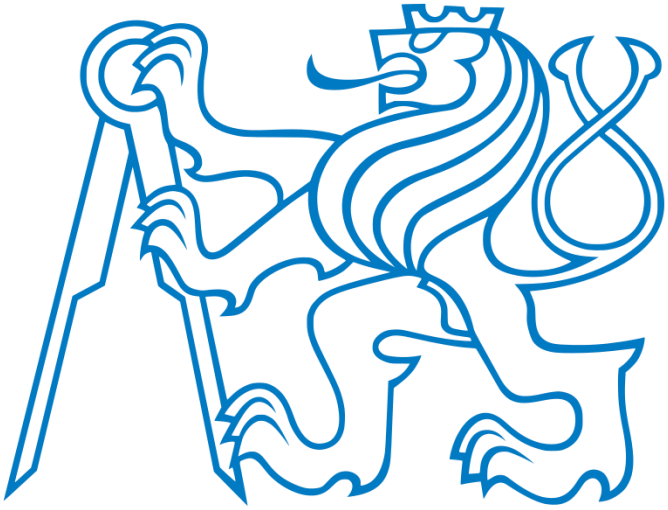
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní

Ústav přístrojové a řídící techniky



Bakalářská práce

ROZPOZNÁVÁNÍ A MANIPULACE DÍLŮ POMOCÍ STROJNÍHO VIDĚNÍ A KOLABORATIVNÍHO ROBOTA

DETECTION AND MANIPULATION OF PARTS USING MACHINE VISION AND COLLABORATIVE ROBOT

Autor: Patrik Brejla

Vedoucí práce: Ing. Matouš Cejnek, Ph.D.

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Brejla** Jméno: **Patrik** Osobní číslo: **508923**

Fakulta/ústav: **Fakulta strojní**

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

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rozpoznání a manipulace dílů pomocí strojového vidění a kolaborativního robota

Název bakalářské práce anglicky:

Part recognition and manipulation using machine vision and collaborative robot

Pokyny pro vypracování:

1) Vybrat vhodný segmentační model pro rozeznávání dílů a otestovat ho.
2) Napsat program, který pomocí segmentačního modelu rozezná díly, lokalizuje je a pošle o nich informaci robotovi.
3) Naprogramovat kolaborativního robota aby dokázal s nalezenými díly manipulovat
4) Otestovat zvolený postup a vytvořený program s několika díly a skutečným robotem

Seznam doporučené literatury:

E. R. Davies, Machine vision: theory, algorithms, practicalities, San Francisco: Elsevier, 2004

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Matouš Cejnek, Ph.D. U12110.3

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

Datum zadání bakalářské práce: **26.04.2024** Termín odevzdání bakalářské práce: **31.05.2024**

Platnost zadání bakalářské práce: _____

Ing. Matouš Cejnek, Ph.D.	prof. Ing. Tomáš Vyhlídal, Ph.D.	doc. Ing. Miroslav Španiel, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	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.

24. 4. 2024
Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím literárních zdrojů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů.

Datum: …… Podpis: ……

Poděkování

Rád bych poděkoval panu Ing. Matoušovi Cejnkovi, Ph.D. za odborné vedení a užitečné rady při tvorbě mé bakalářské práce. Dále bych rád poděkoval mému nadřízenému Cyrillovi Piteau za ochotu a umožnění realizace tohoto projektu ve firmě Doosan Bobcat a také děkuji kolegům v oddělení Automatizace za podporu. Závěrem bych chtěl poděkovat mým rodičům, přítelkyni za jejich zájem a podporu nejen při tvorbě bakalářské práce, ale i po dobu celého studia.

Abstrakt

Abstrakt

Tato bakalářská práce se zaměřuje na problematiku rozpoznávání a manipulaci dílů v průmyslových procesech pomocí strojového vidění a kolaborativního robota. Cílem je navrhnout a otestovat systém, který pomocí kamery identifikuje díly na základě jejich tvaru, pošle potřebné informace robotovi a robot následně provede manipulaci s těmito díly podle specifických požadavků v průmyslovém prostředí. Metodika práce zahrnuje návrh algoritmů pro rozpoznávání, výpočet těžiště a úhlu natočení dílů a naprogramování robota pro příjem dat a manipulaci.

Výsledky ukazují úspěšné a efektivní rozpoznání dílů s vysokou přesností a spolehlivou integraci s kolaborativním robotem. Tato práce ukazuje spolehlivou komunikaci mezi Pythonem a robotem a přináší inovativní možnosti k automatizaci průmyslových procesů do výrobního závodu.

Klíčová slova: Strojové vidění, Python, OpenCV, Label Studio, Kolaborativní robot, Dart Studio

Abstract

This bachelor thesis focuses on the issue of recognizing and manipulating parts in industrial processes using machine vision and collaborative robot. The aim is to develop and test a system that identifies parts based on their shape using a camera, sends the necessary information to the robot and then the robot manipulates these parts according to specific requirements in an industrial environment. The methodology involves process of a algorithm creation for recognition, calculating the center of mass and orientation of the parts, and programming the robot to recieve data and manipulate the parts.

The results demonstrate successful and efficient part recognition with high accuracy and reliable integration with the collaborative robot. This work shows reliable communication between Python and the robot and brings innovative possibilities for automating industrial processes into the manufacturing plant.

Keywords: Machine vision, Python, OpenCV, Label Studio, Collaborative robot, Dart Studio

Seznam zkratek

Cobot	Kolaborativní robot
EMEA	Europe, Middle East, Africa
SPZ	Státní poznávací značka
CMD	Command window – příkazový řádek
Pip	Pip install packages
YOLO	You only look once
TCP	Transmission control protocol
UDP	User datagram protocol
IP	Internet protocol
LS	Label Studio
SAM	Segment Anything Model
YAML	YAML Ain´t Markup Language
DS	Dart Studio
kg	kilogram
mm	milimetr
N	Newton

Obsah

1 Úvod

Jako téma své bakalářské práce jsem si vybral rozpoznávání dílců pomocí strojového vidění, určení jejich polohy na paletě a následnou manipulaci s nimi kolaborativním robotem (dále jako Cobot) ve firmě Doosan Bobcat. Myslím si, že zrovna vidění je nedílnou součástí automatizace u strojů, robotů nebo složitějších procesů ve výrobních závodech. Je to relativně mladá disciplína v tomto oboru, která nabízí mnoho nových možností implementace v průmyslu a ve firmách může být velmi nápomocná k ušetření pracovníků a finančních prostředků.

Cíl tohoto projektu je, aby kamera umístěná nad paletou nebo stolem snímala díly, které se následně vkládají do ohraňovacího lisu. Obraz z kamery bude číst program napsaný v Pythonu a pomocí modelu, vytvořeného v aplikaci Label Studio (dále jako LS), rozpozná na paletě postupně všechny díly podle jejich tvaru a uloží si informaci s jejich číslem (např. 7301668). K docílení budu využívat knihovnu OpenCV. Následně vypočte těžiště dílu, aby robotovi poslal data s jeho přesným umístěním na paletě. Data se budou posílat přes socket do softwaru Dart Studio. V této aplikaci je kolaborativní robot programován pomocí vlastního programovacího jazyku na bázi Pythonu. Cobot bude vědět místo a číslo dílu, díky čemuž může s dílem následně manipulovat a například v aplikaci u již zmiňovaného ohraňovacího lisu by byl schopen zautomatizovat proces a ušetřit firmě pracovníka.

2 Teoretická část

2.1 Doosan Bobcat EMEA s.r.o.

Historie značky Bobcat sahá do roku 1947, kdy Edward Gideon Melroe ve Spojených státech amerických v Gwinneru založil firmu Melroe. Hlavní milník pro spolčenost Bobcat se stal deset let poté. Bratři Kellerové si na své farmě chtěli ulehčit těžkou ruční práci, a tak sestrojili malý tříkolový nakladač, na který v roce 1958 firma Melroe koupila výrobní práva a začala první sériovou výrobu nakladače M60. Pár let poté tyto nakladače začali představovat pod výrobní značkou Melroe Bobcat. Název Bobcat (v překladu rys červený) má reprezentovat tvrdost, rychlost a obratnost. S vývojem dalších strojů expanduje společnost do Evropy, konkrétně do Anglie a v roce 1969 je společnost prodána společnosti Clark Equipment Company. Po dalším rozšíření firmy a příchodem nových typů nakladačů na trh v roce 1995 firmu skupuje Ingersoll-Rand [1] [2].

Obrázek 1: První smykem řízený nakladač Melroe Bobcat [1]

V České republice se stroje Bobcat zahnízdily v roce 2001 po zakoupení společnosti Superstav na Dobříši, tomuto místo se v budoucnu začalo říkat starý závod, kde se o tři roky později vyrobil první Bobcat nakladač 553. Roku 2007 společnost kupuje Doosan Infracore a Bobcat od této doby vystupuje pod názvem Doosan Bobcat. Doosan je firma z Jižní Korey se sídlem v Soulu a je považována za jednu z nejrychleji rostoucích korporací. Toho samého roku na Dobříši otevřeli zbrusu novou továrnu neboli Nový závod. V roce 2014 společnost udělala velkou investici a vedle Nového závodu postavili Inovační centrum, kde dodnes desítky inženýrů z různých zemí vyvíjí nové technologie a stroje. Nachází se zde také jedna z největších protihlukových komor v Evropě [1].

Velký milník nastal v roce 2017 kdy se hlavní sídlo Doosan Bobcat EMEA přemisťuje z Waterloo (Belgie) na Dobříš. Od tohoto momentu je u nás ředitelství pro Evropu, Střední východ a Afriku. V dnešní době sem za prací denně dojíždí přes tisíc lidí a vyrobí se zde 25 000 strojů ročně. V kampusu byla přistavěna ještě jedna budova, a to Bobcat Institute, kam jezdí na proškolení dealeři, zákazníci a operátoři s cílem získání nezbytných znalostí a dovedností se stroji [1].

2.2 Strojové vidění

Stejně jako jsou důležité oči pro člověka při vykonávání jakýchkoliv aktivit je důležité i pro roboty a stroje nějakým způsobem vidět, aby mohli plnit svou práci správně a efektivně. Bez vývoje a postupné implementace strojového vidění by ve firmách nebyla automatizace procesů na takové úrovni jako je dnes.

Strojové vidění je revoluční oblast umělé inteligence. Zabývá se vývojem inteligentních algoritmů a metod pro analýzu fotografií, videí a 3D obrazů. Využívá tedy široké spektrum metod a technik, jako matematické zpracování obrazu, strojového učení a hlubokého učení, aby extrahovalo relevantní informace z vizuálních dat a tato data se následně převedla na užitečná data, která po stroji požadujeme. Hlavním cílem strojového vidění je z těchto vizuálních dat v co nejkratším čase poznat a detekovat různé objekty, vzory, vzdálenosti a podobně [3].

V praxi se dnes strojové vidění používá v mnoha odvětvích průmyslu, ale i v každodenním životě. Jako pár příkladů mohu uvézt automobilový průmysl (dnes snaha o autonomní vozidla), lékařství, bezpečnost, kontrolování SPZ u vozidel a mnoho dalších. Můžeme se s tím setkat v našich každodenních životech. Toto odvětví je již na velmi vysoké úrovni, nicméně je to stále vcelku mladý obor a stále nabízí mnoho prostoru pro další vývoj samotné umělé inteligence a zároveň zdokonalování implementace v průmyslu [3].

2.3 Python

Na světě v dnešní době existuje mnoho programovacích jazyků, jako je Python, Java, C, PHP a další. Každý jazyk je alespoň v jedné věci dobrý a dá se pro konkrétní projekty s trochou cviku dobře aplikovat. Python je výkonný, snadno a rychle pochopitelný jazyk, který se dá uplatnit v mnoha činnostech, jak ve všedním životě, tak v průmyslu. Je intuitivní

na naučení a i úplný začátečník se po krátkém tréninku začne orientovat v základních operacích, funkcích a algoritmech. Je volně přístupný na internetu a nabízí širokou škálu knihoven obsahujících příkazy a funkce pro plnění složitých úkolů pouze v pár řádcích. Má velkou a přátelskou komunitu, což ulehčuje práci s tímto programem a případné řešení problémů a chyb [4] [5].

Python nám umožňuje konat jednoduché operace jako je sčítání, odečítání, násobení a dělení. Pro psaní delších kódů je možné využít klasický textový dokument, kam můžeme napsat náš kód, soubor uložit s příponou *.py* a program spustit v příkazovém řádku (dále jako CMD). Pro zjednodušení existují různé aplikace, jako je třeba Pycharm nebo Visual Studio. Uživatel v nich napíše svůj kód a může ho rovnou spustit a vyzkoušet. Výstup kódu se ukáže v konzoli a pokud je v kódu nějaká chyba, napíše, o jakou chybu se jedná, v jakém je řádku a řešení je mnohdy snadno dohledatelné na oficiálních stránkách Pythonu nebo na uživatelských fórech.

Pro mou práci budu využívat verzi pythonu 3.10.11 v aplikaci Pycharm a knihovny OpenCV, Numpy, Socket, Ultralytics, Shapely.

2.3.1 Instalátor balíčků Pip

Pip je nástroj pro snadnou instalaci, aktualizaci a odinstalaci balíčků (knihoven) do Pythonu. Zkratka Pip znamená „Pip installs packages“ [6].

Instalátor se v operačním systému Windows používá v CMD, kde pro instalaci knihovny stačí jednoduchý příkaz: `pip install <název_knihovny>`. Například pro instalaci knihovny OpenCV by příkaz vypadal takto: `pip install opencv-python`. Pokud by byla požadována konkrétní verze knihovny, stačí kód upravit následovně: `pip install opencv-python==<verze>` [6].

Aktualizace a odinstalace balíčků je intuitivní podobně jako instalace. Pro aktualizaci zadáme příkaz: `pip install --upgrade <název_knihovny>`. Odinstalaci provedeme takto: `pip uninstall <název_knihovny>` [6].

Pro instalaci většího množství knihoven najednou nám nástroj Pip umožňuje využít jednoduchý způsob. V počítači se vytvoří textový dokument obsahující seznam balíčků a jejich verzí. Počet knihoven není omezený. Standardně se tento dokument pojmenovává requirements.txt. Pomocí příkazu: `pip install -r requirements.txt` se nainstaluje vše najednou [7].

2.3.2 OpenCV

OpenCV, neboli Open Source Computer Vision Library [9], je pro Python něco jako oči pro člověka. Má velmi široké využití v průmyslu, akademickém prostředí, výzkumných institucích, ale mohou ji používat i nadšenci, kteří se Pythonu věnují pro zábavu. Můžeme říct, že to je klíčová knihovna pro automatizaci různých procesů. OpenCV dokáže sledovat a detekovat objekty, rozpoznávat obličeje, zpracovávat obraz v reálném čase a mnoho dalších [9].

