připadá např. stanovení poloh ramen pomocí druhé kamery, která by snímala ArUco markery, jimiž by bylo každé rameno osazeno. Co se detekce kontur týče, problém nastával při změnách světelných podmínek. V reálné aplikaci ale lze předpokládat umělé osvětlení, a tedy konstantní podmínky. Případným zvýšením uživatelské přívětivosti by bylo přidání několika přednastavených sad parametrů odpovídajících různému nasvícení. Trasování společně s detekcí by mohlo bát doplněno o schopnost zapamatovat si daný objekt i po jeho zmizení ze záběru (případně jen krátkodobém, např. přejetí rukou). Velkou pomocí by bylo odlehčení a ztužení konstrukce robota, díky čemuž by ramena zaujala polohy přesně odpovídající úhlům nastaveným v GUI.

Aplikace v praxi by byla podmíněna provedením úprav konstrukce a případnou optimalizací řídicího programu. Výsledek ale ukázal, že využití strojového vidění pro řízení robotické ruky je možné i s minimálními náklady. Vytvořený prototyp plní svoji funkci a zadání práce bylo splněno.

5 Bibliografie

1. Český statistický úřad. Využívání informačních a komunikačních technologií v podnikatelském sektoru - rok 2017, leden 2018. *czso.cz.* [Online] 22. 01. 2019. [Citace: 19. 11. 2020.] https://www.czso.cz/csu/czso/automatizovane-sdileni-informaci-o-objednavkach-13m9dhdxdi.

2. MAREK, David, NĚMEC, Petr a FRANČE, Václav. *Automatizae práce v ČR.* Praha : Deloitte, 2018.

3. Český statistický úřad. Aktuální populační vývoj v kostce. *czso.cz.* [Online] [Citace: 20. 11. 2020.] https://www.czso.cz/csu/czso/aktualni-populacni-vyvoj-v-kostce.

4. DevTeam.Space. 10 Examples of Using Machine Vision in Manufacturing. *devteam.space.* [Online] [Citace: 19. 11. 2020.] https://www.devteam.space/blog/10-examples-of-using-machine-vision-in-manufacturing/#one.

5. BCN3D Technologies. BCN3D MOVEO – A fully Open Source 3D printed robot arm. *bcn3d.com.* [Online] [Citace: 21. 11. 2020.] https://www.bcn3d.com/bcn3d-moveo-the-future-of-learning/.

6. ZACH, Patrik. *Uživatelské rozhraní pro polohování robotické ruky BCN3D Moveo.* Praha : ČVUT fakulta strojní, ústav přístrojové a řídicí techniky, 2020.

7. JAIN, Rashmi. 7 Powerful Programming Languages For Doing Machine Learning. *hackerearth.com.* [Online] 02. 01. 2017. [Citace: 07. 11. 2020.] https://www.hackerearth.com/blog/developers/7-programming-languages-and-libraries-used-for-doing-machine-learning/.

8. ŠVEC, Jan. Učebnice jazyka Python (aneb Létající cirkus). [Online] 2002. [Citace: 05. 11. 2020.] http://macek.sandbox.cz/texty/python-tutorial-cz/python-tutorial-cz-a4-2.2.pdf.

9. JetBrains. PyCharm Features. *jetbrains.com.* [Online] 2020. [Citace: 08. 11. 2020.] https://www.jetbrains.com/pycharm/features/coding_assistance.html.

10. OpenCV. OpenCV Library. *opencv.org.* [Online] 2020. [Citace: 07.. 11. 2020.] https://opencv.org/.

11. BRADSKI, Gary a KAEHLER, Adrian. *Learning OpenCV: Computer Vision with the OpenCV Library.* Sebastopol, CA : O'Reilly Media, Inc., 2008.

