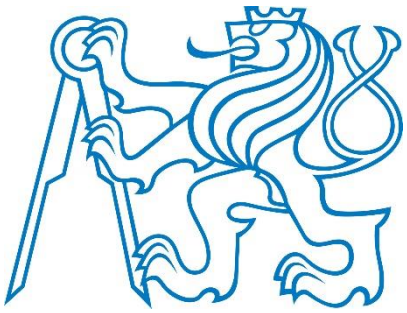
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta strojní – Ústav přístrojové a řídicí techniky



DIPLOMOVÁ PRÁCE

Pracoviště automatizované kontroly výstupu vibračních kruhových zásobníků

Bc. Tomáš Lojík

2017

Vysoká škola: ČVUT v Praze *Fakulta:* strojní

Pracoviště: Ústav přístrojové a řídicí techniky *Akademický rok:* 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE

pro: **Bc. Tomáše LOJÍKA**

obor: Přístrojová a řídicí technika

Název tématu:

Pracoviště automatizované kontroly výstupu vibračních kruhových zásobníků

Název anglicky:

Automated workplace of supervisory control of circle vibratory conveyor's output

Zásady pro zpracování

Navrhněte systém pro automatizovanou kontrolu výstupu vibračního kruhového zásobníku. Systém by měl být obsloužen pomocí počítače s připojenou kamerou a měl by být cenově co nejdostupnější. Systém by měl obsahovat detekci pohybu a orientace dopravované součásti. Součást v ideálním případě prochází zásobníkem po kruhové dráze směrem vzhůru a na výstupu má požadovanou orientaci. Systém musí zaznamenávat chybné průchody součásti a zaseknutí součásti na výstupu. Všechny kritické momenty budou uloženy do videosouborů, aby sloužili k diagnostice zaznamenaných problémů. Součástí výstupu z programu by měl být výpis počtu chyb za sledované období s přiloženou dokumentací. Systém by měl pracovat samostatně a jeho konfigurace na nový druh součásti by měl být co nejjednodušší.

Stručná osnova zadání:

1. Rešerše problematiky rozpoznávání obrazu
2. Návrh variant
3. Detailní rozpracování vhodné varianty
4. Implementace

Rozsah praktické části:

1. Zdrojový kód
2. Složka s podklady pro snadnou instalaci softwaru

Specifikace textové části:

1. technická zpráva
2. min. 50 stran vč. obrázků + přílohy

Seznam odborné literatury:

- RUSS, John C. *The image processing handbook* [online].6th. Boca Raton: CRC Press, 2011. ISBN 1439840458;9781439840450;.
- DAWSON HOWE, Kenneth. *A Practical Introduction to Computer Vision with OpenCV*. 2014. ISBN 1118848454.

Vedoucí diplomové práce: Mgr. Ing. Jakub Jura, Ph.D.

Odborný konzultant Vladimír Balcar (ROX spol. s r.o.)

Datum zadání diplomové práce: 27. 10. 2016

Termín odevzdání diplomové práce: 6. 1. 2017

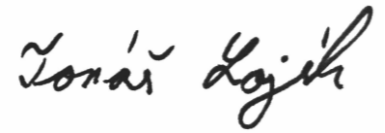
doc. Ing. Jan Chyský, CSc.
vedoucí ústavu

prof. Ing. Michael Valášek, DrSc.
děkan fakulty

V Praze dne: 26. 10. 2016

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci s názvem: „Pracoviště automatizované kontroly výstupu vibračních kruhových zásobníků“ vypracoval samostatně pod vedením Mgr. Ing. Jakuba Jury, Ph.D. s tím, že její výsledky mohou být dále použity podle uvážení vedoucího diplomové práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její spoluautor.



V Praze 6.1.2017 Bc. Tomáš Lojík

Anotační list

Jméno autora:	Bc. Tomáš LOJÍK
Název DP:	Pracoviště automatizované kontroly výstupu vibračních kruhových zásobníků
Anglický název:	Automated workplace of supervisory control of circle vibratory conveyor's output
Rok:	2017
Studijní program:	N2301 Strojní inženýrství
Obor studia:	2301T034 Přístrojová a řídicí technika
Ústav:	12 110, Ústav přístrojové a řídící techniky
Vedoucí DP:	Mgr. Ing. Jakub Jura, Ph.D.
Konzultant:	Vladimír Balcar
Bibliografické údaje:	počet stran 67
	počet obrázků 49
	počet tabulek 3
	počet příloh 4

Klíčová slova: Rozpoznání obrazu, python, vibrační technika, detekce součástí, OpenCV

Keywords: Image recognition, python, vibrating machines, detection of components, OpenCV

Anotace: Tato diplomová práce se zabývá návrhem vhodného systému využívající rozpoznání obrazu pro kontrolu správné funkčnosti vibračního dopravníku.

Abstract: This master thesis describes the design of a proper system that uses image recognition to verify the correct function of the vibrating conveyor.

Obsah

Seznam použitých výrazů

Anaconda2: Balíček Pythonu 2.7 s nejpoužívanějšími knihovnami – jednoduchý na instalaci.

Bitová mapa: Matice, která na místech pixelů uchovává jen 0 nebo 1.

Barvy HSV: Systém značení barev, které využívá místo 3 hodnot barev (RGB) 3 její složky – tón, sytost a jas.

C/C++: Rozšířené programovací jazyky – C++ rozšiřuje C o objektové programování.

Histogram: Grafické znázornění distribuce hodnoty dat pomocí sloupcového grafu.

Java: Nejrozšířenější programovací jazyk dnešní doby.

Kamerový senzor: Kamera, která dokáže nezávisle rozpoznávat obraz např. ve spojení s PLC.

Licence BSD: Licence pro svobodný software.

Mikrovrh: Způsob pohybu, který využívají vibrační dopravníky pro svou funkci.

OPC server: Software, který zprostředkovává komunikaci mezi PC a PLC.

OpenCV: Knihovna pro zpracování obrazu.

Open-source: Počítačový software s otevřeným zdrojovým kódem.

PLC: Programovatelný logický automat.

Pracovní prostor: Oblast, kde program snímá celou scénu.

Prostor pro vyhodnocení: Oblast, kde program snímá součásti a porovnává je se vzorem.

Průmyslu 4.0: Výraz, který se používá pro stále větší rozšíření automatizace ve výrobě.

Python: Programovací jazyk, který byl využit v této práci.

Raspberry Pi: Miniaturní levné PC založené na procesoru ARM, na kterém mohou běžet linuxové distribuce a primitivní OS.

Senzorika: Označení pro čidla a senzory, které se v automatizaci využívají.

Stav OK: Stav, kdy je součást správně zorientována.

Stav NOK: Stav kdy součást není správně zorientována.

Strojové vidění: Obor, který se snaží o rozšíření možností vidění počítačů a jejího přiblížení k lidskému.

SW: Zkratka pro software.

Thresholding: Odstranění přebytečných a nepotřebných informací z obrazu pro lepší možnost zpracování pomocí počítače.

TkInter: Grafické rozhraní integrované v Pythonu.

Vibrační dopravník: Dopravník, který využívá k pohybu dopravovaného materiálu vibrace vytvářené připojeným pohonem.

Vibrační kruhový zásobník (KZ): Druh vibračního dopravníku kruhového tvaru se spirálovitou dráhou.

Seznam použitých veličin

A … plocha soušásti v obraze [pixel^2]

C_x … poloha těžiště shluku součástí v ose X [pixel]

f … hodnota matice na souřadnicích i a j [-]

h_1 … výška jedné součásti [pixel]

g … bit v bitové masce, který značí přítomnost objektu [-]

i … souřadnice obrazu ve svislém směru [pixel]

j … souřadnice obrazu ve vodorovném směru [pixel]

k … počet součástí ve shluku [-]

Mod() … modulo - zbytek po dělení [-]

p … pořadí součásti ve shluku [-]

T … mezní hodnota detekce, nad kterou provedeme ořez obrazu [-]

w_1 … šířka jedné součásti [pixel]

X_P … poloha těžiště p-té součásti v ose X při rozpoznávání shluku součástí [pixel]

x … Gaussova oblast ve vodorovném směru [pixel]

y… Gaussova oblast ve svislém směru [pixel]

Δ … velikost gradientu v obraze při použití Cannyho hranového detektoru [-]

θ … úhel natočení gradientu v obraze při použití Cannyho hranového detektoru [rad]

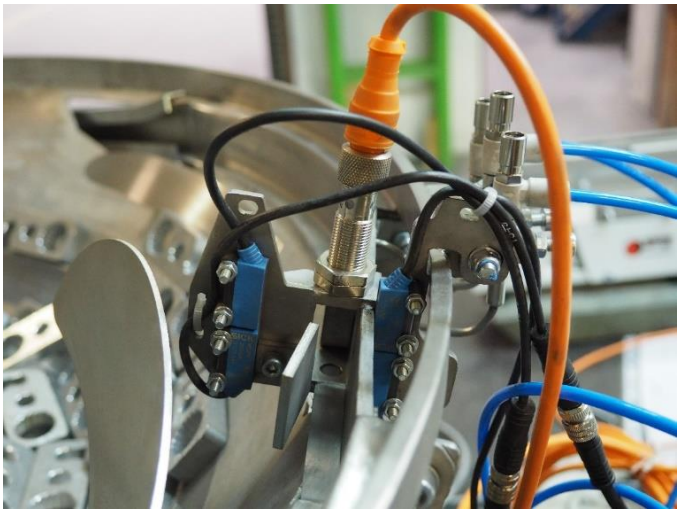
σ…rozptyl – velikost rozostření [pixel]

1. Úvod

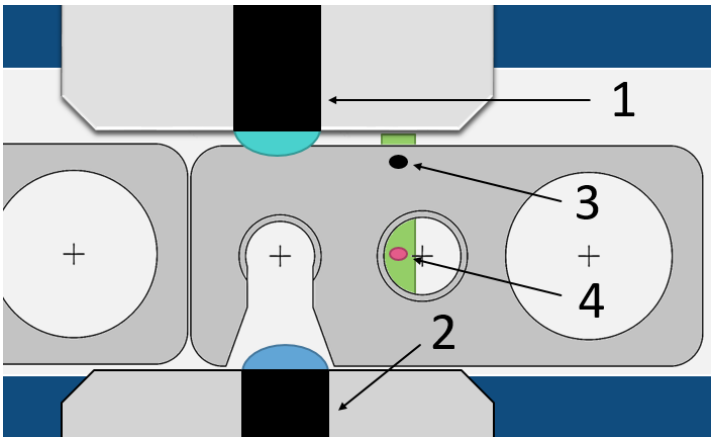
Diplomová práce je zaměřena na návrh, tvorbu a implementaci systému, který by dokázal nahradit lidský faktor v oddělení výstupní kontroly zadávající společnosti, která se zaměřuje na zakázkovou výrobu vibračních zařízení. Mezi tato zařízení patří také vibrační kruhové zásobníky, které dopravují součásti pomocí vibrací na výstup stroje v přesně určeném natočení za pomoci mechanických či senzorických pomůcek. Spolehlivost tohoto řešení musí být ale důkladně kontrolována, než je stroj předán zákazníkovi. Můj systém by měl automaticky kontrolovat spolehlivost jednotlivých vyrobených zařízení tím, že bude sledovat výstup zařízení a kamerou vyhodnocovat orientaci všech součástí vystupujících ze stroje. Zároveň by měl zdokumentovat případné problémy na videozáznam. Celý systém bude naprogramován pomocí Pythonu s knihovnou OpenCV a bude obsluhován na počítači s připojenou kamerou.

1.1. Vznik projektu a návaznost na předchozí práci

V této diplomové práci jsem se rozhodl navázat na mou činnost z předchozího semestru, kdy jsem v rámci Projektu III [1] řešil ve spolupráci s firmou ROX spol. s r.o. rozpoznání orientace součásti, která se pohybovala po kruhovém vibračním dopravníku. V případě detekce nesprávně orientované součásti na výstupu zařízení došlo k jejímu shození zpět do zásobníku pomocí proudu vzduchu. Obrázek 1 zobrazuje fotografii hotového zařízení a na obrázku 2 je znázorněno koncepční schéma, které se skládá ze dvou indukčních čidel (1,2), dvou optických závor (3,4), a můžeme na něm vidět průchod správně orientované součásti zařízením.



1 – Konečná podoba rozpoznávacího místa[1]



2 – Koncept zařízení pro rozpoznání orientace součásti[1]

Zařízení je dnes plně funkční a v provozu, k běhu je použito PLC „LOGO! OBA06“ a k němu připojené výše zmíněné 4 senzory umístěné v zásobníku. Algoritmus v PLC využívá dvou základních částí pro rozpoznání správné orientace. Jedna část využívá otvory součásti – pokud dojde k jedinečné kombinaci sepnutí a rozepnutí čidel, jak lze vidět na obrázku 2 nebo v tabulce T1, zařízení jednoznačně určí, že jde o součást se správnou orientací – nazvěme stavem OK. V opačném případě je součást označena jako špatně orientovaná – NOK. V tabulce T1 níže můžete vidět Karnaughovu mapu s přehledem všech stavů, které mohou senzory zachytit.

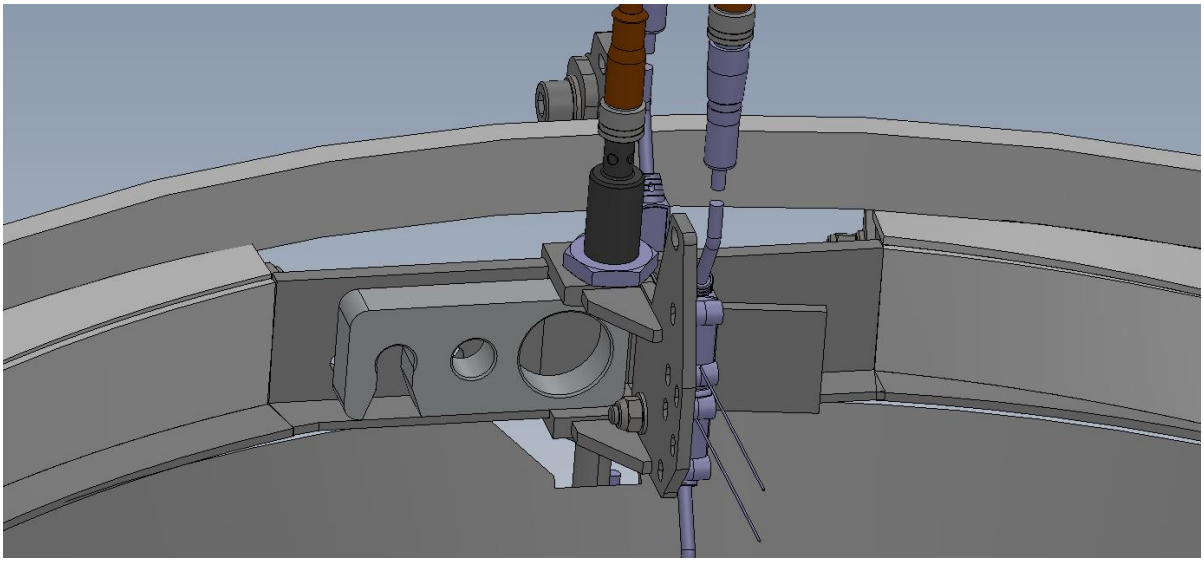
T1 – Karnaughova mapa se znázorněním detekce správné orientace (OK)

12 \ 34	00	01	10	11
00	NOK			
01	NOK			
10	NOK		**OK**	NOK
11	NOK			

Druhá část algoritmu je použita pro eliminaci falešně pozitivních detekcí. Zde spodní indukční a obě optická čidla počítají po příchodu nové součásti počet hran na ni a v případě správného počtu je opět detekována správná orientace. Pokud není, dojde ke shození součásti proudem vzduchu. V případě, že obě části algoritmu neukazují stejný výsledek, dojde k preventivnímu shození.

Během řešení byl kladen důraz na vysokou spolehlivost výsledného řešení, což obnášelo velmi časově náročné testování a pozorování. Řešení bylo nakonec velmi spolehlivé, ale ne

dostatečně. Během jedné osmihodinové směny došlo v průměru k jedné falešně pozitivní detekci a do výstupu stroje se dostal NOK díl. I přesto, že se chybu nakonec podařilo zachytit, trvala diagnostika problému velmi dlouhý čas. Velkým negativem se ukázal být samotný průběh kontrolního procesu, neboť zde byla nutná neustálá pečlivá kontrola pracovníkem, který výstup pozoroval a zároveň pořizoval videozáznam. To vedlo k tomu, že nebylo možné kontrolovat stroj dostatečně rychle, neboť jsou zde omezení pracovní dobou a zároveň i koncentrace během pracovní doby, která s časem pomalu klesá.



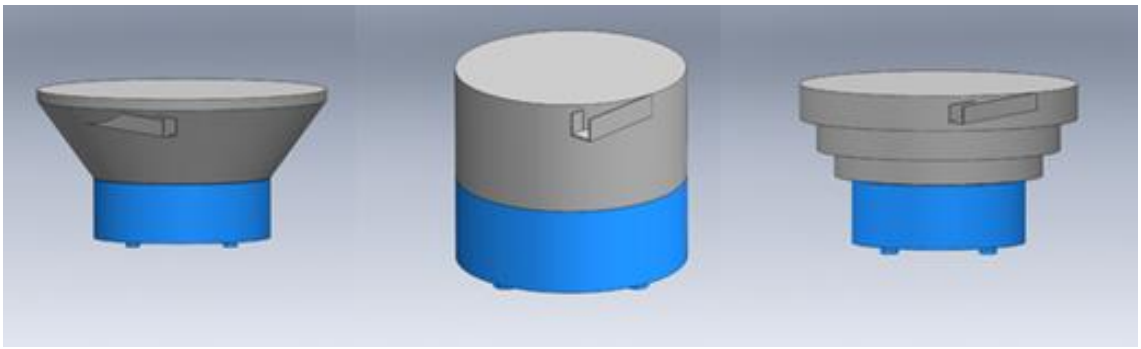
3 - Konstrukční náhled na detekční zařízení a jeho implementace do zásobníku[1]

Tento problém vedl k dalšímu zvážení pokračování projektu. Ještě na počátku práce z Projektu III [1] se uvažovalo, zda by bylo vhodnější použití senzorů nebo kamerového systému, který by dokázal jednotlivé součásti a jejich orientaci rozpoznat. Nakonec byla určena jako vhodnější varianta s indukčními a optickými senzory. Nyní se ale objevila motivace pro sestavení a testování univerzálního kamerového systému, který by dokázal fázi výstupní kontroly vibračních kruhových zásobníků podstatně zkrátit a zefektivnit právě díky využití inteligentního rozpoznání součásti kamerou. Projekt tedy bude pokračovat právě rozpracováním původně zavrhnutého řešení. Než se ale budu zabývat touto problematikou podrobněji, popišme si samotné stroje, které mám v úmyslu kontrolovat.

1.2. Vibrační kruhový zásobník – popis funkce a provedení

Vibrační kruhové zásobníky (zkráceně KZ) jsou speciální podskupinou vibračních dopravníků, které jsou založené na principu dopravy mikrovrhem. Při mikrovrhu dochází k vymrštění součásti nepatrným šikmým vrhem směrem vzhůru vpřed ve směru pohybu. Rychlost pohybu můžeme měnit velikostí amplitudy kmitu a vše záleží na nastavení, které je

závislé na dopravovaném materiálu. Každá součást potřebuje své specifické nastavení, převážně určení frekvence kmitání je velmi závislé na tvaru, velikosti a tuhosti součásti.



4 - Koncepční zobrazení kuželové, válcové a stupňovité varianty[2]

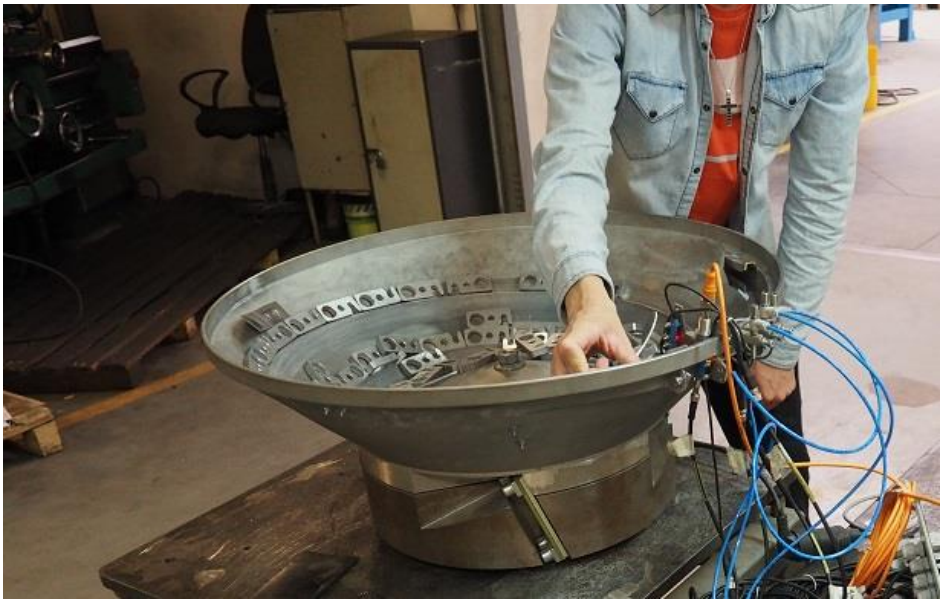
Celý zásobník je tvořen z pohonu a nádoby. Nádoby mají válcový, kuželový nebo stupňovitý tvar se šroubovitými drahami uvnitř a jsou vyráběny v rozličných velikostech od průměru 120 mm až po 600 mm. Kmitání je vytvářeno pomocí elektromagnetického pohonu, který je spojen s nádobou listovými pružinami. Ty jsou natočeny vůči základně ve specifickém úhlu a tomuto úhlu se říká úhel kmitu. Ten zajišťuje, že se v nádobě vytváří šroubový kmit, který posunuje součásti nuceně po kružnici uvnitř nádoby. Součásti postupují ode dna směrem vzhůru po šroubové dráze a v průběhu cesty dochází k jejich rovnání a třídění tak, aby na konci dráhy byly všechny součásti správně orientované a uspořádaně seřazeny za sebou. Součásti jsou orientovány mechanicky nebo pomocí senzorů a pneumatiky.

Vibrační kruhové zásobníky se používají převážně v továrnách, kde probíhá velká sériová výroba a kde mohou zastoupit funkci pracovníka, který bude například umisťovat součásti z neuspořádaného zásobníku přímo do výrobního stroje. To navíc dokáže mnohem levněji, neboť tento druh zařízení nespotřebovává při svém provozu velké množství energie, řádově se jedná o jednotky kW. Základním předpokladem pro tyto operace je ale spolehlivost, neboť výstup musí díly dodávat s velmi vysokou přesností.