V praxi se knihovna OpenCV používá pro jednoduchá upravování obrázků nebo fotek různých formátů (JPG, PNG, BMP a dalších) [8], jako například otáčení, změna velikosti, vkládání textu a objektů, ořezávání a podobně. Příklad můžete vidět na Obrázku 6, kde jsem spojil dva obrázky a přidal mezi ně šipku. U složitějších aplikací už se využívá možnost detekování a rozpoznávání objektů na bezpečnostních kamerách, čtení státních poznávacích značek u aut nebo třeba, jako v mojí aplikaci, hledání dílů na paletě. K mojí aplikaci OpenCV spolupracuje s knihovnou Ultralytics, o které píšu dále. To je také velká výhoda tohoto balíčku, je kompatibilní s velkým množstvím knihoven pro Python a kamer, tím ještě rozšiřuje své možnosti pro své využití v průmyslové praxi [8] [9].

Nejčastěji používané příkazy v Pythonu s knihovnou OpenCV [22]:

`cv2.VideoCapture()` - zapnutí kamery nebo nahrání videa z počítače

`cv2.imread()` - nahrání obrázku/fotky

`cv2.resize()` - změna velikosti obrázku nebo videa

`cv2.imshow()` - zobrazení obrázku nebo videa okně

`cap.release()` - vypnutí kamery (`cap` je proměnná se zdrojem videa)

`cv2.waitKey()` - po stisknutí tlačítka přejde na další krok v kódu

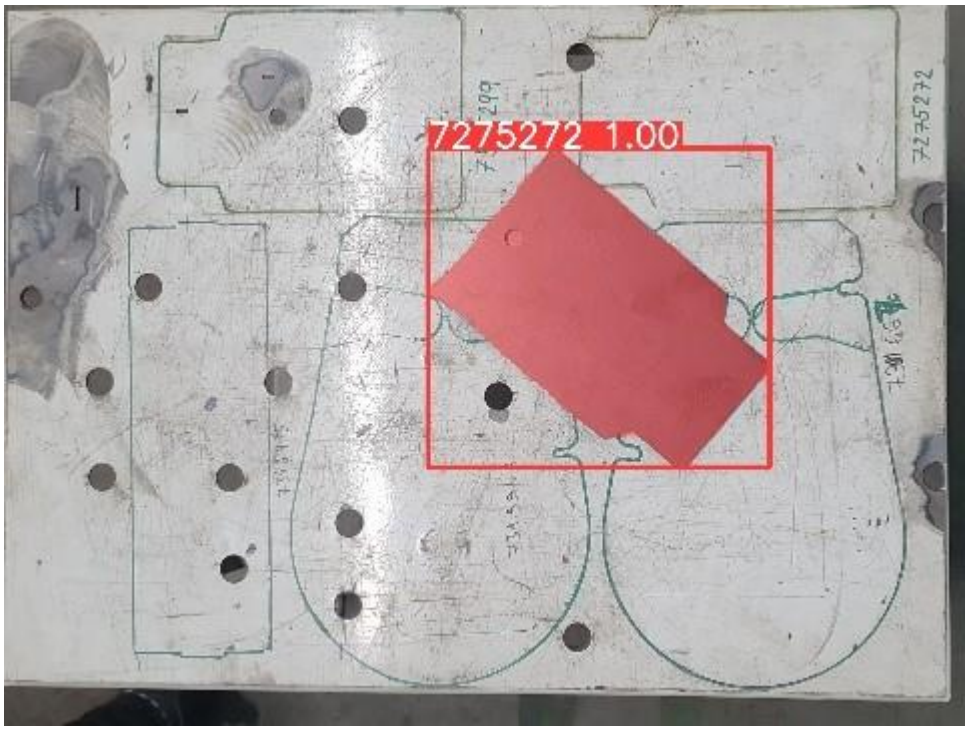
`cv2.destroyAllWindows()` - zavření oken s obrázkem nebo videem

2.3.3 Ultralytics – YOLOv8

Ultralytics je společnost, která se specializuje na vývoji nástrojů a technologií v oblasti strojového vidění s využitím umělé inteligence. Jejich hlavním a nejpoužívanějším produktem je knihovna YOLO. V angličtině je tato zkratka pro You Only Look Once, neboli Podíváš se jen jednou. Tato volně přístupná knihovna disponuje svou rychlostí a snadnou implementací v průmyslu, vědě a výzkumu, podobně jako OpenCV [11].

Z tohoto balíčku využívám v mé práci právě balíček YOLO. Je to revoluční způsob, jak detekovat, segmentovat a rozeznávat objekty a osoby na obrazu z kamery v reálném čase. Od ostatních tradičních metod, které provádějí detekci po různých částech obrazu, tato knihovna provádí detekci přímo pro celý obraz, čímž dosahuje vysoké rychlosti a efektivity, což je v dnešním průmyslu jednou z klíčových vlastností implementace jakékoliv automatizace [11].

V tomto projektu využívám semantickou segmentaci od Ultralytics. Je to pokročilá technika v oblasti strojového vidění. Má za cíl přesně určit, které třídě patří každý pixel v obraze. Rozpozná tedy objekt, který je definovaný připraveným a naučeným modelem (tvorbě modelu se věnuji v praktické části), ohraničí ho polygonem a přiřadí k němu třídu (nebo název) konkrétního objektu, příklad je na Obrázku 2. Výstup dat ze segmentace je dostačující k tomu, abych mohl určit o jaký díl se jedná a kde přesně se nachází [10].



Obrázek 2: Příklad rozpoznaného dílu – překrytý červenou maskou ukazující tvar dílu, ohraničený obdélníkem, na jehož horní straně je uvedeno číslo dílu a jistota rozpoznání

2.3.4 NumPy

NumPy je jednou z dalších knihoven v Pythonu sloužící pro vědecké výpočty a manipulaci s numerickými daty. Rychle a efektivně pracuje s maticemi, poli a vektory. Je vhodná pro zpracování dat, implementaci algoritmů v lineární algebře, statistiku, matematické modelování a strojové učení. Jednou z klíčových funkcí NumPy je Broadcasting, což umožňuje provádět operace mezi poli různých rozměrů bez nutnosti explicitního přeformátování [12]. Integruje s dalšími knihovnami, jako je třeba SciPy, Pandas a Matplotlib.

V praxi se knihovna NumPy často používá pro výzkum ve fyzice, biologii a chemii, strojové vidění pro efektivní manipulaci s daty a tvorbu modelů a například pro finanční analýzu [12].

2.3.5 Shapely

Shapely je knihovna v Pythonu poskytující nástroje pro manipulaci s geometrickými objekty. Mohou to být body, linie nebo třeba polygony. Umožňuje s nimi provádět různé operace užitečné k následné analýze prostorových dat a tvorbě aplikací. Velké využití v praxi má u Geografických informačních systémů (GIS), to znamená mapy, body zájmu a polygonální oblasti a u urbanistického plánování. Já tuto knihovnu využívám pro výpočet těžiště polygonu, respektive pro výpočet těžiště dílu [13].

V mé práci jsem nejprve zkoušel počítat těžiště matematickými operacemi s knihovnou NumPy. Po porovnání těchto dvou metod jsem zvolil metodu s využitím knihovny Shapely. Těžiště dílu tato knihovna vždy přesně najde bez ohledu na to, jak je díl natočený. Na dalším

Obrázku 3 můžete vidět porovnání těchto dvou metod. Na levé fotce je výstup z výpočtu pomocí NumPy a pár matematických operací a na pravé straně je výsledek knihovny Shapely.

Obrázek 3: Porovnání dvou metod pro výpočet těžiště dílu – v levé části je těžiště vypočteno knihovnou NumPy a vyznačeno zelenou tečkou, na pravém obrázku je zobrazen výpočet knihovnou Shapely červenou tečkou

Příkaz pro zjištění středu pomocí Shapely je velmi jednoduchý [13]. Stačí jeden řádek:

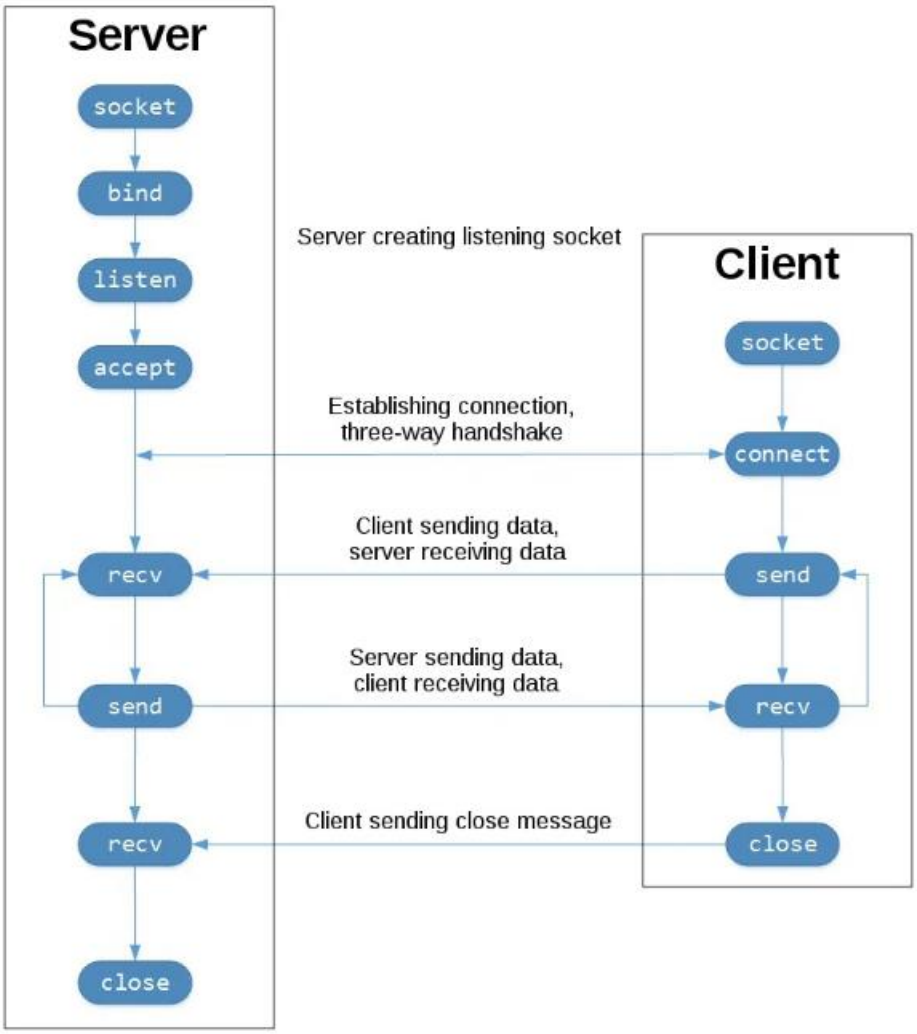
```
center = list(Polygon(maska).centroid.coords)
```

Výsledek se uloží do proměnné `center`, ze které se vytáhnou souřadnice x a y středu polygonu. V proměnné `maska` jsou body masky dílu, který program našel. Konkrétně je to červené zbarvení dílu na levé straně Obrázku 3.

2.3.6 Socket

Balíček Socket v Pythonu poskytuje snadné a efektivní síťové programování. Umožňuje vytvářet sítě a tím i komunikaci mezi počítači a aplikacemi pomocí různých protokolů, jako je TCP (Transmission Control Protocol) nebo UDP (User Datagram Protocol) [14]. K propojení se využívá IP adresa zařízení a zvolený port. Mezi aplikacemi v Pythonu se port může volit libovolně, nicméně u nějakých aplikací je třeba mít specifický port, který by měl uvádět výrobce v dokumentaci.

Hlavní dvojice ke správnému fungování je klient a server. Standardně je na jednom zařízení vytvořen server se svou IP adresou a portem a na ostatních zařízeních se vytvoří klienti kteří se na server mohou připojit. Po připojení už mezi sebou mohou komunikovat jakkoliv, není tam žádné omezení, jestli je zpráva od serveru nebo od klienta. Jediná podmínka pro správné zaslání zprávy je zakódování do bytového formátu, pomocí jednoduchého příkazu `encode()` [15]. Na Obrázku 4 můžete ve schématu vidět, jak server a klient vzájemně pracují.



Obrázek 4: Schéma vzájemné komunikace mezi socket serverem a klientem [15]

Často používané příkazy v Pythonu [15]:

`pip install sockets` - instalace knihovny pomocí instalátoru Pip

`s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
- vytvoření serveru, uložení do proměnné `s`

`s.bind((IP_addresa, PORT))` - otevření serveru

`s.listen()` - hledání dostupných zařízení

`s.accept()` - přijmutí připojení klienta

`s.connect((IP_adresa, PORT))` - připojení klienta k serveru

`s.sendall()` - odeslání zprávy, musí být zakódována

`s.recv(1024)` - přijetí zprávy (max 1024 bytů) v zakódovaném formátu

`s.encode()` - zakódování zprávy do bytového typu

`s.decode()` - rozkódování zprávy z bytů do stringu

Na Obrázku 5 ukazuji příklad kódu pro vytvoření serveru a vypsání IP adresy připojeného klienta.

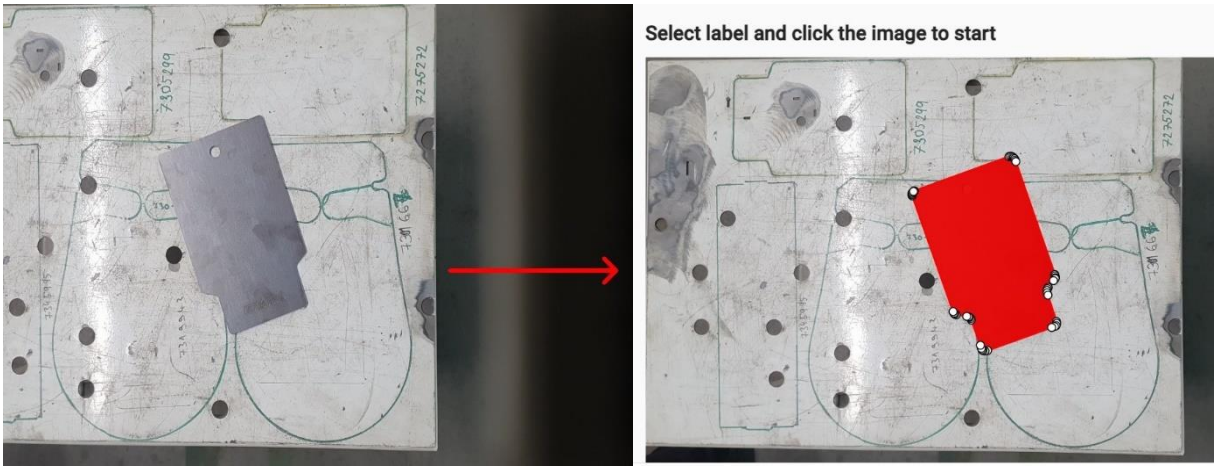
```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 20001))
print("Server is online")
s.listen()
client_socket, address = s.accept()
print(f"{address} has connected to the server")
```

Obrázek 5: Příklad kódu pro vytvoření socket serveru

2.4 Label Studio

LS je volně přístupná platforma pro anotaci dat. Umožňuje vytvářet, spravovat a nadále distribuovat různé anotační projekty pro vývoj a trénování modelů strojového učení. Uživatel zde může anotovat data pro obraz, zvuk, písmo, jazyk a podobně. Následný export dat pro tvorbu modelu má LS vyřešeno velmi dobře, protože může integrovat s externími nástroji, jako třeba TensorFlow, PyTorch, scikit-learn, YOLO a další [16].

