
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

## ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS


# VYUŽITÍ KNIHOVNY OPENCV PRO MECHATRONICKÉ APLIKACE
COMPUTER VISION FOR MECHATRONIC APPLICATIONS USING THE OPENCV LIBRARY

**BAKALÁŘSKÁ PRÁCE**
BACHELOR'S THESIS


**AUTOR PRÁCE** — **Martin Černil**
AUTHOR

**VEDOUCÍ PRÁCE** — **Ing. Michal Bastl**
SUPERVISOR



**BRNO 2018**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

# Zadání bakalářské práce

| | |
|---|---|
| Ústav: | Ústav mechaniky těles, mechatroniky a biomechaniky |
| Student: | **Martin Černil** |
| Studijní program: | Strojírenství |
| Studijní obor: | Základy strojního inženýrství |
| Vedoucí práce: | **Ing. Michal Bastl** |
| Akademický rok: | 2017/18 |

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

## Využití knihovny OpenCV pro mechatronické aplikace

**Stručná charakteristika problematiky úkolu:**

Cílem práce je seznámení se s rozsáhlou open source knihovnou pro počítačové vidění OpenCV a vypracování vzorových úloh z oblasti mechatroniky.

**Cíle bakalářské práce:**

1, Seznámit se s knihovnou OpenCV a s jejím využitím v jazyce Python.
2, Rešerše na počítačové vidění s přihlednutím k možnostem OpenCV.
3, Vytvořit vzorové úlohy využitelné v mechatronice.
4, Zhodnocení výsledků práce

**Seznam doporučené literatury:**

HOWSE, J.: OpenCV computer vision with Python: learn to capture videos, manipulate images, and track objects with Python using the OpenCV library. Birmingham, England: Packt Publishing, c2013.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2017/18

V Brně, dne

L. S.

| | |
|---|---|
| prof. Ing. Jindřich Petruška, CSc.<br>ředitel ústavu | doc. Ing. Jaroslav Katolický, Ph.D.<br>děkan fakulty |

# ABSTRAKT

Tato práce se zabývá seznámením se s knihovnou počítačové vize OpenCV, která je následně implementovaná do tří zaměřením odlišných úloh pomocí programovacího jazyka Python. Těmito úlohami jsou rozpoznání objektu a jeho lokalizace, sledování objektu a detekce změny a stanovení bezpečně vzdálenosti pomocí hloubkové mapy.

# ABSTRACT

This thesis introduces a computer vision library OpenCV, which is subsequently implemented into three distinguishably different problems using Python programming language. These three problems are identifying an object and its location, object tracking and difference detection, safe distance qualification using a depth map.



# KLÍČOVÁ SLOVA

OpenCV, Python, Počítačová vize, Mechatronika, KCF, MIL, Stereovize, Haarovy kaskády, Sledování objektů, Hloubková mapa

# KEYWORDS

OpenCV, Python, Computer vision, Mechatronics, KCF, MIL, Stereovision, Haar cascades, Object tracking, Depth map

# BIBLIOGRAFICKÁ CITACE

ČERNIL, M. Využití knihovny OpenCV pro mechatronické aplikace. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018. 66 s. Vedoucí bakalářské práce Ing. Michal Bastl.

# ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci na téma „Využití knihovny OpenCV pro mechatronické aplikace“ vypracoval samostatně s použitím odborné literatury a pramenů uvedených v seznamu použitých zdrojů.

22. května 2018

.....
Martin Černil

# PODĚKOVÁNÍ

Děkuji tímto Ing. Michalovi Bastlovi za cenné připomínky, výtky a rady při vypracování bakalářské práce.

# OBSAH

# ÚVOD

V době moderních technologií a především díky výpočetní rychlosti počítačů se lze běžně setkávat se systémy založenými na počítačovém vidění. Počítačové vidění umožňuje počítačům zpracovávat obrazový materiál, získávat z něj potřebné informace a ty následně uplatnit. Tyto technologie je možno použít v mnoha odvětvích průmyslu, kde jsou využívány především k automatizaci procesů.

Úvodní kapitola přibližuje obor počítačového vidění, digitálního zpracování dat a s nimi spjaté programové a technologické vybavení použité v dalších kapitolách.

Následující kapitola se zabývá zprovozněním a použitím základních funkcí programovacího jazyka Python a knihovny počítačového vidění OpenCV. Demonstrace funkcí je provedena pomocí jednoduchých úloh, jejichž poznatky jsou užity v další části práce.

Hlavní část této bakalářské práce demonstruje tři zaměřením rozdílné mechatronické úlohy. Jedná se o rozpoznávání objektu díky použití haarových kaskád a jeho následnou lokalizaci, kde pomocí metody počítačového učení byly rozeznávány dva objekty na desce stolu. Dále bylo řešeno rozpoznání změny v obraze a sledování této změny pomocí dostupných sledovacích algoritmů, jmenovitě algoritmů MIL (Multiple Instance Learning) a KCF (Kernelized Correlation Filter). Třetí úloha sestávala ve stanovení bezpečné vzdálenosti díky hloubkové mapě, a to za použití dvou kamer ve specifické konfiguraci pro zajištění stereovize. Podrobněji je problematika těchto úloh zpracována v kapitole tři.

Každá ze tří demonstrovaných úloh byla algoritmicky analyzována a zhodnocena na základě zjištěných omezení a výsledků, a následně byly navrhnuta možná zlepšení a rozšíření.

# 1 TEORETICKÁ ČÁST

## 1.1 Počítačová vize

Počítačové vidění, odvětví výpočetní techniky, se zabývá vývojem softwaru a zařízení schopných interpretovat obrazová data. Vstupující informace se zpracovávají s daným úmyslem, výstupem je požadované porozumění danému obrazu nebo požadovaná informace z něj odvozená.

Hlavním rozdílem počítačového vidění oproti zpracování obrazu je vytvoření „vjemu“ na základě právě vstupních obrazových dat.

## 1.2 Digitální zpracování obrazu

Digitální zpracování obrazu používá přímou interpretaci obrazových dat, vstupem bývají nejčastěji informace z nahrávacích nebo snímacích zařízení, jako jsou kamery či fotoaparáty. V následující části jsou vymezeny některé pojmy pro práci s těmito daty.

### 1.2.1 Obraz jako funkce

Obraz je funkce $f(x,y)$, kde každý pixel má přiřazenu souřadnici $x,y$ a intenzitu 0-255 (kvůli počítání s bity). Pixel je zkrácené označení pro takzvaný picture element, což znamená základní element obrazu. Obraz může mít jeden či více barevných kanálů, například tři: RGB (červená, zelená, modrá), kde jsou každému pixelu přiřazeny právě tři intenzity pro každou z uvedených barev. S obrazem tedy pracujeme jako s maticí, která obsahuje stejný počet seřazených prvků, jako je počet pixelů v obrazu, a každý prvek obsahuje tolik členů, kolik má obraz barevných kanálů.

### 1.2.2 Rozlišení a typy

Rozlišení vyjadřuje celkový počet pixelů v obrazu, například $640 * 480$ vyjadřuje šířku 640 pixelů a výšku 480 pixelů. Existuje mnoho typů obrazových dat, v této práci byly použity nejčastěji datové typy s barevnou hloubkou 32 bitů pro barevné obrazy a 8 bitů pro černobílé obrazy, obojí formátu JPEG a PNG.

## 1.3 Python

Python [2] je variabilní a interaktivní programovací jazyk s intuitivním syntaxem používaný pro skriptování a rapid development. Zahrnuje obsáhlou základní knihovnu objektů a modulů doplněnou propracovanou dokumentací. Mimo to Python rozšiřuje mnoho modulů, rozšíření a nástrojů třetích stran. Python je otevřený software s licencí na Open Source Initiative [1].

Tento programovací jazyk se používá v mnoha odvětvích IT, jako je například tvorba webových aplikací, výzkum a numerické výpočty, vzdělávání nebo vývoj software. Dává také prostor pro kolaboraci, workshopy a otevřenou komunitu složenou z nadšenců i profesionálů různých odvětví. Díky uživatelské přívětivosti je tento programovací jazyk mnohdy užíván pro výuku počítačového programování.

Pro vypracování úloh této bakalářské práce byl použit programovací jazyk Python verze 3.6.3.

## 1.4 OpenCV

OpenCV [3] (Open Source Computer Vision - volně přeloženo jako Otevřený software počítačové vize) je otevřenou knihovnou funkcí pro počítačové vidění. Podporuje programovací jazyky C++, C, Python, Java a operační systémy Windows, Linux, Android, iOS a Mac OS, přičemž obsahuje více než 3 000 optimalizovaných algoritmů z nejrůznějších odvětví počítačové vize. OpenCV je vydáno pod licencí BSD [1], která umožňuje redistribuci a modifikaci zdrojového materiálu, to vše pro komerční i edukační účely.

Použití této knihovny je velmi obsáhlé. Odvětví užití může být od interaktivního umění, kontroly provozu, inspekce výrobků, až po manipulaci obrazů a robotiku.

V této práci byla použita knihovna OpenCV verze 3.3.0.

## 1.5 NumPy

Tento balík pro vědecké výpočty, s pomocí programovacího jazyka Python, je užíván pro práci s n-dimenzionálními maticemi, lineární algebrou, Fourierovými transformacemi a operacemi s multidimenzionálními daty, jako je například právě obrazový materiál. Balík je licencovaný jako open-source BSD. [1]

V této práci byla použita NumPy [4] verze 1.13.3.

## 1.6 Využití v průmyslu a mechatronice

Nejběžnější použití je v oblasti bezpečnostních kamer a jejich výstupu (kontrola lidí, pohyb osob, počítání objektů, analýza dopravy, rozpoznání tváří, identifikace osob), dále převod pohybů na data, například sledování pohybu, výpočet rychlostí nebo rozpoznávání gest, a dále pro inspekci výrobků (papírny, pekárny, textil, výroba mikročipů, kontrola zabalení, atp.). [5]

Využití této technologie je mnohem více. V poslední době atraktivním odvětvím se stává i automobilový průmysl. [6] Zde počítačové vidění umožňuje užití parkovacích asistentů, systémů popojíždění v kolonách a hlídání jízdních pruhů, předvídání nebezpečí či kolize, rozpoznávání značek a samotné autonomní řízení automobilů.

## 1.7 Hardware a software

Tato práce byla zpracována na operačním systému Windows 7 a obrazový materiál, pokud není napsáno jinak, pochází z dvou webkamer Microsoft LifeCam Cinema [7], kde na obr. 1 je možno spatřit právě jednu z nich. V tabulce 1 lze najít její specifikace.