1.3. Princip zajištění spolehlivosti vibračního KZ

Vibrační KZ jsou vyráběny ve většině případů na zakázku a je tedy nutné pro každý druh součásti připravit jedinečné řešení, to následně vyzkoušet a odladit. V tomto procesu je nutné konstantní sledování a při jakémkoliv nedostatku je nutná pomoc technika. Vzhledem k bohatým zkušenostem jsou při testování opravovány jen drobné detaily, ale vzhledem k povaze problematiky je stále nutný nepřetržitý dohled a ostražitost pracovníka, který stroj kontroluje. Při dlouhodobé kontrole může ale pracovník například přehlédnout špatně

orientovaný díl a jeho spolehlivost může být ovlivněna únavou, hlukem a dalšími faktory. Proto nelze spolehlivě testovat v dlouhodobějších časových intervalech a proces tak nemá dostatečnou rychlost ani spolehlivost. V pozdějších fázích, když se spolehlivost zdá dobrá, se již používá pouze kamera na stativu, ale ani ta nedokáže zajistit perfektní kontrolu, neboť je stále nutné později shlédnout záznam. A právě tuto problematiku jsem si vybral jako oblast pro zlepšení v mé diplomové práci.



5 - Proces výstupní kontroly vibračního kruhového zásobníku

1.4. Motivace projektu

Jako řešení této složité situace při výstupní kontrole se ukázalo vytvoření vlastního nebo zakoupení průmyslového systému, který by dokázal pracovat přímo pro tento typ úlohy a dokázal by nahradit nespolehlivý lidský faktor spolehlivějším automatickým systémem. Systém by byl pravděpodobně obsluhován pomocí softwaru na PC. Kamera připojená k počítači by byla díky tomu schopna sledovat pohyb na dopravníku, vyhodnocovat jej, počítat průchodnost a hlavně kontrolovat, zda se na výstupu neobjevil NOK díl a případně zaznamenal celou situaci, při které došlo k detekci falešné pozitivity – za účelem zjištění příčiny poruchy. Vzhledem k různorodosti použitých součástí musí být v systému velké množství nastavení všech důležitých proměnných a ty by měly být snadno měnitelné v grafickém rozhraní programu. Funkční systém by mohl do budoucna ušetřit mnoho hodin testování, ulehčit časovou zátěž na tuto činnost a zefektivnit práci lidí v tomto oddělení. Při zachování stávajícího jednoho zaměstnance by bylo do budoucna možné po zaškolení kontrolovat současně více než jeden stroj, což by přineslo mnohem vyšší efektivitu celého výrobního procesu.

2. Dostupné varianty řešení kontroly

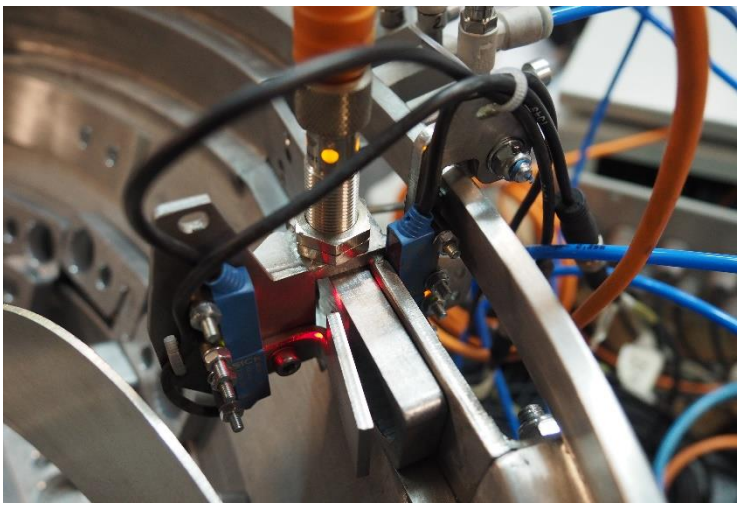
V této kapitole se zaměřím na varianty, jak je možné problematiku kontroly řešit. Jako nejideálnější se zdají být 2 možnosti, a to výběr již některého hotového řešení, nebo využití vlastního systému s pomocí volně dostupných knihoven a nástrojů, který bude náročnější na vývoj, ale bude přesně vyhovovat požadavkům.

2.1. Použití již hotového řešení od specializované firmy

Jako nejefektivnější řešení se z počátku zdálo vybrat vhodný průmyslový produkt, který by byl vhodný pro naše provozní podmínky a aplikovat jej do zkušebny. Podmínkou však byla kompletní samostatnost a upravitelnost řešení, bez dalších zásahů spolupracujících společností a co nejnižší pořizovací cena. Po provedení průzkumu trhu jsem došel k 3 variantám řešení ve 3 cenových kategoriích.

2.1.1. Varianta I - senzorika

První varianta se skládá pouze z optických a indukčních čidel a vychází z mé práce v Projektu 3 [1]. Za poměrně malé náklady bychom navrhli univerzální stojan, který by se umístil na konec výstupu a do kterého by se jednotlivá čidla osadila. Po spojení s PLC a vytvořením vhodného algoritmu bychom rozpoznávali jednotlivé díly a vzdáleně sledovali celý proces videokamerou. V principu by se jednalo o kontinuální snímání, ale pokud by došlo k chybě, rozsvítí se varovné světlo. Na videu by poté stačilo sledovat pouze světlo spuštěné z PLC.

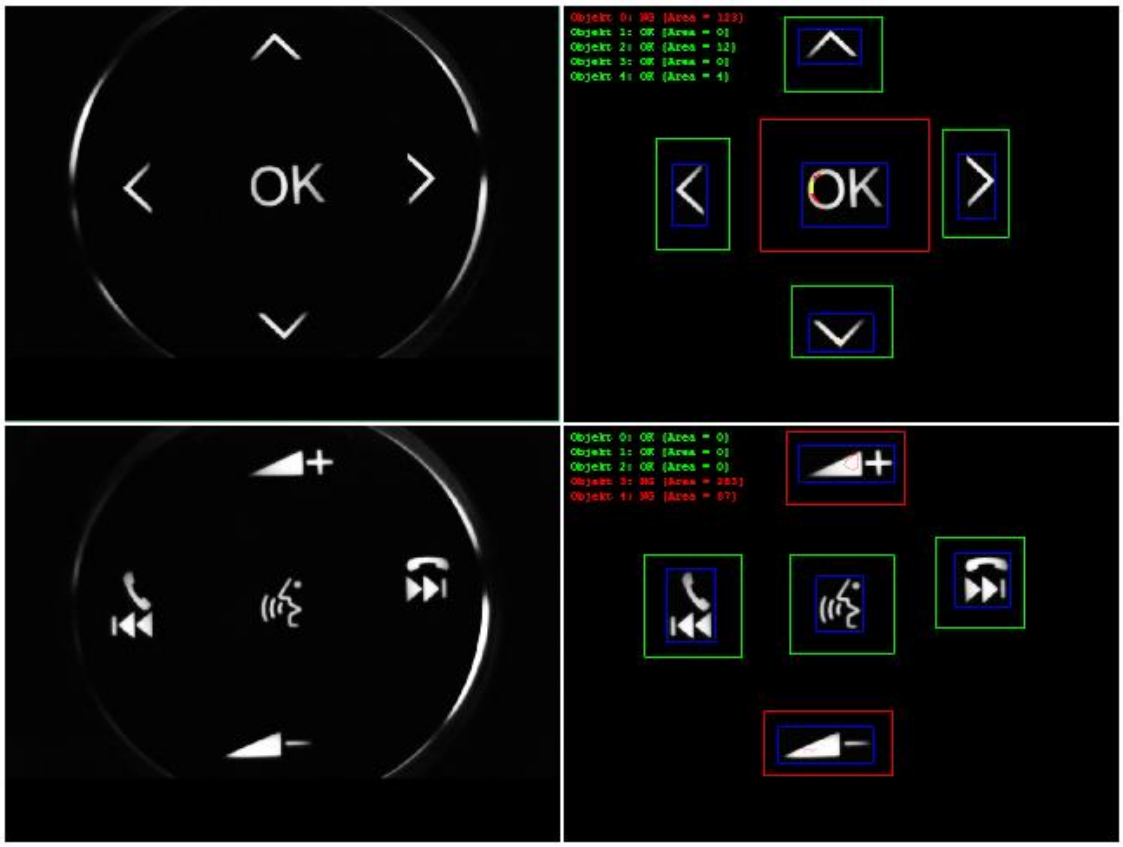


6 - Varianta 1: Vzor pro rozpoznání pomocí senzoriky [1]

Výhodou této varianty jsou velmi nízké náklady a velmi jednoduché zavedení do provozu. Stále však převažují nevýhody, neboť činnost není zcela automatizována a především není příliš modifikovatelná pro další aplikace. Pro každý druh součásti bychom museli vyrobit nový stojan a každá součást by musela mít vlastní instrukce pro PLC. To by do budoucna mohlo vést k dalšímu vytížení jiných pracovníků a stejné časové náročnosti. Z těchto důvodů jsem tuto variantu označil za aplikovatelnou, ale nedostačující.

2.1.2. Varianta II – kamerový systém

Jako druhá možnost se při dalším zkoumání naskytla varianta, která by obsahovala systém pro zpracování obrazu z kamerových systémů. Tento druh systémů je dnes velmi rozšířený a používá se například jako kontrola kvality pro výstup výrobní linky nebo jako průběžná kontrola při výrobě. Jeho implementace by byla také velmi rychlá v závislosti na dodavateli řešení. Mezi dodavatele tohoto řešení na českém trhu patří například firma JHV – ENGINEERING s.r.o. [3] , Hönigsberg & Düvel Datentechnik Czech s.r.o. [4] nebo DFC Design, s.r.o. [5].



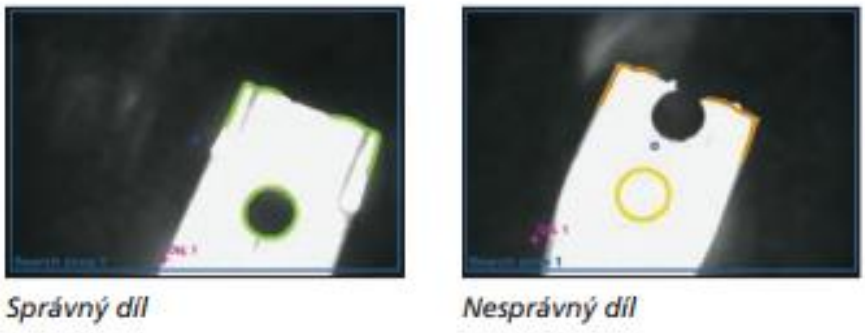
7 - Příklad realizace při výrobě palubních desek [5]

Výhodou tohoto řešení je vysoká přesnost rozpoznání a dokonce je přítomen i software, který by dokázal pracovat s rozličnými druhy součástí. Stačí pouze zaškolená obsluha a vhodné

stanoviště, kde by mohl stroj fungovat. Nevýhodou tohoto řešení je ovšem vysoká pořizovací cena, která se u jednotlivých dodavatelů liší, ale v konečné ceně může přesáhnout více než 100 tisíc korun, pokud počítáme celý systém se zaškolením a veškerým vybavením. Další nevýhodou řešení je zaměření většiny systémů na výrobní procesy, při kterých se počítá s nasazením systému pro dlouhodobé použití v jedné aplikaci. V tomto směru je pro naše účely tato varianta použitelná, ale ekonomicky nevýhodná, předimenzovaná, a tudíž nevhodná.

2.1.3. Varianta III – Kamerové senzory

Speciální skupinou, na kterou jsem při svém hledání narazil, jsou kamerové senzory. Ty kombinují výhody obou předchozích skupin jako je levná pořizovací cena, snadná implementace do výroby a dobrá adaptivita na nový druh součásti. Kamerový senzor je velmi malé odolné zařízení, které lze upevnit na jakýkoliv stojan a následně lze připojit buď k počítači, kde pomocí speciálního softwaru probíhá rozpoznání součástí, nebo je možné provést připojení pouze k libovolnému PLC, kde může například zastavovat běh zásobníku a spustit varovný signál. Celkově se škála nastavení SW pro naše použití zdá být vyhovující a pořizovací cena kamery bez softwaru je přibližně 20 tisíc korun[6], což by bylo vhodné řešení pro naše potřeby. Významnými dodavateli těchto zařízení jsou v České republice firma SICK spol. s r.o. [7] nebo Ifm Electronic spol. s r.o.[8].



8 - Příklad použití kamerového senzoru [8]

I toto řešení má ale své nedostatky. Po konzultaci s dodavateli jsme dospěli ke dvěma překážkám, které při současné konfiguraci senzorů není možné snadno vyřešit. Největší problém představuje doprava součástí v těsném závěsu za sebou, někdy dokonce s téměř nulovou mezerou. V případě vibračních dopravníků je ale tento jev velmi častý, někdy dokonce žádaný. Pak ale není možné, aby kamera zaznamenala přesný obrys součásti a tudíž není možné ani definovat, jestli je součást správně orientována. Dalším faktorem je pohyb. Pokud by součást zastavila v některém detekčním bodě, bylo by možné součást detekovat a ověřit,

ale během pohybu není možné vhodně určit jakýkoliv referenční bod a tudíž se jedná o velký problém, který by při detekci mohl způsobovat velkou nepřesnost.

Další nevýhodou je nemožnost přidání speciálních funkcí do softwaru čidla, takže naše možnosti jsou značně omezeny pouze na předpřipravené funkce uvnitř programu. Pokud bychom do budoucna měli nějaký speciální požadavek, museli bychom se nejspíše znovu obrátit na dodavatelskou společnost, nebo dokonce využít řešení jiného dodavatele.

9 - Produkt firmy SICK [7]

Toto řešení je tedy pravděpodobně nejblíže k realizovatelnosti jak z ekonomického, tak z funkčního hlediska, ale stále se nejedná o řešení, které by se dalo bez obtíží implementovat.

2.2. Vlastní open-source řešení

V návaznosti na předchozí podkapitolu lze prohlásit, že se mi nepodařilo naleznout zcela uspokojivé řešení celého problému, zvláště v těchto dvou oblastech:

- Univerzální kamery navržené pro průmyslové sledování a třídění součástí obecně velmi špatně pracují s pohybujícími se součástmi, které cestují po dopravníku s velmi malými rozestupy.
- Pokud se podařilo najít nějaké funkční a spolehlivé řešení, většinou šlo o velmi finančně náročný stroj.

Tak jsem se rozhodl vytvořit čtvrtou variantu, která ale na rozdíl od předchozích bude probíhat bez přispění externího dodavatele a bude pod plnou kontrolou autora.

2.2.1. Popis řešení

Jako další možnost tedy nastává vydat se cestou vlastního vývoje a navrhnout software pomocí programovacího jazyka Python, který společně s knihovnou OpenCV dokáže zpracovávat a vyhodnocovat obraz pořízený obyčejnou kamerou připojenou k počítači. Je to

tedy kombinace všech 3 předchozích variant, která ale může být, na rozdíl od výše zmíněných systémů, plně přizpůsobena a připravena na naši konkrétní aplikaci. V případě vlastního softwaru bych jako autor měl plnou kontrolu nad správou systému a případné drobné úpravy by nemusely být řešeny s externím dodavatelem. Systém by mohl být dále vyvíjen pro potřeby výstupní kontroly a později by případně mohl sloužit i jako supervizní systém, který by dokázal nahradit senzoriku přímo v pracovním procesu vibračních zásobníků, pokud by výstup nemohl být kontrolován jinak.

10 - Zdrojový obraz z kamery při testování pro další zpracování

Celý systém bude fungovat na principu strojového vidění a bude rozpoznávat pohybující se objekty v zásobníku. V obraze bude umístěn kontrolní prostor, kde bude probíhat kontrola podobnosti součásti. Tam systém podle předem nafocených vzorů porovná, zda jde o NOK součást, nebo jestli předchozí detekce proběhla v pořádku a procházejí pouze OK součásti. Systém bude porovnávat vnitřní kontury objektu a nebude tedy ovlivněn problémem při průchodu součástí v těsném závěsu. Pokud systém detekuje NOK součást, nebo pokud dojde k náhlé situaci, kdy díly nebudou odcházet ze zásobníku a dojde k jejich zaseknutí, systém rozpozná problém a provede záznam do zvláštního videosouboru. V případě zaseknutí součásti dojde také k upozornění obsluhy. Systém tedy bude schopen kontinuálně kontrolovat zásobník několik hodin a v krajních případech i několik dnů. Výstupem tohoto testování bude složka videosouborů, kterou obsluha prohlédne a vytvoří zprávu pro technika, který je zodpovědný za opravy zjištěných závad. V případě potřeby bude probíhat kontinuální videozáznam s přidanou vizualizací celého procesu.

2.2.2. Výhody a nevýhody řešení

Mezi hlavní přednosti celého řešení je:

a) Výhradní návrh řešení pro potřeby konkrétní činnosti.

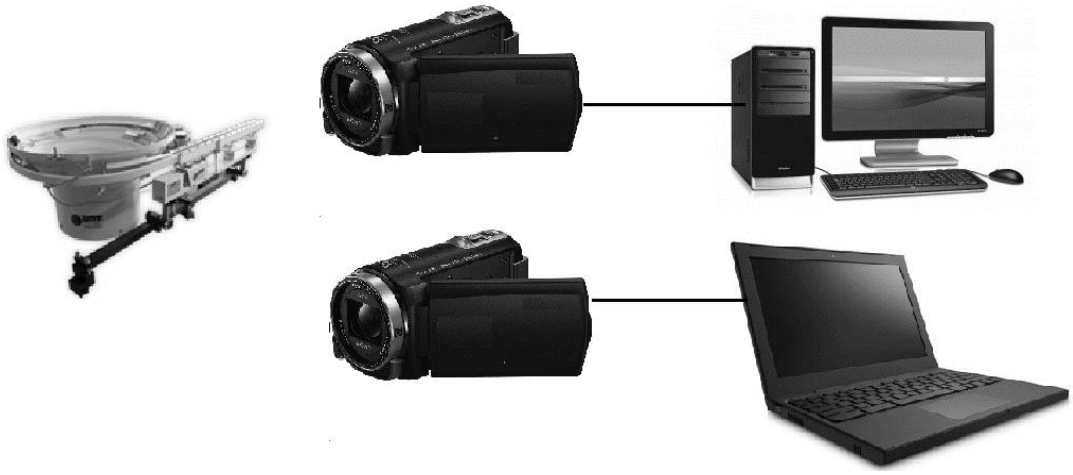
b) Prostor pro budoucí vylepšení a snadná upravitelnost.
c) Řešení je cenově přijatelné, veškeré investice jsou jen do vývoje aplikace a nákupu HW zařízení.
d) Možnost použití stávajícího hardwaru.
e) Řešení bude fungovat při pohybu součástí těsně za sebou.

I v tomto případě ale zůstávají negativa:

a) Software je bez jakékoliv podpory.
b) Spolehlivost řešení není ani zde zcela zaručena.
c) Mohou se objevit problémy při vývoji.
d) V budoucnu bude potřeba aplikaci renovovat a udržovat.

2.2.3. Vhodný hardware

Při návrhu systému budu počítat s tím, že na pracovišti bude umístěna jedna kamera na stavitelném stojanu a jeden stolní počítač či notebook. Celá sestava bude umístěna u kontrolovaných zařízení. Při požadavku na dlouhodobé testování, které bude probíhat déle než je pracovní směna zaměstnance kontrolního oddělení, je potřeba zajistit počítač, který bude vhodný pro běh 24/7, například speciální server.



11 - Koncepční schéma hardwaru

Ve zvláštním případě může nastat situace, že počítač bude nutné umístit mimo dosah vibračního zásobníku. V tom případě bude možné využít IP kameru, která obsahuje ethernetový port a bude možné ji připojit do firemní počítačové sítě. Poté již můžeme provádět test vzdáleně například z kanceláře, abychom měli nepřetržitý přehled o stavu

kontroly a nebylo nutné zůstávat u stroje. V případě testování více strojů současně bude nezbytné využít více počítačů tak, aby jejich počet odpovídal počtu testovaných strojů.

2.3. Zhodnocení a výběr vhodného řešení

Ze všech variant popsaných v této kapitole je možné reálně uskutečnit pouze dvě, které by mohly spolehlivě fungovat po dlouhou dobu bez větších problémů a nebyly by zároveň příliš finančně nákladné. A těmi jsou zaprvé instalace kamerových senzorů, kde bych ale musel vyřešit problém s pohybem více dílů v řadě, a zadruhé návrh vlastního softwarového řešení. Vzhledem k tomu, že software kamerových senzorů není zcela přesně určen pro aplikaci ve zkušebně a musel bych při sestavení hledat některé kompromisy, rozhodl jsem se pro vlastní návrh, který bude vytvořen přesně na míru zadaným požadavkům. V budoucnu by tento systém mohl posloužit také jako základ pro nahrazení orientace v zásobníku, kdy by byl místo senzoriky použit právě tento systém rozpoznání obrazu. I to byl jeden z důvodů, proč jsem se rozhodl právě pro tuto variantu, kterou v následujících kapitolách popíšu podrobněji. V tabulce T2 si můžete podrobně prohlédnout vhodnost jednotlivých řešení v nejkritičtějších oblastech rozhodování (10 značí nejlepší a 0 nevhodné).

T2 - Zhodnocení všech variant a jejich aspektů od 0 do 10 (0 - nevhodné, 10 - nejlepší)

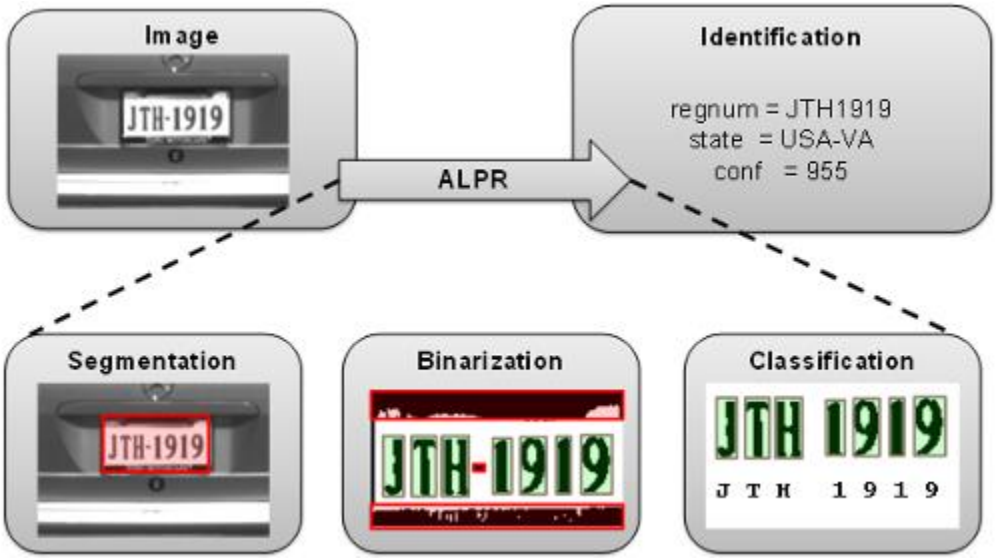
Varianta	*Cena*	*Variabilita*	*Rychlost nasazení*	*Spolehlivost*	*Provozní náklady*
Senzorika	10/10	4/10	8/10	4/10	8/10
Kamerový systém	3/10	9/10	7/10	10/10	5/10
Kamerové senzory	8/10	7/10	8/10	7/10	9/10
Vlastní řešení	6/10	10/10	5/10	9/10	10/10

3. Strojové vidění a jeho využití v praxi

Strojové (chcete-li počítačové) vidění dnes můžeme nalézt téměř ve všech průmyslových odvětvích, ve kterých hraje význam automatizace a nahrazení lidské práce. Jak se samotná technologie zlepšuje, dostává se i do dalších sfér jako je například automobilový průmysl a samotné řízení automobilu. To naznačuje, že již žijeme v době, kdy je možné, aby strojové vidění dosahovalo lepších a spolehlivějších výsledků než vidění lidské. Strojové vidění je tak často spojováno s jeho velkou rolí v příchodu další průmyslové revoluce, tzv. „Průmyslu 4.0“.