Pro mou aplikaci je nejdůležitější, že dokáže vyexportovat anotovaná data ve formátu YOLO. To nám dá zazipovaný soubor s oanotovanými fotkami a ke každé z nich přiřazený textový dokument podle názvu. V textovém dokumentu jsou data o objektech (třídách), které se na fotce nebo obrázku vyskytují, a souřadnice všech bodů polygonu anotovaných v LS (viz Obrázek 6).



Obrázek 6: Ukázka oanotování fotky dílu v Label Studiu – na levé straně je originální fotografie dílu, na pravé straně je jeho označený tvar

Komunitní podpora a rozvoj LS vůbec nechybí. Platforma je pravidelně aktualizována a vývojáři stále pracují na nových inovacích a funkcích. V roce 2023 uvedli na veřejnost nové rozšíření, které anotuje fotky skoro úplně samo. Toto rozšíření má zkratku SAM a znamená to Segment Anything Model. Uživatelům to velmi várazně urychlí práci při anotovnání. Místo

manuálního anotování fotek pomocí označování polygonů stačí jedno kliknutí a objekt se označí sám. Osobně jsem si toto rozšíření vyzkoušel a funguje to velmi dobře, bohužel to nemohu vyexportovat ve formátu YOLO, který já potřebuji, proto je to pro mou aplikaci nepoužitelné. Podporuje to např. formát JSON [17].

2.5 Doosan Robotics

Společnost Doosan Robotics, jakožto součást Doosan Group, je přední globální výrobce robotů do průmyslu. Zaměřuje se na vývoj a výrobu těchto robotů, které jsou využívány po celém světě, převážně ve společnostech, které vlastní Doosan [18].

Hlavně se zaměřují na kolaborativní roboty neboli roboty, kteří jsou schopni spolupracovat s lidmi a tím ulehčí práci a sníží bezpečnostní nároky. Na trhu nabízejí čtyři typy kolaborativních robotů – H, M, A, E. Dále se firma zaměřuje přímo na robotické systémy navržených pro konkrétní průmyslové aplikace [19].

Roboti ze všech tříd se vyznačují pohybem v šesti osách, vysokou přesností a precizností. Třída robotů „H“ má největší využití u manipulace s díly. Má největší nosnost ze všech a to 25 kg. Třídy „A“ a „M“ jsou si velmi podobné. Kolaborativní roboty typu „M“ označují jako nejkvalitnější, proto zkratka „M“ stojí za slovem „Masterpiece“. Roboti „A“ jsou zase hbitější a úspornější. Poslední skupina robotů „E“ má největší využití v potravinářském průmyslu, jelikož je skladný, má certifikáty a akreditace za velmi dobrou hygienu a sanitaci [19].

2.5.1 Doosan Collaborative Robot M1013

V mém projektu používám Cobota M1013 (viz Obrázek 7), protože tento typ máme ve firmě. Tento stroj má maximální nosnost 10 kg a dosah 1300 mm, což jsou pro mou aplikaci optimální hodnoty. Společnost uvádí přesnost pohybů s tolerancí ± 0,05 mm [19]. Klíčovými prvky jsou vysoká bezpečnost, díky šesti moderním senzorům na kontrolování kroutícího momentu, to souvisí s dalším prvkem, a to je silové ovládání pohybů pomocí funkce Force Control. Doosan se pyšní také kvalitně zpracovaným rozhraním pro základní programování robota, které usnadňuje život operátorům. Slouží k tomu tablet (neboli Teach Pendant) připojený k řídící jednotce Cobota [19]. Operátor zde může nejen ovládat a programovat jednoduché funkce díky DART Platform v systému, může také konfigurovat a nastavovat nástavce na hlavu Cobota, měřit hmotnost nástavců nebo dílů nebo například vytvářet kolaborativní zóny, kde se v programu pohybuje se sníženou rychlostí, protože zde může být riziko kontaktu s lidskou osobou.

Obrázek 7: Doosan kolaborativní robot M1013 [22]

Cobot má veliké spektrum využití. Montáž různých zařízení, konkrétně třeba šroubování nebo utahování, manipulaci

s díly, leštění, broušení, nanášení lepidla, a dokonce i svařování [19]. Pro každou aplikaci je třeba ke Cobotovi pořídit konkrétní koncový efektor. Tyto efektory většinou nabízejí jiné firmy než Doosan Robotics, efektory jsou buď kompatibilní s Coboty od Doosan Robotics nebo jsou vyráběny přímo na ně. Já používám koncový efektor od firmy On Robot, a to vakuovou savku.

2.5.2 DART Studio

Dart Studio je softwarová platforma vyvinutá právě firmou Doosan Robotics, používaná k programování a ovládání průmyslových a kolaborativních robotů. Software má uživatelsky přívětivé rozhraní se širokou škálou nástrojů pro tvorbu a optimalizaci aplikací a programů pro roboty. Vše funguje s vysokou efektivitou a přesností, přesně jak by bylo od takovéto platformy požadováno.

K tomuto programu společnost poskytuje i manuál na programování, který usnadní učení a používání tohoto programovacího jazyka. Jazyk běží na bázi Pythonu, takže pro zaběhlejší programátory je jednoduché zde tvořit různé cykly a aplikace. Při spuštění programu je možné nejdříve všechny pohyby robota vidět v simulaci, kam můžeme nahrát 3D modely okolních objektů a proces nasimulovat přesně tak, jako je tomu ve skutečnosti.

Dart Studio je kompatibilní s různými externími zařízeními a technologiemi v průmyslovém prostředí. Cobot může tedy komunikovat jak s jinými roboty, CNC stroji nebo jinými programy a tím snáze můžeme docílit automatizace konkrétního procesu.

V průmyslu se tento software může využít v různých aplikacích, stejně jako Cobot od společnosti Doosan Robotics, jako například v automatizaci výroby, manipulaci s materiálem, montážních operacích, kontrolách, testováních a podobně.

3 Praktická část

V praktické části mé bakalářské práce popíši celý proces od původní tvorby datasetu modelu na rozeznávání dílů, až po vzájemnou komunikaci Dart Studia (dále jako DS) a Pythonu při hledání a manipulaci s díly.

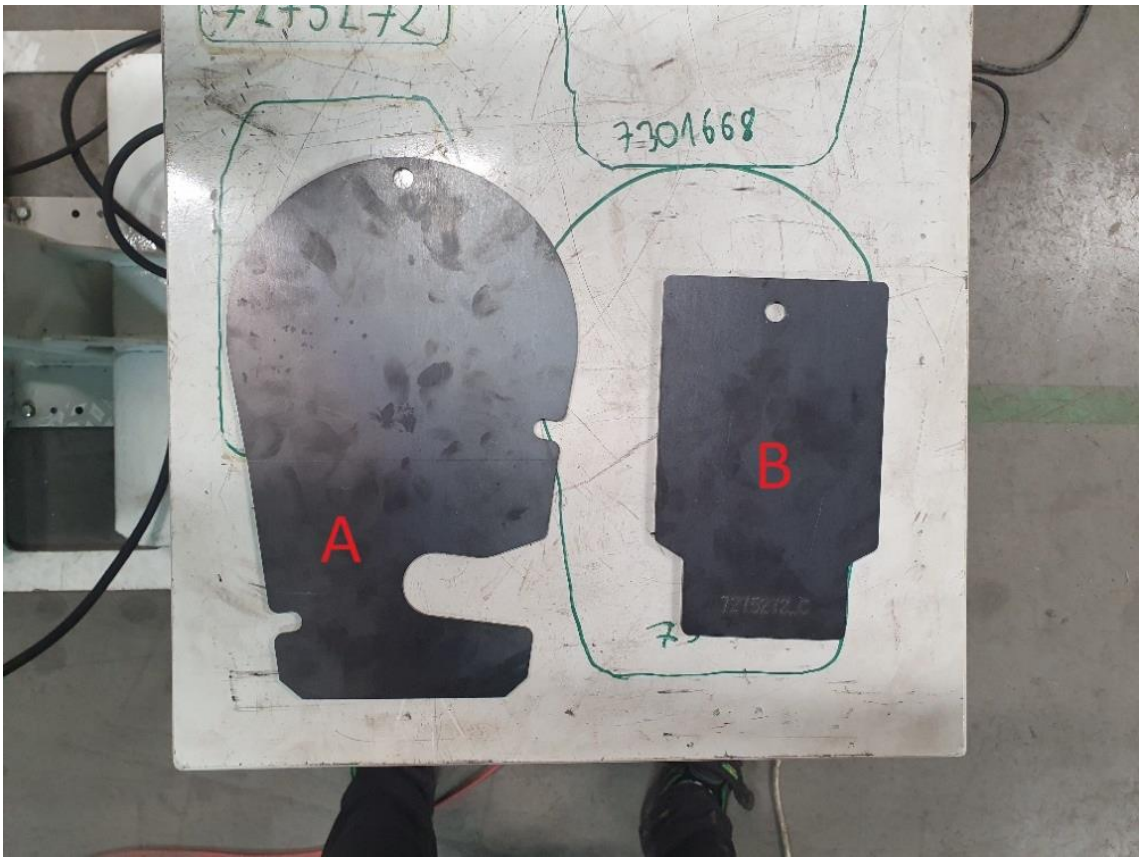
3.1 Rozeznávání dílů

K docílení toho, aby program napsaný v Pythonu rozeznal jednotlivé díly, je potřeba vytvořit model, který využívá strojového učení. Funguje na tom principu, že ze souboru fotografií dílů, které se oanotují v LS, se pomocí příkazů v Pythonu vytvoří model. Učí se právě z tohoto datasetu fotografií a bodů, které definují tvar konkrétních dílů. Model se poté připojí k obrazu zachytávaného z kamery a v reálném čase je schopný rozeznat typ dílů. V této části využívám v Pythonu knihovny YOLO od Ultralytics a OpenCV.

3.1.1 Tvorba datasetu

Datasetem je nazývána složka nebo soubor fotek použitý k naučení modelu. Abych ulehčil tuto práci a ukázal možnost automatizace i tohoto jednoduchého úkolu, vytvořil jsem program v Pythonu komunikující s Cobotem, který ve správnou chvíli vyfotí díl v náhodné poloze.

Nejprve jsem si stanovil pro každý díl plochu na testovacím stole, kam Cobot může díl položit, aniž by spadnul na zem. Jelikož je tento projekt v testovací fáze, tak jsem model vytvářel pro dva díly 7301667 a 7275272, označme je jako díl „A“ a „B“. Díly můžete vidět na Obrázku 8. Pro každý díl jsem tedy definoval pole, ve kterém se může pohybovat na stole s rozměry 675 x 490 mm.



Obrázek 8: Díly „A“ a „B“ používané v tomto projektu

Po proměření jsem stanovil, že díl „A“ by měl být nejblíž 150 mm od kraje stolu a díl „B“ 100 mm. Aby fotografie dílů byly co nejvíce odlišné, vytvořil jsem ještě proměnnou `rotation`, která Cobotovi řekne o kolik stupňů má díl pootočit od 0° do 180°. Proměnné v Pythonu jsou tedy pojmenovány a definovány takto:

```
partA_X = range(0, 200)
partA_Y = range(0, 400)
partB_X = range(0, 300)
partB_Y = range(0, 500)
rotation = range(0, 180)
```

Tato pole nemusí být na milimetry přesná, proto jsem je zaokrouhlil na celé stovky a jsou zadány pro osu X a osu Y.

Před spuštěním programu nad stůl umístím na držák kameru snímající celou pracovní plochu, která bude následně dělat fotky dílů a kameru připojím k programu v Pythonu. Toto popíši dále ve vysvětlování skriptu. V dalším kroku přepnu signál pro savku na Cobotovi na hodnotu 1, aby se zapnula, a přisaji požadovaný díl. Cobot tedy začíná program již s přisátým dílem.

Ještě je třeba říci, že oba programy běží ve for cyklech s počtem opakování, který si stanovím. Zadám na začátek tedy for cyklus: `for i in range(počet_dílů):`. Jinak by skript proběhnul jednou a program by se ukončil a tím by to ztrácelo veškerý smysl, takto se může opakovat třeba stokrát.

Nejdříve ze všeho je třeba propojit Python (klient) a DS (server) přes socket. V DS vytvořím server s portem 20002 pro simulátor nebo 20001 pro reálného Cobota:

```
sock = server_socket_open(port)
```

Python připojím na tento socket se stejným portem a IP adresou robota:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP_adresa, port))
```

Aby se oba programy správně propojily, musím nejdříve spustit server a až potom klienta, jinak by program nefungoval.

Pro zkrácení kódu jsem na jeho začátku vytvořil funkci pro výběr a poslání náhodného čísla z výše uvedených intervalů, zakódování čísla do bytů a poslání robotovi přes socket. Pro komunikaci Pythonu a Cobota zde tedy využiji knihovny Socket, která přes síť přepošle informaci zakódovanou v bytovém formátu. Ještě předtím program musí poslat informaci o tom, kolik znaků mají jednotlivé souřadnice, aby přečetl přesně tolik bytů ve zprávě. Toto je malý nedostatek, na který jsem narazil. DS totiž při přijímání zpráv přes socket přečte pouze konkrétní množství znaků najednou. Jelikož se tato čísla budou různě měnit od jednociferných po tříciferná, je třeba znát jejich počet znaků. Funkci jsem pojmenoval `cord_send()` a vypadá následovně:

```
def cord_send(x):
    random_number = random.choice(x)
    length = len(str(random_number))
    mess_len = str(length).encode()
    s.send(mess_len)
    mess_number = str(random.number).encode()
    s.send(mess_number)
```

Tato funkce je pro díl vyvolána vždy třikrát, a to pro výběr náhodné souřadnice na ose X, Y a náhodného stupně pro otočení dílu. Funkci zadám tedy například takto: `cord_send(partA_X)`. Pomocí příkazu `random.choice()` vybere náhodné číslo, dále si zjistí kolik znaků tato informace obsahuje příkazem `len()` a zadáním `s.send()` odešle informaci o délce a obsahu zprávy do DS. Následně program čeká dokud mu nepřijde zpráva zpět od DS, že díl úspěšně umístil na náhodnou pozici a odjel na kraj stolu.

DS přijímá postupně všechny zprávy, rozkóduje je, vytvoří proměnnou s těmito třemi souřadnicemi a s dílem se nad místo přesune. Díl položí na stůl a odjede na domovskou pozici, kterou má definovanou na rohu stolu, aby kamera viděla na stůl s dílem bez jakýchkoliv překážek. Jakmile zaparkuje v domácí pozici, pošle zprávu „ok“ do druhého programu.

Zpráv z Pythonu přijme, rozkóduje a převede na číselný datový formát přesně 6, kvůli informaci o počtu znaků. Následující část kódu použije tedy pro každou souřadnici zvlášť.