**Obr.1: Microsoft LifeCam Cinema [7]**

| Parametr | Hodnota |
|---|---|
| Výrobce | Microsoft |
| Rozlišení | 1280x720 |
| Typ senzoru | Snímač 720p (HD rozlišení) |
| Autofocus | ano |
| Rozhraní | USB 2.0 |
| Mikrofon | ano |
| Snímků za sekundu | Až 30 FPS |

**Tab. 1: Specifikace kamery Microsoft LifeCam Cinema [7]**

# 2 SEZNÁMENÍ SE S KNIHOVNOU OPENCV A JAZYKEM PYTHON

Následující část se zabývá instalací a základními funkcemi OpenCV, které byly následně implementovány do tří úloh, které byly hlavním výstupem této bakalářské práce.

## 2.1 Instalace OpenCV a Python

Aktuální možnosti instalace programovacího jazyka Python [3] lze nalézt na oficiálních stránkách.

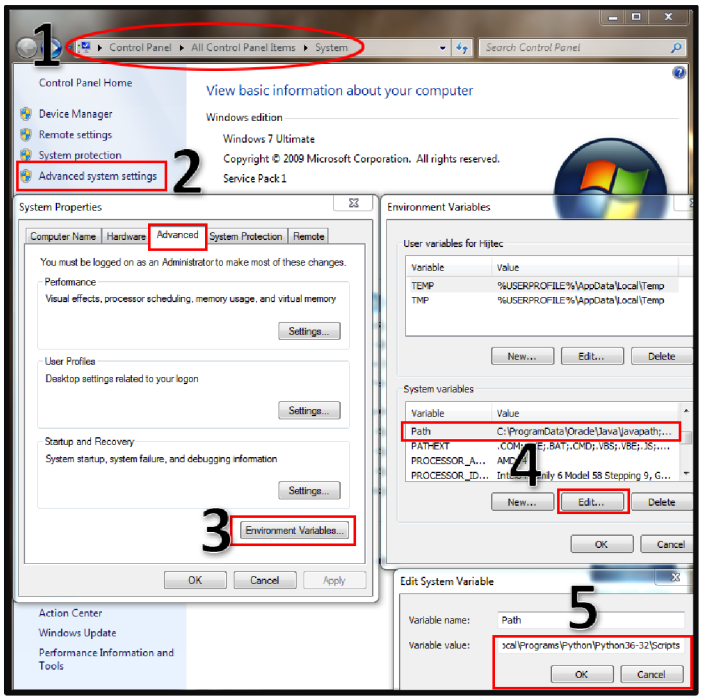
Zvolený Python verze 3.6.3 pro tutu bakalářskou práci má již zabudovaný Pip, používaný pro instalaci a správů balíčků z Python Package Index (PyPI) [8], ve kterém jsou uvedeny odkazy k jednotlivým balíčkům.

Pro instalaci knihovny OpenCV je nutno otevřít příkazový řádek a použít následující příkaz:

```
pip install opencv-python==3.3.0.10
```

V případě chyby je potřeba přidat volání Pipu v proměnné PATH, kterou lze nalézt na operačním systému Windows dle obr. 2, kde se následně přídá k proměnné PATH cesta k instalaci Pythonu, standartně tato:

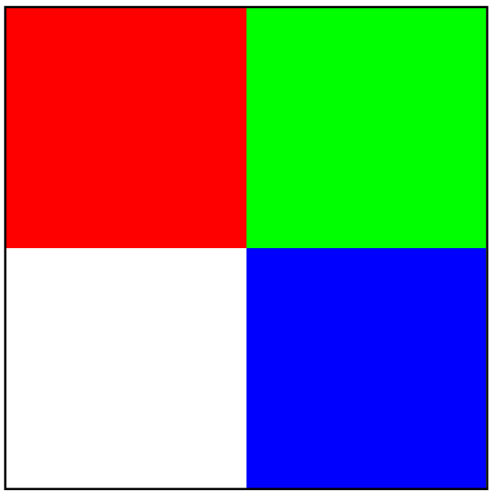
C:\Users\"Název účtu"\AppData\Local\Programs\Python\Python36-32\Scripts.




**Obr. 2: Přidání Pip do PATH**

## 2.2 Načtení obrazu a základní operace

S úspěšně nainstalovaným OpenCV lze načíst následující obrázek:



**Obr. 3: 2x2 pixely**

S obrazem bylo manipulováno skrze následující skript:

```
import cv2                                  #import knihovny OpenCV
import numpy as np                          #import knihovny Numpy

img = cv2.imread("soubor")                  #do proměnné img načten "soubor"

    #Příklad "soubor" může být např.: obraz.png nebo uveďte celou cestu k cíli

adv_img = cv2.imread("soubor", argument)    #za argument lze zvolit následující:

    #cv2.IMREAD_COLOR         - (1)Standartní, načte barevný obraz bez průhlednosti
    #cv2.IMREAD_GRAYSCALE     - (0)Načte obraz v odstínech šedé
    #cv2.IMREAD_UNCHANGED     - (-1)Načte barvy včetně průhlednosti (alpha kanál)

print(img)                                  #zobrazení matice obrazu img
print(adv_img)                              #zobrazení matice obrazu adv_img

cv2.imshow("jméno_okna",img)                #vytvoření okna se zobrazením img
cv2.imshow("jméno_okna2",adv_img)           #vytvoření okna se zobrazením adv_img

cv2.waitKey(0)                          #program čeká na vstup klávesnice
cv2.destroyAllWindows()                 #zavře všechny vytvořené okna

cv2.imwrite("test.png",img)             #uloží img jako PNG soubor se jménem test
```

Skript vrátil přes funkci print následující matice *img* a *adv_img*, pro které byl použit argument 0 (načtení v odstínech šedé).

$$img = \begin{bmatrix} [0 & 0 & 255] & [0 & 255 & 0] \\ [255 & 255 & 255] & [255 & 0 & 0] \end{bmatrix}$$

$$adv\_img = \begin{bmatrix} 147 & 200 \\ 255 & 95 \end{bmatrix}$$

Každý prvek matice $img$ reprezentuje jeden pixel. Je zřejmé, že OpenCV používá nekonvenční systém BGR (modrá, zelená, červená) oproti známému RGB (červená, zelená, modrá), a to z důvodu dřívější konvence barevných kanálů. Matice $adv\_img$ obsahovala již pixely převedené do odstínů šedé. Pro běžné užití a zobrazení mohly jednotlivé členy nabývat hodnot 0-255.

## 2.3 Velikost, kreslení, vložení textu, konverze obrazu

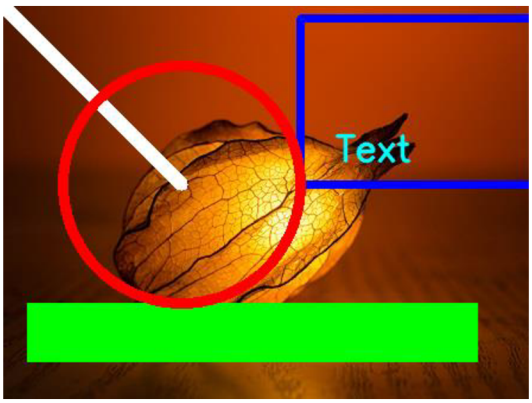
Pro uživatelsky přívětivý výstup vytvořených úloh bylo nutno uvést následující operace. V této části je ukázána práce se základními fonty a vykreslenými obrazci demonstrovanými na obr. 4. Předvedeno v následujícím skriptu:

**Obr. 4: Prosvětlený květ [9]**

```
import cv2
import numpy as np
img = cv2.imread("flowerbud.png")           #načtení obrázku
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #převod do odstínů šedé

print(img.shape)            #pozor, vrací řady, sloupce (převrácené rozlišení)
print(img.size)             #vrací celkový počet pixelů
print(img.dtype)            #vrací datový typ

img[250:300, 20:400] = [0,255,0];       #část obrazu je vyplněna zelenou barvou
cv2.line(img, (0,0), (150,150), (255,255,255), 10);
#čára v img, začátek čáry, konec čáry, barva čáry, šířka čáry v pixelech
cv2.rectangle(img, (250,10), (450,150), (255,0,0), 5);
#obdélník v img, jeho začátek, konec, barva čáry a šířka v pixelech
cv2.circle(img, (150,150), 100, (0,0,255), 8);
#kružnice v img, její střed, poloměr v pixelech, barva, šířka kružnice

font = cv2.FONT_HERSHEY_SIMPLEX; # volba fontu
cv2.putText(img, "Text", (280,130), font, 1, (255,255,0),2 ,cv2.LINE_AA);
#kam, text, začátek, font, velikost, barva, , antialiasing, tloušťka, modulace

cv2.imshow("img",img)
cv2.imshow("gray",gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Výsledkem bylo následující:

$$img.shape = (331, 500, 3) \qquad img.size = 456500 \qquad img.dtype = "uint8"$$




**Obr. 5a: Výstup img ze skriptu**

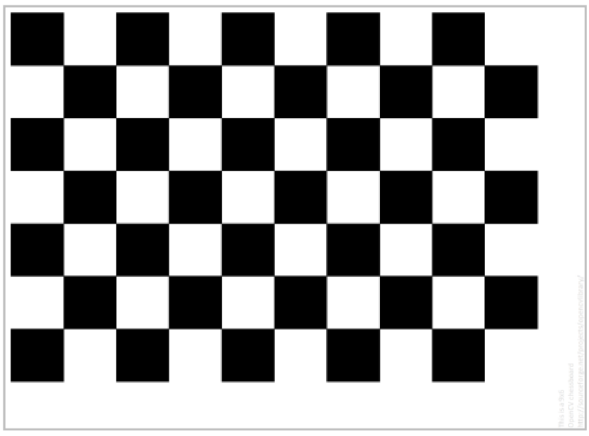
**Obr. 5b: Výstup gray ze skriptu**

Z obrázku 5a bylo zřejmé, jak jednotlivé operace následovaly po sobě díky překrývání obrazců. Funkce $img.shape$ vrátila řádky, sloupce a kanály matice obrazu, $img.size$ zase celkový počet pixelů v obrázku. Fontů a datatypů je velké množství, další informace lze nalézt v dokumentaci knihovny OpenCV. [10] Některé funkce OpenCV očekávají datový typ uint8 (čísla přirozená) a některé float32 (s desetinnou čárkou). V obr. 5b lze vidět převod do odstínů šedé.

## 2.4 Kalibrace kamer

Každá běžná čočka není perfektní a projevují se u ní zobrazovací vady. Na použitých kamerách pro tuto bakalářskou práci bylo zjištěno malé soudkovité zklenutí, jež muselo být korigováno.

V každé z provedených úloh bylo nutné provést kalibraci kamer. Tímto krokem byla kompenzována radiální a tangenciální deformace obrazu kamery. Všechny rovné hrany musejí být rovné i na obrazu, proto se používají kalibrační obrazy, viz následující obrázek 6.

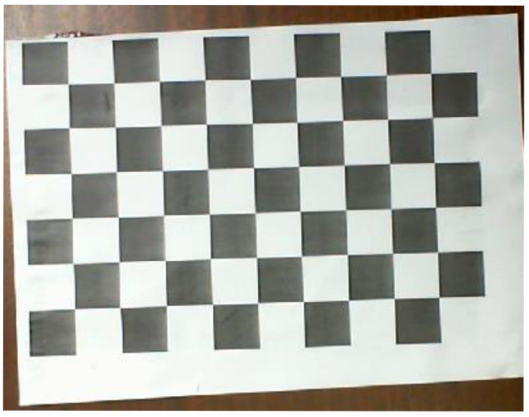


**Obr. 6: Kalibrovací obrazec OpenCV [11]**

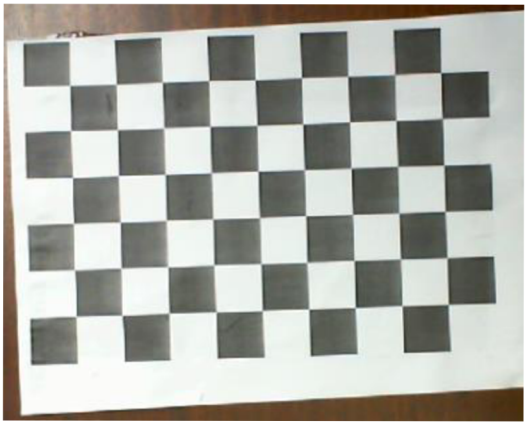
Je nutno najít pět koeficientů deformace a samotnou matici kamery $3 * 3$, jež obsahuje ohniskovou vzdálenost a polohu ohniska. Pro aplikace je dále nutné deformace kompenzovat, a to například dodáním obrazů známého vzoru, u kterého zajišťujeme jeho tvar a velikost - například šachovnice (viz obr. 6). V tomto vzoru se hledají specifické body (u šachovnice rohy jednotlivých čtverců). [12]

Jsou známy jejich souřadnice v reálném světě a souřadnice v obrazu kamery. S těmito daty se následně vyřeší matematický problém, díky kterému se získají jednotlivé deformační koeficienty.

Byla použita funkce $cv2.calibrateCamera()$, jež vrací matici kamery, koeficienty deformace, rotační a translační vektory (poslední dva vhodné pro kontrolu, kde porovnáme nalezený vzor se vzorem vygenerovaným po kalibraci). Tyto veličiny úzce souvisí s funkcí $cv2.undistort()$, která přetváří originální obraz na obraz kalibrovaný. [12]



**Obr. 7: Originální obrazec**



**Obr. 8: Zkalibrovaný obrazec**

Změny mezi obr. 7 a obr. 8 nebyly na první pohled zřejmé, což značí, že kamera byla přibližně věrná ideální kameře se snímacím úhlem 90°. S použitou panoramatickou kamerou (120°) už byly výsledky velmi nepřesné. Relativní chyba kalibrace s použitými kamerami se pohybovala okolo 0,05 %. Kalibrace se vyhotovuje především pro rozšířenou realitu a stereovizi, kdy pro kalkulaci hloubkové mapy pomocí triangulace by se i malá změna úhlu promítla v nezanedbatelnou nepřesnost na výstupu.

# 3 APLIKACE OPENCV NA MECHATRONICKÉ ÚLOHY

Hlavním cílem této bakalářské práce bylo demonstrovat a aplikovat získané znalosti jazyka Python a knihovny OpenCV na třech zaměřením rozdílných mechatronických úlohách spojených s počítačovou vizí.

Těmito úlohami se staly:

1. Rozpoznání objektu a jeho lokalizace
2. Sledování objektu a detekce změny v obraze
3. Hloubková mapa – stanovení bezpečné vzdálenosti pomocí stereovize

Úlohy byly vypracovány samostatně s použitím citovaných zdrojů.

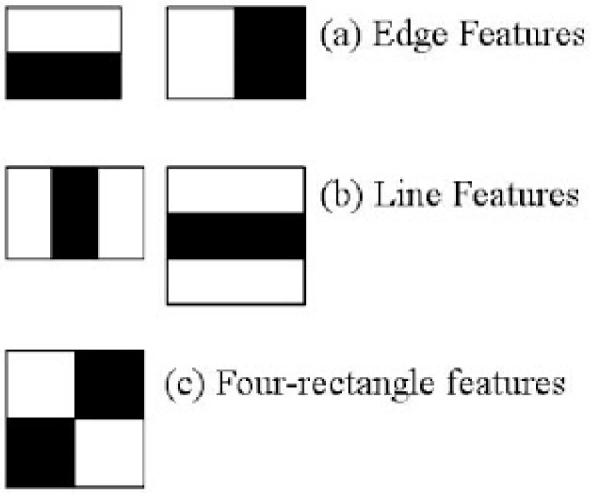
## 3.1 Rozpoznání objektu a jeho lokalizace

OpenCV disponuje funkcí $cv2.CascadeClassifier.detectMultiScale()$, [13] která pomocí zadané haarovy kaskády detekuje objekty v obraze a navrací obdélníky obklopující detekovaný subjekt.

Následující úloha byla zpracována za účelem rozpoznání dvou odlišných významných předmětů na desce stolu a upozornění na případnou absenci těchto předmětů v jimi přidělené lokaci. Detekované předměty jsou zobrazeny na následujícím obrázku 9.

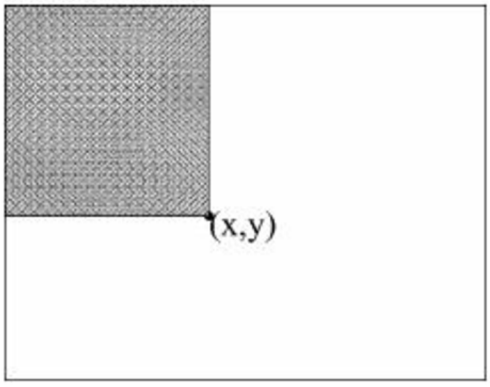

**Obr. 9: Předměty k rozeznání**

V roce 2001 zveřejnili Paul Viola a Michael Jones metodu rozpoznávání předmětů použitím Haarových prvků. Jedná se o metodu počítačového učení, jejímž výstupem byl klasifikátor vytrénovaný na mnoha pozitivních, resp. negativních obrazech, ve kterých se nacházel, resp. nenacházel hledaný předmět. [14] Tento algoritmus byl představen na úkolu rozpoznávání obličeje.



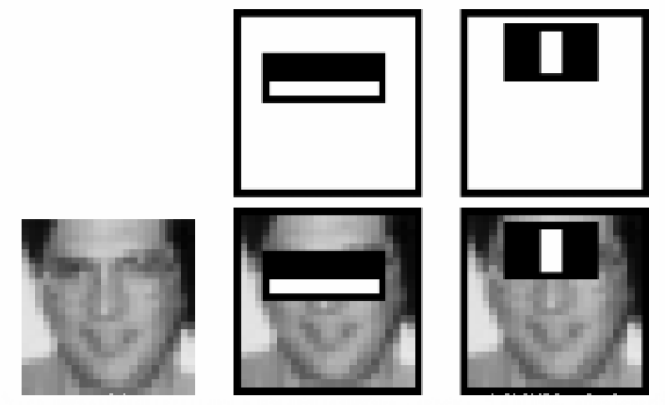

**Obr. 10: Prvky používáné pro trénink klasifikátoru [15]**

Prvky zobrazené v obr. 10 se vypočítaly v zadaném rozlišení přes celý obraz. Pro výpočet prvku byla odečtena suma pixelů z bílé části od sumy pixelů v černé části. Při použití všech pozic a všech druhů prvků pro obraz $24 * 24$ pixelů se získalo přes 160 000 unikátních prvků. Aby se nemusely počítat sumy pixelů ve všech iteracích, byl zaveden tzv. integrální obraz, kdy každý pixel je převeden na číslo, jež je sumou všech pixelů nad a vlevo od něj, viz obr.11.




**Obr. 11: Integrální obraz [15]**

Většina vypočtených prvků byla nevýznamných, duplicitních nebo nadbytečných. Pomocí adaptivního zesilování (AdaBoost [16]) se získají takové prvky, které se opakují a jsou jednoznačné. Na lidském obličeji lze nalézt dva nejvýznamnější prvky splňující následující kritéria. Oblast očí je tmavší oproti oblasti podoční a také je tmavší než oblast můstku nosu, viz obr. 12.



**Obr. 12: Významné Haarovy prvky lidského obličeje - slabé klasifikátory [17]**

Tyto prvky nazýváme slabými klasifikátory [14], jelikož nejsou schopné samy o sobě určit objekt. Výsledný klasifikátor se určí jako vážená suma slabých klasifikátorů. Celkový počet kontrolovaných prvků se sníží na přibližně 6 000 u okna $24 * 24$ pixelů.

Většina obrazu ovšem neobsahuje obličej a počítat 6 000 prvků pro každou pozici je neefektivní. Tato situace byla vyřešena užitím kaskády klasifikátorů, kdy prvky byly seřazeny do několika fází (např. 1, 10, 25, 50,...). Jakmile okno neprošlo jednou z fází, neobsahuje obličej a přechází se na další pozici. Průměrně se na jednu pozici zkontroluje 10 prvků.

Haarovy kaskády jsou velmi ovlivněné kvalitou a kvantitou vstupních dat. Více než 1 000 pozitivních a 1 000 negativních obrazů vytvoří dostatečně přesný klasifikátor. [18]

### 3.1.2 Trénink haarovy kaskády

Pro trénink kaskády byly snímky zaznamenány z konstantní výšky, úhlu a vzdálenosti od stolu. Stůl byl osvětlen výkonnou LED lampou umístěnou na stole. S předměty bylo manipulováno tak, aby předmět byl vždy celý v záběru kamery. V průběhu pořizování záznamu byla měněna intenzita osvětlení a natočení předmětů vůči snímacímu zařízení, viz následující obr. 13.

**Obr. 13. Pořizování tréninkových dat**

V důsledku nepřesné automatizace ořezu obrazu bylo pro vytvoření pozitivních snímků využito programu BatchCrop [19], který snímky oříznul.

Následně se pomocí 100 pozitivních snímků, resp. 2 000 negativních snímků obsahujících, resp. neobsahujících námi požadovaný předmět vytrénovala Haarova kaskáda, která byla použita pro detekci objektů ze záběrů kamery v reálném čase. Samotná tréninková data lze nalézt v příloze 1 a 2, náhled tréninku je rozebrán v následující části včetně užitých parametrů.

**Obr. 14: Vybrané ukázky pozitivních snímků**

**Obr. 15: Vybrané ukázky negativních snímků**

V obr. 14 a obr. 15 lze pozorovat typické snímky pro trénink haarových kaskád. Program $samples\_creation.bat$ spustil program pro načtení všech pozitivních snímků a vytvořil z nich vektor specifikující umístění snímků, jejich velikost a oblasti obsahující požadovaný objekt v těchto snímcích.