3.1. Úvod do problematiky strojového vidění

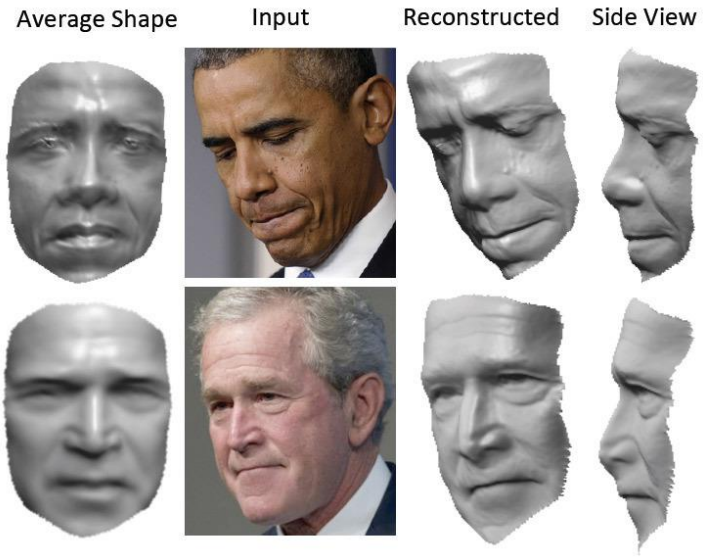
Strojové vidění není věc vůbec nová, jeho použití bylo velmi rozšířené již na přelomu tisíciletí a prvotní zmínky v této problematice se objevily již na přelomu 70. a 80. let minulého století. Velmi rozšířené bylo převážně při kontrole povrchu, automatizovaných montážních činnostech, při bezdotykovém měření nebo při identifikaci v sériové výrobě na výrobních linkách.[9], [10] Rozpoznání textu v obrazu (OCR) se objevilo dokonce již na počátku 30. let 20. století a do počítačové sféry proniklo o 30 let později. [11]



12 - Příklad průběhu zpracování SPZ pomocí OCR[12]

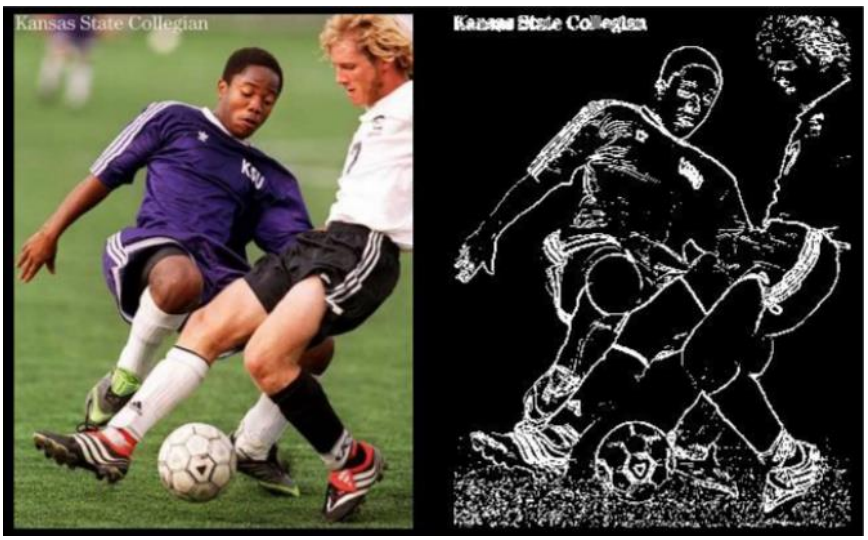
Do každodenního života následně proniklo strojové vidění také, a to při zavedení funkce rozpoznání textu do domácích skenerů a později v digitálních fotoaparátech s detekcí obličejů. Během postupného vývoje došlo k přechodu na rozpoznání 3D obrazu, kdy byla pro detekci použita další kamera, která dokázala celý proces posunout na lepší úroveň. Tento krok naznačuje, že se celý proces snaží co nejvíce přiblížit k fungování lidského oka a mnoho poznatků z biologie již bylo aplikováno pro dosažení lepších výsledků.[13] Nejvýznamnějšími

obory, kde je dnes strojové vidění nenahraditelné, jsou hlavně lékařství, výrobní procesy, bezpečnostní systémy, fotografický průmysl a automobilový průmysl.



13 - Příklad použití 3D detekce při rozpoznání obličejů[14]

Princip strojového vidění funguje na základě postupného zpracování a převedení surového obrazu na jednotlivé složky obrazu, ze kterého počítačový program dokáže rozpoznat konkrétní informaci. Pokud bychom použili pouze surový obraz, počítač vidí jen matici RGB čísel, která pro něj ale nemají žádný konkrétní význam. Obraz se převádí na obraz ve stupních šedi, histogram, bitovou masku (v závislosti na daných kritériích) nebo matici obsahující složky jasu. Pomocí složitějších algoritmů můžeme provést dále detekci hran nebo hledání předem známého vzoru v obrazu. [13]



14 - Příklad aplikace detekce hran na obrázek[13]

V posledních letech se vývojáři v této oblasti nechali znovu inspirovat přírodou a do technologie rozpoznání obrazu zapojili neuronové sítě. To přineslo obrovský skok v této

oblasti a dnes tak můžeme najít tak pokročilé aplikace, které dokáží rozpoznat nejen to, jaké objekty se na vložené fotografii nachází, ale i jaké činnosti jsou vykonávány[15]. Na obrázku 15 můžete vidět úspěšnost algoritmu od firmy Google v roce 2014, zprava doleva můžeme vidět od nejpřesnější po nejméně odpovídající. Toto všechno zajišťuje učící algoritmus, který se s každou další aktivitou zlepšuje a s dalším vývojem bychom se tak mohli skutečně brzy dočkat zařízení, které by skutečně mohlo svět okolo sebe rozpoznávat, stejně jako lidské oko.[16]

15 – Přesnost algoritmu detekce scény vytvořeným společností Google. [15]

3.2. Metody analýzy obrazu a detekce objektů

Po získání surového obrazu je většinou nutné provést základní processing, jako je odstranění šumu, úprava jasu a kontrastu či oříznutí. Velikost zásahu je přímo úměrná kvalitě objektivu a světelným podmínkám při expozici. Jakmile máme obraz předzpracovaný, můžeme přejít k dalšímu kroku, kterým je segmentace. Existuje mnoho metod segmentace obrazu a všechny vedou k podobnému výsledku, který v obraze vyhledá základní kontury a tvary, se kterými lze dále pracovat. Podívejme se na pár z nich.

Thresholding (prahování)

Thresholding je nejzákladnější a výpočetně nejméně náročná metoda segmentace, kterou lze jednoduše aplikovat na obraz pouze s malými úpravami. Nezbytná úprava je alespoň převod na obraz ve stupních šedi nebo na barevný obraz ve formátu HSV (Hue, Saturation, Value – viz obrázek 16). Tuto metodu lze použít na objekty, které jsou v kontrastu se svým pozadím, například černé na bílém pozadí nebo barevné objekty na pozadí jiné barvy. V tomto případě můžeme využít naměřené meze intenzity barvy (např. z histogramu) a odstranit z obrazu všechny body, které jsou pod nebo nad touto mezí. Tím nám vznikne bitová mapa,

která obsahuje pouze 1 nebo 0 a která značí, zda je daný bod v rozsahu či nikoliv. Vezměme v potaz, že chceme separovat bílé pozadí, které se v intenzitě stupních šedi přibližuje hodnotě 255. Jako hodnotu *T* si zvolíme číslo, které bude menší než nejtmavší bod pozadí a aplikujeme vztah z rovnice :

$$g(i,j) = \begin{cases} 1, & f(i,j) < T \\ 0, & f(i,j) \geq T \end{cases} \quad (1)$$

kde *g* … bit ve výsledném obrazu, značí přítomnost objektu

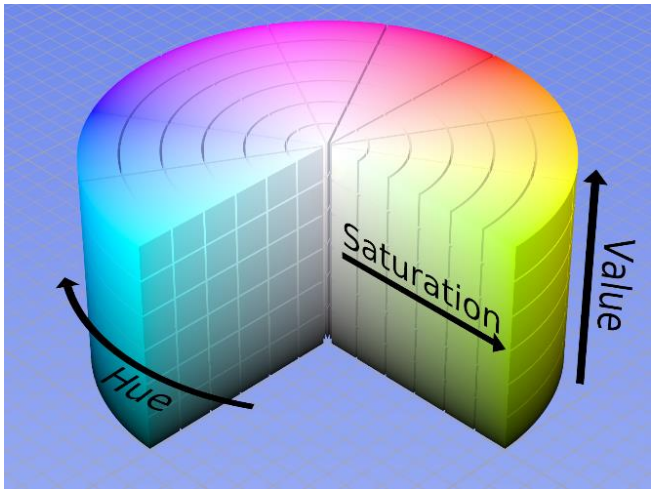
f … hodnota matice na souřadnicích *i* a *j*

T … mezní hodnota detekce, nad kterou provedeme ořez obrazu

i … souřadnice ve svislém směru

j … souřadnice ve vodorovném směru

Výsledek je maska ve formě matice, která obsahuje oblasti, ve kterých se vyskytují hledané objekty. Nyní můžeme provést další úpravy, jako jsou například detekce hran nebo aplikace masky na původní obraz. Dalšími variantami thresholdingu může být další rozšíření mezních hranic a tím získání více masek. Pokročilejší varianta může být s adaptivním prahováním, kde je mezní bod určen automaticky v závislosti na pozadí.



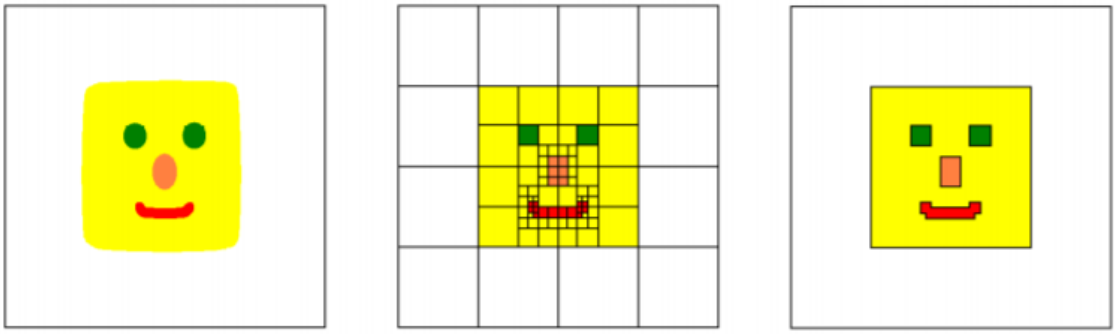
16 - Grafické znázornění barevné palety HSV[17]

Regionální metody

Regionální metody se používají pro homogenizaci členitého obrazu. Z obrazu ve výsledku vystoupnou oblasti se stejnými vlastnostmi, například povrch hledaného objektu. Základní dvě metody jsou spojování a štěpení obrazu. Při spojování obrazu zvolíme spojovací kritérium, tedy hodnotu, o kterou se mohou sousední pole lišit, aby došlo ke spojení. Pak už jen spustíme algoritmus, volitelně můžeme přidat parametry např. pro počáteční bod hledání nebo směr

spojování. Výsledky se liší v závislosti na zadaném kritériu, pokud jej ale zvolíme správně, můžeme dostat stejné výsledky jako při detekci hran, v obrazu s vysokým šumem dokonce i lepší.

Při štěpení oblastí dochází k podobnému principu jako při spojování, ale v opačném pořadí. Nejprve vezmeme celý obraz, který postupně dělíme na další menší oblasti, až každá oblast bude ve vztahu k sousedním odpovídat zadanému kritériu. Zvláštní variantou může být použití štěpení a spojování, kdy dojde nejprve ke štěpení obrazu do čtvercových oblastí a poté se vyhodnocuje, zda by bylo možné některé sousední oblasti spojit dle slučovacích kritérií. Příklad použití tohoto algoritmu na jednoduchý obraz můžete vidět na obrázku 17. Vlevo vidíme originál, uprostřed po použití štěpení a vpravo výsledek po štěpení a sloučení.



17 - Znázornění použití štěpení a slučování v praxi [13]

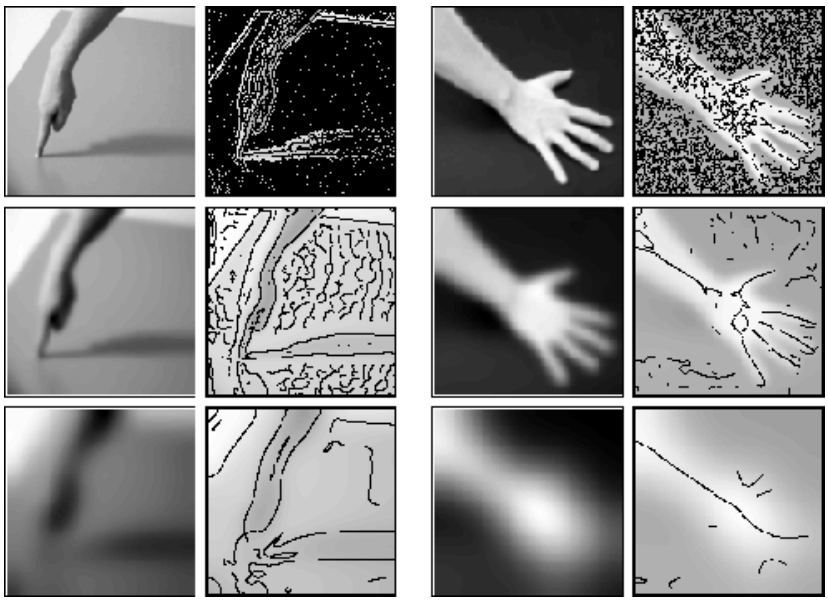
Detekce hran

Mezi další rozšířené metody segmentace patří detekce hran, která je založena na předpokladu, že jednotlivé pixely na hranici objektů mají velký rozdíl hodnot jasu. Při detekci hran tak použijeme hodnoty jasů na jednotlivých pixelech, na které aplikujeme první nebo druhou derivaci a hledáme lokální maxima v případě první, anebo průchody nulou v případě derivace druhé. Před použitím této metody je ale z obrazu třeba odfiltrovat šum, neboť poté může dojít k detekci falešných hran zapříčiněných právě šumem.

<u>Rozdělení typů algoritmů pro detekci hran:</u>

- Hledání maxima při použití první derivace
- Hledání průchodu nulou při použití druhé derivace
- Pokročilejší algoritmus využívající lokální aproximace obrazové funkce parametrickým modelem
    - Robertsův operátor
    - Operátor Prewittové

- Sobelův operátor
- Cannyho hranový detektor



18 - Detekce hran a jednotlivé výsledky při použití filtru šumu [18]

Cannyho hranový detektor

Nejpopulárnější hranový detektor se nazývá Cannyho hranový detektor a vynalezl jej John F. Canny v roce 1986 [19]. Celý algoritmus se skládá z několika kroků:

- Redukce šumu – hranové detektory obecně jsou velmi citlivé na šum a právě šum dokáže jejich výsledky značně znehodnotit. Proto je nutné nejprve šum nepatrně odfiltrovat pomocí Gaussova filtru. Filtr využívá konvoluce obrazu s elementem určeným Gaussovou funkcí

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

Kde: x … Gaussova oblast ve vodorovném směru
y… Gaussova oblast ve svislém směru
σ…rozptyl – velikost rozostření

- Nalezení intenzity gradientu v obrazu – po přiměřeném rozostření obrazu použijeme Sobelův algoritmus pro zjištění první derivace v obrazu pro vertikální Δ_x a horizontální směr Δ_y . Následně provedeme sloučení směrů pro každý pixel pomocí vztahů:

$$Velikost\ gradientu\ \Delta = \sqrt{\Delta_x^2 + \Delta_y^2} \quad (3)$$

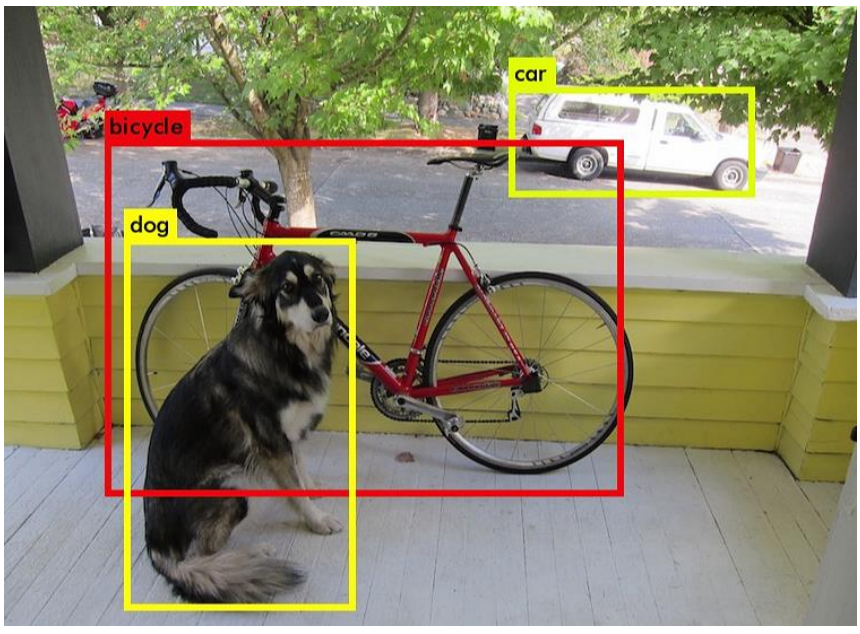
$$Úhel\ \theta = tan^{-1}\left(\frac{\Delta_y}{\Delta_x}\right) \quad (4)$$

- Nalezení lokálního maxima – jakmile dostaneme směr a sklon pro jednotlivé pixely, můžeme analýzou pole zjistit lokální maxima. V těchto bodech je očekávána hrana nebo přechod. Tyto body zaneseme do vznikající masky – minima označíme jako bílé body a vše ostatní jako černou oblast.

- V této fázi máme obraz, který obsahuje všechny hrany, dokonce i ty nejmenší, a musíme tedy provést další thresholding, abychom z obrazu vyfiltrovali tenké a nepatrné čáry, které pro nás nejsou důležité. Tato filtrace spočívá v tom, že v každém lokálním minimu stanovíme určitou horní hranci, kterou musí funkce ve svém okolí dosáhnout. Pokud funkce protne a překročí horní hranici ve svém okolí, je gradient natolik velký, že se zcela určitě jedná o výraznou hranu. Tímto procesem získáme z obrazu důležité hranice a vzniklou masku (viz. Obrázek 14) můžeme použít pro detekci tvarů a objektů.

Rozpoznání objektů

Po segmentaci, kdy jsme dokázali detekovat kontury hledaného objektu, je nyní nutné provést kategorizaci a rozpoznat objekt. Objekt porovnáváme dle barvy, tvaru, kulatosti nebo těžiště s předem definovanými předlohami, které jsou uvedeny jako výchozí tvary. Předloha může být jedna nebo více a algoritmus porovnává, ke kterému objektu či skupině je objekt na obrázku nejblíže. Ne vždy je detekce přesná, občas může dojít k falešné detekci. Je třeba správně určit, jak velkou míru podobnosti požadujeme a musíme nalézt hranici mezi tím, kdy algoritmus detekuje falešné objekty a kdy žádné.



19 - Příklad použití kompexnějšího algoritmu pro detekci objektů [20]

I pro rozpoznání objektu existuje mnoho druhů algoritmů a metod, od těch nejjednodušších, které dokážou označit tvary jako je kruh, čtverec, obdélník, atp., až po náročnější, které dokážou podle předem zadaných předloh odlišovat například auta, motorky a chodce na ulici – takový můžete vidět na obrázku 19.

3.3. OpenCV – úvod a používané metody

Abychom nemuseli všechny metody a algoritmy složitě programovat, existuje knihovna nástrojů, která obsahuje až okolo 500 funkcí pro manipulaci s obrázky a videem a jejich analýzu[21]. Tuto knihovnu budu využívat k vývoji softwaru pro sledování součástí. OpenCV je vytvářena jako svobodný software pod licencí BSD a je tedy zdarma pro akademické i komerční využití. Její velkou předností je možnost zpracování obrazu v reálném čase a také její univerzálnost. Lze ji spustit na všech rozšířených operačních systémech (Windows, Mac, Linux, iOS) a je možné ji integrovat do většiny rozšířených programovacích jazyků jako je C++, C, Python a Java.

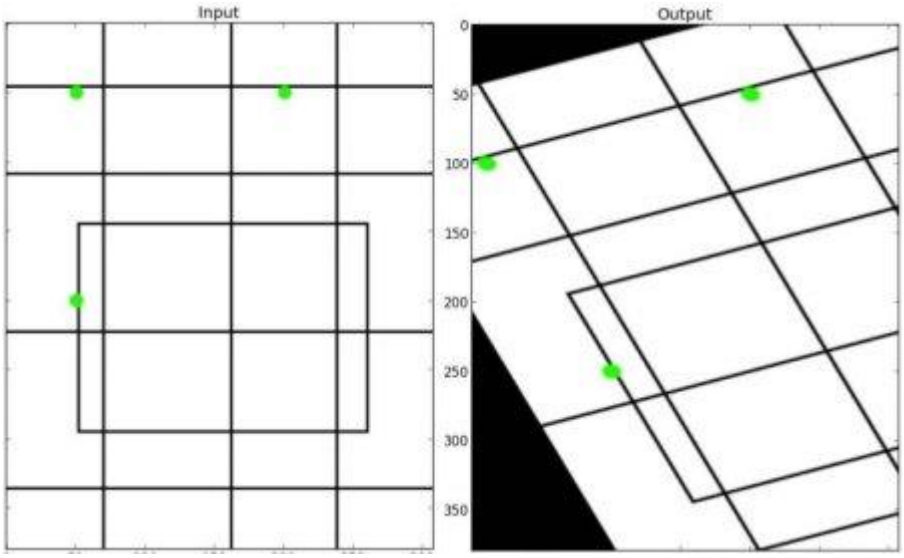
3.3.1. Základní zpracování videa a fotografií

Mezi zpracováním fotografie a videa nedělá knihovna OpenCV žádný velký rozdíl, neboť každé video je bráno jako sled fotografií. Pokud tedy dojde k aplikaci filtru, jsou aplikovány na fotografie, stejně jako na video. To je oproti běžným softwarům pro zpracování videa velký rozdíl. Mezi nejzákladnější nástroje, které v OpenCV můžeme použít je načtení videa či fotografie ze souboru či videokamery pomocí funkce *imread*. Čtení z kamery probíhá velmi jednoduše a není třeba žádného nastavování, pokud máte videokameru označenou jako výchozí zdroj videa v operačním systému. Pokud načteme obraz, můžeme jej upravit a následně zapsat do souboru pomocí funkce *imwrite.*

20 - Použití funkce pro převod do odstínů šedi[21]

Mezi nejzákladnější funkce patří úprava barev v obrazu. Funkce *COLOR_BGR2GRAY* převede obraz do odstínů šedi (viz obrázek 20) a *COLOR_BGR2HSV* se zase postará o převod z barev BGR do HSV (viz. Podkapitola 3.2). Systém BGR je totožný se systémem barev RGB, ale v matici každého pixelu jsou barvy místo červená-zelená-modrá přeskupeny na kombinaci modrá-zelená-červená. Pokud aplikujeme některý z filtrů, budeme si nejspíše chtít výsledek zobrazit a k tomu slouží funkce *imshow*.