```
res, length = server_socket_read(sock, length = 1)
length.decode(“utf-8“)
int(length)
res, partA_X = server_socket_read(sock, length)
partA_X.decode(“utf-8“)
int(partA_X)
```

Domovskou pozici má uloženou v proměnné `home`. Od této pozice odečítá přijaté souřadnice a uloží si je do nové proměnné `place_part` pro díl na indexy 0, 1, 5, neboli pro souřadnice X, Y a otočení, jako je vidět v následujících řádcích kódu:

```
place_part = home
place_part[0] = home[0] - partA_X
place_part[1] = home[1] - partA_Y
place_part[5] = home[5] - rotation
```

Nyní má DS uloženou novou proměnnou s polohou položení dílu, kam se přesune za jednu vteřinu příkazem `movel`:

```
movel(place_part, time = 1)
```

Díl položí na stůl a vypne savku příkazem `set_digital_output(1, 0)`. Číslo jedna definuje signál (savku) a 0 definuje přepsání hodnoty, aby se savka vypnula a pustila díl.

Po přesunu do domácí pozice pošle informaci Pythonu, aby mohl pokračovat ve skriptu:

```
msg = “ok“
msg = msg.encode()
server_socket_write(sock, msg)
```

Python celou dobu čeká na tuto zprávu tímto while cyklem:

```
while True:
   msg.recv(1024)
   msg = msg.decode(“utf-8“)
   if msg == “ok“:
         break
```

Příkaz `msg.recv(1024)` přijímá zprávu od zařízení připojeného přes socket. Číslo 1024 nám značí počet bytů, který má program přešíst, je to zároveň maximální možný počet bytů, který je schopný přijmout. Zprávu přijme opět ve formě bytů, je tedy třeba ji rozkódovat funkcí `msg.decode("utf-8")`. Jakmile přijme zprávu, ukončí while smyčku příkazem `break` a pokračuje ve skriptu.

V další části kódu začne Python využívat knihovnu OpenCV, okamžitě vyfotí díl v náhodné poloze a uloží jej do určené složky. Fotku uloží s daným názvem a každý další cyklus zvětší její index o 1. Kdyby tomu tak nebylo, tak by Python ukládal všechny fotky pod jedním názvem a tím pádem by ve složce byla neustále pouze jedna fotka, která by se po každém cyklu přepsala za novou.

```
k = 1
cap = cv2.VideoCapture(kamera)
while cap.isOpened():
   success, frame = cap.read()
   if success:
         cv2.imwrite(r"cesta_do_složky/fotografie_%d.jpg" % k, frame)
         break
   k += 1
   cap.release()
```

Do proměnné `cap` se uloží funkce z knihovny OpenCV umožňující načtení a zachycení obrazu z kamery. Do závorky se zadá informaci o kameře. Je možné zadat například `0` na otevření webkamery na notebooku, `1` pro otevření externí kamery připojené například přes USB kabel, nebo IP adresu, na kterou kamera přenáší svůj obraz. Já pro testování používal kameru z mého telefonu a obraz jsem sdílel přes IP adresu pomocí aplikace IP Webcam. Tato aplikace vytvoří server na localhostu a přes Wifi připojení přenáší obraz, který mohu načíst v Pythonu.

Další řádky této části kódu znamenají, že pokud je kamera otevřena a program čte obraz, tak dokola pokračuje v daném `while` cyklu a z proměnné `frame` můžeme obraz buď zobrazit nebo uložit jednu fotografii právě pomocí funkce `cv2.imwrite()`, kam se zadá cesta do námi zvolené složky pro ukládání fotografií a proměnná `frame`, která programu řekne odkud má brát obraz. V této části: `"…/fotografie_%d.jpg" % k` je skriptu řečeno, aby právě každý obrázek ukládal s jiným očíslováním, které se mění v proměnné `k`. Proto se v kódu hned za uložením obrázku přičte do `k` jednička. Následně se vypne kamera příkazem `cap.realease()` a tím se i ukončí `while` cyklus.

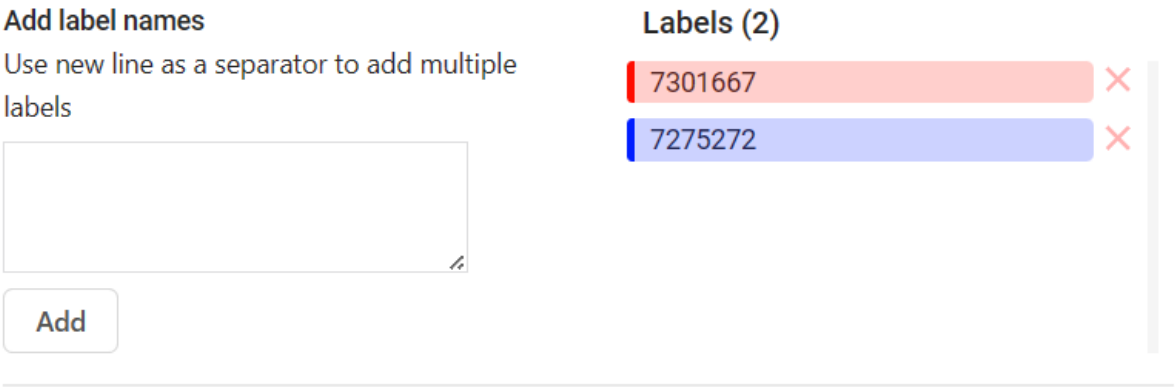
Python v dalším kroku pošle informaci do DS, obdobně jako v předešlém kroku, že vyfotil fotografii. Jakmile DS přijme tuto zprávu, Cobot se vrátí na pozici `place_part`, vezme díl a pošle zprávu Pythonu o uchopení dílu. V tento moment je segment `for` cyklu na konci a vrací se znovu na začátek a tato smyčka se opakuje přesně tolikrát, kolikrát zadáme.

Tímto automatizovaným procesem Cobot s Pythonem vytvoří dataset pro jeden a následně druhý díl. Nechal jsem vytvořit 50 fotografií pro každý díl.

3.1.2 Anotace dílů v Label Studiu

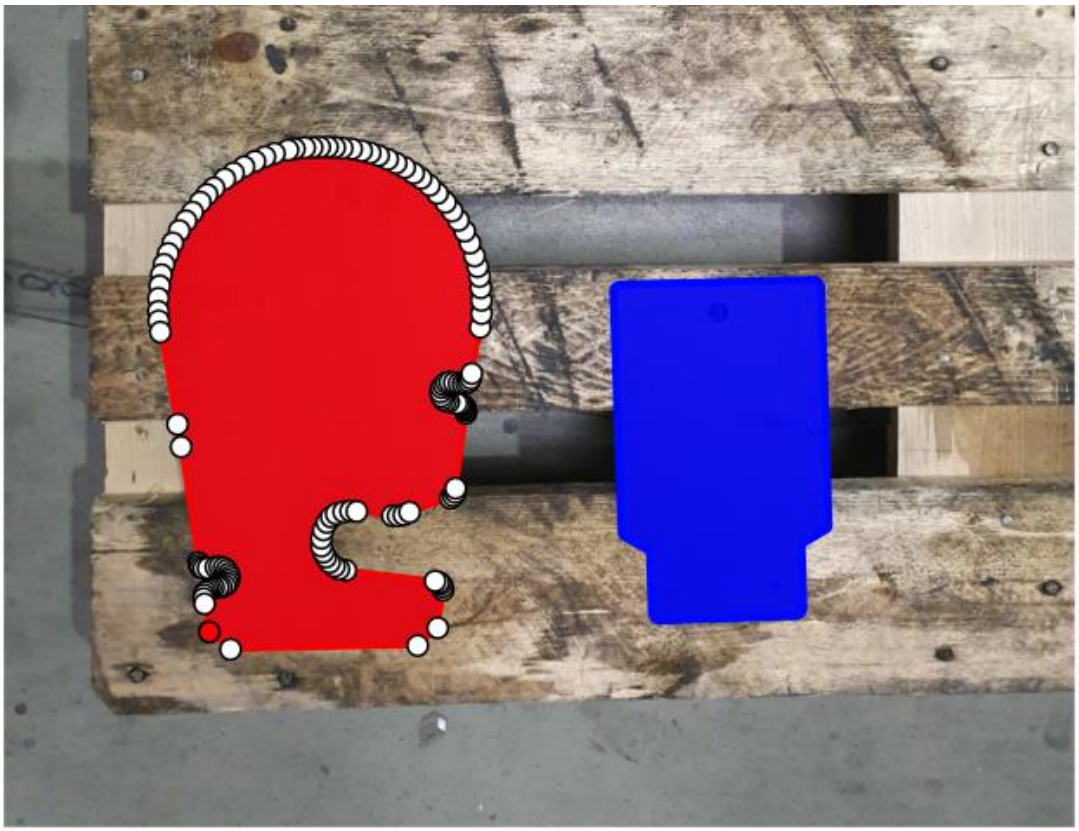
Dalším krokem je práce v LS. Pro instalaci stačí do CMD zadat příkaz `pip install label-studio`. Za malou chvíli se "aplikace" stáhne a můžeme ji v CMD spustit příkazem `label-stuio`. Po krátkém načítání se otevře okno internetového prohlížeče s úvodní stránkou LS. Pokud je aplikace používána poprvé, je třeba se zaregistrovat a začít pracovat na anotaci fotografií.

Nejprve musíme založit nový projekt, naimportovat vytvořený dataset, v tomto případě celkově 100 fotografií, a zvolit správnou šablonu pro anotování fotografií. Pro tuto aplikaci jsem zvolil šablonu „Semantická segmentace s polygony". V nastavení šablony se zadají názvy dílů, jako je vidět na Obrázku 9, aby následně vytvořený model byl schopen rozeznat díly od sebe a určit správně jejich třídy neboli názvy.



Obrázek 9: Nastavení názvu dílů v LS

Celá procedura s označováním všech dílů je časově velmi náročná. Pro znázornění je na Obrázku 10 ukázáno označení body jednoho dílu. V případě že těchto dílů musíte udělat například sto, se celý proces může protáhnout na několik hodin. Byla tu vidina urychlení tohoto procesu pomocí Segment Anything Modelu (dále jako SAM), ale to bohužel nešlo kvůli formátům, které SAM exportuje. Více jsem se o tomto modelu rozepsal v teoretické části.

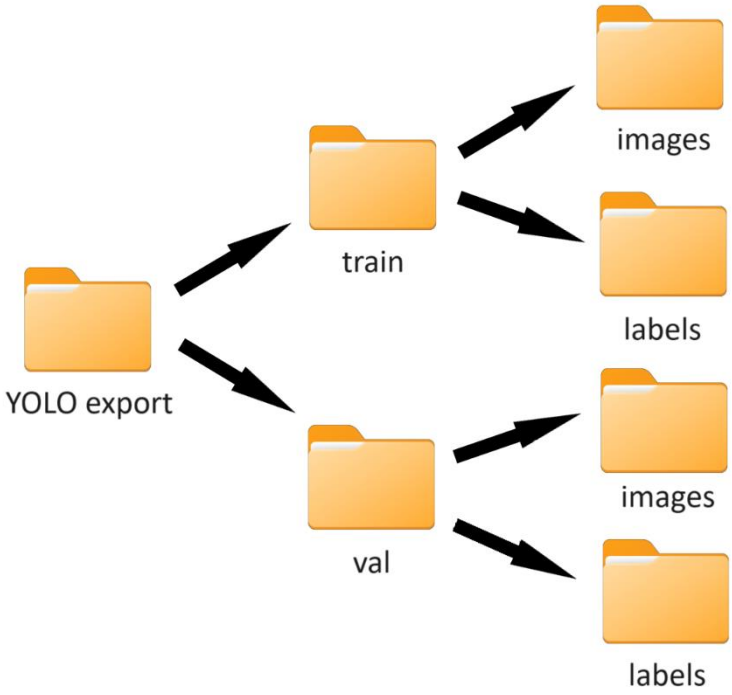


Obrázek 10: Zvýraznění všech bodů na oanotovaném dílu

3.1.3 Úprava vyexportované složky z LS

Po dokončení anotace všech fotografií dílů se vše vyexportuje ve formátu YOLO. Do počítače se uloží zazipovaný soubor, ve kterém jsou dvě složky, jeden textový dokument a jeden „json“ dokument. Dokumenty v sobě mají informace o třídách anotovaných dílů a jejich názvů, například `0: 7301667`, takže třída 0 je pro díl „A“. Jedna složka pod názvem „images“ má v sobě všechny fotografie a druhá složka „labels“ obsahuje textové dokumenty a každý odpovídá názvem fotografii, ke které patří. V těchto dokumentech jsou pro každý díl na fotografii body námi označeného polygonu neboli souřadnice x a y bílých bodů na Obrázku 9. Na začátku řádku s body je vždy uvedena třída dílu (např. 0, 1…).

Tento stažený soubor je potřeba upravit pro správnou tvorbu modelu. Musí se zde vytvořit dvě nové složky „train“ a „val“ a v každé z této složce vytvoříme ještě složky „images“ a „labels“. Fotky v původní složce „images“ rozdělíme v poměru 4:1, takže mých 100 fotek jsem rozdělil na 80 a 20. Osmdesát fotografií umístíme do složky „images“ v „train“ a dvacet jsem přesunul do „images“ ve „val“. Pro lepší pochopení jsem uspořádání složek znázornil ve schématu na Obrázku 11. Textové dokumenty se souřadnicemi zkopírujeme všechny do složek „labels“. Nyní můžeme smazat původní složky, které se stáhly po exportu.



Obrázek 11: Schéma uspořádání složek pro tvorbu modelu na rozpoznávání dílů

3.1.4 YAML dokument

Data pro tvorbu modelu jsou připravena, nicméně je ještě třeba napsat krátký textový dokument s informacemi o umístění těchto dat v počítači a o třídách a názvech dílů. Tento dokument vypadá následovně:

```
path: C:\Users\patrikbrejla\PycharmProjects\label-studio\data
train: train\images
val: val\images
nc: 2
names:
    0: 7301667
    1: 7275272
```

Do proměnné `nc` (neboli number count) se zadá počet dílů a do `names` přiřadíme názvy dílů ke konkrétním třídám. Po napsání tohoto kódu uložíme dokument s koncovkou „.yaml“ (v mém případě jako „`segmentation.yaml`“), protože nechceme klasický textový dokument. Soubor YAML má totiž zjednodušenou syntaxi, používá značení pro různé datové typy a je nutný pro tento krok. Je tedy navržen tak, aby byl snadno čitelný jak pro lidi, tak pro stroje.

3.1.5 Tvorba modelu pro rozeznávání

Na oficiálních stránkách knihovny YOLO stáhneme předtrénovaný model od Ultralytics. Máme na výběr z více možností, které se liší podle preciznosti učení modelu - nano, small, medium, large, extra large (viz Obrázek 12), tím pádem volíme podle požadavků na kvalitu modelu. Já zvolil variantu small (`yolov8s_seg.pt`), protože model bude rozeznávat v podstatě 2D obrazce položené na stole nebo paletě. Při výběru large nebo extra large modelu by mohlo dojít k přetrénování modelu, a to by pro výsledný výstup nebylo přívětivé.

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

Obrázek 12: Seznam předtrénovaných modelů od Ultralytics [21]

Ná stránkách pro YOLO segmentaci nalezneme kód do Pythonu pro natrénování a vytvoření modelu. Tento krátký skript jsem zkopíroval a vložil do PyCharmu, kde jsem ho upravil pro mé data a soubory. Aby vše fungovalo, je třeba nainstalovat do Pythonu knihovnu YOLO od Ultralytics zadáním příkazu `pip install ultralytics-yolo` do CMD.

Skript pro natrénování modelu:

```
from ultralytics import YOLO

model = YOLO(„yolov8s_seg.pt“)

model.trian(data=“segmentation.yaml“, epochs=50, batch=16, imgsz=640)
```

Prvním krokem je naimportování knihovny YOLO. Poté se načte předtrénovaný model a uloží se do proměnné `model`. Příkazem `model.train()` začne učení modelu. Je třeba zde do parametru `data` zadat napsaný yaml dokument, `epochs` nastaví kolikrát má program projet trénovací dataset a příslušné anotační soubory, `batch` neboli velikost dávky určuje, kolik obrázků bude model najednou zpracovávat, `imgsz` je parametr pro velikost vstupních obrázků, kterou jsem nastavil na 640 z důvodu zkrácení času trénování a zároveň vyhovující účinnosti při rozeznávání.

Během učení můžeme sledovat v jaké epoše program zrovna je a jestli vše probíhá správně, jako na Obrázku 13.

Obrázek 13: Průběžné výstupy při učení modelu v Pythonu

Po dokončení tohoto programu se ve složce projektu Pycharmu vytvoří nová složka s názvem „runs“, kde můžeme najít námi vytvořený model pro rozeznávání dílů. Konkrétně se model nachází zde: runs\segment\train\weights\best.pt. Soubor „best.pt“ je tedy finální výsledek tohoto procesu a může se použít v hlavním programu v Pythonu pro rozeznání dílů, o tomto budu psát v další kapitole.

3.2 Python kód pro rozeznání a lokalizaci dílů

V této kapitole popíši celý skript pro rozeznávání a lokalizaci dílů v Pythonu. Budu popisovat postupně důležité segmenty kódu, aby bylo vše srozumitelné a jasné.

3.2.1 Import potřebných knihoven

Před samotným načtením knihoven do programu musím všechny knihovny nainstalovat do Pythonu v CMD pomocí příkazu `pip install <název_knihovny>` a stáhnout je do projektu v Pycharmu. Knihovny stáhnu v nastavení Pycharmu: Settings\Project\Python Interpreter, a naimportuji je do skriptu pomocí funkce `import` následovně:

```
from ultralytics import YOLO

import cv2

import socket

import numpy as np

from shapely.geometry import Polygon

import time

from shapely.geometry import MultiPoint

import math

from math import atan2, degrees
```

Funkce jednotlivých knihoven jsem vysvětlil v teoretické části bakalářské práce. Neuvedl jsem tam ještě knihovny `time`, ta mi umožňuje pozastavit program po určitý čas příkazem `time.sleep(počet_sekund)`, a `math`, ze které využiji `atan2` a `degrees`, které využiji při výpočtu úhlu natočení dílů.

3.2.2 Připojení socket klienta k serveru

Pro přenos informací do DS musím stejně jako u tvorby fotografií do datasetu připojit socket klienta v Pythonu k serveru v DS a uložím jej do proměnné `s`.

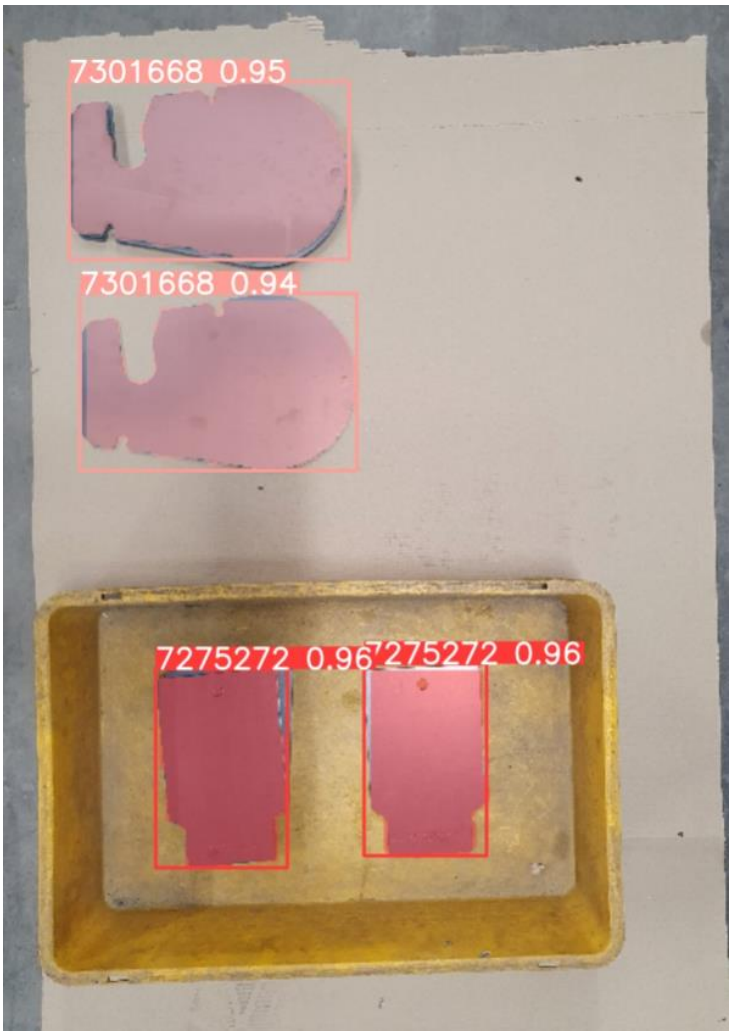
```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("IP_adresa", port))
```

3.2.3 Spuštění a propojení kamery s modelem

Vytvořený model uložím do proměnné `model` tak, že do parametru funkce `YOLO` zadám cestu v počítači k modelu `best.pt`. Jelikož obraz z kamery budu po každém rozeznání dílu vypínat a poté znovu zapínat, tak už tady začnu s `while` smyčkou, aby program neustále běžel a nevypnul se po rozpoznání prvního dílu.

```
model = YOLO(r"C:\...\best.pt")
while True:
        cap = cv2.VideoCapture("http://10.52.97.161:8080/video")
        while cap.isOpened():
              success, frame = cap.read()
              if success:
                    results = model(frame)
                    annotated = results[0].plot()
                    video = cv2.resize(annotated, (860, 540))
                    cv2.imshow(“YOLO segmentace“, video)
```

Spuštění kamery a uložení do proměnné jsem naprogramoval stejně jako v tvorbě datasetu. V proměnné `results` dám do parametru funkce `model` výstup z kamery `frame`. Tímto příkazem se obraz zanotuje a rozezná všechny prvky co jsou v modelu. Pro vizualizaci výsledků vytvořím novou proměnnou `annotated`, která nám poskytne snímek s výsledky, kde model ohraničí rozeznaný díl obdélníkem a připíše k němu název dílu, jako je ukázáno na Obrázku 14, ten zobrazím příkazem `cv2.imshow()`, popřípadě mohu upravit jeho velikost pomocí `cv2.resize()`. Upravení velikosti obrazu je pouze pro účely vyhovujícího zobrazení na obrazovce, nikoliv aby se změnily vlastnosti kamery nebo ukládané fotografie.



Obrázek 14: Rozpoznání dílů vytvořeným modelem, u každého dílu je napsáno jeho číslo a jistota, kterou model díl poznal

3.2.4 Výtah důležitých dat z výstupu segmentace

Pomocí výsledků v proměnné `results` můžeme nejen vizualizovat výsledek segmentace, ale také dostat výstup dat a dále s nimi pracovat v proměnných. Na oficiálních stránkách YOLO je dokumentace s informacemi, jak tato data získat.

Tato data dostaneme ve formátu tensor, to je základní stavební prvek pro reprezentaci dat neboli n-rozměrné pole s libovolným počtem dimenzí. Tyto výsledky musíme tedy převézt na jiný datový typ pro lepší manipulaci s nimi. Část kódu pro získání výsledků segmentace vypadá následovně:

```
for i in results:
    classes = i.boxes.cls
    confidences = i.boxes.conf
    masks = i.masks.xy
```

Pro můj projekt mi stačí tato data, kdybych je vytisknul pro Obrázek 13, dostal bych toto:

```
classes: tensor([0., 0., 1., 1.])
confidences: tensor([0.9640, 0.9580, 0.9500, 0.9436])
masks: [array([800.1, 2545], …, [1045, 2545]], dtype=float32), array...]
```

Do proměnné `classes` uložím třídy rozpoznaných dílů. Vím tedy, že třída `0` je díl „A“ a třída `1` je díl „B“. V `confidences` mám uložené jistoty, se kterými program rozpoznal všechny díly a do `masks` se uloží všechny body polygonů jednotlivých dílů. Jsou to velké matice, proto jsem je zde zobrazil ve velmi zkrácené verzi. Je dobré, že všechna data se ukládají postupně pro jednotlivé díly, takže s nimi mohu pracovat kontinuálně po dílech, a to v dalším `for` cyklu:

```
h = 0
for name, confidence, mask in zip(classes, confidences, mask):
     h += 1
     if confidence >= 0.85:
         ...
     else:
         print(f“Chyba v rozeznání dílu {h}“)
         continue
```

Tento cyklus se pro uvedený příklad opakuje tedy čtyřikrát a vždy s daty pro jeden díl. Zavedl jsem i podmínku, která program pustí dál pouze tehdy, když je jistota větší než 85 %. Při nesplnění této podmínky program napíše do konzole „Chyba v rozeznání dílu {h}“ a pokračuje od začátku příkazem `continue`. Díky proměnné h vím přesně o jaký díl se jedná. S každým dílem se hodnota `h` zvětší o 1 a v f-stringu mi funkce print vytiskne informaci o konkrétním dílu nebo špatně rozeznaném objektu, který zrovna probíhá cyklem.

3.2.5 Získání názvu dílu

Na začátku psaní této části kódu si zadefinuji funkci `socket_send()` pro posílání zpráv přes socket do DS. Jinak pokračuji za posledním `for` cyklem, který jsem uvedl v předešlé podkapitole.

```
def socket_send(msg):
   msg = bytes(str(msg), “utf-8“)
   s.send(msg)
```

Vždy když budu chtít odeslat zprávu, stačí vyvolat funkci `socket_send()` a ušetřím v kódu pár řádků pro lepší přehlednost.

Hlavním cílem této části je vytáhnout z výsledků název dílu, vypočítat jeho těžiště a úhel natočení, aby na sebe díly mohl Cobot pokládat nebo s nimi manipulovat se stejnou orientací.

Získání čísla dílu je jednoduchý úkol za použití podmínky `if`:

```
if int(name) == 0:
   part = 7301667
elif int(name) == 1:
   part = 7275272
socket_send(part)
```

Vycházím tedy z očíslovaných tříd z anotace dílů. Až se v programu bude nacházet více dílů, pouze budu rozšiřovat tento řetězec podmínek. Jakmile se uloží číslo dílu pro proměnné, informaci pošle do DS.

3.2.6 Výpočet těžiště dílu

Další částí je výpočet těžiště dílu. Momentálně jsou v systému díly s velkou plochou bez otvorů. Pro díly s otvorem nebo například ve tvaru L bych souřadnice středu jednoduchou geometrií poupravil pro správné místo uchopení.

Takto vypadá část skriptu pro získání těžiště:

```
points = np.array(maska, np.int32)
polygon = cv2.polylines(annotated, [points], True, (255, 0, 0), 3)
center = list(Polygon(maska).centroid.coords)
```

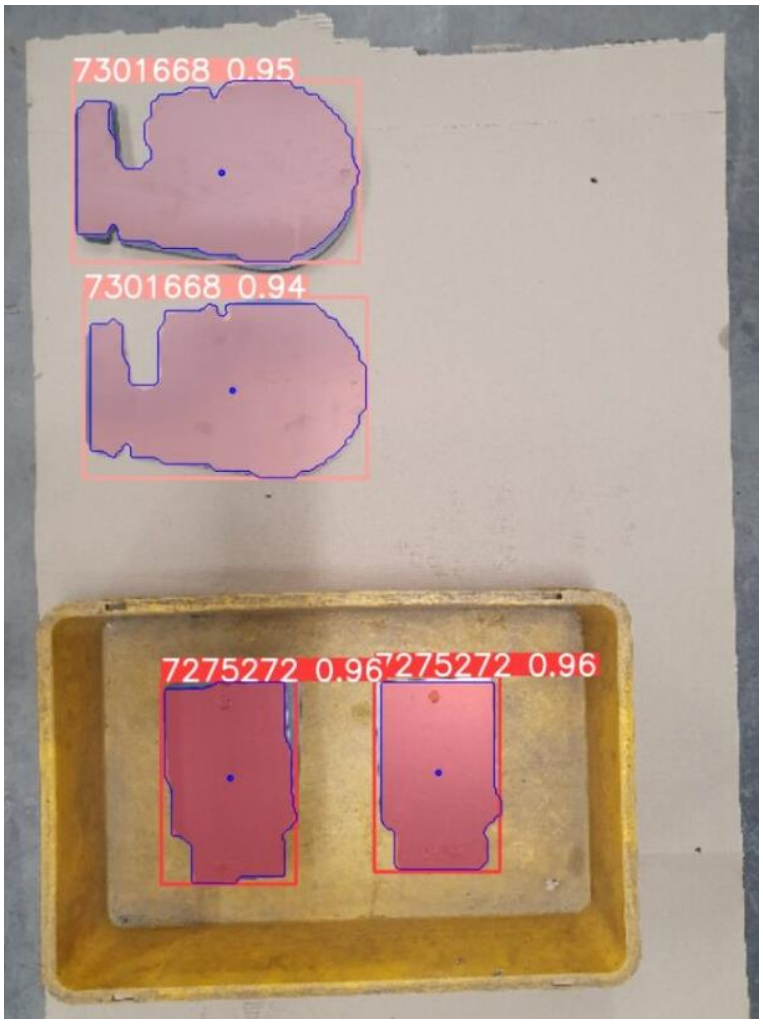
Příkazem `np.array()` konvertuji datový typ `maska` na pole numpy s datovým typem `np.int32`, to je obvykle nutné pro následující funkci z OpenCV pro vizualizaci polygonu, která tyto body polygonu vykreslí a propojí v proměnné `polygon` a tím přepíše výstup z obrazu uložený v `annotated`. Tento krok není nutný pro funkčnost celého programu, je to pouze pro ukázku toho, co se postupně v programu odehrává. Důležitá část nastává v dalším řádku, kdy využiji funkce Polygon z knihovny Shapely a do `center` se uloží souřadnice středu polygonu neboli dílu. Kdybych hodnoty vytisknul v takovémto tvaru, dostanu toto:

```
center: [(848.2865849233475, 2903.4495414337484)]
```

Je vidět, že střed to našlo správně, jenom je třeba hodnoty upravit do správného tvaru, abychom je mohli zobrazit a poslat do DS. Toto provedu v těchto krocích:

```
center_coords = center[0]
center_x = int(round(center_coords[0], 0))
cetner_y = int(round(center_coords[1], 0))
center_array = (center_x, center_y)
circle = cv2.circle(polygon, center_array, (8), (255, 0, 0), 8)
```

Prvním řádkem získáme souřadnice bez hranatých závorek. Dále vytáhnu x a y souřadnici, zaokrouhlím jej na celé číslo a převedu na datový typ `int`. Pro zobrazení pomocí OpenCV si souřadnice uložím do vektoru `center_array` a funkcí `cv2.circle()` jej vytisknu do obrázku. Tento výstup je ukázán na Obrázku 15.



Obrázek 15: Zobrazení polygonů (modrý okraj) a středů dílů (modrý bod)

Informace o souřadnicích středu dílu už je připravena na poslání pomocí vytvořené funkce `socket_send()`. Nejdříve k tomu musím poslat ještě délku těchto číslic, ze stejného důvodu jako při tvorbě datasetu pro LS.