```
createsamples.exe -info positive/info.txt -vec vector/facevector.vec -w 20 -h 20
```

Běžně volená velikost prvků se pohybuje od $10 * 10$ do $25 * 25$ pixelů na prvek. Zvolená velikost prvku pro trénink haarovy káskády v této úloze byla $20 * 20$ pixelů.

Parametry samotného tréninku byly zvoleny takto:

```
haartraining.exe -data cascades -vec vector/facevector.vec -bg negative/bg.txt -npos 3000 -nneg 7000 -nstages 20  -mode ALL -w 20 -h 20 -nonsym -nsplits 2 -maxFalseAlarmRate 0.5 -minHitRate 0.9999
```

Kde *–data* určila typ výstupních dat (kaskáda), *-vec* načetl vytvořený vektor z $samples\_creation.bat$, *-bg* načetl negativní snímky, -npos, resp. *–nneg* určil počet použitých pozitivních, resp. negativních snímků. Zvolením *–nstages* 20 bylo použito celkem 20 fází tréninku. *–mode ALL* zastupovalo použití veškerých haarových prvků, *-w,-h* muselo souhlasit s velikostí jednotlivých prvků. *–nonsym/sym* představovalo asymetrii/symetrii předmětu a *–nsplits* zastupoval počet možných větvení slabých klasifikátorů. Velmi důležitými parametry byly *–maxFalseAlarmRate*, který určoval kolik procent negativních snímků mohlo být považováno jako tzv. falešné pozitivum a *–minHitRate*, jenž zastupoval minimální procento detekce objektu v pozitivních snímcích. [20]

Trénink každé kaskády trval v průměru 20 minut a pro každou kaskádu bylo použito přibližně 165 000 haarových prvků. 3 000 pozitivních snímků bylo generováno pomocí 100 originálně pořízených pozitivních snímků a 3 700 negativních snímků bylo použito z veřejně dostupného balíčku obrazových dat. [21]

### 3.1.3 Detekce objektů

Detekce jednotlivých objektů byla realizována pomocí funkce OpenCV $cv2.CascadeClassifier.detectMultiScale()$, jež jako detektor načetla vytrénovanou haarovu kaskádu, soubor typu XML. V rámci dodaného obrazu algoritmus vyhledal objekt vyhovující kaskádě a dalším parametrům uvedeným dále v kapitole.

Skript načetl jednotlivé kaskády, následně inicializoval kameru a v jednotlivých snímcích detekoval požadované předměty, které následně ohraničil zadanou barvou, viz následující příklad:

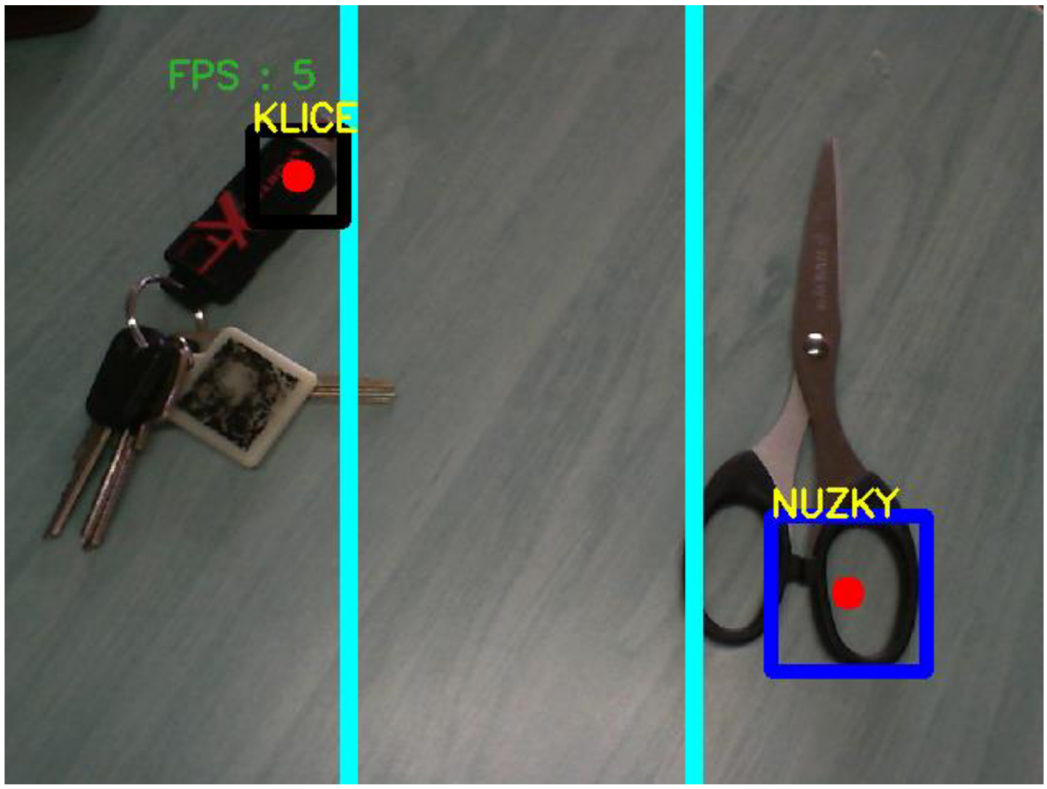
```
key = cascade.detectMultiScale(
     img,               #ve kterém obrazu hledat
     minNeighbors=8,    #minimální počet blízkých pozitivních nálezů
     minSize=(70,70),   #minimální velikost nalezeného objektu
     maxSize=(80,80));  #maximální velikost nalezeného objektu
for (cx,cy,cw,ch) in key:
        cv2.rectangle(img, (cx,cy), (cx+cw, cy+ch), (0,0,255), 5);
             #vykreslí obdélník na oblast detekovaného objektu červené
             barvy čarou velkou 5px pro každý detekovaný objekt key
```

Z důvodu velkého počtu falešných pozitiv a mnoho lokací detekce byla použita funkce $cv2.groupRectangles()$, která sjednotila překrývající se ohraničení detekovaných objektů a potlačila ty, jež se nacházely v obraze samostatně. Současně užitím skriptu s časovačem preferovala takové oblasti, ve kterých mohla sjednotit více ohraničení.

Pokud kaskáda nedetekovala objekt, v paměti zůstala zachována jeho poslední poloha a po zadaném čase se na výstupním obrazu objevilo hlášení o chybě detekce daného objektu.

### 3.1.4 Vytvoření inventářního systému

Inventářní systém byl realizován za pomoci pozice detekovaného objektu v obraze získáného funkcí $cv2.detectMultiScale()$. Obraz byl rozdělen na 3 oblasti, ke dvěma oblastem byly přiděleny konkrétní objekty. Vizualizace systému je zobrazena na obr. 16.

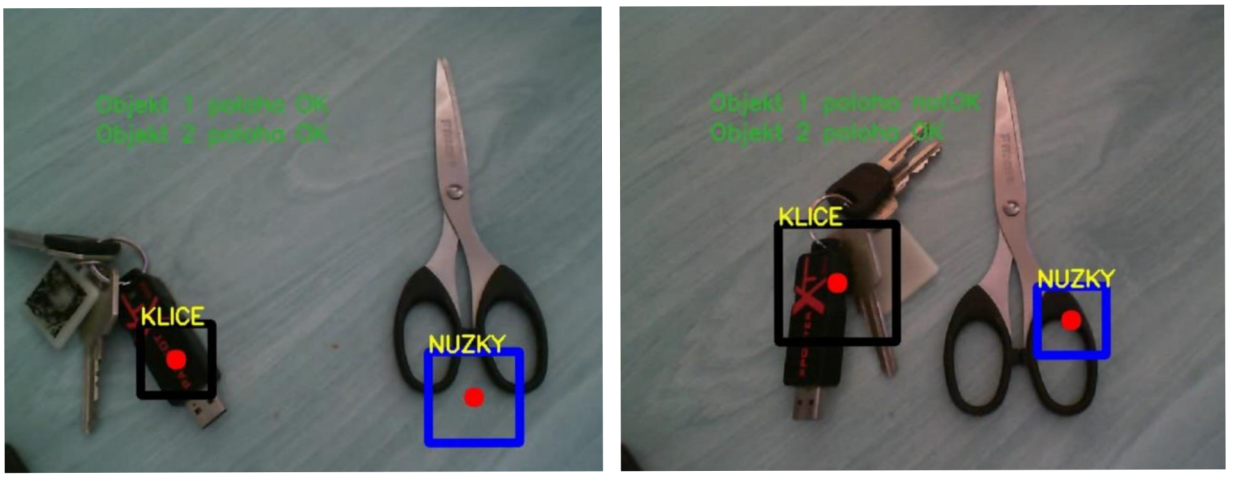



**Obr. 16: Vizualizace inventářního systému**

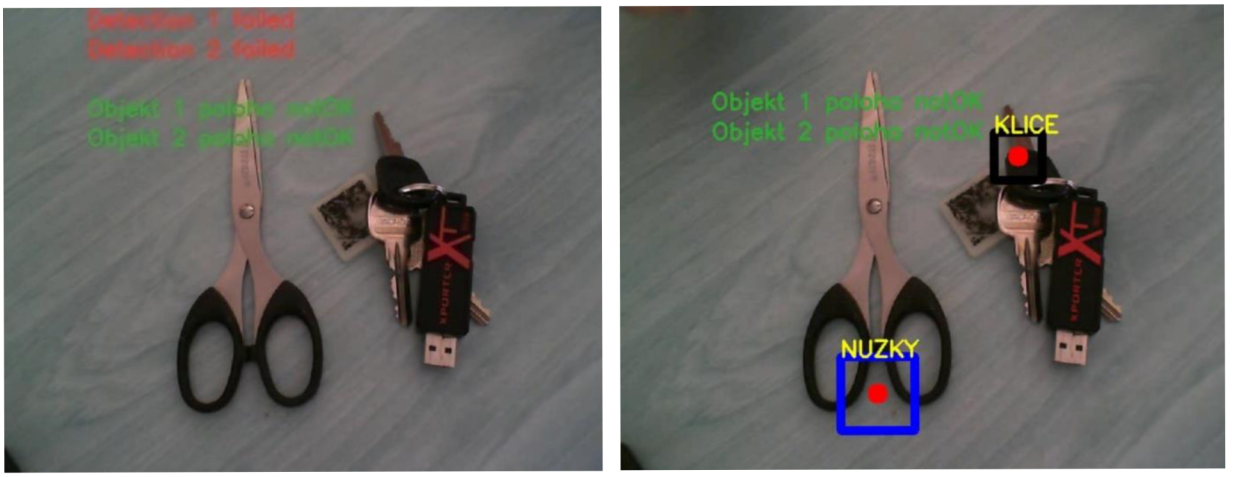
Na obr. 16 lze vidět korektní pozice předmětů, střední oblast neměla přidělený objekt.

### 3.1.5 Demonstrace algoritmu

Úloha byla ověřena v podmínkách nejvíce napodobujících pořízení tréninkových dat. Pro osvětlení byla užita stejná LED lampa a záběry z kamery byly pořizovány z podobného úhlu a vzdálenosti. Vybrané úspěšné a neúspěšné detekce společně s kontrolou polohy objektu lze vidět v obr. 17 a obr. 18.




**Obr. 17: Úspěšná detekce (vlevo), částečně korektní poloha objektů (vpravo)**




**Obr. 18: Neúspěšná detekce (vlevo), nekorektní poloha objektů (vpravo)**

Výsledný skript lze nalézt v příloze 3.

### 3.1.6 Zhodnocení úlohy

S přihlédnutím ke způsobu trénování haarových kaskád bylo použito 100 pozitivních snímků pro zajištění dostatečně přesné detekce. Pro nedostatečně tvarově a obsahově unikátní předměty bylo nutné zajistit velmi podobné světelné podmínky jaké byly použity při získání tréninkových dat. Mezi negativní snímky bylo nutné zahrnout i pozadí (deska stolu), na kterém byly pořizovány tréninková data.

Vhodná volba objektů je klíčovým prvkem kvalitní detekce. V této úloze byly demonstrovány objekty statické (nůžky) i relativně se měnící (klíče). Objekt musel být dostatečně rozlišný od pozadí, nejlépe svým obrysem vytvářet spojitou oblast a obsahovat nějakou nesourodost (oblasti jiné barvy, změny tvaru, atp.).

Parametry detekce musely být voleny pro každý objekt samostatně, na takové hodnoty, aby objekt byl po většinu času spolehlivě sledován a většina falešných detekcí byla eliminována. Mezi tyto parametry patřily počet sousedících detekcí, rozměrová kompenzace, minimální a maximální velikost ohraničení, počet sjednocených ohraničení a časová prodleva pro sledování oblasti s menším počtem sjednocených ohraničení.

Při mnou zvolenému relativně malému počtu tréninkových dat detekce často označovala předměty za jiné. Komerční užití vyžaduje databáze až tísíců pozitivních snímků. Díky tréninku s parametrem *maxFalseAlarmRate 0.5* mohlo být až 50 % negativních snímků označeno v tréninku jako pozitivní, ovšem snížením tohoto parametru se doba tréninku exponenciálně zvyšovala bez znatelnějších zlepšení v kontrolovaném prostředí úlohy. Při dalším snížení hodnoty tohoto parametru trénink nedokázal vykompenzovat požadavky tréninku, který v návaznosti selhal.

Z důvodu komplexnosti a vizuální nejednoznačnosti relativně se měnícího objektu (klíče) algoritmus tento objekt detekoval méně přesněji než objekt statický (nůžky).

Samotná implementace sledování polohy, uživatelské odezvy a kontroly detekce fungovala spolehlivě a přesně. Prostory inventářního systému byly modifikovatelné v souřadnicích $x$ i $y$.

V komerčním užití se velmi často kombinuje detekce se sledováním objektu. Detektor označí nejpravděpodobnější kandidáty na sledování, kteří jsou následně použiti jako vstupy pro sledovací algoritmy. Úloha by mohla být rozšířena právě o tuto funkci.

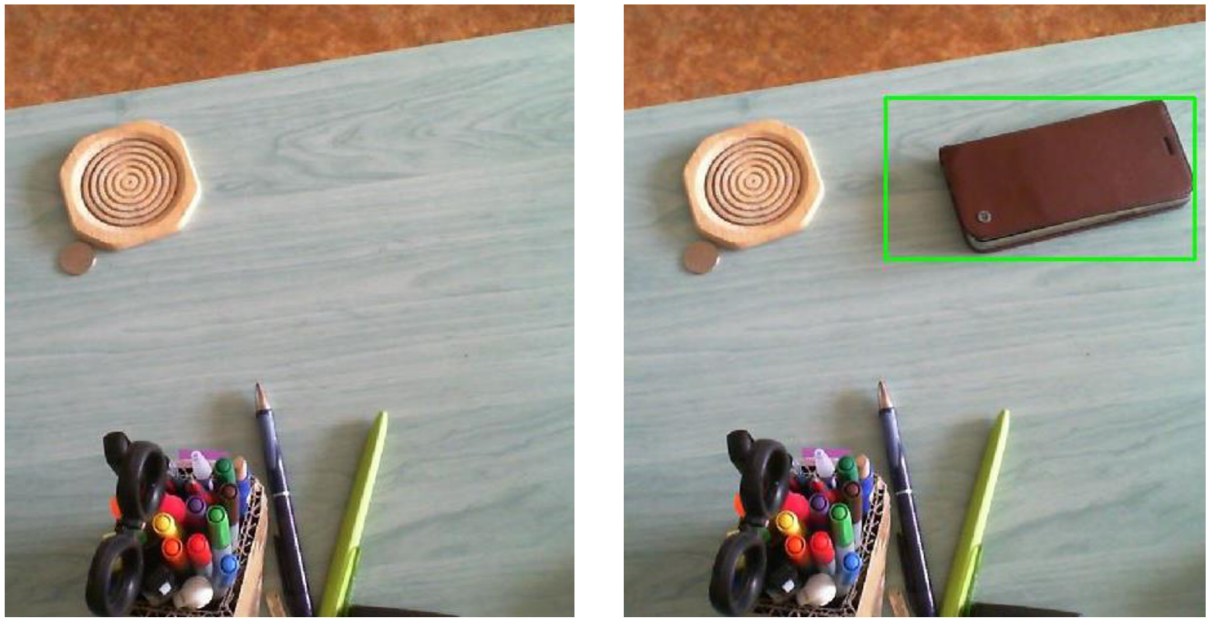
## 3.2 Sledování objektu a detekce změny

Další mechatronickou aplikací ke zpracování byla detekce změny a sledování pohybujícího se objektu. Prvním krokem bylo zjištění změny v obraze, její zaznamenání a použití v algoritmech pro sledování objektu. Poté se sledovala trajektorie hmotného bodu nahrazujícího sledovaný objekt, využitá v algoritmu pro počítání objektů a hru PONG.

### 3.2.1 Detekce změny

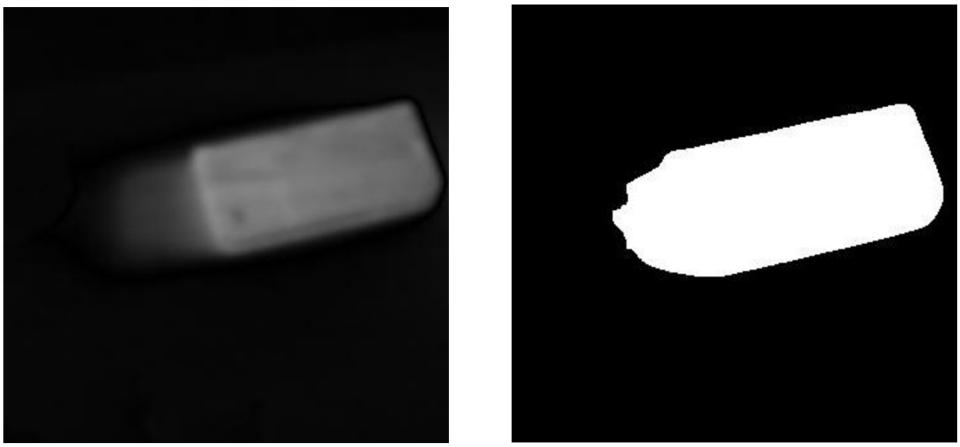
Po spuštění skriptu se uložil první obraz, se kterým byly následně všechny další porovnávany pro určení změny v obrazu. Aby se dostal předmět celý do prostoru záznamu kamery, bylo zavedeno časové prodlení po detekci první změny v obraze. Změnu lze spatřit v obr. 19.