Pokud bychom chtěli použít geometrické transformace, jsou zde předpřipraveny funkce pro změnu velikosti obrazu (*resize*), posuv obrazu (*warpAffine*) či rotaci (*getRotationMatrix2D*). Mezi méně běžné metody patří změna afinity (*getAffineTransform*), která dokáže zkosit úhel pohledu tak, aby zůstala zachována rovnoběžnost všech čar v obraze. A také metoda pro změnu perspektivy obrazu (*getPerspectiveTransform*), která dokáže např. opravit úhel pohledu na žádaný objekt na fotografii.



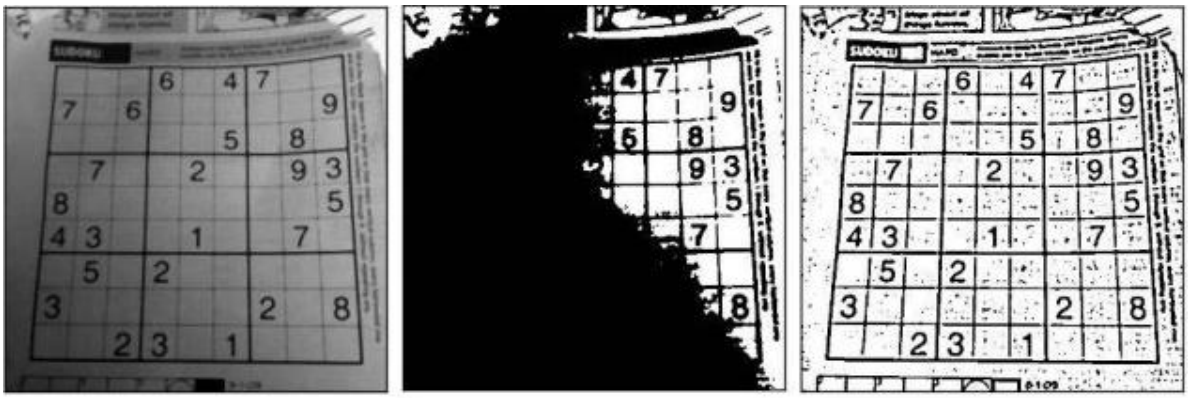
21 - Použití změny afinity[19]

3.3.2. Pokročilé metody zpracování obrazu

Pokud budeme chtít náš algoritmus přiblížit strojovému vidění, musíme využít pokročilejší funkce, které nám dovolí skutečně rozpoznávat a zpracovávat obraz.

Mezi velmi důležitou funkci patří thresholding, který nám umožní provést základní analýzu obrazu tak, jak bylo popsáno v kapitole 3.2 v části thresholding. Jednoduchý thresholding (*threshold*) nám zpracuje obraz dle zadaných kritérií na masku, se kterou lze dále pracovat např. při rozpoznání objektu před kontrastním pozadím. Jednoduchý thresholding ale vykazuje špatné výsledky při zpracování nerovnoměrně osvětleného obrazu, v tomto

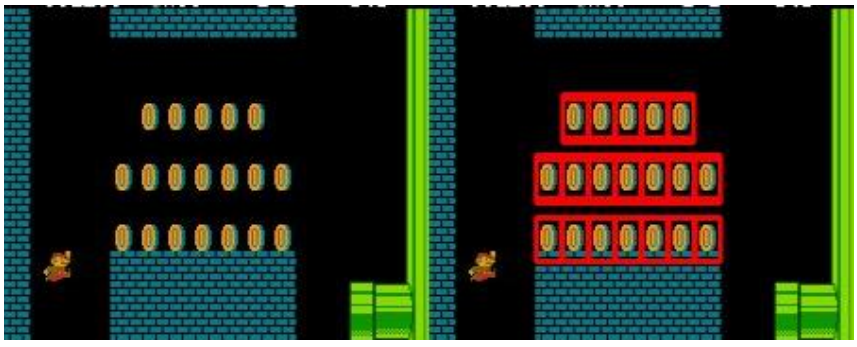
případě je lepší použít adaptivní thresholding, který rozčlení obraz do několika menších čtverců a podle adaptivních kritérií provede analýzu. Na obrázku 22 můžete vidět rozdíl mezi použitím jednoduchého (uprostřed) a adaptivního thresholdingu (vpravo).



22 - Použití jednoduchého a adaptivního thresholdingu[19]

Pokud chceme, aby strojové vidění dokázalo rozpoznat samotný objekt, je třeba najít v obraze nějaké souvislosti, které je možné později porovnat se vzorem. Mezi důležité metody patří detekce hran, jak bylo zmíněno v podkapitole 3.2 na straně 22. Pokud dokážeme zachytit hrany v obraze a převést je na vektorové kontury, můžeme danou konturu porovnávat. V OpenCV můžeme využít přímo Cannyho hranový detektor pomocí funkce *Canny.* Jako parametry této funkce musíme vložit zdrojový obraz, maximální hodnotu pro finální thresholding a velikost Sobelova jádra pro detekci gradientu.

Jako další velmi důležitý algoritmus je porovnání se šablonou *matchTemplate*, ten používá jiný přístup než výše zmíněný hranový detektor. Základem je vzor, který do algoritmu vložíme a ten pak prozkoumává postupně celý obraz a snaží se vyhledat shodu, která by odpovídala kritériu podobnosti vložené jako parametr. Výhoda této metody je velká adaptabilita na nový vzor a při vysoké podobnosti i velmi dobré výsledky. Nevýhodou ovšem naopak může být falešná detekce některých částí obrazu, které ale objektem vůbec nejsou a mají jen např. stejnou barvu.

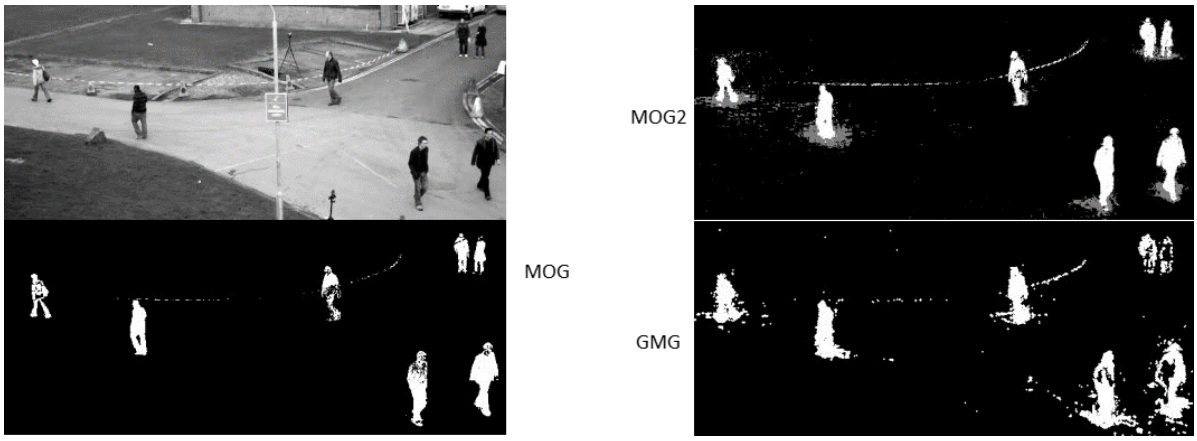


23 - Výsledek hledání objektu pomocí šablony[19]

3.3.3. Zpracování videa

Knihovna OpenCV má také své specifické nástroje pro zpracování videa, které jsou založeny právě na návaznosti jednotlivých snímků za sebou. Pojďme si představit ty nejdůležitější.

Mezi nejpoužívanější kroky při zpracování videa je odstranění pozadí. Je to jedna ze základních metod při zpracování videa. Předpokladem je ale pro celý proces statická kamera, neboť základním principem je detekce pohybujících se částí obrazu oproti statickému pozadí. V knihovně máme připraveny 3 algoritmy, které můžeme pro tento úkon využít a z každého dostaneme jiné výsledky. Tyto algoritmy jsou *createBackgroundSubtractorMOG, createBackgroundSubtractorMOG2* a *createBackgroundSubtractorGMG*. Každý algoritmus přistupuje k problému trochu odlišným způsobem, všechny ale mají stejný předpoklad, a to ten, že se snaží pomocí analýzy předchozích snímků a využití teorie pravděpodobnosti vykreslit kompletní obraz pozadí, i když se např. v obraze po celou dobu pohybuje spousta lidí.



24 - Porovnání výsledků po separaci pozadí[19]

Dalším velmi důležitým nástrojem je funkce *CamShift*, která dokáže usnadnit rozpoznání objektů. Nástroj vychází z metody *Meanshift*, ale je nepatrně rozšířen. Abychom mohli funkci použít, je třeba obraz nejprve upravit a odstranit z něj pixely, které pro naše rozpoznání nebudou důležité - například pokud vše odfiltrujeme a ponecháme pouze pixely v barvě lidské kůže, můžeme detekovat obličej. Při aplikaci vložíme parametr, který nám přibližně určí velikost a počátek hledané oblasti. Algoritmus nalezne počet pixelů v dané oblasti a určí společné těžiště pixelů. Pokud se těžiště neshoduje se středem oblasti, je nutné oblast posunout dále, dokud se těžiště nebude nacházet ve středu oblasti. Po detekci je nástroj schopen touto iterací rozpoznat i posunutí oblasti na dalších snímcích a metoda je tak vhodná

pro sledování pohybujících se objektů. Rozšíření pro *CamShift* spočívá v tom, že během pohybu je schopný dle okolností upravovat velikost a natočení oblasti.

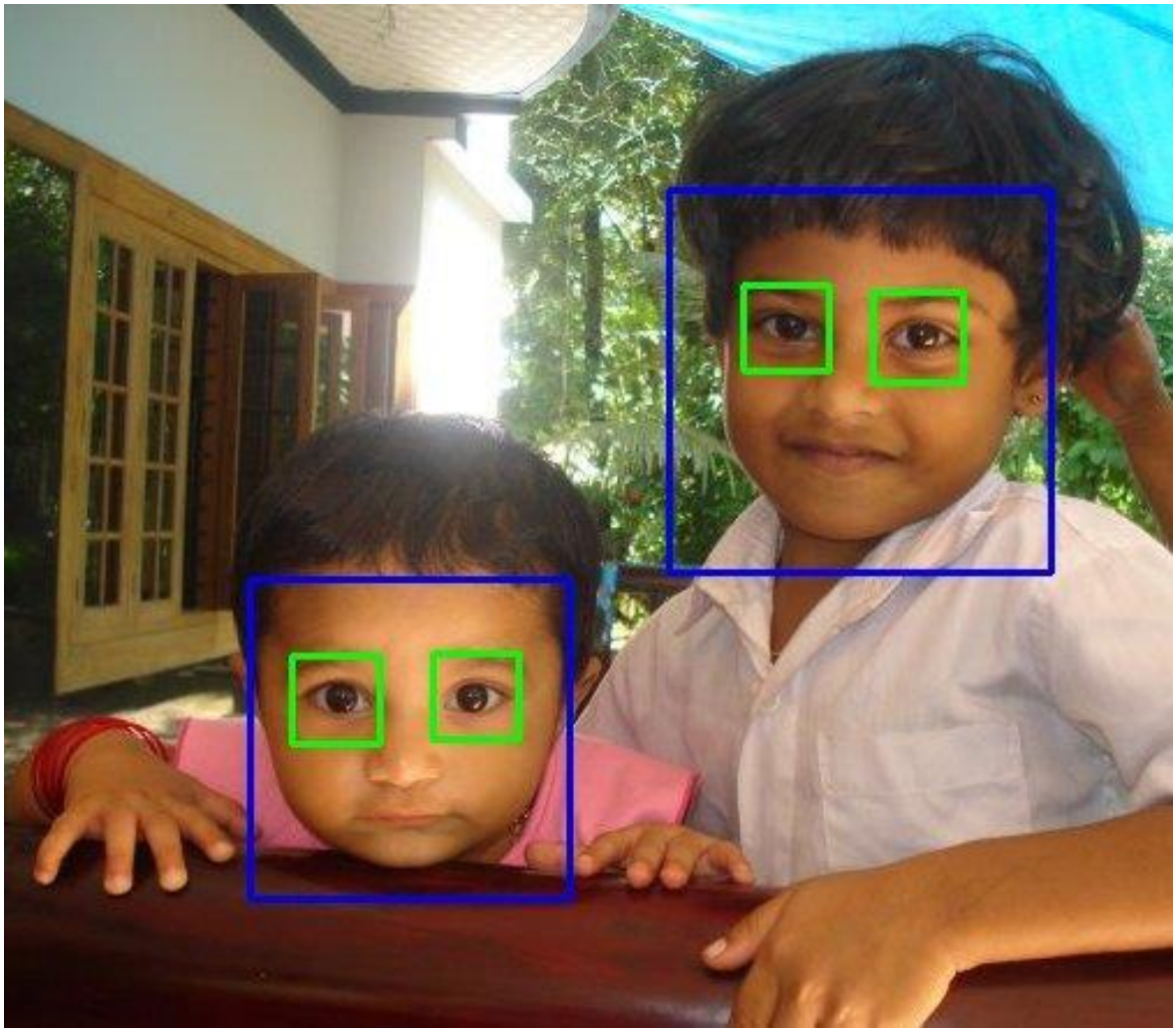
Jestliže je nutné zajistit sledování objektů, je možné také využít další nástroj, který je pro tuto aktivitu určen a jmenuje se *OpticalFlow.* Na rozdíl od předchozího nástroje, zde se porovnávají snímky tak, jak jdou za sebou a jako výsledek máme křivku pohybu, která se může zakreslit do obrazu. Hlavním rozdílem oproti předchozímu je tedy to, že zde sledujeme pouze vektor a nikoli celý objekt, předpokladem jsou neměnící se pixely objektu a zároveň statické pozadí.

25 - Příklad aplikace OpticalFlow[19]

3.3.4. Detekce objektů

Všechny výše zmíněné nástroje využívali především geometrii, kterou se snažili aplikovat do obrazu, ale stále šlo jen o rozpoznání tvarů či pixelů. Teď se podívám na jeden z nejpokročilejších nástrojů v OpenCV. A to je rozpoznání objektů pomocí metody *Haar cascades*. Ta využívá systému učení a trénování pomocí velkého množství pozitivních a negativních obrázků podle toho, o který předmět máme zájem. Například chceme-li rozpoznat v obraze psa, musíme vytvořit předlohu, do které vložíme tisíc fotografií psa a tisíc fotografií, kde pes není. Algoritmus provede analýzu pro trénování klasifikátoru, a jakmile jej následně použijeme na fotografii určenou k prohledání, algoritmus najde a označí na obrázku psa. Takto existuje již tisíce klasifikátorů, které jsou zaměřeny na věci z každodenního života, jako jsou například auta, lidé, obličeje, oči nebo zvířata. Pokud máme dostatečně kvalifikovaný

klasifikátor, dostaneme se na vysokou přesnost při hledání objektů v obraze a to až 95%. Občas může být kvalita ovlivněna šumem, rozostřením nebo částečným překrytím objektu. Nevýhodou této metody je to, že musíme pro každý objekt vytvořit novou galerii snímků a pokud chceme dosáhnout velké přesnosti, je třeba snímků velmi mnoho, a to až několik tisíc. Proto je tato metoda hůře využitelná pro použití v aplikaci, kde potřebujeme rychlé a snadné adaptování na nový vzhled součásti.



26 - Využití klasifikátoru pro detekci očí a obličeje[19]

3.4. Python – použití programovacího jazyka

Jako programovací jazyk pro moji práci jsem zvolil Python ve verzi 2.7, neboť tento jazyk je velmi rozšířený na akademické půdě, zároveň je uživatelsky intuitivní i pro méně zkušené programátory a jeho spolupráce s OpenCV je na velmi dobré úrovni. Python vznikl roku 1991 v Nizozemsku a jeho autorem je Guido van Rossum. Aktuální verze je 3.6, ale souběžně probíhá také vývoj verze 2.7, která by se měla postarat o plynulý přechod programátorů na verzi 3, neboť programy pro Python 2 a 3 nejsou mezi sebou vzájemně kompatibilní. Verze 2.7

ponechává syntaxi Pythonu 2, ale přidává volitelně některé prvky z 3 a zároveň poukazuje na funkce, které již ve verzi 3 nebudou funkční.

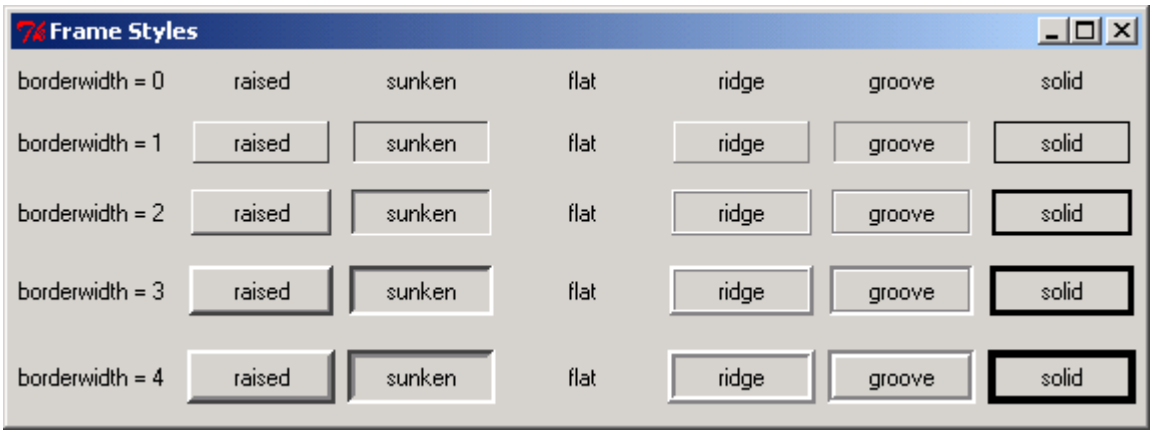
Co se týče srovnání s jinými programovacími jazyky, patří Python do interpretovaných, interaktivních a objektově orientovaných jazyků. Výkonem sice zaostává za C, C++ nebo Javou, ale co ztrácí na rychlosti, to se mu daří dohnat v přehlednosti kódu. Některé programy mohou být napsány až s dvakrát menším počtem řádků, než srovnatelné pro C nebo Javu. Výrazným pomocníkem ve výkonu může být nástroj zvaný PyPy[22], který je kompatibilní se současným kódem v Pythonu, ale dokáže jej kompilovat do kompaktnější podoby, stejně jako to dělá již v základu jazyk C nebo Java. Programy v pythonu jsou spouštěny jinak přímo ze zdrojových kódů a není třeba žádná jejich kompilace. Jako velká nevýhoda Pythonu může být bráno to, že je vyvíjen převážně nadšenci a nemá žádnou stabilní základnu jako například Oracle a jeho Java, proto může některé uživatele odradit poněkud nepřehledná dokumentace, oproti již zmiňované Javě nebo C/C++. Dle srovnání společnosti TIOBE je aktuálně Python na 5. místě dlouhodobé popularity, kam se vyšplhal po roce 2011 a kde se aktuálně nachází už jen za skupinou jazyků C/C++/C# a dlouhodobě vedoucí Javou.[23]

T3 - Tabulka vývoje dlouhodobé popularity a umístění Pythonu dle společnosti TIOBE [23]

Programming Language	2016	2011	2006	2001	1996	1991	1986
Java	1	1	1	2	14	-	-
C	2	2	2	1	1	1	1
C++	3	3	3	3	2	2	5
C#	4	4	6	9	-	-	-
Python	5	7	7	18	26	-	-

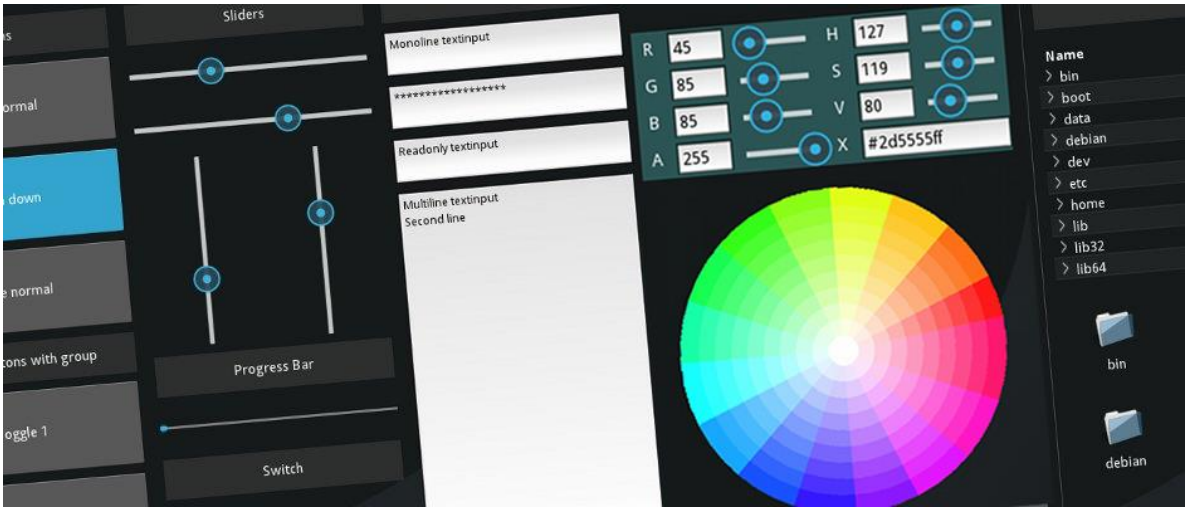
3.4.1. GUI pro Python

Při vývoji softwaru, který bude dále obsluhován pouze zaškolenou obsluhou, je třeba brát v potaz složitost ovládacího rozhraní a jeho ovládání, které by mělo být co nejjednodušší. Proto naprogramuji pro ovládání jednoduché uživatelské rozhraní, které bude obsluha ovládat. Pro Python existují desítky nadstaveb, které zajišťují běh uživatelského rozhraní, jsou velmi různorodé, každá varianta má svá specifika a aktuálně jich je přes 30 v aktivním vývoji[24]. Dokonce lze nalézt GUI, které se zaměřuje pouze na využití ve webových aplikacích. Při našem vývoji se zaměříme převážně na ta GUI, která jsou kompatibilní se systémem Windows, na kterém probíhá vývoj a pravděpodobně na něm poběží i v provozu, ač by se nabízela také varianta s použitím Linuxu.



27 - Knihovna pro GUI TkInter integrovaná do Pythonu [25]

Python má přímo ve své instalaci připraven nástroj pro tvorbu GUI zvaný TkInter, je přímo integrován do Pythonu, takže jeho kompatibilita i použití je bezproblémové, i když jeho vzhled a možnosti jsou značně omezené oproti ostatním. Dále existují knihovny, které je třeba dodatečně přidat - jsou jimi například Kivy, libavg, Pyglet, TraitsUI, pywebview nebo WxPython. Mezi v současné době nejpoužívanější externí grafické rozhraní patří Kivy, které vzniklo v roce 2011 a velkého rozmachu se dočkalo převážně díky modernímu vzhledu, podpoře na platformě Android a iOS a také modernímu přístupu k ovládání. V mém softwaru využiji Tkinter, neboť jeho základní výbava mi pro tyto účely vystačuje a není třeba přidávat další knihovny.