12. NumPy. numpy.org. *numpy.org.* [Online] 2020. [Citace: 08. 11. 2020.] https://numpy.org/.

13. PyQt5 5.15.4. *pypi.org.* [Online] 10. 03. 2021. [Citace: 04. 05. 2021.] https://pypi.org/project/PyQt5/.

14. KADOUF, Hani Hunud a MUSTAFAH, Yasir Mohd. Colour-based Object Detection and Tracking for Autonomous Quadrotor UAV. [Online] 2013. [Citace: 20. 12. 2020.] https://iopscience.iop.org/article/10.1088/1757-899X/53/1/012086/pdf.

15. VIOLA, Paul a JONES, Michael. Rapid Object Detection using a Boosted Cascade of Simple Features. [Online] 2001. [Citace: 07. 12. 2020.] https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

16. DALAL, Navneet a TRIGGS, Bill. Histogram of Oriented Gradients for Human Detection. [Online] [Citace: 10. 12. 2020.] http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

17. LOWE, David G. Distinctive Image Features from Scale-Invariant Keypoints. [Online] 2004. [Citace: 19. 12. 2020.] https://doi.org/10.1023/B:VISI.0000029664.99615.94.

18. BAY, Herbert, a další. Speeded-Up Robust Features (SURF). [Online] 2008. [Citace: 19. 12. 2020.] ftp://ftp.vision.ee.ethz.ch/publications/articles/eth_biwi_00517.pdf.

19. ROSTEN, Edward a DRUMMOND, Tom. Machine Learning for High-Speed Corner Detection. [Online] 2006. [Citace: 19. 12. 2020.] https://www.researchgate.net/publication/215458901_Machine_Learning_for_High-Speed_Corner_Detection.

20. CALONDER, Michael, a další. BRIEF: Binary Robust Independent Elementary Features. [Online] 2010. [Citace: 19. 12. 2020.] https://link.springer.com/chapter/10.1007/978-3-642-15561-1_56.

21. RUBLEE, Ethan, a další. ORB: an efficient alternative to SIFT or SURF. [Online] 2011. [Citace: 19. 12. 2020.] https://www.researchgate.net/publication/221111151_ORB_an_efficient_alternative_to_SIFT_or_SURF.

22. Ren, SHAOQING, a další. Faster R-CNN: Towards Real-Time Object Detectionwith Region Proposal Networks. [Online] 2017. [Citace: 22. 11. 2020.] https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

23. REDMON, Joseph, a další. You Only Look Once: Unified, Real-Time Object Detection. [Online] 2016. [Citace: 11. 22. 2020.] https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.

24. LIU, Wei, a další. SSD: Single Shot MultiBox Detector. [Online] 2016. [Citace: 23. 11. 2020.] https://arxiv.org/pdf/1512.02325.pdf%EF%BC%89.

25. HUANG, Jonathan, a další. Speed/accuracy trade-offs for modern convolutional object detectors. [Online] 2017. [Citace: 23. 11. 2020.] https://arxiv.org/pdf/1611.10012.pdf.

26. MADRID-CUEVAS, Francisco J., a další. Automatic generation and detection of highly reliable fiducial markers under occlusion. *researchgate.net.* [Online] 06. 2014. [Citace: 12. 02. 2021.] https://www.researchgate.net/publication/260251570_Automatic_generation_and_detection_of_highly_reliable_fiducial_markers_under_occlusion.

27. MI, Tzu-Wie a YANG, Mau-Tsuen. Comparison of Tracking Techniques on 360-Degree Videos. *mdpi.com.* [Online] 29. 06. 2019. [Citace: 04. 12. 2020.] https://www.mdpi.com/2076-3417/9/16/3336.

28. KALAL, Zdeněk, MIKOLAJCZYK, Krystian a MATAS, Jiří. Forward-Backward Error: Automatic Detection of Tracking Failures. *dspace.cvut.cz.* [Online] 2010. [Citace: 04. 12. 2020.] https://dspace.cvut.cz/bitstream/handle/10467/9553/2010-forward-backward-error-automatic-detection-of-tracking-failures.pdf;jsessionid=9124D982CF44C53E877BE9F0FCA66F87?sequence=1.