```
"""Porovnání s prvním snímkem"""
    frameDelta = cv2.absdiff(firstFrame, gray);
              #diference mezi pozadím a aktuálním obrazem
    thresh = cv2.threshold(frameDelta, 50, 255, cv2.THRESH_BINARY);
              #přes threshold zvýrazním změny obrazu a převedu je na
              "data" pro detekci kontur
    thresh = cv2.dilate(thresh, None, iterations=2);
              #cv2.dilate redukuje nevhodný šum
```



**Obr. 19: Odhalená změna v obraze po prodlevě 5 sekund**

Příkaz $cv2.threshold()$ odstranil nedostatečně velké změny v obraze (částečná eliminace stínu, jemná změna osvětlení atp.), načež se pomocí příkazu $cv2.findContours()$ našly vhodné, dostatečně velké a spojité oblasti ke sledování viz obr. 20.



**Obr. 20: Absolutní diference (vlevo) a následný threshold (vpravo)**

### 3.2.2 Sledovací algoritmy KCF a MIL

Samotný princip těchto metod spočívá v představení obrazu sledovaného objektu danému algoritmu, jenž se „naučí“ rozeznávat objekt od jeho okolí. Velký důraz je kladen na obrazy, které neobsahují sledované objekty (stacionární lokace, budovy, pozadí, okolí objektu, atd.). Počet těchto tzv. negativních snímků se snažíme z důvodu výpočetní náročnosti omezit, ovšem za cenu spolehlivosti sledování.

Sledovací algoritmus **MIL** (Multiple Instance Learning) [22] je založen na pořízení prvotní fotky objektu a jeho blízkého okolí jako pozitivních vzorků a vzálenějšího okolí jako negativních vzorků. V každém snímku videozáznamu následně algoritmus porovnal vzorky z předchozího snímku vůči aktuálnímu a určil nejvhodnější posunutí sledovacího okna.

| Výhody | Nevýhody |
|---|---|
| **Relativně výpočetně rychlý** | Při úplném zakrytí již nesleduje objekt |
| **Stabilní i při parciálním zakrytí objektu** | Neumí zpracovat objekt, který opustí snímek.<br>Při relativně rychlém pohybu objektu nestabilní |

**Tab. 2: Výhody a nevýhody sledovacího algoritmu MIL [23]**

**KCF** (Kernelized Correlation Filter) [24] pracuje na principu posouvání vstupního objektu a hledání filtru pro konvoluci, který by minimalizoval chybu pro odhad následujícího snímku získaného regresní analýzou (ridge regression – regularizace).

| Výhody | Nevýhody |
|---|---|
| **Velmi rychlý** | Při úplném zakrytí již nedokáže sledovat objekt (chyba sledování) |
| **Stabilní i při parciálním zakrytí objektu** | Při velmi rychlém pohybu objektu jej nedokáže sledovat. |

**Tab. 3: Výhody a nevýhody sledovacího algoritmu KCF [23]**

### 3.2.3 Aplikace sledovacích algoritmů po detekci pohybu

Po zaregistrování změny a vytvoření obdélníku, který danou změnu obklopuje se inicializovaly zvolené trackery. Výstupem trackeru byla proměnná $ok$ zastupující úspěšnost trackování a proměnná $bbox$ s informací o nové pozici obdélníku. Pro zpětnou vazbu uživatele byly zavedeny popisy trackeru, počet snímků za sekundu a stav sledování.

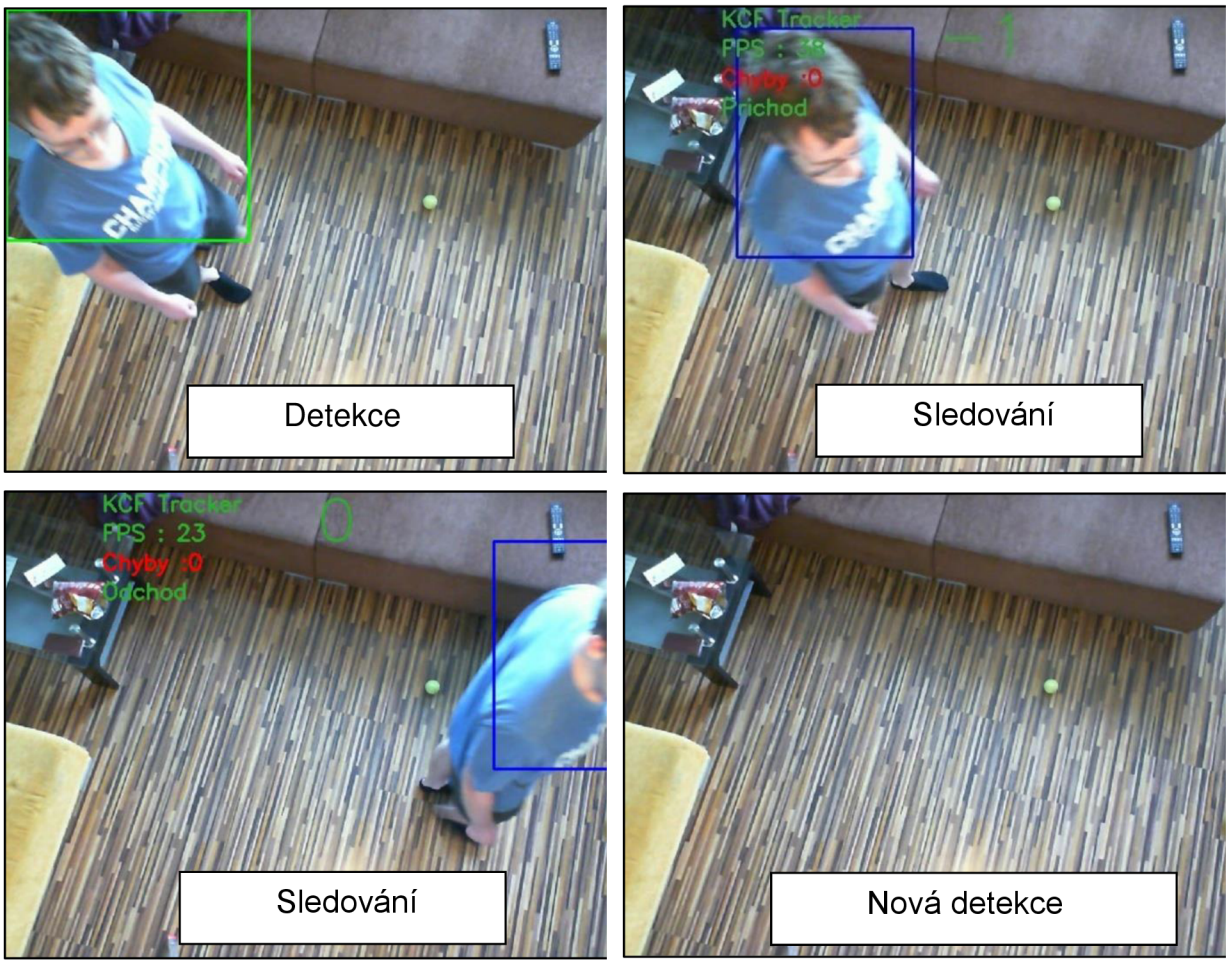
V rámci skriptu jsou k dispozici sledovací algoritmy BOOSTING, MIL, KCF, TLD, MEDIANFLOW a GOTURN dostupné v balíčku *opencv-contrib-python* získaného pomocí Pip. [23, 8]

V následující části skriptu je přiblížena zpětná vazba a fungování trackeru:

```
# Vstup kamery
ok, frame = cap.read()
# Start časovače
timer = cv2.getTickCount()
# Update trackeru
ok, bbox = tracker.update(frame)
# Výpočet FPS (snímky za sekundu)
fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
if ok:
    # Úspěšný tracking
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1) # vykreslení nového obdélníku
else :
    # Neúspěšný tracking
    cv2.putText(frame, "Trackovani neuspesne", (100,80),
                cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)
# Zobrazení typu trackeru
cv2.putText(frame, tracker_type + " Tracker", (100,20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
# Zobrazení FPS
cv2.putText(frame, "FPS : " + str(int(fps)), (100,50),
            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
#Trajektorie
xt = int(bbox[0]+bbox[2]/2) #souřadnice x středu plochy
yt = int(bbox[1]+bbox[3]/2) #souřadnice y středu plochy
cv2.circle(mask, (xt,yt), 3, (255,255,255), -1) #vykreslení bodu trajektorie
```

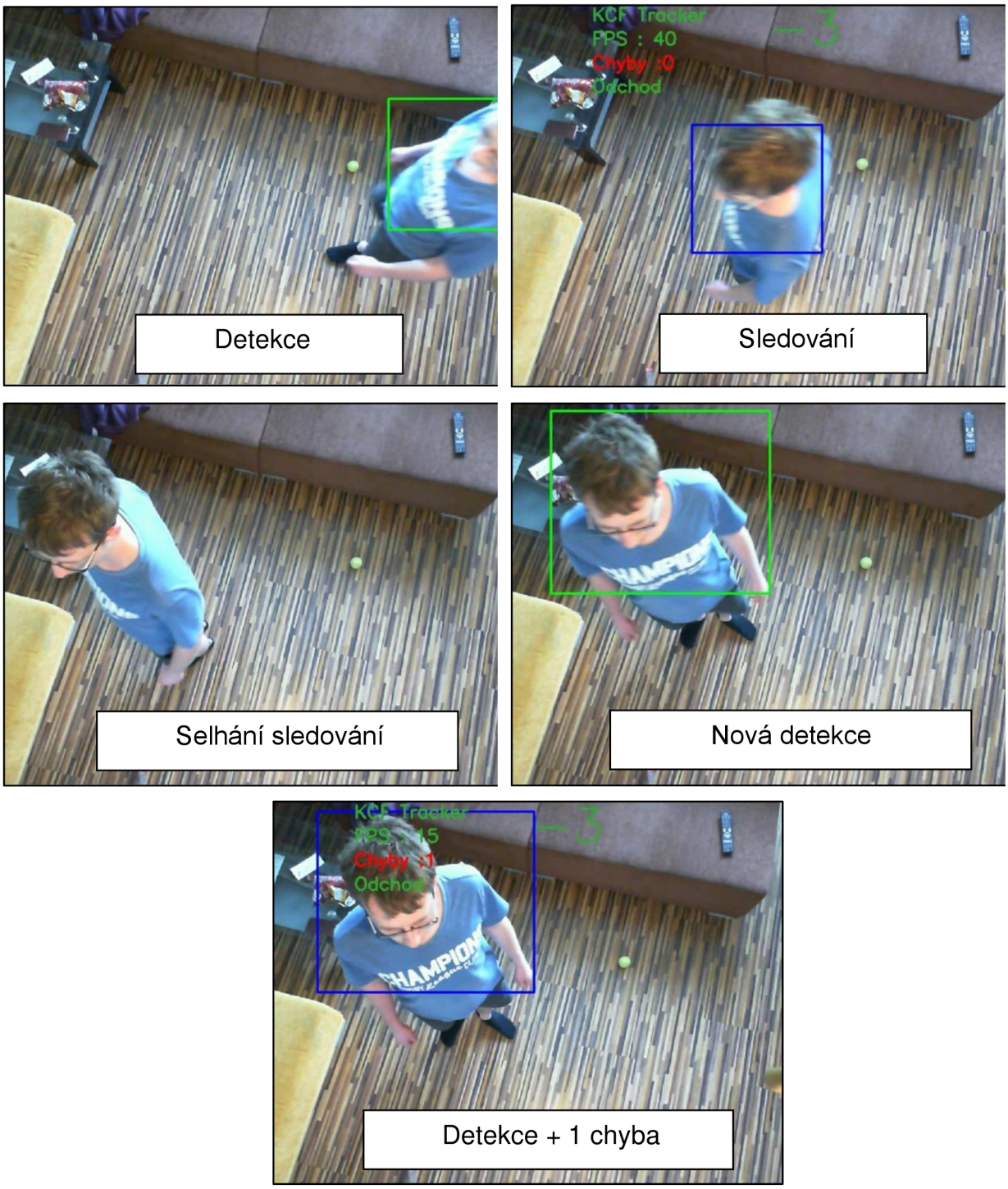
### 3.2.4 Počítání předmětů/lidí

Použitím vypočtené trajektorie sledovaného objektu bylo implementováno počítání objektů, které se pohybovaly z levé strany obrazu do pravé a naopak. V jednom, resp. opačném směru se k celkovému počtu objektů jeden přičetl, resp. odečetl. Získané obrazové materiály byly zaznamenány do obr. 21 a v příloze 4 se nachází výsledný skript.




**Obr. 21: Demonstrace algoritmu počítání předmětů**

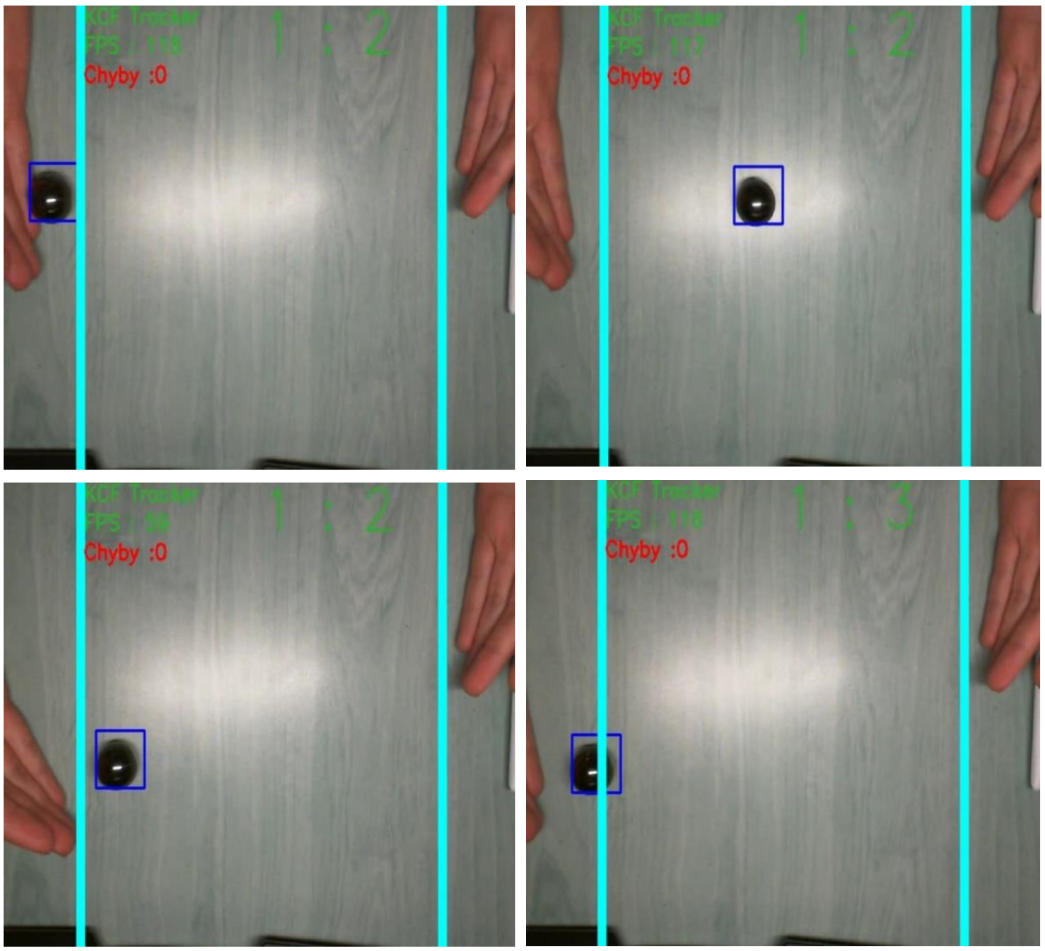
Jakmile objekt opustil levou nebo pravou stranu obrazu, sledovací algoritmus opět hledal další významnou změnu ke sledování. V případě, že sledovací algoritmus selhal v průběhu sledování (objekt opustil sledovanou oblast jiným způsobem nebo měl příliš vysokou rychlost), program tuto chybu zaregistroval a zobrazoval jejich celkový počet, jak je naznačeno na obr 22. Takto program činil dokud nebyl vypnut.




**Obr. 22: Demonstrace detekce chyby**

### 3.2.4.1 Hra PONG

Upravení této úlohy dalo za vznik hře PONG pomocí počítačové vize. Detekce změny probíhala pouze v barevném pásmu odpovídajícímu sledovanému objektu (železná kulička), a to pro eliminaci sledování jiných objektů. Jakmile těžiště obrazce sledované kuličky překročilo mezní hranici, protějšímu hráči byl přičten bod. Obrazový materiál se nachází na obr. 23 a výsledný skript v příloze 5.




**Obr. 23: Demonstrace hry PONG**

Program byl rovněž vybaven detekcí chyby, viz obr. 22. Obnovení funkce počítání skóre proběhlo až po vstoupení do střední části obrazovky, jak bylo znázorněno na obr. 16, a to pro zamezení opětovnému přičítání skóre, když se kulička pohybovala v okolí „brankové čáry“ stolu.

### 3.2.5 Zhodnocení úlohy

V porovnání nejvyužívanějších trackerů **MIL** a **KCF** s přihlédnutím k realizované úloze se použil tracker **KCF**, a to díky spolehlivému hlášení ztráty hledaného objektu a menší výpočetní náročnosti, což vedlo ke zvýšení kontrolovaných snímků za sekundu. Pro běžná užití se tento tracker jevil jako nejuniverzálnější.

Realizace této úlohy měla jistá omezení a předpoklady:

1. V průběhu úlohy musely být zajištěny neměnné světelné podmínky.
2. Kamera musela být zajištěna proti posunu.
3. Algoritmus dokázal sledovat pouze první objekt, jenž vstoupí do snímacího prostoru kamery.
4. Úloha musela být korigovaná na dané použití, minimální velikost detekované změny a doba prodlení po detekci první změny musela být vhodně zvolena.
5. Objekt pohybující se vysokou rychlostí algoritmus nebyl schopen spolehlivě sledovat.
6. Sledovaný objekt musel být dostatečně barevně odlišný od pozadí.

Úloha také trpěla ojedinělými falešnými pozitivy způsobené změnami obrazu při automatickém zaostření kamery, jež nelze deaktivovat a program bylo nutno spustit znovu. Tento problém by se mohl vyřešit použitím jiného záznamového zařízení.

V případě zajištění výše zmíněných podmínek a omezení byl algoritmus stabilní a spolehlivý, přičítání a odečítání objektů probíhalo bezchybně a uživatelská odezva dodávala relevantní informace.