28 - Ukázka GUI vytvořeného pomocí Kivy[26]

Pokud bychom nechtěli vytvářet GUI jen pomocí kódu, je možné využít nástroje pro tvorbu grafického rozhraní, které nám ušetří spoustu práce, dokáží s nimi pracovat i začátečníci, ale ztratíme plnou kontrolu nad vytvářením kódu, což může být v určitých ohledech nevýhoda. Jedním z nich je například *Visual TkInter IDE*, který je určen přímo pro

TkInter, jak je patrno z názvu. Bohužel mnoho těchto nástrojů už je pár let bez větší aktualizace, a tak je jich většina zastaralých, i přesto ale stále použitelných.

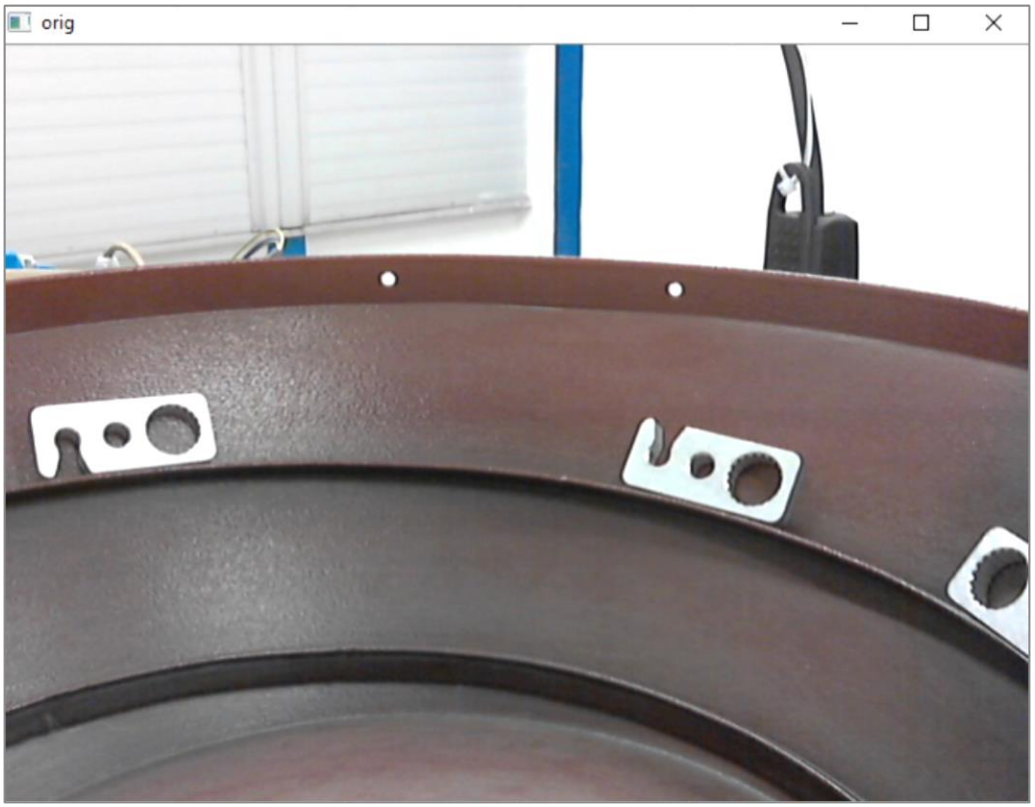
3.4.2. Instalace dalších modulů

Python je hned po instalaci vhodný k programování všech základních programů, občas ale potřebujeme jeho funkce doplnit o další knihovny, které jeho funkčnost rozšíří, například již výše zmíněná knihovna OpenCV či další nástroje pro komunikaci s vnějšími periferiemi a programy. Pro tento účel je v Pythonu zakomponován nástroj zvaný Pip, ten dokáže po zadání příkazu s názvem balíku do konzole Pythonu stáhnout danou knihovnu z internetu. Příkaz pro instalaci balíku přes Pip je „*python -m pip install název_balíčku*“. Uživatel si tak ulehčí práci, neboť nemusí pracně stahovat a importovat knihovny tak, jak to bylo nutné dříve. Stále to neplatí pro všechny knihovny, občas je nutné ji stáhnout ručně, ale pro nejpoužívanější knihovny je tato funkce plně k dispozici.

4. Rozpracování zvolené varianty

Vývoj systému strojového vidění založený na open-source nástrojích

Po zvážení všech variant a prozkoumání podkladů je dalším krokem vytvoření vlastního programu. V následující kapitole se zaměřím na celý průběh projektu a popíši postup od samého počátku až po finalizaci. V tuto chvíli je zajištění instalace Pythonu a všech potřebných částí, včetně prostudování dokumentace k OpenCV a Pythonu, hotové a začnu rovnou na prvních řádcích programu.



29 - Původní snímek bez úprav tak, jak byl zaznamenán kamerou

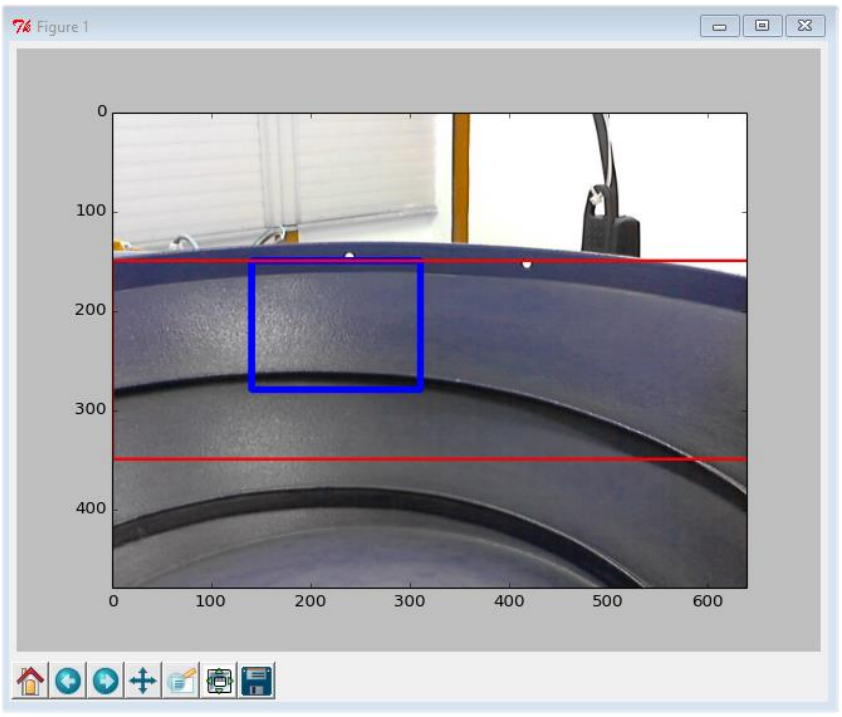
Můj prvotní krok bylo pořízení zkušebních video-podkladů pro tvorbu programu, abych nemusel vše odlaďovat přímo v provozu. Ve firmě ROX spol. s r.o. jsem natočil několik záběrů na zkušebních zásobnících, ve kterých jsem nechal se pohybovat vzorové součásti různě orientované, aby bylo rozpoznání co nejrychlejší. Záběry jsem pořídil pomocí zkušebního algoritmu, který jsem napsal přímo v Pythonu. Ten pouze přijal obraz z kamery a uložil jej na úložiště do videosouboru. Během vývoje jsem umístil do hlavního programu část, která místo kamery načítá obraz ze souboru na disku, ale jde pouze o změnu parametru tak, že během chvíle je možné program přepnout, aby využíval připojenou kameru. Tímto způsobem jsem navrhl a otestoval páteřní část programu, kterou jsem později testoval a odlaďoval v provozu.

Ukázka kódu pro zaznamenání zkušebního videa:

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc('M','J','P','G')
out = cv2.VideoWriter('output_new.avi',fourcc, 18.0, (640,480))
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    out.write(frame)
    cv2.imshow('gray',gray)
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
out.release()
cap.release()
cv2.destroyAllWindows()
```

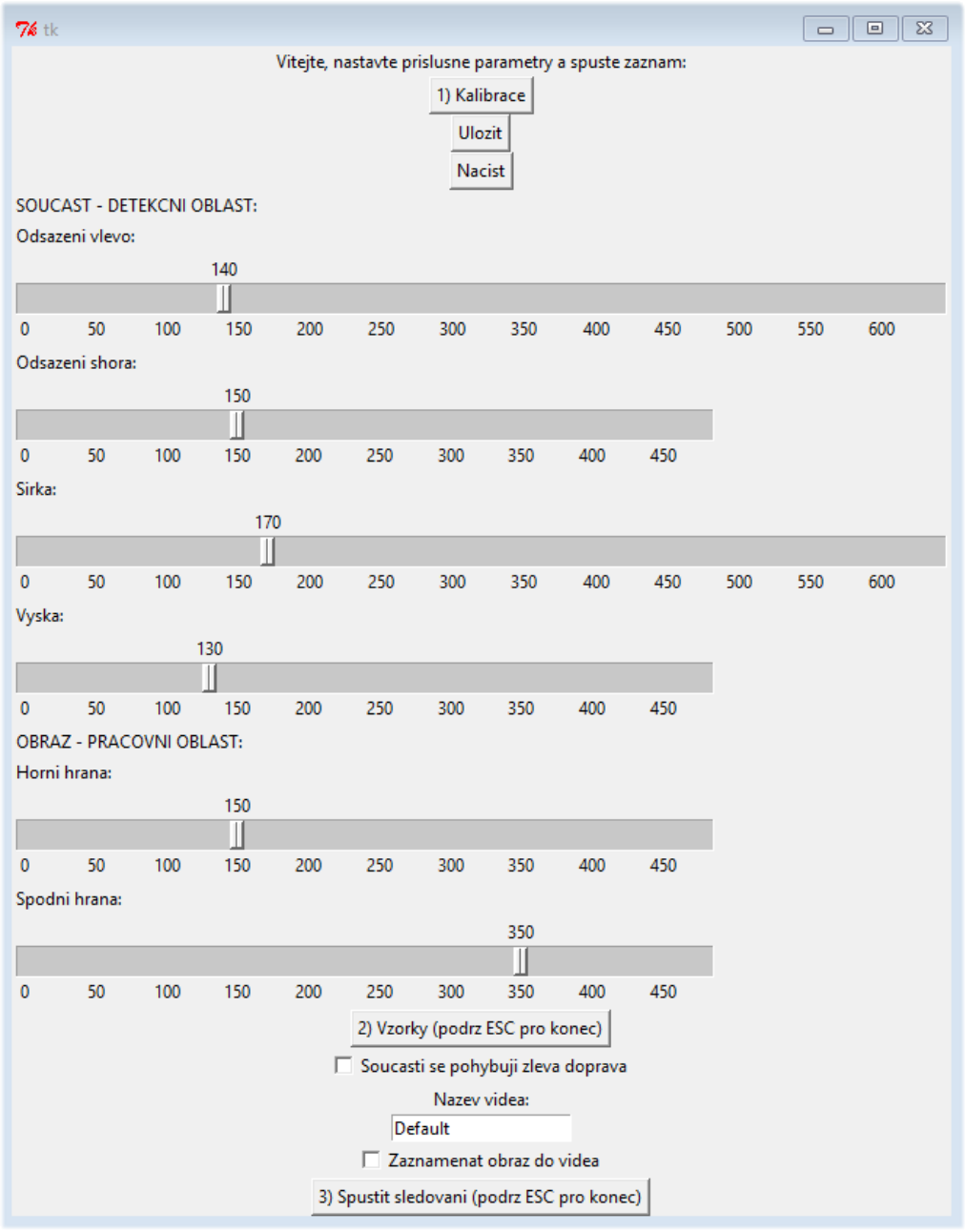
4.1. Kalibrační část a uživatelské rozhraní

Po spuštění programu se uživateli zobrazí úvodní konfigurační menu, které bylo vytvořeno za pomoci vestavěné knihovny TkInter. V něm je zapotřebí provést kalibraci kamery. Uživatel musí po každém umístění kamery do nového místa nebo při změně druhu součásti provést kalibraci.



30 - Kalibrační okno

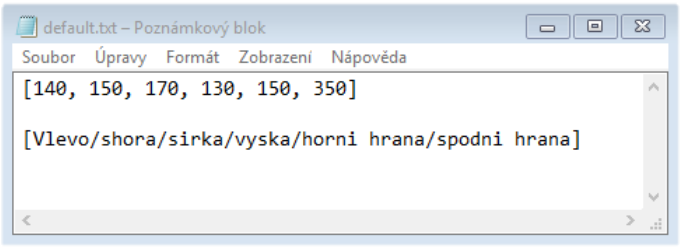
Kalibrace spočívá ve vyhrazení pracovního prostoru, kde procházejí součásti, a také prostoru pro vyhodnocení, který by měl mít stejné rozměry jako snímaná součást. Pracovní prostor je omezen pouze horní a spodní hranou, neboť se počítá s tím, že součásti budou procházet v celé šíři záběru zleva doprava či zprava doleva.



31 - Úvodní menu programu

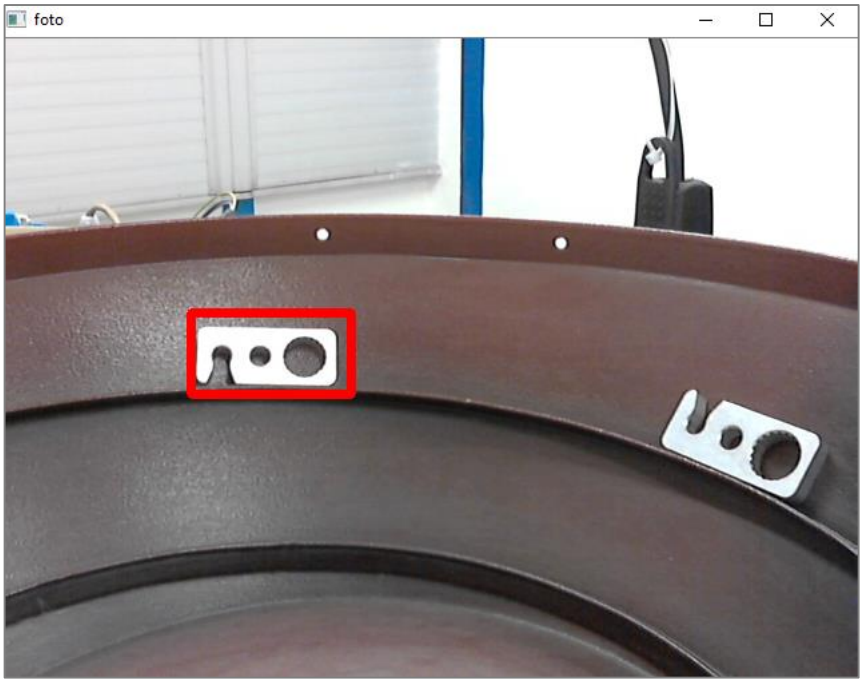
Kalibrace probíhá tak, že program načte obraz z kamery a vykreslí jej do grafického okna, z kterého lze vyčíst souřadnice. Uživatel najde příslušné souřadnice, ty poté nastaví na příslušných posuvnících a opět spustí grafické okno, aby mohl zkontrolovat správnost nastavení. Jakmile uživatel provede kalibraci oblastí, může dané parametry uložit na disk, aby s nimi mohl poté znovu pracovat, například po znovuotevření programu, neboť pokud

program zavřeme, jednotlivé parametry se vrátí do výchozí hodnoty. Samozřejmostí je tedy nejen uložení hodnot, ale i jejich načtení z disku. Program při ukládání na disk využívá souborů s koncovkou *.txt, které je jednoduché upravovat v jakémkoliv textovém editoru. Formát zápisu parametrů do souborů je jednořádkové pole „[Levé odsazení / Odsazení shora/ Šířka oblasti / Výška oblasti / Horní hrana / Spodní hrana]“. První 4 parametry se týkají prostoru pro vyhodnocení a poslední dva jsou pro pracovní prostor.



32 - Konfigurační soubor uložený na disku

Druhým krokem je nafocení vzorků pro danou součást. Tu je třeba zaznamenat ve všech polohách, ve kterých by mohla projet zásobníkem před kamerou. Ve většině případů to budou 2 nebo 4 varianty. Vzorek je nutné umístit do oblasti pro vyhodnocení a stisknout klávesu A, S, D, F podle toho, jak je součást natočena – A je vyhrazena pro OK součást, ostatní pro NOK. Oblast by měla přesně odpovídat velikosti součásti. Vzorek je tímto umístěn na disk a později načten do běžícího procesu rozpoznávání. Uložením vzorku se přepíše předchozí soubor, je tedy nutné ho zálohovat ručně, pokud nebudete chtít tento proces znovu opakovat. Jakmile vytvoříme vzorky, máme téměř vše připraveno pro běh samotného procesu rozpoznávání.



33 – OK součást v poloze pro zaznamenání vzoru

Než spustíme samotné rozpoznávání, je nutné zvolit směr chodu součástí, nastavit jméno pro zaznamenané video a také označit, zda chceme či nechceme záznam ukládat jako video. Pak je již možné spustit proces stisknutím tlačítka „Spustit sledování“, pokud jej budeme chtít znovu vypnout, stačí podržet klávesu ESC. Ve stavovém diagramu v příloze 3 jsou části pro kalibraci zobrazeny zeleně, zaznamenání vzorků tyrkysově a hlavní menu žlutě.

Ukázka principu kódu pro zaznamenání šablon:

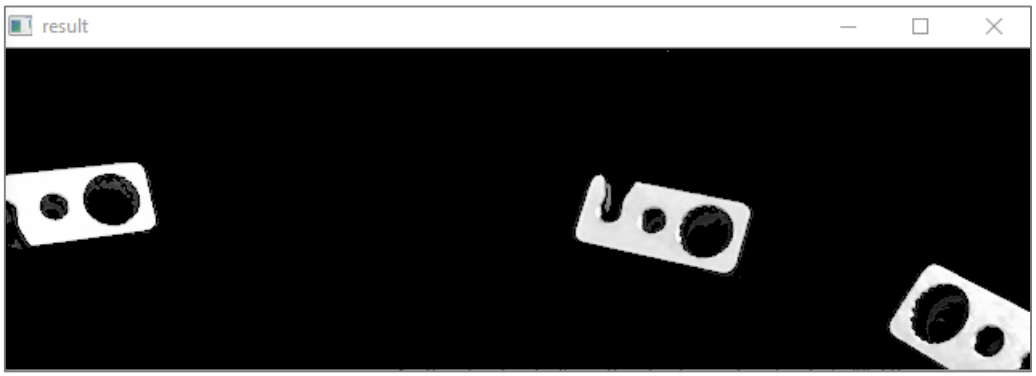
```
while (1):                                  // Smyčka pro kontinuální záznam
       frame_cam = cap.read()               // Načtení aktuálního snímku
      if frame_cam is None:                 // Pokud nelze snímek načíst
            print 'Konec'                   // Výpíše zprávu a ukončí smyčku
            break
      crop_kalibr = frame_cam[self.var2.get():self.var4.get()+self.var2.get(),
      self.var1.get():self.var3.get()+self.var1.get()]   // Oříznutí snímku podle součásti
      cv2.imshow('foto_C',crop_kalibr)              // Zobrazení výřezu
      if cv2.waitKey(1) & 0xFF == ord('a'):         // Detekuje povel na klávesnici
            cv2.imwrite('kalibr0.png', crop_kalibr)     // Zapíše do souboru
            k=0                             // Vynuluje počítadlo
      …..                                   // Takto stejně pro S, D, F
      if k<10:
          cv2.putText(frame_cam,'_____OK!',(0,25), font, 1, (200,255,155), 2, cv2.LINE_AA)
          k=k+1             // Po stisknutí klávesy a zaznamenání snímku se na obrazovce
                               vypíše potvrzující hláška
      cv2.rectangle(frame_cam,(self.var1.get(),self.var2.get()),(self.var3.get()+self.var1.
      get(),self.var4.get()+self.var2.get()),(0,0,255),5)
                                        // Zobrazení hraničení prostoru do obrazu
      cv2.imshow('foto',frame_cam)      // Zobrazení aktuálního snímku
      q = cv2.waitKey(1) & 0Xff         //Klávesa ESC ukončí smyčku
      if q == 27:
            break
cv2.destroyAllWindows()                 // Po ukončení smyčky jsou zavřena všechna okna
```

4.2. Princip čtení obrazu

Po spuštění smyčky kontroly dochází k načtení snímku ze souboru nebo kamery, obraz je oříznut podle pracovního prostoru, který jsme definovali v konfiguračním prostředí. To ulehčí výpočetní práci a zrychlí běh programu. Vzhledem k tomu, že program musí běžet v reálném čase, je to velmi důležitý faktor. Po oříznutí provedeme důležitý krok pro separaci pozadí. Po vyzkoušení všech předpřipravených separátorů pozadí jsem se u některých setkal s technickými problémy a u některých s velkým poklesem výkonu. Rozhodl jsem se použít vlastní jednoduchý algoritmus, který spoléhá na statické a neměnné umístění kamery a stabilní

světelné podmínky ve zkušební stanici. Program při svém spuštění načte snímaný prostor bez jakéhokoliv dílu a snímek uloží na disk jako vzor. Tyto úkony shrnují inicializační fázi.

34 - Pracovní oblast s vyznačenou oblastí pro rozpoznání před separací pozadí



35 - Obraz po separaci pozadí obsahuje pouze pohybující se součásti

V tomto okamžiku program převezme další snímek a jako první operaci provede oříznutí, jak bylo popsáno výše. Snímek je následně porovnán se vzorem a je vytvořena rozdílová bitová maska, která dle zadaného kritéria porovná rozdíl obou snímků a do masky tedy zanese jakýkoliv pohybující se předmět, který tam při inicializaci nebyl. Jelikož obraz z kamery ale může obsahovat šum a snímky nejsou zcela identické, dojde k zobrazení šumu i na masku. Je tak nutné aplikovat binární thresholding, který snímek vyčistí a zbydou jen obrysy pohybujících se větších celků. Následně je mapa aplikována na zdrojový snímek a dojde k odstranění pozadí. To nám následně pomůže najít obrysy a samotné součásti. Tato část je ve stavovém diagramu v příloze 3 vykreslena oranžově.

Ukázka principu kódu:

```
_, frame_cam = cap.read()                          // Načtení snímku z kamery
frame = frame_cam[self.var5.get():self.var6.get(), 0:640]      // Oříznutí
cv2.imwrite('kalibr.png',frame)                    // Záznam kalibračního 1. snímku
while (True):                                      // Smyčka pro kontinuální snímání
      frame_cam = cap.read()
      frame = frame_cam[self.var5.get():self.var6.get(), 0:640]
      template = cv2.imread('kalibr.PNG',1)        // Načtení kalibračního vzoru
```