29. SIDHU, Rumpal Kaur. Tutorial on Minimum Output Sum of Squared Error Filter. *mountainscholar.org.* [Online] 2016. [Citace: 04. 12. 2020.] https://mountainscholar.org/handle/10217/173486.

30. LUKEŽIČ, Alan, a další. Discriminative Correlation Filter Tracker withChannel and Spatial Reliability. *researchgate.net.* [Online] 2018. [Citace: 04. 12. 2020.] https://www.researchgate.net/publication/310953390_Discriminative_Correlation_Filter_with_Channel_and_Spatial_Reliability.

31. JENKINS, Francis A. a WHITE, Harvey E. *Fundamentals of Optics.* místo neznámé : The McGraw-Hill Companies, Inc., 1976. ISBN 0-07-256191-2.

32. TANG, Zhongwei, a další. A Precision Analysis of Camera Distortion Models. [Online] 2017. [Citace: 20. 12. 2020.] https://ieeexplore-ieee-org.ezproxy.techlib.cz/stamp/stamp.jsp?tp=&arnumber=7885103.

33. ZHANG, Zhengyou. A Flexible New Technique for Camera Calibration. [Online] 2000. [Citace: 20. 12. 2020.] https://ieeexplore-ieee-org.ezproxy.techlib.cz/stamp/stamp.jsp?tp=&arnumber=888718.

34. VALÁŠEK, Michael, ŠIKA, Zbyněk a BAUMA, Václav. *Mechanika B.* Praha : Vydavatelství ČVUT, 2004. ISBN 80-01-02919-0.

35. ROSERBROCK, Adrian. Simple object tracking with OpenCV. *pyimagesearch.com.* [Online] 23. 07. 2018. [Citace: 01. 05. 2021.] https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/.

36. tutorialspoint. OpenCV - Overview. *tutorialspoint.com.* [Online] 2020. [Citace: 07. 11. 2020.] https://www.tutorialspoint.com/opencv/opencv_overview.htm.

37. CHOUDHURY, Ambika. 10 Best Python Libraries For Computer Vision. *analyticsindiamag.com.* [Online] 28. 09. 2020. [Citace: 07. 11. 2020.] https://analyticsindiamag.com/10-best-python-libraries-for-computer-vision/.

38. TKALČIČ, Marko a TASIČ, Jurij F. Colour spaces - perceptual, historical and applicational background. [Online] [Citace: 20. 12. 2020.] https://web.archive.org/web/20090306063001/http://ldos.fe.uni-lj.si/docs/documents/20030929092037_markot.pdf.

39. LearnOpenCV. LearnOpenCV. *learnopencv.com.* [Online] [Citace: 04. 12. 2020.] www.learnopencv.com.

40. LOULOUDI, Athanasia. Evaluation of a Bisensor System for 3D Modeling of Indoor Environments. *researchgate.net.* [Online] 2010. [Citace: 21. 01. 2021.] https://www.researchgate.net/publication/297737545_Evaluation_of_a_Bisensor_System_for_3D_Modeling_of_Indoor_Environments.

6 Přílohy

Příloha 1: zdrojový kód, soubory s kalibračními maticemi, soubory pro generování GUI, obrázky

```
|   aruco.py
|   camera_calibration.py
|   main.py
|   mainwindow.py
|   object_detection.py
|   output.doc
|   resource_rc.py
|   secondary.py
|   serial_port.py
|   tracker.py
|   source.qrc
|   mainwindow.ui
|   secondary.ui
|   logo.png
|   Motor.jpg
+---calibration
|       dist.npy
|       mtx.npy
|       ret.npy
|       rvec.npy
|       tvec.npy
```

Příloha 2: upravený firmware Marlin

Příloha 3: záznam chodu kalibrace

Příloha 4: záznam chodu detekce