```
socket_send(len(center_x))
socket_send(len(center_y))
socket_send(center_x)
socket_send(center_y)
```

3.2.7 Výpočet úhlu natočení dílu

Jako základ pro výpočet úhlu natočení využiji metodu z knihovny Shapely, která mi vypočte čtyři body obdélníku označujícího nejmenší plochu dílu. Nevyznačí to tedy červený obdélník jako vytvořený model, ale pootočený obdélník o stejný úhel jako díl.

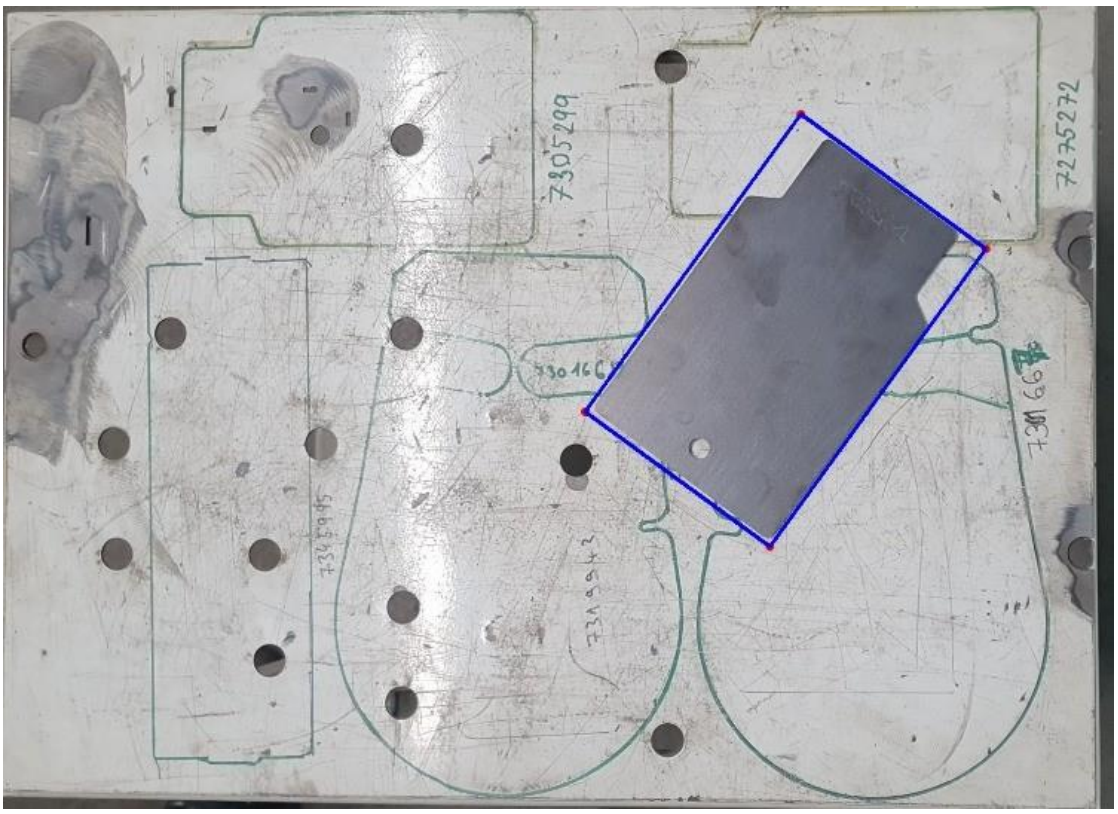
```
rectangle = MultiPoint(points).minimum_rotated_rectangle
x, y = rect.exterior.coords.xy
```

Použil jsem již zadefinovanou proměnnou `points` jako argument pro zadání bodů. Dostanu pět souřadnic pro x a y. Jelikož se jedná o obdélník, měly by body být čtyři, ale metoda `minimum_rotated_rectangle` je implementována tak, že vytvoří bodů pět. Ničemu to ale nevadí, jenom jsou tam dva body stejné. Pokud všechny body vytisknu, data vypadají následovně:

```
x: array('d', [752.627027027027, 1031.9351351351352, 1270.4270270270272,
991.1189189189191, 752.627027027027])
```

```
y: array('d', [523.6378378378379, 139.5891891891892, 313.0378378378379,
697.0864864864866, 523.6378378378379])
```

Postupný proces výpočtu budu ukazovat na fotografii jednoho dílu, aby bylo vše lépe vidět. Vzniknul tedy modrý obdélník viditelný na Obrázku 16.



Obrázek 16: Vizualizovaný obdélník ohraničující nejmenší plochu dílu

Jelikož souřadnice bodů na fotografii mají počátek v levém horním rohu, mohu s jistotou uložit do proměnné bod nejvíc vpravo a nejníže. Ten vpravo bude mít vždy největší hodnotu pro osu x a ten dole pro y. Body ale s postupným otáčením dílu mohou měnit své indexy v proměnných, které jsem ukázal výše, proto budu volit tyto dva body jako optimální k dosažení vždy stejného postupu ve výpočtu a získám je použitím funkce `max`:

```
max_x = int(max(x))
```

```
max_y = int(max(y))
```

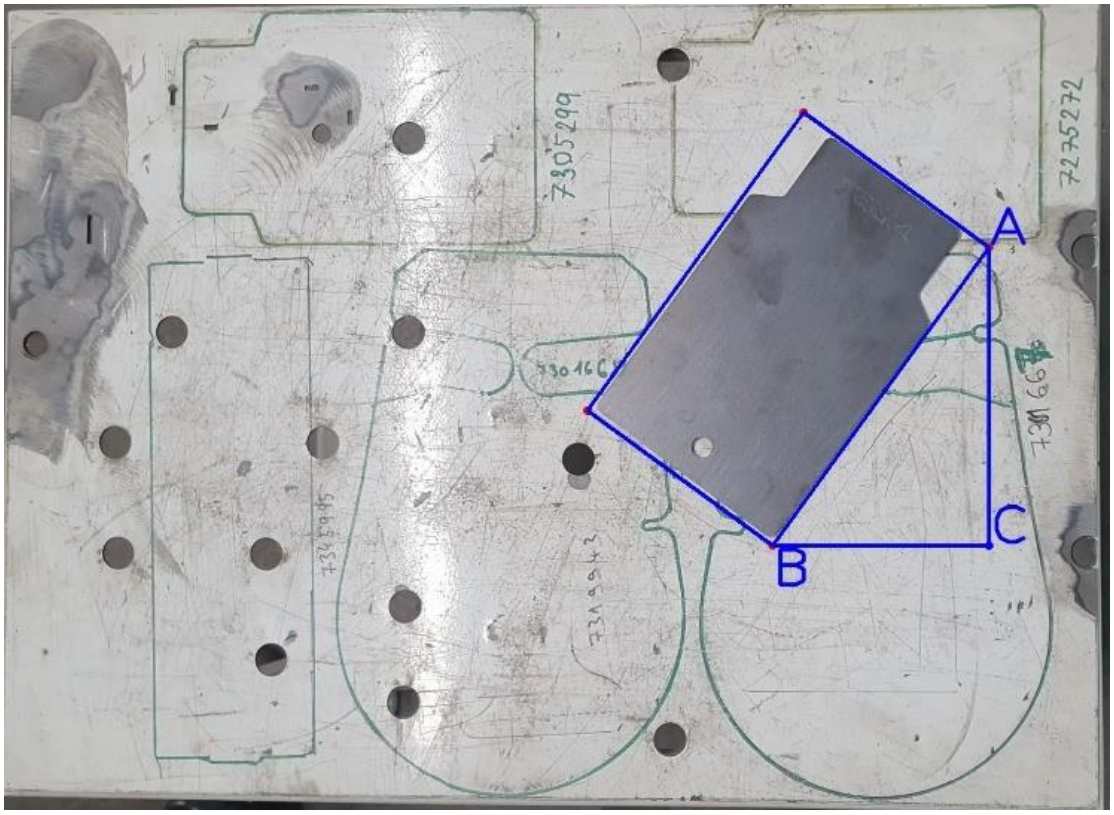
Uložím si do proměnných tyto dvě maximální hodnoty a v následujícím `for` cyklu jednoduchými podmínkami `if` a `elif` k nim přiřadím druhou hodnotu, tímto krokem budu mít v proměnných body A a B.

```
for xx, yy in zip(x, y):
   if int(xx) == max_x:
         point_A = (int(xx), int(yy))
   elif int(yy) == max_y:
         point_B = (int(xx), int(yy))
```

K těmto dvěma bodům vytvořím třetí bod C potřebný pro výpočet. Jeho souřadnice budou vždy maximální hodnota x a maximální hodnota y. Bod C tedy mohu do proměnné uložit takto:

```
point_C = (max_x, max_y)
```

Vznikne mi tím trojúhelník se stranami o známých délkách a budu moct vypočítat úhel natočení pomocí funkce arkustangens. Trojúhelník ABC je znázorněn na Obrázku 17.



Obrázek 17: Ukázaný trojúhelník ABC pro výpočet úhlu natočení dílu

Délky stran zjistím funkcí `dist` z Pythonovské knihovny Math. Finální úhel zjistím pomocí již zmíněné matematické funkce arkustangens a následně jej převedu z radiánů na stupně funkcí `degrees` z té samé knihovny:

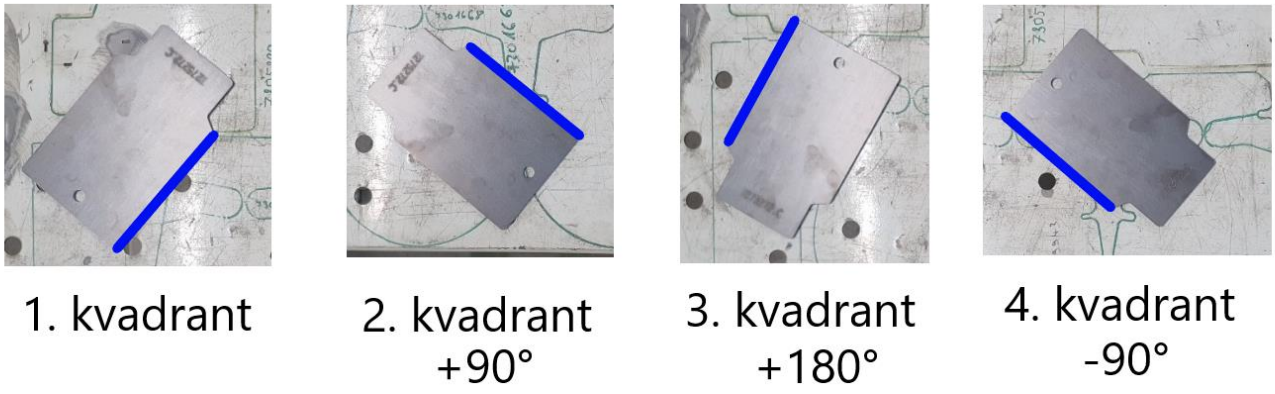
```
len_AC = math.dist(point_A, point_C)
len_BC = math.dist(point_B, point_C)
angle = degrees(atan2(len_AC, len_BC))
```

Úhel vyšel pro tuto fotografii 54°, což odpovídá a tím mohu prohlásit výpočet jako úspěšnou operaci. Pokud by ale díl byl natočen jinak, úhel otočení by odpovídal jiné straně.

Robot by ve finále díl podle úhlu otočil, ale mohlo by dojít k rozdílu 90, 180 nebo 270 stupňů. Proto si pootočení dílu rozdělím na čtyři kvadranty. Podle poměru stran a vzdáleností jejich středu od těžiště dílu zjistím, v jakém kvadrantu se díl nachází, abych mohl upravit úhel o již zmíněný počet stupňů. Této problematice se věnuji v další podkapitole.

3.2.8 Určení kvadrantu dílu a úprava konečného úhlu

Díly jsem do čtyř kvadrantů rozdělil postupně po 90° pootočení podle jedné strany, pro kterou jsem prováděl výpočet. Pro lepší pochopení jsem ukázal otočení dílu a původní stranu vyznačil modře na Obrázku 18.



Obrázek 18: Rozdělení dílu do čtyř kvadrantů podle jeho natočení

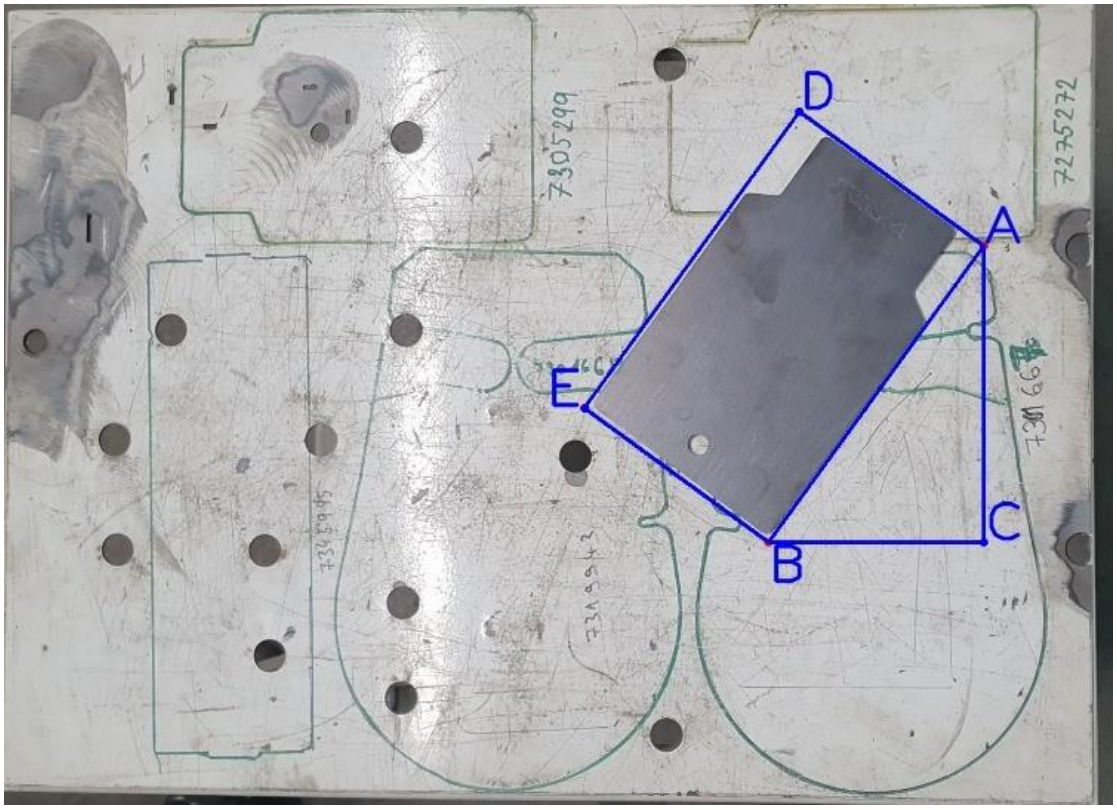
Pro správné určení kvadrantu jsem se rozhodl nejdříve zjistit přibližnou orientaci dílu poměry stran a poté porovnáním vzdálenosti středu dílu od vzdálenější hrany. Jelikož díly v drtivé většině budou mít nějaké zúžení nebo nepravidelný tvar, tak mohu s jistotou pracovat s tím, že vždy jedna strana bude ke středu blíž než druhá a tím odvodím, v jakém konkrétním kvadrantu se díl nachází.

Pro začátek ještě musím dostat dva zbývající body do proměnných. Začnu bodem D nacházejícím se nalevo proti směru hodinových ručiček od bodu A. Získám ho oindexováním všech souřadnic obdélníku a jakmile se nějaké hodnoty rovnají bodu A, tak bod D bude mít o 1 menší index. V kódu tento krok vypadá následovně:

```
for h in range(len(x)):
    D_x = int(x[h])
    D_y = int(y[h])
    coord = (D_x, D_y)
    if coord == point_A:
        point_D = (int(x[h - 1]), int(y[h - 1]))
```

Poslední bod E získám jednoduchou podmínkou. Jakmile se souřadnice nerovnají ani jednomu z bodů, tak právě ten bude hledaný bod E:

```
for E_x, E_y in zip(x, y):
    if (int(E_x), int(E_y) != point_A and (int(E_x), int(E_y) != point_B
    and (int(E_x), int(E_y) != point_D:
        point_E = (int(round(E_x, 0)), int(round(E_y, 0)))
```

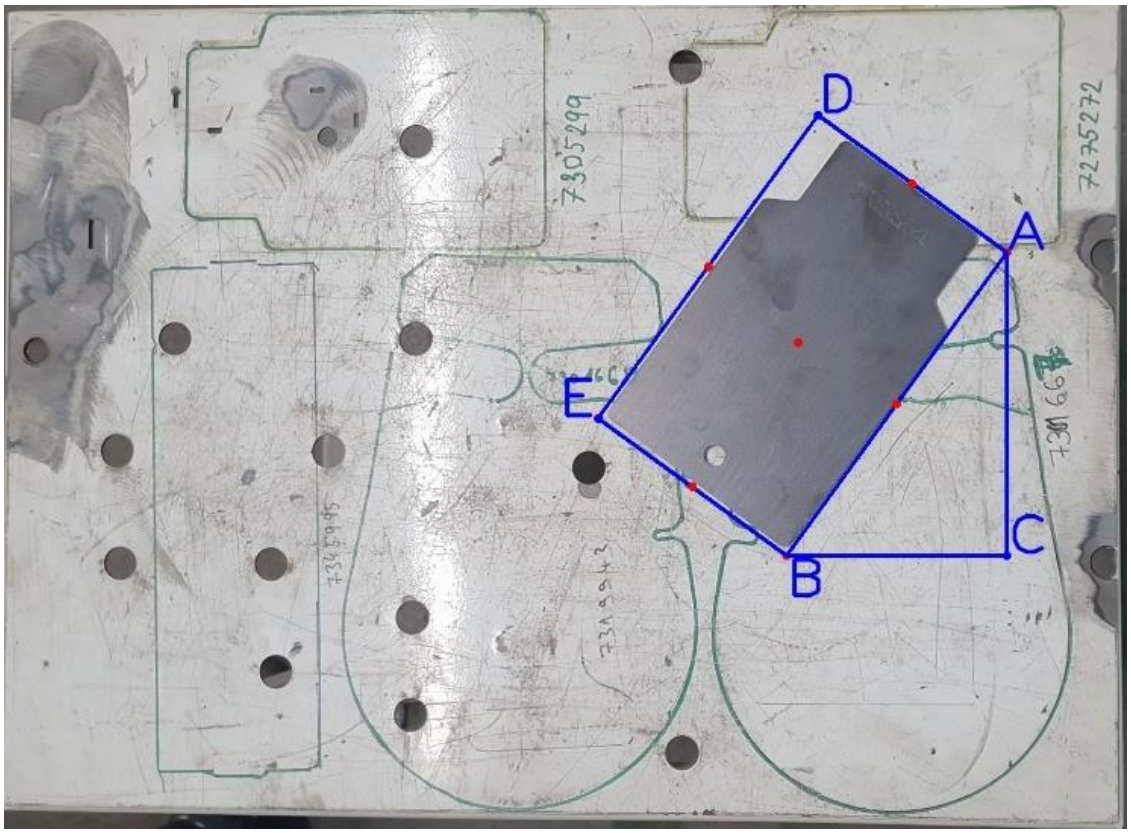


Obrázek 19: Zobrazení všech bodů obdélníku a trojúhelníku při výpočtu úhlu natočení

Po uložení všech bodů, které jsou vidět na Obrázku 19, do proměnných mohu vypočítat body v polovině přímky mezi jednotlivými body. Uvedu příklad pro jednu úsečku, u ostatních budu postupovat obdobně:

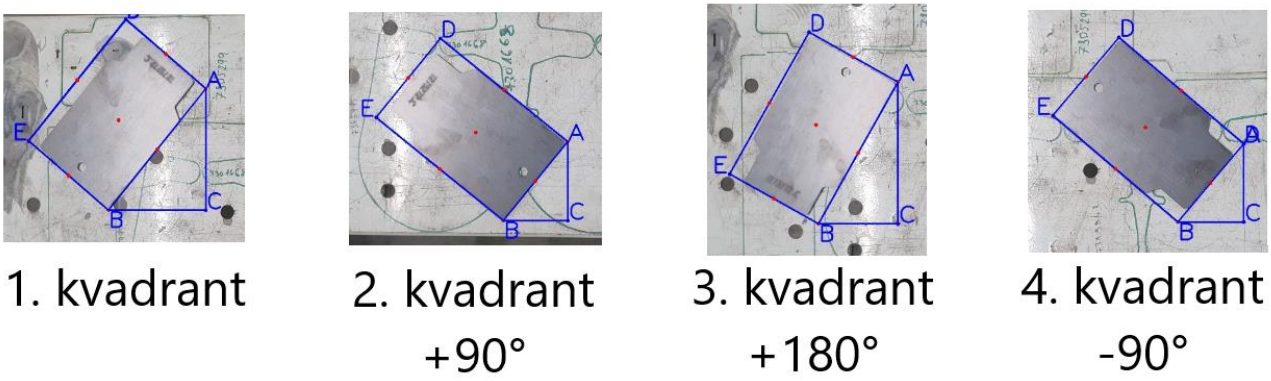
```
AB_x = ((point_A[0] + point_B[0])/2)
AB_y = ((point_A[1] + point_B[1])/2)
middle_AB = (AB_x, AB_y)
```

Pro výpočet středu úsečky AB jsem sečetl souřadnice x bodu A a B a vydělil dvěma, to samé pro y souřadnice a následně i pro ostatní úsečky. Středy úseček jsou červeně zvýrazněny na Obrázku 20. Pomocí příkazu `math.dist()` jsem si změřil vzdálenosti mezi body obdélníka a mezi středem dílu a středu jednotlivých úseček.



Obrázek 20: Zobrazené středy úseček obdélníka kolem dílu pro zjištění, v jakém kvadrantu se díl nachází

Nyní mám všechny potřebné body a vzdálenosti pro zjištění, v jakém kvadrantu se díl nachází. Pokud úsečka AB bude delší než úsečka EB a střed dílu bude blíž ke středu EB a ne DA, tak se jedná o první kvadrant. Kdyby byl střed dílu blíž středu DA, díl by byl ve třetím kvadrantu. Jakmile by úsečka EB byla delší než AB a střed by byl blíž středu AB, šlo by o druhý kvadrant, ve čtvrtém kvadrantu by díl byl, kdyby byl střed dílu blíž středu ED. Pro lepší pochopení všechny varianty ukážu na Obrázku 21.



Obrázek 21: Zobrazení dílu ve všech kvadrantech s popsanými body

Tato metoda, se kterou jsem přišel může u různých dílů fungovat odlišně, jak už jsem zmínil. Při přidání dalších dílů je třeba provézt pár testů a popřípadě upravit střed dílu nebo podmínky při porovnávání vzdáleností. Na konci kódu pošlu úhel do DS pomocí mé funkce `socket_send()` a Cobot se začne přesouvat k dílu a bude pokračovat ve svém programu, který popíši v další kapitole.

Na konci skriptu v Pythonu jsem napsal následující krátký while cyklus, který čeká na přijmutí zprávy od DS, aby vypnul kameru a začal opět úplně od začátku celého kódu.

```
while True:
   msg = s.recv(1024)
   msg = msg.decode(“utf-8“)
   if msg == “ok“:
         cap.release()
         cv2.destroyAllWindows()
         break
```

Používám stejnou while smyčku jako u tvorby datasetu. Jednoduše program v Pythonu čeká, než Cobot dokončí svou část kódu a pošle zprávu „ok“. Jakmile přijme tuto zprávu, vypne kameru, ukončí celý loop, a vrátí se na začátek.

3.3 Kód v Dart-Studiu pro robota

V této kapitole podrobně vysvětlím skript napsaný pro robota v DS. Bude se jednat o vytvoření socket serveru, nastavení domácí pozice, přijímání zpráv z Pythonu a následných pohybech a manipulování s díly na základně přijatých informací.

3.3.1 Vytvoření socket serveru

Abych mohl Python propojit s DS, musím na začátku kódu vytvořit server s konkrétním portem:

```
sock = server_socket_open(port)
```

V DS je tento krok velmi jednoduchý, je pouze důležité znát správný port. Pro zkoušení programu v simulaci je port 20002 a pro fungování na reálném Cobotovi je port 20001.

Pro kontrolu správného vytvoření a připojení serveru si mohu zobrazit status serveru a to následovně:

```
state = server_socket_state(sock)
if state == 1:
   tp_log(“Connected“)
```

Funkce uložená v proměnné `state` mi může dát dvě hodnoty. Hodnota 0 znamená, že server není připojený, hodnota 1 značí správné vytvoření a připojení serveru ke klientovi. Podmínka `if` plní funkci, že pokud je status serveru 1, napíše do konzole „Connected“ a skript pokračuje v dalších krocích.

3.3.2 Nastavení domácí pozice Cobota

Jak už jsem zmiňoval v předešlých kapitolách, tak mám kameru nastavenou levým horním rohem na levý horní roh stolu, abych ulehčil přepočet souřadnic. Z tohoto důvodu nastavím

domácí pozici robota přesně na toto místo také. Z výkresové dokumentace koncového efektoru (savky) jsem si zjistil, že má rozměry 100 x 100 mm. Přiložím efektor robota nejdřív z jedné strany stolu, potom z druhé a uložím si tyto pozice do proměnných. To samé udělám pro výšku stolu. Aby hlava robota byla přesně v rohu stolu, tak od x-ové souřadnice z pravé strany stolu odečtu 50 mm a od souřadnice y z přední strany stolu také. Tímto docílím vycentrování efektoru přesně nad roh stolu a jeho souřadnice vypadají pro tuto konkrétní pozici takto:

```
x_way = posx(-297.24, 369.88, -93.03, 61.52, 179.07, -29.63)

y_way = posx(-395.21, 382, -95.95, 71.45, 178.91, -19.53)

z_way = posx(-528.29, 226.56, -22.55, 61.72, 178.57, -28.7)
```

Tyto pozice získám stisknutím ikony v DS, která mi dá aktuální souřadnice robota v lineárních souřadnicích `posx`. Následně vytvořím původní domácí pozici `home`:

```
home = posx(x_way[0]-50, y_way[1]-50, z_way[2]+120, x_way[3], x_way[4], x_way[5])
```

K z-ové souřadnici jsem přičetl 120 mm, aby Cobot nezačínal přesně na stole, ale měl pod sebou nějaké místo, tím pádem by nemělo dojít k žádné kolizi s díly. Na indexy 3, 4 a 5 jsem uložil stejné hodnoty jako v pozici pro získání x-ové korekce. Tyto souřadnice slouží k natočení hlavy a toto zaručí, že koncový efektor Cobota bude začínat ve vertikální poloze.

3.3.3 Příjem dat z Pythonu

V dalším kroku začnu while smyčkou pro celý zbytek kódu, ze stejného důvodu jako u Pythonu, aby se kód opakoval, dokud na stole budou nějaké díly.