```
diff = cv2.absdiff(frame,template,100)      // Porovnání aktuálního snímku a vzoru
      thresh0 = cv2.threshold(diff, 40, 255, cv2.THRESH_BINARY)[1]  // Thresholding
      thresh = cv2.cvtColor(thresh0, cv2.COLOR_BGR2GRAY)  // Převod do stupňů šedi
res = cv2.bitwise_and(frame,frame, mask= thresh) // Aplikace masky a separace pozadí
                                                    – výsledek
```

4.3. Sledování pohybu objektů v obraze

Jakmile je separováno pozadí, je možné provést detekci objektů na snímku. V této fázi vezmu výsledný snímek z minulé kapitoly 4.2 a aplikuji na něj funkci *findContours.* Ta nalezne a umístí do pole všechny kontury ploch, které se na snímku nacházejí. Následně použijeme for cyklus pro každou konturu. Nejprve zjistíme její plochu, pokud je plocha menší než zadaná mez, která je určena z rozměrů vzorové součásti, není možné, aby se jednalo o součást a je tedy automaticky potlačena. Pokud splní tuto podmínku, předpokládá se, že se jedná o objekt na dráze zásobníku a je určeno jeho těžiště. V každém snímku je připraveno pole souradnice[], které obsahuje informace o čísle aktuálního snímku, X a Y souřadnici polohy těžiště, číslo součásti a její orientaci. Orientace a číslo součásti jsou ve většině případů převzata z předchozího snímku, pokud ale přijde do obrazu nová součást, dostane číslo 0 a orientaci neznámou „-1“.

36 - Detekce kontury a očíslování

Po nalezení všech kontur musíme provést synchronizaci s předchozím snímkem. V paměti zůstává vždy uložena aktuální souřadnice, ale také její kopie, která obsahuje informace o předchozím snímku a je nutné zajistit, aby například číslo součásti a informace o orientaci zůstala zachována a nepřesunula se na jiný objekt. V každém snímku je tedy detekováno, zda nějaký objekt přibyl nebo naopak z obrazu zmizel. Pokud se počet součástí nezměnil nebo se snížil, je vše jednoduché a dojde pouze k ověření, zda číslo součásti leží v podobné oblasti jako na předchozím snímku a pokud ne, je provedena korekce.

Pokud však do obrazu přibyla nová součást, je nutné nejprve nalézt objekty, které odpovídají těm z předchozího snímku. To se provede porovnáním blízkých souřadnic a následné přiřazení vlastností z předchozího snímku. Tím zajistím kontinuitu informací o dané součásti. Dále si označím objekty, které vstoupily do obrazu, a přiřadím jim dočasnou hodnotu 0, která značí, že objekt v předchozím snímku nebyl nalezen. Na konec už jen přiřadím nové součásti číslo, které dle pořadí následuje a označím jej jako nerozpoznaný díl. To provedu pokaždé, když přijde nová součást nebo se souřadnice liší o více, než je dle podmínek připuštěno. V tom případě pravděpodobně jeden díl odjel a přijel v ten samý okamžik další. To je velmi málo pravděpodobné, ale uskutečnitelné. Ve stavovém diagramu z přílohy 3 je tento úsek vyznačen modře.

Ukázka principu synchronizace (Uvnitř While oddílu):

```
cnts = cv2.findContours(thresh_shape.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                                            // Nalezení kontur na snímku
for c in cnts:
        M = cv2.moments(c)                  // Pro všechny kontury vypočteme jejich
moment
        if M["m00"] >= det_hranice:         // Pokud je kontura dostatečně velká
            cX = int(M["m10"] / M["m00"])   // Vypočteme X a Y souřadnici těžiště
            cY = int(M["m01"] / M["m00"])
            souradnice[shape][0] = screen   // Každému obrysu patří jeden prvek pole
            souradnice[shape][1] = cX      // Který obsahuje časovou značku a souřadnice
            souradnice[shape][2] = cY
            if shape_prev>shape: //Pokud se nezvětšil počet dílů a souř. odpovídá předch.:
              if abs(souradnice[shape][1] - souradnice_prev[shape][1]) <=safe:
                 souradnice[shape][3] = souradnice_prev[shape][3]
                                            //Nastavíme předchozí hodnoty
                 souradnice[shape][4] = souradnice_prev[shape][4]
            else:                          // Každý nový díl je označen výchozí hodnotou
                souradnice[shape][3] =0
                souradnice[shape][4] =-1
            shape = shape+1
//SYNCHRONIZACE
if shape <= shape_prev:                     //Pokud je součástí stejně nebo méně:
    for c in range(shape):     // Zjistím, jestli každá součást odpovídá součásti se
                               stejným indexem z předchozího snímku
            if abs(souradnice[c][1] - souradnice_prev[c][1]) >safe:    //Pokud NE
                for i in range(shape_prev):
                    if abs(souradnice[c][1] - souradnice_prev[i][1]) <= safe:
                        souradnice[c][3]=souradnice_prev[i][3]
                        souradnice[c][4]=souradnice_prev[i][4]
                           //Najdu jinou součást, která byla v bezprostředním okolí
if shape > shape_prev:                          // Pokud součást přibyla:
    for c in range(shape_prev):     // Nejprve porovnáme součásti na předchozích a
                                      aktuálních indexech v poli, zda odpovídají
            if abs(souradnice[c][1] - souradnice_prev[c][1]) > safe:
                found=0
                for i in range(shape_prev):
                    if abs(souradnice[c][1] - souradnice_prev[i][1]) <= safe:
                        souradnice[c][3]=souradnice_prev[i][3]
                        souradnice[c][4]=souradnice_prev[i][4]
                        found=1     // Pokud součásti neodpovídají, nalezneme v předchozích
                                    jinou blízkou a pokud nalezneme tam, přiřadíme
                                    informace z předchozího snímku
```

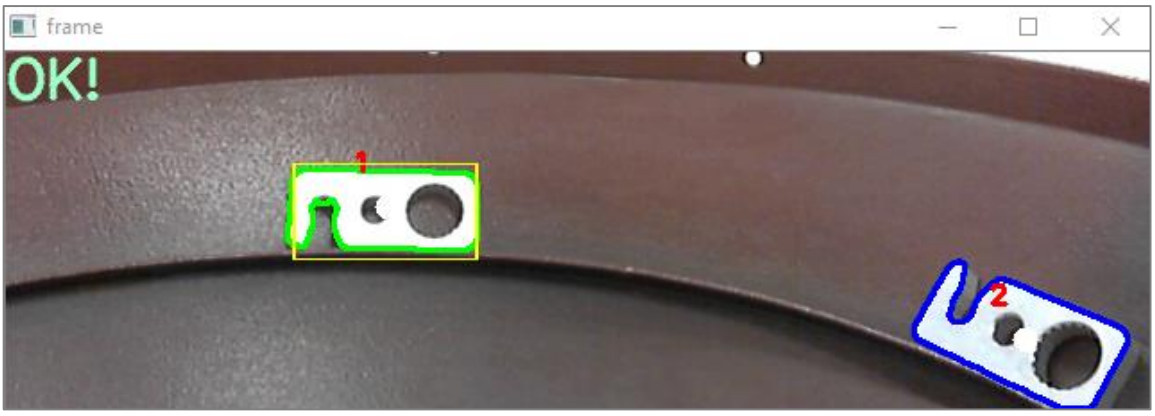
```
                if found==0:
                    souradnice[c][3]=casti          // Pokud nenalezneme, přiřadíme
                                                    součásti nové číslo podle pořadí
                    casti=casti+1                   // Zvedneme počítadlo součástí
                    souradnice[c][4] =-1            // Nastavíme neidentifikovaný díl

    rozdil = shape - shape_prev
    for c in range(rozdil):
        for i in range(shape_prev):
            if abs(souradnice[shape-1-c][1] - souradnice_prev[i][1]) <= safe and
               abs(souradnice[shape-1-c][2] - souradnice_prev[i][2]) <= safe:
                souradnice[shape-1-c][3]=souradnice_prev[i][3]
                souradnice[shape-1-c][4]=souradnice_prev[i][4]
                                        // Následně porovnáme nové součásti
                                         s předchozími, jestli se některá nepřesunula na
                                         jiný index. Pokud ne zůstává ji číslo 0
for c in range(shape):
        if souradnice[c][3]==0:
                    souradnice[c][3]=casti
                    casti=casti+1       // Nakonec zjistíme, jestli má některá součást
                                        stále číslo 0 a pokud ano přiřadíme nové číslo
                                        součásti…
                    souradnice[c][4] =-1 // … a označíme jako neidentifikovaný díl.
if shape == shape_prev:
     for c in range(shape):
            if abs(souradnice[c][1] - souradnice_prev[c][1]) >safe/4 and
            (souradnice[c][3] == souradnice_prev[c][3]):
                    stop = stop + 1      // Detekce zaseknutí součásti
```

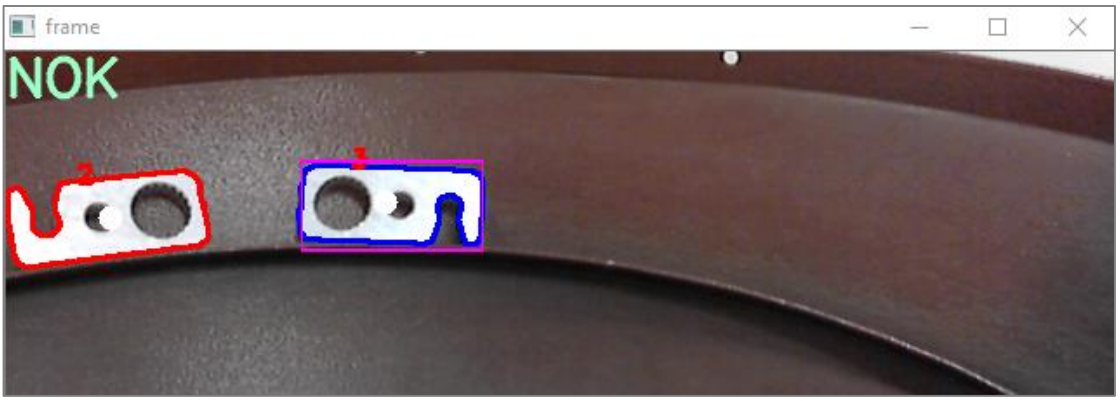
4.4. Princip zjištění orientace součásti

Během konfigurační části dochází k nastavení pracovního prostoru a také prostoru pro vyhodnocení. To je prostor, kde je součást nasnímána ještě během přípravy na sledování, nejlépe ve všech polohách, kterými by mohl zásobníkem projíždět, ale hlavně v té správně orientované a žádané. Součásti jsou vyfoceny do vzorových souborů a s těmi jsou potom dále porovnávány zaznamenané snímky. Jakmile je spuštěno rozpoznávání, program načte oříznutý snímek z kamery, který je ale ještě převeden do odstínů šedi. Tento proces je již součástí zpracování z předchozí kapitoly.

V programu již máme načteny vzorové předlohy součástí, taktéž v odstínech šedi a tak můžeme v pomocí funkce *matchTemplate* porovnat vždy aktuální snímek se vzorovými a v pracovní oblasti hledat shodu. Jelikož máme v algoritmu nastavenou požadovanou přesnost 95%, dojde k nalezení shody ve většině případů až okolo prostoru pro vyhodnocení, pokud bych chtěl provést detekci dříve, musel bych nastavit menší přesnost, ale pak už by mohlo dojít k záměně jednotlivých vzorů a k falešné detekci. Vzhledem k tomu, že není požadována shoda 100%, je nalezeno většinou mnoho duplicitních a překrývajících se shod, které je třeba odfiltrovat pro správné počítání a detekci součástí.



37 - Detekce OK součásti (1) a její označení



38 - Detekce NOK součásti (3) a označená NOK součást (2)

Odstranění duplicitních součástí spočívá ve vytvoření pole o počtu elementů rovných počtu všech detekcí. V poli jsou umístěny souřadnice pro každou jednu detekci a ty jsou s pomocí *for* cyklu porovnány mezi sebou. Pokud je vzdálenost dvou souřadnic menší než je polovina šířky nebo délky součásti, jedná se s největší pravděpodobností o stejnou součást a duplicitní souřadnice je z pole odstraněna. Nakonec vznikne pouze pole se všemi jedinečnými součástmi. Tato součást je zanesena do obrazu pro vizuální kontrolu. Při rozpoznání OK součásti dojde k porovnání souřadnice součásti se souřadnicemi součástí rozpoznaných podle obrysů, a pokud je shoda s OK, zapíše se tato informace do pole „souřadnice“, které shlukuje informace o procházejících dílech. Obrys dílu se pak zbarví do zelena, značící OK stav. Pokud je detekován NOK, zbarví se červeně. Tento úsek kódu je ve stavovém diagramu v příloze 3 vyobrazen fialově.

<u>Ukázka principu kódu (Uvnitř While oddílu):</u>

```
det1 = cv2.matchTemplate(frame_orig,template1,cv2.TM_CCOEFF_NORMED)
                                                  // Vyhledání šablony ve snímku
threshold = 0.95                                   // Procento shody
loc = np.where( det1 >= threshold)   // Nalezení souřadnic vyšších než je procento shody
for pt in zip(*loc[::-1]):                         // Pro všechny body:
```

```
                part[i][0]=pt[0] + w2/2          // Přiřadíme souřadnice …
                part[i][1]=pt[1] + h2/2          // … středu součásti do pole
                i=i+1
                cv2.rectangle(frame_orig, pt, (pt[0] + w1, pt[1] + h1), (0,255,255), 1)
                                                 // Vykreslení obdélníku do obr.
                cv2.putText(frame_orig,'OK!',(0,25), font, 1, (200,255,155), 2,
                                                 cv2.LINE_AA)  // Vykreslení textu „OK"
for p in range(i):                               // Všechny souřadnice
            for k in range(i):                   // Porovnám s ostatními
                if k!=p:                         // Kromě „sami se sebou"
                    if abs(part[p][0]-part[k][0])<w1/4 and abs(part[p][1]-
                                                 part[k][1])<h1/4:
                        part=np.delete(part,(i-p-1),axis=0) // Pokud je vzdálenost
                                                            menší než půl součásti - smaž
distance_part=math.sqrt(w1*w1+h1*h1)/4           // výpočet rádiusu okolí součásti
for a in range(shape):                           // pro všechny součásti
            for b in range(part.shape[0]):           // a všechny obrysy
                x=abs(souradnice[a][1]-part[b][0])   // zjisti vzdálenost x a y
                y=abs(souradnice[a][2]-part[b][1])
                distance=math.sqrt(y*y+x*x)      // Vypočet vzdálenosti
                if distance<distance_part:       // Pokud je vzdálenost v radiusu
                    souradnice[a][4] = 1         // Zapiš do souřadnice OK součást (1)
```

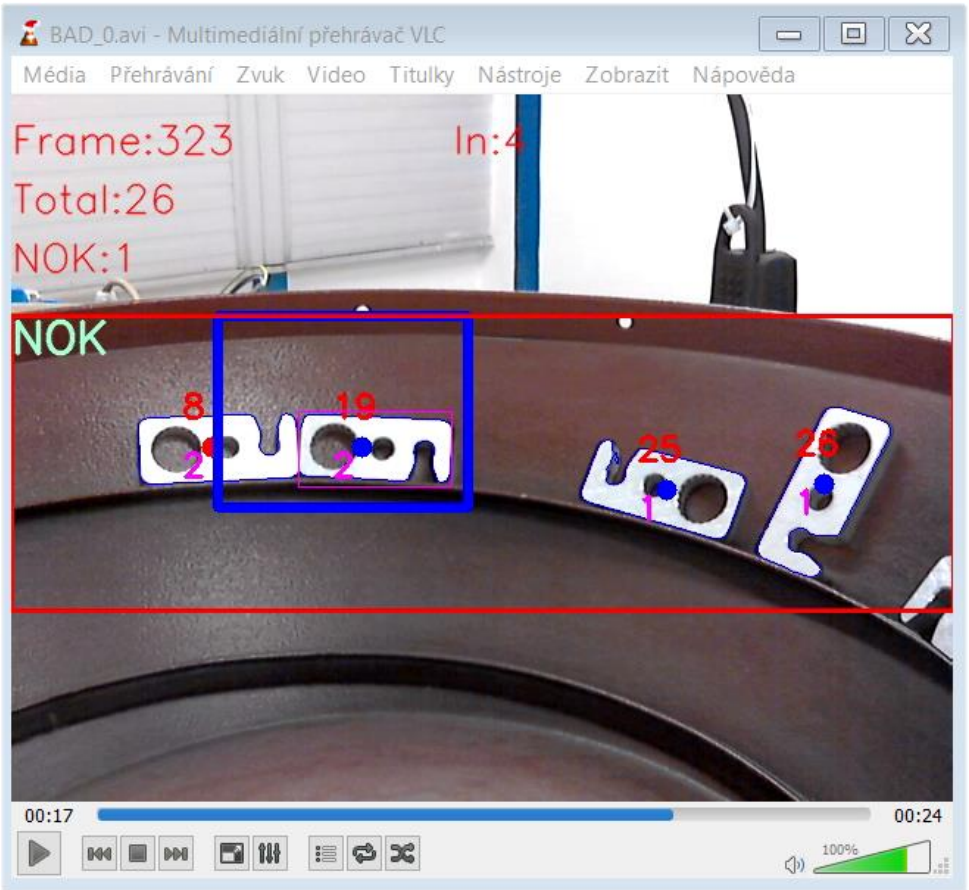
4.5. Výstup do videosouborů a protokol o sledování

Jedním z kritérií bylo zaznamenání chybných průchodů do videosouboru a také výstupní protokol. Knihovna OpenCV má připraven příkaz *VideoWriter* pro ukládání videa do souboru, který využiji. V úvodním menu je možné nastavit, zda budeme chtít během snímání zaznamenat i obraz. Ve většině případů to bude nejspíše nutné, ale například pro účely testování není tento krok zcela nezbytný. Pokud tedy tuto možnost zaškrtneme, máme v našem kódu připraven blok, který se postará o uložení každého snímku do videosouboru. Uložen je veškerý obsah společně s info-grafikou pro lepší přehlednost, pokud by například byla chyba pouze na straně programu, aby to bylo možné odhalit. Název videosouboru je dán názvem, který jsme zvolili v menu a délka každého souboru je maximálně 30 sekund pro snadnější prohlížení. Soubory jsou chronologicky očíslovány např. pro název DP:

- DP_0.avi
- DP_1.avi
- DP_2.avi …

Každé video průběžně zobrazuje aktuální snímek (Frame), počet zaznamenaných kusů (Total) a počet prošlých NOK součástí. V druhém sloupci je zobrazen počet součástí v obraze

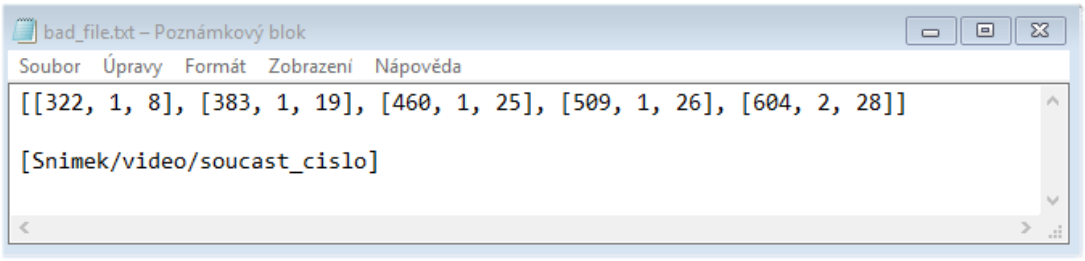
(In). Na disk jsou v reálném čase ukládány všechny snímky, zachovány však na konci sledování zůstanou jen ty, které obsahují průchod NOK součásti. Pokud tedy během jedné hodiny projde jeden špatný díl, budeme mít po hodině pouze 90 vteřin z okolí tohoto okamžiku a nemusíme sledovat celý záznam. Princip postupného mazání je jednoduchý, program při zakládání nového videa vždy zkontroluje pole „vider“(video_error), které obsahuje 2 elementy, které vypovídají informace o aktuálním a předchozím souboru. Pokud je během kontroly hodnota [0,0], značí to skutečnost, že během nahrávání posledního a předposledního videa nebyla nalezena žádná NOK součást a předposlední video je promazáno. Pokud je stav jiný, znamená to, že nemůžeme video smazat. Tato kontrola probíhá pokaždé při zakládání nového videa a zaručí nám 30 vteřinový náhled před a po průchodu NOK součásti.



39 - Náhled výsledku v přehrávači

Společně s videozáznamem je na konci rozpoznávání zapsán také konečný výpis průchodů všech NOK součástí do *.txt souboru. Program za běhu vytváří list hodnot, které vždy doplní, pokud detekuje, že se některá ze součástí změnila hodnota stavu z „nerozpoznáno“ na NOK. V listu je pro každý průchod zaznamenáno číslo součásti společně s číslem snímku a videa, kde je chyba zaznamenána, aby mohla být snadno diagnostikována. Jakmile rozpoznání skončí, je list vypsán do souboru. Na obrázku 40 můžeme vidět příklad

výstupu do textového souboru a na obrázku 39 můžeme vidět průchod první NOK součásti. Tento úsek kódu je ve stavovém diagramu v příloze 3 vyobrazen šedě.



40 - Výpis průchodů NOK součástí

Ukázka principu tvorby listu průchodů NOK součástí:

```
bad = []                                                    // Deklarace listu
while (True):                                               // Smyčka rozpoznávání
      for a in range(shape):                                // Pro všechny součásti
          if souradnice[a][4] == -1 and ((souradnice[a][1]<self.var1.get() and left ==0) or
             (souradnice[a][1]>(self.var1.get()+self.var3.get()) and left==1)):
      // Pokud je součást nedetekována a nachází se za detekčním pásmem – označíme jako NOK
              souradnice[a][4]=0                            // Značka pro NOK
              if not zaznam:     // Pokud neprobíhá nahrávání, bude v zápise číslo videa -1
                  video = -1
              temp=[screen,video,souradnice[a][3]]//Zapíšeme snímek,č.videa,číslo součásti
              bad.append(temp)                              //… do listu bad
              if zaznam:  // Pokud nahrávám, zapíšu značku, aby nedošlo ke smazání videa
                  vider[video%2]=1
bad_file = open("bad_file.txt","w")          // Na konci provedu zápis do bad_file.txt
bad_file.write(str(bad))
bad_file.write('\n\n'+'[Snimek/video/soucast_cislo]')
bad_file.close
```

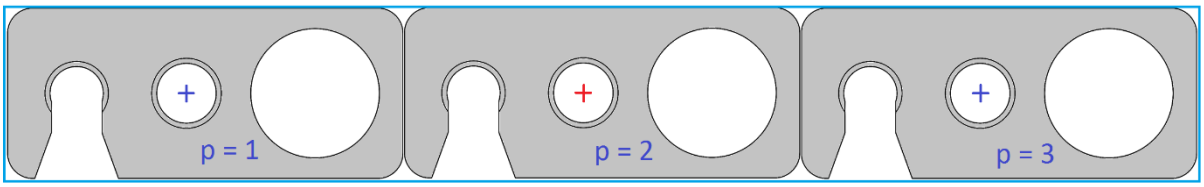
Ukázka principu záznamu:

```
if zaznam:                                      // Blok je spuštěn jen pokud je zapnut záznam
      fourcc = cv2.VideoWriter_fourcc('M','J','P','G')   // Kodek pro ukládání
      timing = 0                                        // Inicializace proměnných
      video = 0
      vider=[0,0]
while (True):                                           // Smyčka sledování
      if zaznam:
          fps = 18.0                                    // Počet snímků za sekundu pro video
          name=self.video_name.get()                    // Název videa z menu
          if 'out' not in locals():                     //Pokud není spuštěno nahrávání, spusť
              out = cv2.VideoWriter(name+'_'+str(video)+'.avi',fourcc, fps, (640,480))
  // Nahrávání spuštěno – informace o názvu, kodeku, počtu snímků za sekundu a rozlišení
          else:                                         // Pokud nahrávání již běží
              if timing>(30.0*fps):                     // A počítadlo dosáhlo 30s
                  if video!=0 and vider==[0,0]://Pokud je předchozí video určeno k smazání
```

```
                os.remove(name+'_'+str(video-1)+'.avi')      // Smaž jej
            if (vider[1-video%2]==-1):       // Pokud je jedna ze značek -1 (Po NOK)
                vider[video%2]=0               // Pro příští snímek nastav 0
            if (vider[video%2]==1):            // Pokud je jedna ze značek 1 (NOK)
                vider[1-video%2]=-1           // Pro příští snímek nastav -1 (Po NOK)
            if (vider[video%2]==0 and vider[1-video%2]==-1):
                vider=[0,0]                    // Pokud v aktuálním ani předchozím
                                                  snímku nebyla NOK součást, nastav
                                                  hodnotu 0,0
            video=video+1
             out.release()                     // Přechod na další video
             timing=0
             out = cv2.VideoWriter(name+'_'+str(video)+'.avi',fourcc, fps, (640,480))
        timing=timing+1
        out.write(frame_cam)                   // Záznam aktuálního snímku
if zaznam:
    out.release()                              // Na konci programu vypni stream
```

4.6. Rozpoznání objektů v těsné blízkosti za sebou

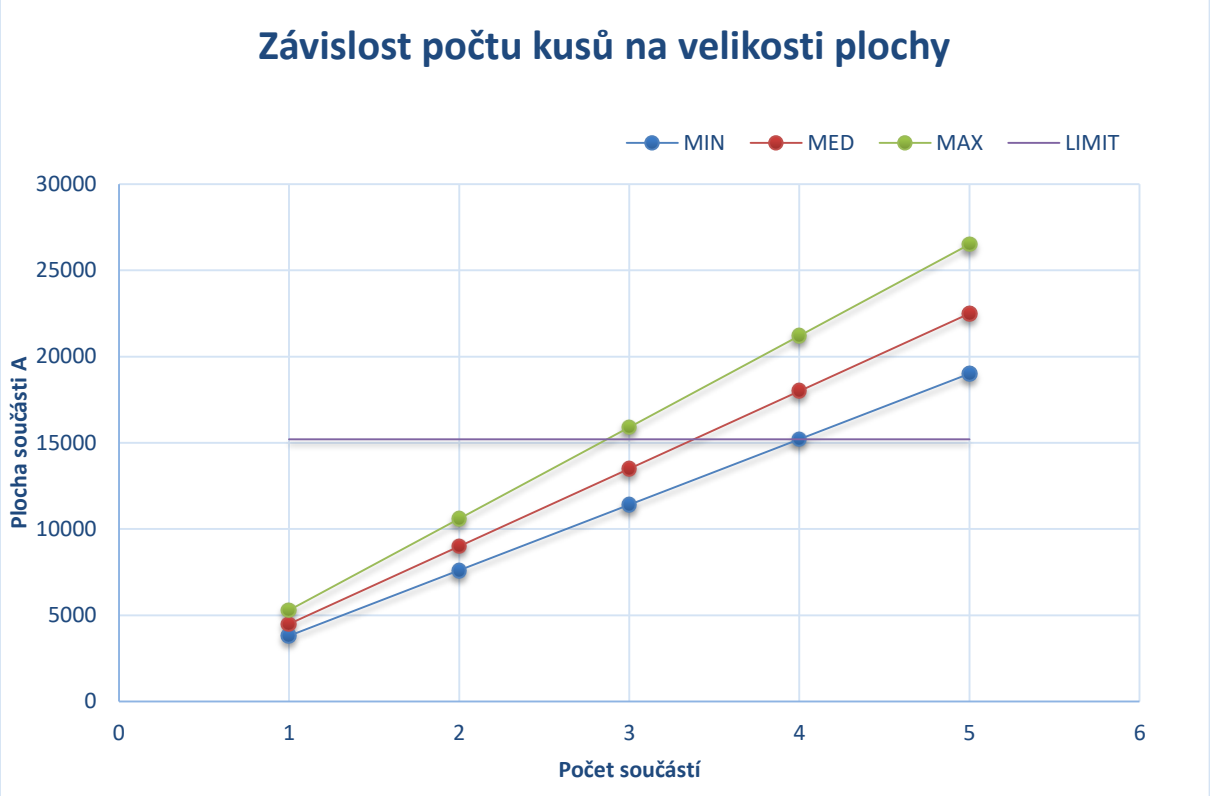
V tomto stavu můžeme program již testovat, neboť obsahuje všechny náležitosti, které jsou potřeba pro jeho základní běh a rozpoznávání. V tomto stavu je schopen rozpoznat, zaznamenat a uložit do výpisu všechny NOK součásti, které by se na výstupu objevily. Možnost použití je ale silně omezena jen na ta vibrační zařízení, kde jsou součásti odděleny od sebe a po dráze nejedou v těsném závěsu za sebou, těch je ale velmi málo. Pro použití na více zařízeních je nutné rozpoznat i více součástí za sebou.



41 - Zobrazení detekce 3 součástí vedle sebe

Rozpoznání provedu na základě výpočtu plochy jedné součásti při kalibraci. Jakmile je načtena šablona OK součásti, je vypočtena z její výšky a šířky přibližná velikost plochy součásti ve čtverečných pixelech. Tento údaj můžeme porovnat s plochou součásti vypočtenou z kontury, která byla popsána v kapitole 4.3. Například řekněme, že zvolená součást má plochu A = 4500 pixelů^2, pokud budu sledovat její průchod obrazem, dostal jsem experimentálně hodnoty 3800-5300 pixelů^2. Tento interval jen přibližně roven A ± A/6. Pokud daná kontura má plochu v intervalu okolo 9000 pixelů^2, mohu s velkou pravděpodobností říci, že se jedná o dvě součásti jedoucí za sebou. Pokud bych to měl zobecnit, mohu takto sledovat *k* součástí, pokud jejich plocha bude v intervalu $k.A \pm A/6$ pixelů^2. S každou další součástí roste také ale

rozsah, ve kterém se intervaly pohybují. Graf 1 zobrazuje, že při průběhu 4 a více součástí za sebou dojde k překrytí intervalů a není tedy zcela jednoznačné, kolik součástí je na obrázku. Z tohoto důvodu je třeba omezit pracovní plochu na maximálně 3 součásti vedle sebe. Další cestou může být natočení kamery tak, aby součásti byly po celou dobu průchodu po dráze vzdáleny stejně od objektivu a nedocházelo tak ke zkreslení vlivem vzdálenosti. Tímto způsobem je možné snížit velikost intervalu a dosáhnout tak větší přesnosti a zvýšit detekci. To je v praxi velmi žádané, neboť při průchodu více součástí za sebou se může stát, že bude procházet více dílů za sebou bez mezery a dojde tak ke zmatení programu, neboť ten nebude vědět, kde přesně jednotlivé součásti začínají a kde končí. V tomto případě bude muset spoléhat jen na kontrolu pomocí porovnání se vzorem a nebude možné spolehlivě počítat součásti, proto je vždy lepší dosáhnout vyšší detekce např. okolo 4 až 5 součástí.



Graf 1 - Závislost počtu kusů na velikosti plochy součásti

Když znám hodnotu plochy součástí a hodnota se pohybuje v intervalu, který mohu považovat za spolehlivý, je nutné danou oblast rozdělit mezi jednotlivé součásti. Pokud znám velikost oblasti a její těžiště, které se pro každou oblast vždy vypočítává, mohu alespoň přibližně popsat umístění těžiště pro jednotlivé součásti ve shluku. Za předpokladu, že součásti se pohybují po dráze ve vodorovném směru, jsem odvodil vzorec pro výpočet polohy jednotlivých těžišť:

$$X_P = C_x + \left(-\frac{k}{2} - mod\left(\frac{k}{2}\right) + p\right) * w_1 - \left|mod\left(\frac{k}{2}\right) - 1\right| * \frac{w_1}{2} \quad (5)$$

Kde:

X_P	Poloha těžiště p-té součásti v ose X (Na obr. 41 modře)
C_x	Poloha těžiště shluku součástí v ose X (Na obr. 41 červeně)
k	Počet součástí ve shluku (Na obr. 41 – k = 3)
Mod()	Modulo - zbytek po dělení
p	Pořadí součásti ve shluku (zleva doprava – viz obrázek 41)
w_1	Šířka jedné součásti

Jakmile jsem provedl teoretickou analýzu problému a našel řešení, je nutné aplikovat výše uvedené skutečnosti do programu. Je nutné provést změny převážně v grafickém zobrazení – nejdříve byl pro označení součástí zbarven obrys součásti, ale to teď již není možné a tak jsem se rozhodl pro označení prostřednictvím zbarvení těžiště součásti, které přesněji označuje právě jednu součást. Zásah do samotného programu vyžadoval přidání dalšího for cyklu ihned za výpočet těžiště a plochy součásti / shluku součástí. Jakmile je spočtena plocha a z ní počet součástí, je ve for cyklu vložen výše uvedený vzorec, který pro každou součást uvnitř shluku vypočte těžiště a přidá jej do pole souřadnic jako novou součást. Nyní již tedy nepočítáme počet součástí dle množství obrysů, ale množství těžišť. Každá součást má také ve svém elementu přidáno pole, které označuje, zda byla součást ve shluku a pokud ano, kolik součástí se ve shluku objevilo. Tato informace je poté zobrazena ve videu. To slouží převážně pro diagnostiku případných problémů, které mohou s touto metodou nastat.

Během testování jsem zjistil, že tato metoda je velmi nápomocná a dokáže součásti označit velmi spolehlivě. Jediná výjimka nastává, když se součásti objevují v zorném poli kamery, která nejprve vidí jednu součást a po chvíli teprve přijíždí další a plocha se postupně plynule zvětšuje. Dostaneme se tak na hranice jednotlivých intervalů, proto jsem na začátku dráhy umístil tzv. slepé místo, součást a její obrys je detekován, ale dokud se její těžiště neposune dostatečně daleko od okraje zorného pole, nejsou součástem přiřazována žádná čísla a algoritmus je ignoruje, dokud nemají dostatečně velkou plochu pro jednoznačnou identifikaci. Pokud bychom vzali část programu z kapitoly 4.3, většina kódu zůstane stejná jen mezi řádky vypsané níže vložíme nový blok kódu. Tento úsek kódu je ve stavovém diagramu v příloze 3 vyobrazen červeně.

```
cX = int(M["m10"] / M["m00"])          // Vypočteme X a Y souřadnici těžiště
cY = int(M["m01"] / M["m00"])
     --- ZDE VLOŽÍME NOVÝ BLOK ---
```

```
souradnice[shape][0] = screen          // Každému obrysu patří jeden prvek pole
```

Ukázka principu detekce více součástí ve shluku:

```
k=int((M["m00"]/plocha_ref)+0.5)       // Detekce počtu součástí
mod=k%2                                // Zjištění sudého / lichého počtu
for p in range(1,k+1):                 // Pro každou zjištěnou součást ve shluku:
    dist=int(cX+((-(k/2)-mod+p)*w1)-(0.5*w1*abs(mod-1)))       // Najdi souřadnici
    if (dist < 0.6*w1 and left==0)or(dist > 640-(0.6*w1) and left==1):
        continue     // Pokud je těžiště v slepé oblasti, přeskoč na další díl
    temp=[]
    for j in range(6):
        temp.append(0)
    souradnice.append(temp)              // Přidání souřadnice do pole souřadnic
    souradnice[shape][5]=k               // Přiřazení hodnot
    souradnice[shape][1] = dist
    souradnice[shape][2] = cY
```

4.7. Zajištění prostředí pro chod programu

V tomto stavu je již program připraven pro nasazení do provozu a testování. Poslední částí, která je nutná zajistit je snadná instalace na další počítače, které budou použity pro testování. Vzhledem k povaze programovacího jazyka Python, je nutné, aby počítač, který bude chtít provozovat aplikaci, měl nainstalovaný Python ve verzi minimálně 2.7.12 a do něj importovány následující knihovny (nejlépe instalovat v takovém pořadí jak jdou za sebou):

- Numpy 1.11.2 – Knihovna pro práci s maticemi
- **OpenCV 3.1.0 – Knihovna pro práci s obrazem**
- Matplot 1.3.0 – Knihovna pro vykreslení grafů
- **Imutils – Nástroj pro manipulaci s obrazem**
- Copy – Nástroj pro vytváření kopií elementů
- Math – Nástroj pro matematické výrazy
- Tkinter (min. Revision 81008) – Grafické prostředí
- Os – Nástroj pro zpřístupnění interface operačního systému

Pro snadnou implementaci všech knihoven jsem se rozhodl pro využití aplikace Anaconda2 4.2.0, která v sobě obsahuje Python ve verzi 2.7.12 a také téměř všechny potřebné knihovny, které byly výše zmíněny. Po instalaci je nutné již pouze stáhnout z oficiálních stránek knihovnu OpenCV a nástroj Imutils. Ty po stažení nakopírujeme do výchozí složky „C:\Program

Files\Anaconda2\Lib\site-packages“. Tímto je prostředí připraveno a nyní již pouze stačí spustit samotný program buď přímo pomocí souboru *.py, který se asociuje na příkazový řádek pythonu, nebo přes editor Pythonu, který je přiložen – první způsob je doporučen.

5. Uvedení do zkušebního provozu

Jakmile je vše připraveno, je třeba provést zkoušku v provozu. Jelikož jde o první zkoušky, provedl jsem první spuštění pouze na zkušebních zásobnících se zkušebními díly.

5.1. Kamera a stojan

Aby bylo vše připravené k testování, bylo nutné vybrat vhodnou kameru, která by dokázala spolupracovat s programem a dala se umístit na stojan. Pro mé zkušební účely jsem zvolil mou osobní kameru Logitech HD Webcam C270, která je schopná záznamu v rozlišení 720p (1280×720 pixelů). Jedná se o obyčejnou webkameru, ale pro další využití v budoucnu bude nejspíše nutné investovat do kvalitnější kamery s vyšším rozlišením, pro zkušební účely je tento model ale naprosto dostačující. Náš program je schopen spolupracovat s každou kamerou připojenou k PC, kterou lze provozovat jako webkameru, což je s pomocí speciálního softwaru možné u téměř všech moderních videokamer, dokonce by se dalo brát v úvahu použití např. fotoaparátu mobilního telefonu, které jsou v dnešní době velmi kvalitní. Podrobnosti o specifikacích použité kamery naleznete v příloze 1.

42 - Kamera při zkušebním provozu umístěna na stojanu

Kamera musí být uchycena ke stolu, aby nedošlo k rozkmitání obrazu vlivem běhu zásobníku a nebyla tím ovlivněna kvalita rozpoznání. Pro uchycení jsem se rozhodl využít hotových komponent firmy ROX, které se využívají pro montáž senzorických čidel k zásobníkům. Pro uchycení kamery k stojanu jsem si vyrobil vlastní plechový díl, na který se kamera nasune a díky gumové podložce na úchytu kamery drží velmi stabilně. Jako

komponenty stojanu jsem využil tři montážní kostky, které dovolují posunutí a rotaci po dvou kulatinách průměru 12mm a délky 500mm. Schéma sestavy najdete v příloze 4.

43 - Zobrazení stojanu s upevněnou kamerou u zkušebního zásobníku

5.2. Zkušební provoz

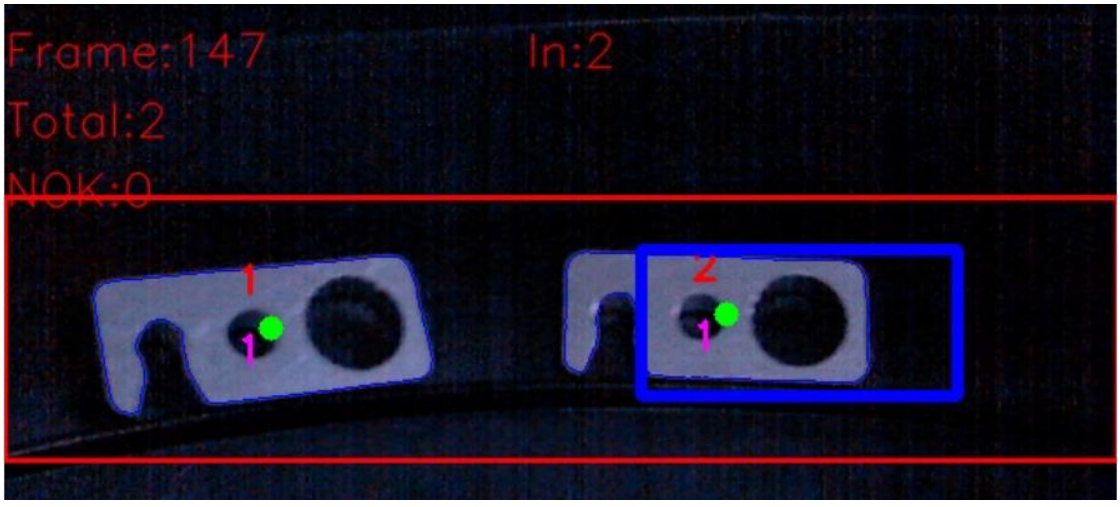
Při sestavení stojanu jsem se snažil naleznout optimální polohu pro kameru. Vzhledem k omezeným ostřícím vzdálenostem, jsem nemohl kameru umístit moc blízko dráhy, ale nakonec jsem našel vyhovující polohu, která zabírala přibližně na šířku 4 součástí, což bylo přibližně 500 mm kolmo od dráhy, kde součásti procházely. Na tuto vzdálenost byl také konstruován samotný program, aby měl dostatečný přehled o celé situaci na dráze.

Po sestavení stojanu, umístění kamery a připojení k počítači je možné zahájit testování. Pro zkušební provoz jsem zvolil notebook s procesorem Intel Core-i7 6500 taktovaný na 2,5 GHz. Během rozpoznávání se vytížení dostalo na 30%, což značí, že při použití moderního hardwaru není s výkonem žádný problém. Pokud bych chtěl na test využít starší stroj, mohl by se nedostatek výkonu projevit snížením počtu snímků za sekundu a obraz by přestal být zcela plynulý. Bohužel jsem ale neměl v době testování k dispozici žádný alternativní počítač.

Jako první krok byla změna programu v oblasti zdroje obrazu, během vývoje jsem používal připravené video, teď bylo nutné vytvořit další verzi programu se zdrojem v připojené kameře. Následně bylo možné provést konfiguraci jednotlivých oblastí a nasnímání vzorků pro srovnání. Celá tato procedura zabrala přibližně 5 minut, ale vzhledem k tomu, že šlo o první použití programu v praxi, může se tento čas v budoucnu zkrátit.

44 - Detail pohledu kamery na zásobník při hledání vhodné polohy

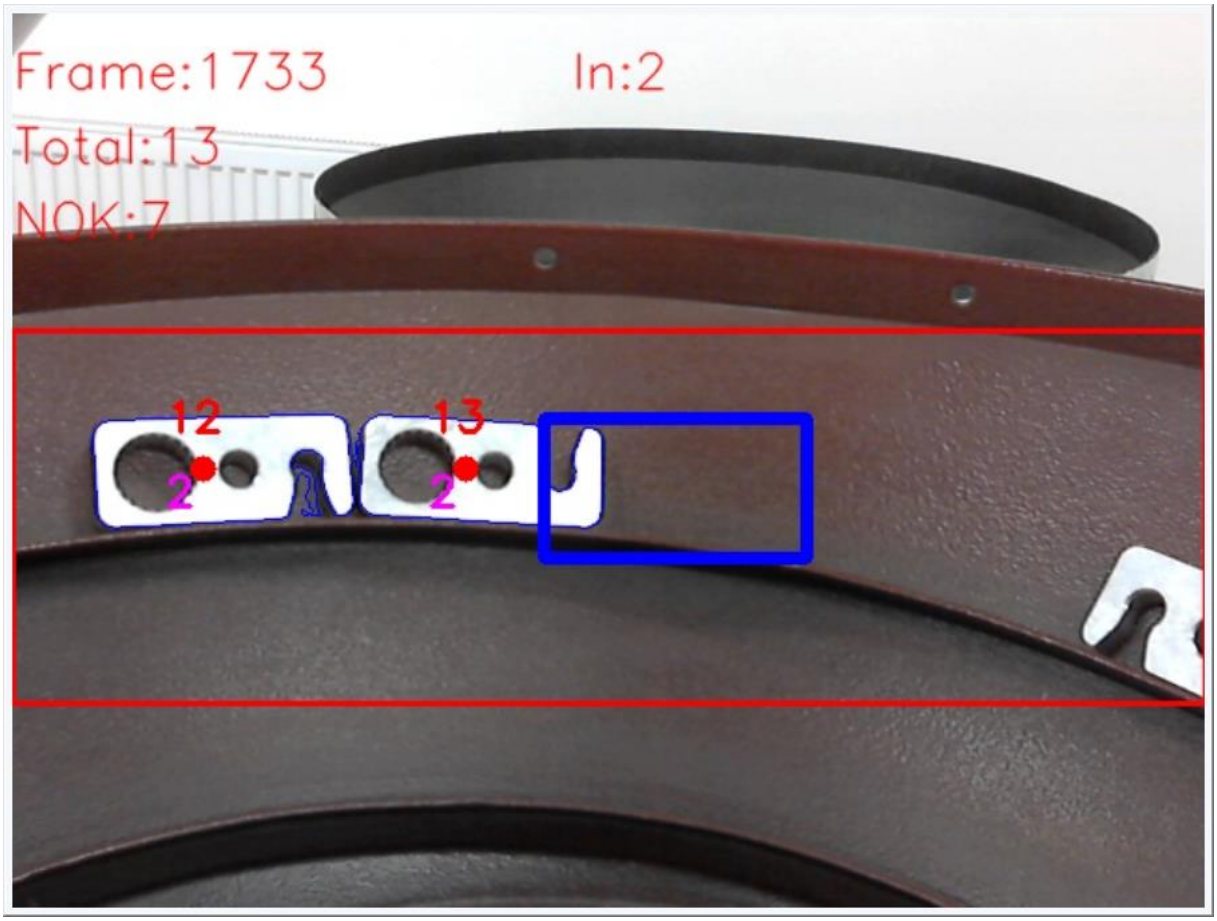
První zkouška proběhla za denního osvětlení bez žádného dalšího přidaného světla. První pokusy proběhly v pořádku, ale proměnlivé venkovní světlo s časem pomalu sláblo a tak se měnili i světelné podmínky uvnitř zkušebního pracoviště. Bylo by tedy nutné přibližně každou hodinu provádět další kalibraci, což není v tomto případě žádané. Změna světla také působila na filtr pozadí, do kterého se dostával šum a detekce tak byla v mnoha případech ovlivněna plochami, které nebyly součástmi, ale algoritmus je falešně detekoval jako NOK. Výsledný protokol tak obsahoval více falešných NOK, než těch skutečných.



45 - Test provozu při špatných světelných podmínkách

Bylo tedy jasné, že provoz musí probíhat pod konstantním osvětlením. Jak ale venkovní světlo slábloo, byly podmínky již tak špatné, že se lesklé součásti velmi viditelně zvýraznily nad pozadí. Provedl jsem tedy test ve špatných světelných podmínkách a zjistil jsem, že program dokáže fungovat i ve špatných podmínkách, dokonce v některých případech byly výsledky lepší, než během předchozích testů. Opět se ale projevil faktor změny osvětlení, kdy se ztmavujícím se světlem algoritmus špatně rozeznával OK součásti a vyhodnocoval je často jako NOK. Co se týče šumu, ten se ale velmi snížil, jak můžete vidět na obrázku 45, součásti jsou velmi dobře viditelné a obrys je detekován téměř bezchybně.

Na závěr jsem v místnosti zajistil stabilní osvětlení, které by mělo zajistit vyžadované konstantní světelné podmínky, a provedl jsem další sérii testů. Tato série byla ze všech nejúspěšnější, ale bohužel i v tomto případě docházelo k nepatrnému zašumění při drobné změně světelných podmínek a tím pádem k falešné detekci NOK součástí, jak je vidět na obrázku 47. Detekce samotných součástí proběhla ale ve všech případech bezchybně a spolehlivost v tomto ohledu je velmi vysoká. Dokonce i detekce více součástí byla bezproblémová, jak je vidět na obrázku 46

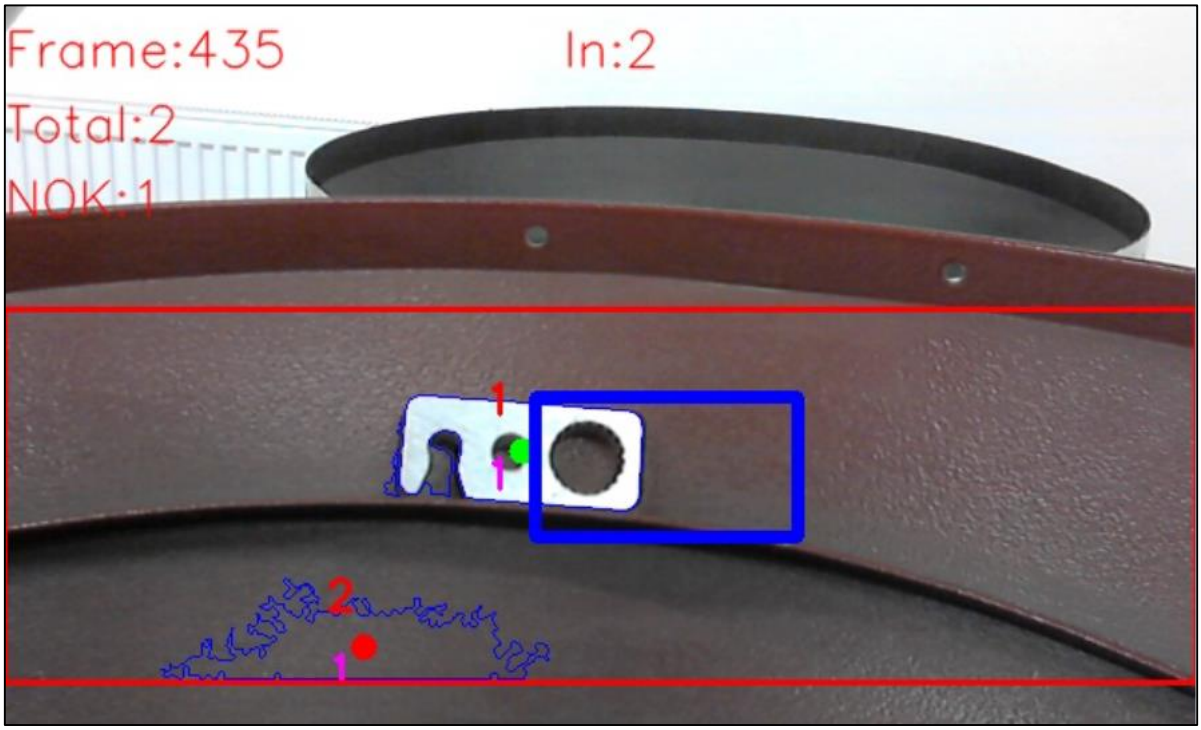


46 – Detekce dvou součástí za sebou v praxi

5.3. Korekce a poznatky z testování

Během zkušebního provozu se objevily problémy, které nebylo možné předem nijak odhalit. Největší z nich se týkal kvality snímaného obrazu. Při záznamu zkušebního videa tento problém nebyl patrný a objevil se až při zkušebním provozu. Z počátku dělalo největší problém osvětlení, které bylo vyřešeno spuštěním stropních zářičů. Jako další krok by se v budoucnu mohlo umístit světlo přímo na stojan s kamerou, aby bylo zamezeno tvorbě stínů v obraze. Po zajištění stabilního prostředí ale nebylo vše zcela vyřešeno.

Největší další slabinou byla kamera, která automaticky regulovala jas a kontrast obrazu při každé menší změně podmínek. Občas se tak stalo, že z těchto důvodů byl obraz zcela nečitelný, pokud byl skok velký, nebo se na obraze objevili plochy, které algoritmus vyhodnotil jako další součást a v některých případech ji vyhodnotil jako NOK součást. Příklad můžete vidět na obrázku 47, kdy je vidět, že součást byla současně označena.



47 - Zobrazení příkladu falešné detekce

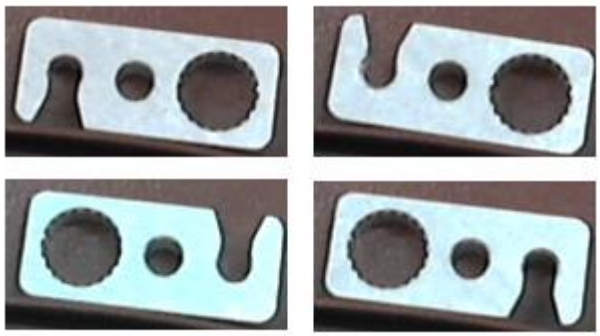
Jako řešení tohoto problému se nakonec ukázalo využití softwaru od výrobce kamery, který dovoluje kompletní nastavení parametrů a také umožní vypnutí všech autokorekcí, takže obraz již zůstává stejný po celou dobu snímání a zkoušky již proběhly zcela bez problémů. Vzniká nám tak další požadavek na vhodnou kameru a to je ten, že by měla být plně nastavitelná a neměla by mít pevně nastavenou autokorekci.

6. Budoucí rozvoj aplikace

Po úspěšném zkušebním provozu je aplikace připravena pro použití v ostrém provozu. Stále ale je mnoho oblastí, ve kterých je nutné či alespoň doporučené provést vylepšení před předáním zaškolené obsluze. Tato vylepšení už přesahují rozsah zadání DP a pro jejich realizaci nezbylo dostatek času. Vývoj ale bude probíhat dál v rámci spolupráce mezi školou a zadávající společností.

- **Šablona součásti pomocí masky**

V aktuálním stavu je šablona pro orientaci součásti pouze běžná fotografie s pozadím a je nutné pro každé natočení provést další snímání tak, jak je znázorněno na obrázku 48. Jako vylepšení by bylo vhodné provést odstranění pozadí, tak jak se to provádí při hledání kontury v obraze a tuto šablonu pouze uložit jako černobílou masku součásti, kterou následně porovnáme s obrazem z kamery. Toto vylepšení by zpřesnilo rozpoznávání a zároveň bychom také pomocí funkce „plocha kontury“ dostali přesnou hodnotu plochy vzorové součásti, kterou musíme nyní zjišťovat dost nepřesně z rozměrů vzorových fotografií. Za předpokladu, že by rozpoznávací místo bylo přesně vodorovné, mohli bychom provést záznam pouze pro OK součást a zbylé vzorky bychom otočily pomocí algoritmu. Mohli bychom tak výrazně zkrátit proces nastavení a také zvýšit přesnost snímání více součástí za sebou.



48 - Nasnímané vzory natočení součásti (OK vlevo nahoře, zbylé NOK)

- **Rozpoznání intervalu plochy jedné součásti**

V mém programu jsem odchylku plochy součásti při pohybu po kruhové dráze zjistil experimentálně. Za předpokladu, že budeme kameru umisťovat na stejné nebo alespoň přibližné místo, bude tato hodnota vyhovovat a program bude pracovat správně. Pokud bychom ale chtěli kameru umístit na jiné místo, bude nutný zásah přímo do konstant programu. Bylo by tedy vhodné umístit do programu další inicializační funkci, která by sledovala průjezd jedné součásti po dráze a během cesty by sledovala, jak se plocha mění. Na

konci průjezdu by funkce vypsala interval plochy a zapsala jej do proměnných v programu. Hodnota by se dala uložit společně s kalibračními daty. Tato funkce by také zlepšila spolehlivost sledování více součástí za sebou. Funkce by také byla schopna rozeznat, kdy se jednotlivé intervaly překrývají a upozornit uživatele, jaký je ideální počet součástí jedoucích za sebou.

- **Interaktivnější výběr oblastí a zachování hodnot i po uzavření aplikace**

Zlepšení by také bylo dobré v oblasti uživatelského rozhraní při výběru pracovní oblasti a oblasti pro vyhodnocení. Aktuálně je nutné odečíst z grafu a nastavit na posuvnících, ale uživatelsky příjemnější by bylo přímo označení v grafu. Jedná se ale pouze o kosmetické vylepšení, které by však mohlo celý proces urychlit. Jako možnost řešení by bylo využít knihovnu Matplot, nebo změnit celé grafické rozhraní do GUI Kivy, které tuto interaktivitu podporuje. Po nalezení ideálních hodnot by byla užitečná funkce, která by dokázala při zavření programu automaticky uložit aktuální nastavení konfigurace např. do souboru, a při dalším spuštění aplikace by se hodnoty načetly zpět.

- **Komunikace s OPC serverem**

Jako další funkci, která by se v některých případech využila je komunikace s PLC, přesněji tedy s OPC serverem, který zprostředkovává komunikaci. 1 z 10 zásobníků využívá senzory, které jsou řízeny PLC. V tomto případě by bylo vhodné umístit do videa signalizaci stavů v PLC, které by nám pomohlo diagnostikovat například špatnou funkci některého senzoru, nebo jeho špatné umístění vůči součásti. Jelikož ale nejsou PLC použity ve většině případů, nebyla tato funkce integrována již do základního programu.

- **Miniaturizace – Raspberry Pi**

Jak jsem již zmínil v kapitole 5.2 na straně 55, nemohl jsem provést test výkonosti na slabších strojích, ale velkým přínosem do tohoto projektu by byl chod na zařízení, které by bylo miniaturní a nepotřebovalo by pro svůj provoz plnohodnotné PC. Vzhledem k možnostem bych velmi preferoval např. Raspberry Pi. To je možné pomocí vzdáleného přístupu ovládat také po síti, takže by stačilo dané zařízení připojit do firemní sítě, nebo pouze ethernetovým kabelem k nejbližšímu počítači, který by sloužil pouze ke konfiguraci a spuštění rozpoznávání. Následné výsledky by se již ukládaly na firemní síť a bylo by možné je prohlížet odkudkoliv. Zároveň by se snížila cenová náročnost celého zařízení, neboť Raspberry Pi je oproti klasickému PC značně levnější.

- **Převod do C++ nebo Javy a vytvoření kompaktnějšího programu**

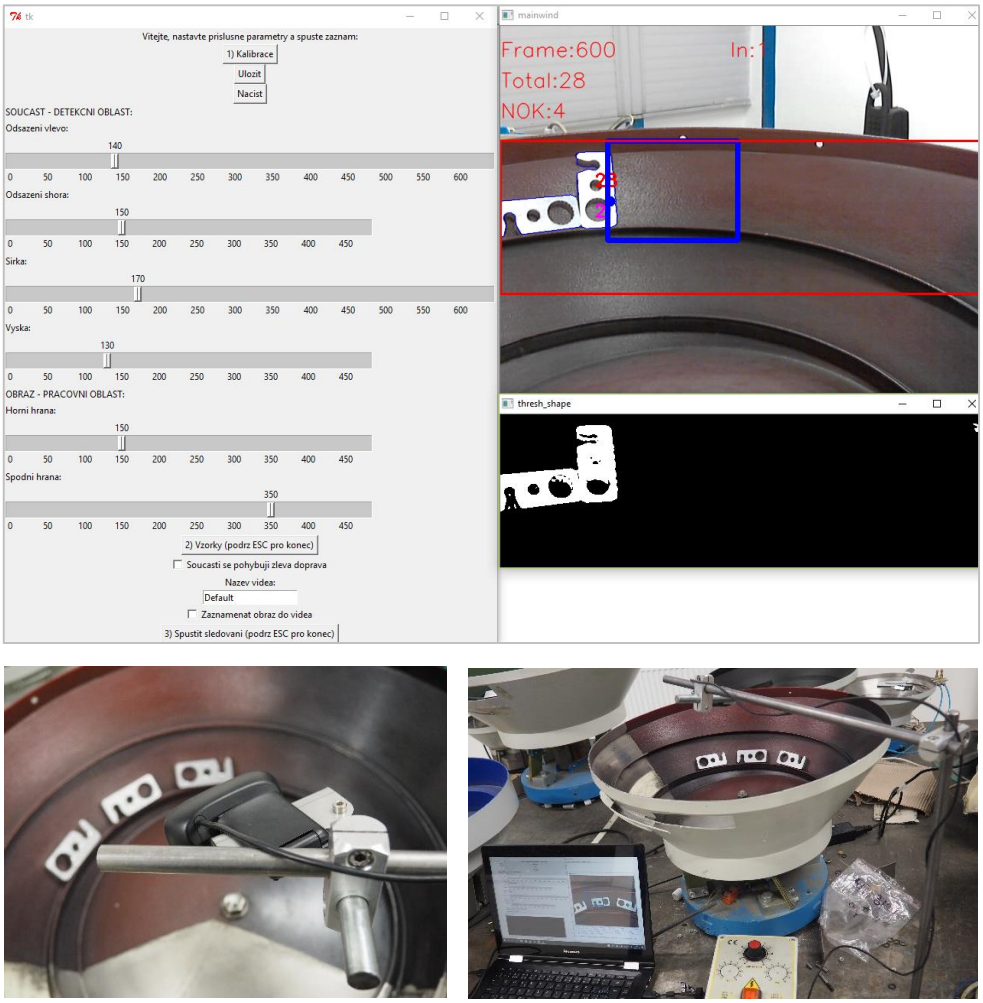
Ačkoliv je Python vhodný pro tento druh aplikace, stále jsou zde jisté limity pro výkon programu zvláště v návaznosti na předchozí bod o miniaturizaci. Během vývoje jsem nenarazil na větší limity, ale po přidání dalších funkcí se můžeme dostat do bodu, že již nebude Python svým výkonem stačit. Bylo by tedy vhodné uvažovat do budoucna o převodu aplikace do některého z výkonnějších jazyků jako je C++ nebo Java. Další výhodou by byla kompilace výsledného programu, který by nepotřeboval žádné další knihovny (jako je tomu nyní), ale nacházel by se na počítači pouze jako spustitelný soubor. To by v mnohém zrychlilo nasazení do dalších zkušebních počítačů a zařízení.

- **Vylepšení algoritmu pro detekci více součástí ve shluku**

Ačkoliv se podařilo vyřešit problematiku pohybu více součástí za sebou, stále má algoritmus svá omezení. Po aplikaci výše popsaných zlepšení bych se mohl dostat na velké přesnosti, ale stále budu limitován počtem detekovatelných součástí, i kdyby se toto číslo mohlo pohybovat kolem 10. To mě stále limituje převážně u menších součástí, a tak by bylo vhodné vymyslet a otestovat algoritmus, který by dokázal součásti nejen detekovat v místě pro rozpoznání, ale zároveň by dokázal sledovat jejich následný pohyb. Tím bychom mohli obejít předchozí algoritmus a dostat se na mnohem větší přesnost. K tomu bychom mohli využít funkci OpticalFlow, kterou jsem popsal v kapitole 3.3.3. Bohužel vývoj takového řešení již přesahuje časový rámec této práce, a tak jsem zde chtěl tímto alespoň naznačit její budoucí vývoj.

7. Závěrečné zhodnocení

V této práci bylo mým úkolem vytvořit systém pro detekci a sledování objektů na dráze vibračního kruhového zásobníku. Po navržení 4 variant – 1. použití stojanu osazeného senzory, 2. průmyslového kamerového systému, 3. kamerových senzorů a 4. návrh vlastního řešení z volně dostupných zdrojů, jsem nakonec zvolil variantu poslední – tedy návrh vlastního systému. Po nastudování potřebné dokumentace jsem za pomoci programovacího jazyka Python a knihovny OpenCV vytvořil program, který dokáže rozpoznávat pohyb a orientaci součástí v zásobníku. Zadání tedy bylo splněno ve všech bodech. Kvůli časové tísni nebylo možné výsledný produkt otestovat na skutečném výrobku, který bude předán zákazníkovi, ale pouze na zkušebních strojích. Test proběhl úspěšně a systém dokázal detekovat všechny součásti, které se na dráze objevily. V návaznosti na 6. kapitolu bude dále probíhat vývoj aplikace v rámci spolupráce s firmou ROX spol. s r.o. V příloze této práce je uveden zdrojový kód, stavový diagram programu a specifikace kamery s výkresem sestavení jejího stojanu.



49 - Konečná podoba funkčního systému (program, kamera, celkový pohled)

8. Použitá literatura

[1] T. Lojík, „Projekt III - Rozpoznání orientace součásti pomocí senzoriky". 20-čer-2016.
[2] „ROX.cz >> Ke stažení". [Online]. Dostupné z: http://rox.cz/kestazeni_003kz200.php. [Viděno: 22-lis-2016].
[3] „Kamerové systémy | JHV". [Online]. Dostupné z: http://www.jhv.cz/jednoucelove-stroje/kamerove-systemy. [Viděno: 20-lis-2016].
[4] „Možnosti strojového vidění - H&D International Group". [Online]. Dostupné z: https://www.hud.cz/moznosti-strojoveho-videni/. [Viděno: 20-lis-2016].
[5] „Zpracování obrazu z kamerových systémů". [Online]. Dostupné z: http://www.dfcdesign.cz/cz/sluzby/zpracovani-obrazu-z-kamerovych-systemu. [Viděno: 20-lis-2016].
[6] „Průmyslové vidění :: Kamerové senzory :: Senzory pro rozpoznávání objektů :: Provedení - typ :: Kvádrový". [Online]. Dostupné z: https://www.ifm.com/ifmcz/web/pmain/020_010_030_020_010_060.html. [Viděno: 22-lis-2016].
[7] „2D Vision / SICK". [Online]. Dostupné z: https://www.sick.com/cz/cs/product-portfolio/vision/2d-vision/c/g114859. [Viděno: 22-lis-2016].
[8] „Průmyslové vidění :: Kamerové senzory :: Senzory pro rozpoznávání objektů :: Provedení - typ :: Kvádrový". [Online]. Dostupné z: http://www.ifm.com/obj/kamerove-senzory-ifm.pdf. [Viděno: 22-lis-2016].
[9] D. H. Ballard a C. M. Brown, *Computer vision*. Englewood Cliffs, N.J: Prentice-Hall, 1982.
[10] G. Holub, „Průmyslové zpracování obrazu v automatizaci - Časopis Světlo - Odborné časopisy", Odbornecasopisy.cz. [Online]. Dostupné z: http://www.odbornecasopisy.cz/svetlo/casopis/tema/prumyslove-zpracovani-obrazu-v-automatizaci--16974. [Viděno: 26-lis-2016].
[11] H. F. Schantz, *The history of OCR, optical character recognition*. Manchester Center, Vt.: Recognition Technologies Users Association, 1982.
[12] „OCR Technology", *Q-Free*. [Online]. Dostupné z: https://www.q-free.com/product/ocr-technology/. [Viděno: 28-lis-2016].
[13] „Metody rozpoznání objektů v obrazu.pdf | Fakulta biomedicínského inženýrství". [Online]. Dostupné z: http://www.fbmi.cvut.cz/files/predmety/3528/public/Metody%20rozpozn%C3%A1n%C3%AD%20objekt%C5%AF%20v%20obrazu.pdf. [Viděno: 26-lis-2016].
[14] „Moving face 3D reconstruction - Hum3D Blog", *Humster3D Store*. [Online]. Dostupné z: https://hum3d.com/blog/moving-face-3d-reconstruction/. [Viděno: 28-lis-2016].
[15] J. Condliffe, „Google's Image Recognition Software Can Now Describe Entire Scenes", *Gizmodo*. [Online]. Dostupné z: http://gizmodo.com/googles-image-recognition-software-can-now-describe-ent-1660033808. [Viděno: 28-lis-2016].
[16] „Vision API - Image Content Analysis", *Google Cloud Platform*. [Online]. Dostupné z: https://cloud.google.com/vision/. [Viděno: 26-lis-2016].
[17] „File:HSV color solid cylinder.png - Control Systems Technology Group". [Online]. Dostupné z: http://cstwiki.wtb.tue.nl/index.php?title=File:HSV_color_solid_cylinder.png. [Viděno: 28-lis-2016].
[18] „Edge detection." [Online]. Dostupné z: http://www.nada.kth.se/~tony/cern-review/cern-html/node12.html. [Viděno: 29-lis-2016].
[19] „Welcome to OpenCV-Python Tutorials's documentation! — OpenCV-Python Tutorials 1 documentation". [Online]. Dostupné z: http://opencv-python-tutroals.readthedocs.io/en/latest/index.html. [Viděno: 03-pro-2016].

[20] „YOLO: Real-Time Object Detection". [Online]. Dostupné z: http://pjreddie.com/darknet/yolo/. [Viděno: 29-lis-2016].

[21] J. Dalal a S. Patel, *Instant OpenCV starter: get started with OpenCV using practical, hands-on projects*. Birmingham: Packt Pub., 2013.

[22] „PyPy - Welcome to PyPy". [Online]. Dostupné z: http://pypy.org/index.html. [Viděno: 05-pro-2016].

[23] „TIOBE Index | TIOBE - The Software Quality Company". [Online]. Dostupné z: http://www.tiobe.com/tiobe-index//. [Viděno: 07-pro-2016].

[24] „GuiProgramming - Python Wiki". [Online]. Dostupné z: https://wiki.python.org/moin/GuiProgramming. [Viděno: 07-pro-2016].

[25] „Define GUI in a class : UI Class « GUI Tk « Python". [Online]. Dostupné z: http://www.java2s.com/Code/Python/GUI-Tk/DefineGUIinaclass.htm. [Viděno: 17-pro-2016].

[26] „Kivy: Cross-platform Python Framework for NUI". [Online]. Dostupné z: http://kivy.org/. [Viděno: 08-pro-2016].

[27] D. Marr, *Vision: a computational investigation into the human representation and processing of visual information*. Cambridge, Mass: MIT Press, 2010.

[28] V. Mařík, O. Štěpánková, a J. Lažanský, *Umělá inteligence*. Praha: Academia, 1993.

[29] „HD Webcam C270 - Logitech Support". [Online]. Dostupné z: http://support.logitech.com/en_in/product/hd-webcam-c270. [Viděno: 02-led-2017].

9. Seznam obrázků

10. Seznam tabulek