```
while state == 1:

   for k in range(1):
```

Využiji zde proměnné `state`, aby program jel pouze tehdy, když bude otevřený server a jelikož budu pohybovat vždy s jedním dílem, tak jsem zde zavedl `for` cyklus, který se bude opakovat právě jednou. Za touto smyčkou už začnu s příjmem dat z Pythonu, obdobně jako v kapitole pro tvorbu datasetu:

```
res, part = server_socket_read(sock, length = 7)
```

Proměnná `res` má v sobě uložené různé hodnoty, záleží na stavu přijetí zprávy. Pokud zprávu přijme, tak vypíše počet přijatých bytů, jinak by vypsal chyby nebo problém, který nastal. Já tuto proměnnou nepoužívám, ale musí být uvedena ke správnému fungování funkce. Do proměnné `part` se uloží již přijatá zpráva z Pythonu, a to je číslo dílu.

Kód pokračuje již pro konkrétní díly, uvedu příklad pro díl „A“ (7301667):

```
if part == b“7301667“:

   res, len_data = server_socket_read(sock, length = 1)

   len_data = len_data.decode()

   len_data = int(len_data)
```

```
res, data = server_socket_read(sock, length = len_data)
data = data.decode()
data = int(data)
```

Díly si rozdělím podmínkou `if`, tedy pokud se proměnná `part` s uloženou informací o čísle dílu rovná v tomto případě 7301667, tak program vybere tuto část kódu. V podmínce se tato proměnná rovná `b“7301667“`, protože to nemůže být klasický string, ale bytový datový typ. Nejprve přijmu informaci o počtu znaků souřadnice a následně přijmu konkrétní souřadnici. Je tu stejný malý problém s délkou odeslaných informací, jako bylo při tvorbě datasetu. Těchto šest řádků zopakuji třikrát a uložím si přijatá data. Souřadnice x a y v pixelech do proměnných `pixel_x` a `pixel_y` a úhel do proměnné `angle`.

3.3.4 Přepočet souřadnic na milimetry

Jelikož souřadnice dílů posílám v pixelech, musím je přepočítat do milimetrů, abych měl jak získat pozici pro uchopení dílu. Toho dosáhnu přepočtem pixelů pomocí vypočteného koeficientu. Jakmile mi dá Python souřadnice dílu (např. (500, 400)), změřím si reálnou vzdálenost dílu od krajů stolu. V milimetrech je tato souřadnice (248, 186). Koeficient získám vydělením souřadnice v pixelech souřadnicí v milimetrech:

$$K_x = \frac{500}{248} = 2{,}0161290322581$$

$$K_y = \frac{400}{186} = 2{,}1505376344086$$

Pro větší přesnost jsem si stůl rozdělil na dva intervaly (0; 739) a (740; 1480). Pouze s jedním koeficientem by Cobot netrefil střed dílu s vyhovující přesností z důvodu ohnutí obrazu na kameře. Koeficienty jsem si poté uložil do proměnných podle toho, v jakém intervalu se díl nachází použití `if` podmínek:

```
if int(pixel_x) > 739:
    x_koef = 2.148148148148
    y_koef = 2.145098039215
else:
    x_koef = 2,0161290322581
    y_koef = 2.1505376344086
```

Koeficienty jsem schválně nechal nezaokrouhlené, kvůli větší přesnosti. S těmito koeficienty již mohu přepočítat souřadnice na milimetry. Musím si dát ale pozor na orientaci souřadnicových systémů. Kamera a Cobot ho mají totiž v tomto rozpoložení o 90° posunutý, tím pádem souřadnice `pixel_x` je pro robota souřadnice v y ose. Přepočet vypadá takto:

```
mm_y = pixel_x/x_koef
mm_x = pixel_y/y_koef
```

Pro uložení pozice uchopení dílu už pouze vytvořím novou polohu `posx()` s vypočtenými hodnotami a posunu tam robota příkazem `movel()` a uložím si tuto pozici také ve formátu `posj()` příkazem `get_current_posj()`, kterou dále budu moct využít.

```
part_posx = posx(home[0] - mm_x, home[1] - mm_y, z_way[2] + 120,
home[3], home[4], home[5])

movel(part_posx, time = 2)

part_posj = get_current_posj()
```

Výhoda `posj` oproti `posx` je využití v pohybu Cobota, ten si vypočte hladkou dráhu a popřípadě udělá i oblouk v pohybu, takže se nepohybuje lineárně v přímce. Dále to využiji v pohybu z jednoho stolu na druhý, tedy přibližné otočení o 180° celého robota.

3.3.5 Uchopení a manipulace dílů robotem

Jelikož kamera kouká na stůl ze shora, tak nemůže určit kolik dílů na sobě se nachází a tím pádem ani program pro robota nemůže vědět v jaké konkrétní výšce má díl uchopit. Proto využiji funkcí Force a Compliance. Cobot zde využije své senzory na měření působících sil na ramena a mohu ho naprogramovat tak, aby pomalu hledal díl. Jakmile na něj narazí, zastaví se a uloží si tuto polohu. Tento algoritmus výrobce uvádí přímo v programovací příručce pro uživatele a vypadá následovně:

```
q1_dir = [0, 0, 1, 0, 0, 0]

force = [0, 0, -40, 0, 0, 0]

task_compliance_ctrl([500, 500, 500, 100, 100, 100], time = 0.8)

set_desired_force(force, q1_dir, time = 0.5)

force_check = 40

force_condition = check_force_condition(DR_AXIS_Z, max = force_check)

while (force_condition):

   force_condition = check_force_condition(DR_AXIS_Z, max = force_check)

   if force_condition == 0:

         height, sol = get_current_posx()

         break


release_force(0.8)

release_compliance_ctrl()

set_digital_output(1, 1)

movel(get_part, time = 1)
```

V prvních dvou řádcích definuji proměnné pro pohyb v ose Z (index 2) dokud působící síla nebude -40 N. Funkcí `task_compliance_ctrl()` nastavím tuhost jednotlivých ramen

robota a následně funkcí `set_desired_force()` nastavím právě tu kontrolu síly -40 N v ose Z. Aby while smyčka mohla proběhnout, musím ještě zadefinovat proměnnou, která v sobě obsahuje funkci `check_force_condition()`, ta jako výstup může mít buď True nebo False. Mně to vrátí hodnotu True a proto program bude pokračovat do while smyčky. Tam program neustále kontroluje tento status a jakmile se savka robota zastaví na dílu při síle 40 N, tak se status přepne na 0 (False), program si uloží tuto pozici do proměnné `height` a ukončí while cyklus. Následně se vypnou funkce Force a Compliance a Cobot, nacházející se savkou přímo na dílu, přepne digitální výstup savky na 1, tím pádem díl uchopí a následně se vrátí do výšky 120 mm nad stůl.

Pokud by Cobot pracoval ve výrobě, vždy by na sobě bylo více dílů najednou, a k tomu se hodí uložené pozice v různých výškách. Ještě je třeba zadefinovat tloušťku materálu do proměnné `material`. Nyní mohu určit celkový počet dílů na stole tímto výpočtem:

```
material = 3
amount_pieces = (height[2] - z_way[2])/material
amount_pieces = round(amount_pieces, 0)
amount_pieces = int(amount_pieces)
```

V proměnné `height[2]` mám uloženou výšku stolu plus výšku sloupce dílů. V `z_way[2]` je definovaná čistě výška stolu. Tyto dvě hodnoty od sebe odečtu, vydělím tloušťkou materiálu, zaokrouhlím na celé číslo funkcí `round()` a dostanu celkový počet dílců ve sloupci. S touto důležitou hodnotou budu pracovat v další podkapitole.

Cobotovi manuálně zadefinuji polohu kam má konkrétní díl položit. Udělám to funkcí `posj()`, aby robot nekolidoval sám do sebe, protože se musí otočit přibližně o 180°. V lineárním pohybu by tento krok udělal přímočaře a v podstatě by naboural sám do sebe. Touto variantou si vypočte trajektorii ve tvaru půlkruhu a tím díl přenese bez jakýchkoliv kolizí.

```
place_part = posj(174.75, -27.56, -106.69, 1.1, -45.18, -97.56)
```

Jakmile se nad tuto pozici dostane, opět spustím funkce Force a Compliance pro případ, kdyby se na odkládacím místě už nějaké díly nacházely. Jakmile se tyto funkce ukončí, přepnu digitální výstup savky na 0, aby pustila díl a uložím si tuto polohu pro hodnotu výšky odkládacího stolu s možným počtem dílů na místě.

3.3.6 For cyklus pro pokládání dílů

Aby Cobot pracoval rychleji a efektivněji při přemisťování stejných dílů ve sloupci, napsal jsem for cyklus, který tento proces urychlí:

```
for i in range(amount_pieces):
    movej(part_posj, time = 3.5)
    height[2] = height[2] - material
    movel(height, time = 1)
    set_digital_output(1, 1)
```

```
movel(part_posx, time = 0.8)
movej(place_part, time = 3.5)
end_height[2] = end_height[2] + material
movel(end_height, time = 1)
set_digital_output(1, 0)
movej(part_posj, time = 1)
```

V DS mám uložené veškeré výšky a počet dílů. Cobot tedy položí první díl ze sloupce a vrátí se na to samé místo, kde díl vzal. Od této uložené pozice odečte tloušťku dílu a přesune se tam rychleji, protože nepoužije funkce Force a Compliance, nijak se tedy nezpomalí. Díl vezme a přesune ho nad odkládací pozici, kde zase přičte tloušťku materiálu od této uložené polohy. Dílec položí, pustí a pojede pro další díl. Toto se bude opakovat přesně tolikrát, kolik je ve sloupci dílů.

3.3.7 Ukončení programu pro robota

Po dokončení všech kroků v předchozí podkapitole se Cobot přesune zpět do domovské pozice pomocí funkce `movej()` a odešle zprávu „ok“ do programu v Pythonu. Tím mu dá signál, aby začal s hledáním dalších dílů.

```
movej(home_j, time = 4)
msg = "ok"
msg = msg.encode()
server_socket_write(sock, msg)
```

Cobot se přesune na domovskou pozici a program se vrátí opět na začátek a čeká na zprávu s informací o čísle dílu a jeho souřadnicích z Pythonu.

Pokud by Python už žádný díl nenašel, program v DS provede následující krok:

```
elif part == None:
   server_socket_close(sock)
   state = 0
   break
break
```

Jelikož na stole není žádný díl, tak hodnota proměnné `part` bude `None`, tím pádem se kód dostane do této části, kde vypne server a jeho status nastaví na hodnotu 0 a tím ukončí celý while cyklus a tedy i celý program.

4 Závěr

Cílem této práce bylo napsat program v Pythonu pro rozeznávání dílů na paletě a naprogramování robota, aby s těmito díly následně mohl manipulovat.

Pomocí knihovny YOLO jsem vytvořil model pro rozeznávání dílů. Tento model jsem následně nahrál do Pythonu a propojil jej s obrazem z kamery. Pokračoval jsem s vytažením dat rozpoznaného dílu, zjistil souřadnice středu dílu a vypočetl jeho úhel natočení. Tato data jsem z Pythonu pomocí socket serveru poslal do Dart Studia, ve kterém jsem naprogramoval robota, aby přijmul všechny důležité informace o dílu, následně nad díl najel, uchopil ho a přenesl na určené místo.

Má bakalářská práce je projekt na testování kolaborativního robota ve firmě Doosan Bobcat EMEA. Finální úspěšná verze programu má velký význam pro naše oddělení Automatizace, jelikož jsem ukázal, jak robota programovat v Dart Studiu a možnost propojení přes socket s Pythonem. Nyní máme více možností s potenciální implementací Cobota do výroby, například automatizace u ohraňovacímu lisu.

Do budoucna plánuji zařídit novou kameru s vysokým rozlišením, aby rozpoznávání dílů a výsledky z toho byly ještě přesnější. Budu se také snažit vymyslet držák na kameru umístěný přímo na robotovi, aby kameru mohl sám automaticky zkalibrovat. Jako finální krok pro úspěšnou implementaci bych považoval přidat více dílů do tohoto systému a otestovat jej v konkrétním průmyslovém procesu.

5 Bibliografie

[1] Doosan Bobcat EMEA, „bobcatdobris.cz,“ 2024. [Online]. Available: https://bobcatdobris.cz/#historie. [Přístup získán 6. 3. 2024].

[2] Doosan Bobcat, „doosanbobcat.com,“ 2024. [Online]. Available: https://www.doosanbobcat.com/en/about/history. [Přístup získán 6. 3. 2024].

[3] D. E. Roy, Machine vision: theory, algorithms, practicalities, San Francisco: Elsevier, 2004.

[4] P. S. Foundation, „python.org,“ 2024. [Online]. Available: https://www.python.org/about/. [Přístup získán 6. 3. 2024].

[5] H. H. Simon Yuill, „Python,“ 2021. [Online]. Available: https://citeseerx.ist.psu.edu/document. [Přístup získán 6. 3. 2024].

[6] P. Developers, „Pip documentation,“ 3. 2. 2024. [Online]. Available: https://pip.pypa.io/en/stable/. [Přístup získán 9. 4. 2024].

[7] T. OpenCV, „OpenCV,“ 2024. [Online]. Available: https://opencv.org/. [Přístup získán 9. 4. 2024].

[8] J. Howse, OpenCV computer vision with python, Birmingham: Packt Publishing, 2013.

[9] G. Jocher, R. Munavar a E. Ayush, „docs.ultralytics.com,“ 13. 3. 2024. [Online]. Available: https://docs.ultralytics.com/. [Přístup získán 26. 3. 2024].

[10] G. Jocher a A. Exel, „docs.ultralytics.com,“ 1. 3. 2024. [Online]. Available: https://docs.ultralytics.com/tasks/segment/. [Přístup získán 26. 3. 2024].

[11] N. Developers, „numpy.org,“ 2022. [Online]. Available: https://numpy.org/doc/stable/. [Přístup získán 26. 3. 2024].

[12] S. Gillies, „Shapely documentation,“ 2024. [Online]. Available: https://shapely.readthedocs.io/en/stable/. [Přístup získán 9. 4. 2024].

[13] A. Singh, Socket programming with Python, Ajit Singh, 2019.

[14] N. Jennings, „Socket Programming in Python (Guide),“ 2018. [Online]. Available: https://realpython.com/python-sockets/#background. [Přístup získán 9. 4. 2024].

[15] „Label Studio,“ Human Signal, 2024. [Online]. Available: https://labelstud.io/guide/get_started. [Přístup získán 26. 3. 2024].

[16] C. Hoge, „Getting Started Using Segment Anything Model with Label Studio,“ 24. 8. 2023. [Online]. Available: https://labelstud.io/blog/get-started-using-segment-anything/. [Přístup získán 9. 4. 2024].

[17] Doosan Robotics, „Doosan Robotics,“ 2024. [Online]. Available: https://www.doosanrobotics.com/en/Index. [Přístup získán 16. 4. 2024].

[18] P. P. Authority, „Python Packaging User Guide,“ 3. 4. 2024. [Online]. Available: https://packaging.python.org/en/latest/. [Přístup získán 9. 4. 2024].

[19] Doosan Robotics, „Doosan Robotics products,“ 2024. [Online]. Available: https://www.doosanrobotics.com/en/products/series/m1013. [Přístup získán 16. 4. 2024].

[20] HumaRobotics, „HumaRobotics,“ 2024. [Online]. Available: https://www.humarobotics.com/produit/revetement-anti-taches/. [Přístup získán 8. 5. 2024].

[21] Q. Burhan, G. Jocher a Q. Laughing, „Instance Segmentation,“ 8. 5. 2024. [Online]. Available: https://docs.ultralytics.com/tasks/segment/. [Přístup získán 8. 5. 2024].

[22] OpenCV, „OpenCV-Python Tutorials,“ 13. 5. 2024. [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html. [Přístup získán 14. 5. 2024].