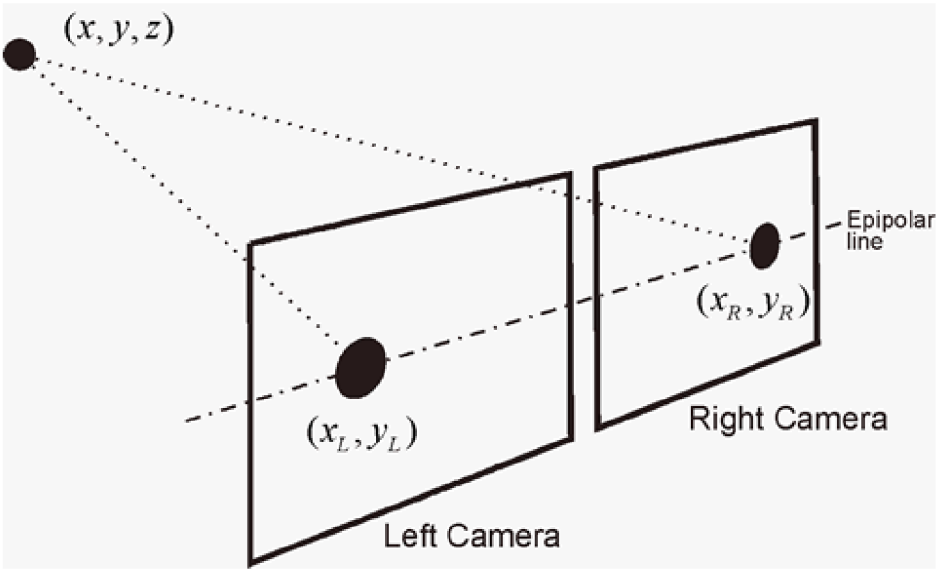
Úlohu by bylo možné dále rozšířit o sledování více objektů najednou, o záznam trajektorie objektu či o obnovení kontrolního obrazu, pokud se objekt nebude pohybovat déle než zadaný čas.

## 3.3 Hloubková mapa – stanovení bezpečné vzdálenosti pomocí stereovize

Poslední realizovaná mechatronická úloha byla stanovení bezpečné vzdálenosti pomocí hloubkové mapy získané pomocí stereovize. Za použití dvou kamer umístěných dle následujícího obr. 24.

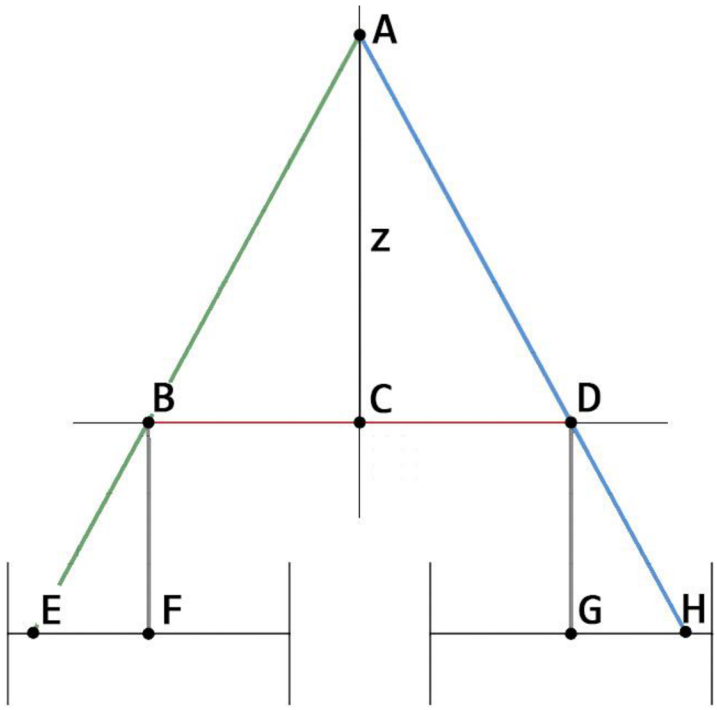

**Obr. 24: Vlastní implementace stereokamery**

Bod v prostoru je na obou kamerách zobrazen jinde, a pokud lze nalézt sobě odpovídající projekce tohoto bodu ve snímcích kamer, je možno odhadnout v jaké vzdálenosti od kamery se nachází. Pokud bychom snímky z obou kamer sjednotili, bližší body by měly své projekce dále od sebe, narozdíl od vzdálenějších bodů, které by se pohnuly minimálně. Vizuální reprezentaci [25] nejběžnější formy stereovize lze vidět na následujícím obr. 25.




**Obr. 25: Princip stereovize [25]**

Tato metoda zjišťování polohy se nazývá triangulace. Tento trigonometrický problém je řešen pomocí obr. 26 a rovnic (1) a (2).




**Obr. 26: Princip triangulace pozice**

Pixel **A** obsahuje barvu své pozice. Stereovize poskytuje dva obrazy stejné scény, pozice je definována v souřadnicovém systému $x, y$, jak je naznačeno na obr. 25. Souřadnice $z$ zastupuje vzdálenost od kamer. Světlo z bodu **A** je skrz ohnisko kamery (body **B** a **D**) promítáno na snímací plochu jako body **E** a **H**. Je nutno znát vzdálenost kamer **BD**.

Při této konfiguraci je použito podobnosti trojúhelníků:

- **ACB** a **BFE**
- **ACD** a **DGH**

Posuv lze vypočítat následovně:

$$d = EF + GH = BF\left(\frac{EF}{BF} + \frac{GH}{BF}\right) =$$

$$BF\left(\frac{EF}{BF} + \frac{GH}{DG}\right) = BF\left(\frac{BC + CD}{AC}\right) = BF * \frac{BD}{AC} = \frac{k}{z} \quad (1)$$

, a tedy:

$$d = \frac{k}{z} \quad (2)$$

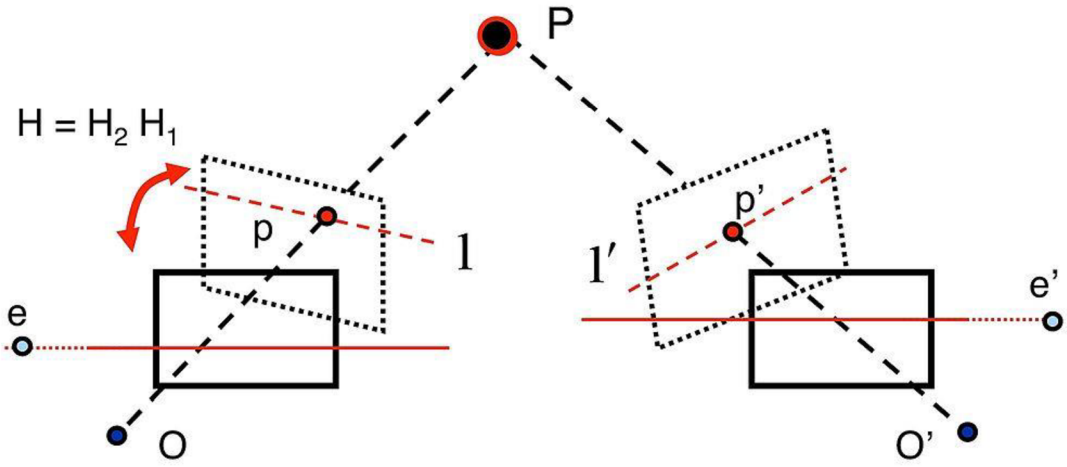
, kde $k$ [mm] je součin vzdálenosti kamer a vzdálenosti snímací plochy od ohniska kamery, $d$ [mm] je posuv (součet vdáleností promítnutých bodů **E** a **H** od kolmého průmětu ohniska na snímací plochu **F** a **G**) a $z$ [mm] je vzdálenost pixelu od kamer.

### 3.3.1 Kalibrace Stereokamery

Obě kamery musely být kalibrovány samostatně pro kompenzaci optických defektů, podrobněji řešeno již v kap. 2.4. Kamery poté musely zjistit, jak jsou vůči sobě posunuty a natočeny.

Tento problém vyřešila funkce $cv2.stereoCalibrate()$, jejímž výstupem byly právě translační a rotační vektor levé kamery vůči kameře pravé. Pomocí funkce $cv2.stereoRectify()$ byl zjištěn vzorový obraz obou kamer promítnutý do jedné roviny, použitý v další funkci $cv2.initUndistortRectifyMap()$. Tato funkce generovala výslednou transformaci pro každou z kamer, použitou ve funkci $cv2.remap()$ pro kompenzaci všech výše uvedených veličin.

Princip výše uvedeného postupu (tzv. rektifikace stereo-obrazu) je znázorněn na následujícím obr. 27, kde 1 a 1‘ zastupují originální obrazový vstup a O a O‘ jejich promítnutí do jedné roviny.




**Obr. 27: Princip rektifikace stereo-obrazu [26]**


**Obr. 28: Levý a pravý obraz, originální (nahoře) a zkalibrovaný (dole)**

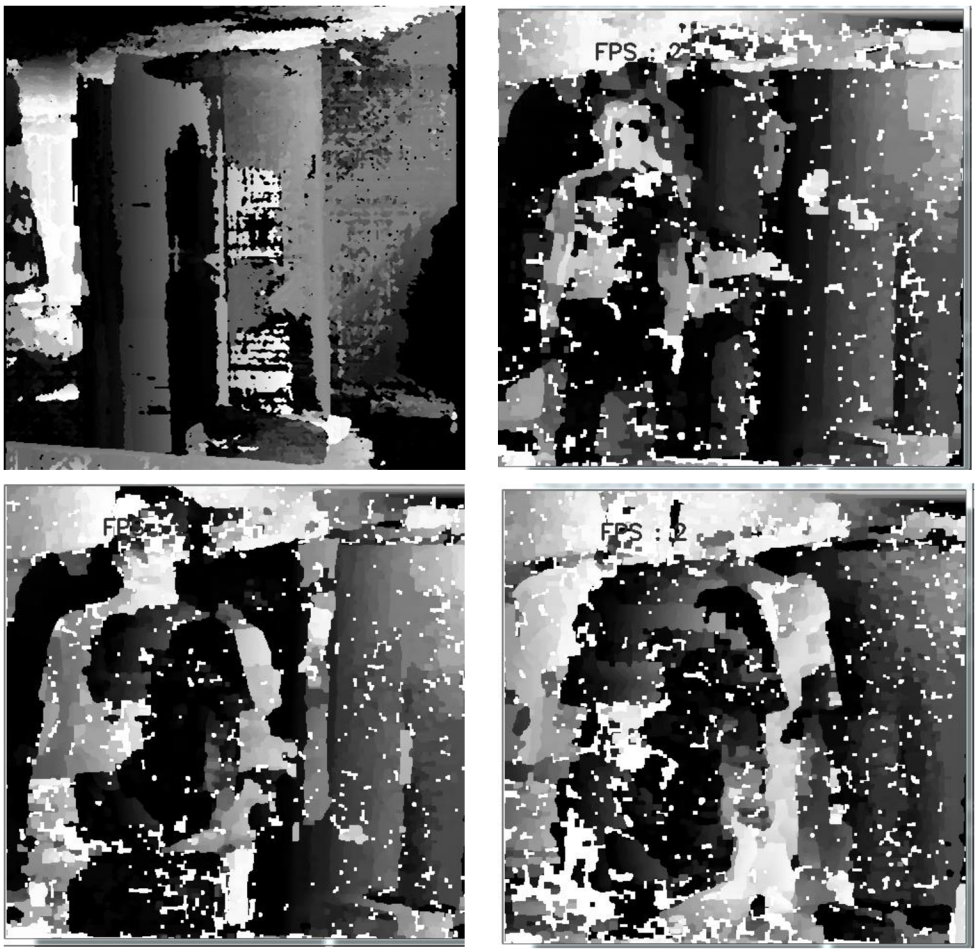
V obr. 28 lze porovnat originální a zkalibrované záběry z kamer. Značná deformace (za předpokladu zajištění polohy kamer) mohla být způsobena nevhodným natočením jedné z kamer, nedostatečně přesnou kalibrací či autofokusem, který nebylo možné deaktivovat a velmi často zkresloval kalibrovací obrazec.

### 3.3.2 Hloubková mapa

V této části bylo nutno nalézt skupinu pixelů odpovídající stejnému bodu ve snímcích obou kamer. Realizace a samotné vytvoření hloubkové mapy bylo řízeno OpenCV funkcí $cv2.stereoSGBM()$ (SGBM – semi-global block matching, pozn. překlad: semi-globální blokové párování) [27, 28] s následujícími parametry:

```
"""STEREO_SGBM - popis parametrů"""
stereo = cv2.StereoSGBM_create(
        minDisparity = -64,      #minimální možná disparita(nesourodost)
        numDisparities = 192,    #maximální možná disparita(nesourodost)
        blockSize = 5,           #velikost oblasti ke spárování
        uniquenessRatio = 1,     #procentuální rozpětí pro
                                  potvrzení správnosti párování
        speckleRange = 2,     #maximální variace disparity v každém spojitém prvku
        speckleWindowSize = 150,#maximální velikost spojité oblasti,
                                  kdy ještě šum v této oblasti bude anulován
        disp12MaxDiff = 10,      #maximální diference mezi pixely obou snímků
        P1 = 600,     #trest změny sousedících pixelů o 1 hodnotu
        P2 = 2400     #trest změny sousedících pixelů o více než 1 hodnotu
        )
```

Příklad pořízené hloubkové mapy lze vidět na následujícím obr.29:




**Obr. 29: Ukázka pořízené hloubkové mapy**

Jak lze vidět na obr. 29, okraje mapy jsou zatíženy velkou chybou. Pro aplikaci hloubkové mapy na bezpečnou vzdálenost bylo nutno výsledný obraz upravit, a to pomocí následujících funkcí:

```
roi = cv2.selectROI(dispmap)

frameDelta = cv2.absdiff(firstFrame, dispmap);
frameDelta = np.uint8(frameDelta*255)
thresh = cv2.threshold(frameDelta, 50, 245, cv2.THRESH_BINARY)[1];
frameDeltaROI = frameDelta[int(roi[1]):int(roi[1]+roi[3]),
int(roi[0]):int(roi[0]+roi[2])]
dispmapROI = np.uint8(dispmap*255)
dispmapROI = dispmapROI[int(roi[1]):int(roi[1]+roi[3]),
int(roi[0]):int(roi[0]+roi[2])]
```

, kde *roi* zastupoval konečnou oblast sledování, *frameDelta* obsahoval změnu vůči pozadí, *thresh* eliminoval šum a *frameDeltaROI* a *dispmapROI* pracovaly už s oříznutým výsledným obrazem.

### 3.3.3 Stanovení bezpečné vzdálenosti a zpětná vazba

Z již upraveného obrazu hloubkové mapy se zaznamenala střední hodnota, jež byla použita jako kritérium pro stanovení vzdálenosti objektu. Je nutno podoknout, že s volbou této veličiny zajišťujeme rozměrovou stálost objektu vstupujícího do prostoru snímaného kamerami.

Algoritmus byl kalibrován na dvě odezvy:

| | |
|---|---|
| **DANGER** | Objekt se nachází v nebezpečné zóně – **varování** |
| **STOP** | Objekt je příliš blízko – **zastavit stroj** |

**Tab. 4: Odezvy algoritmu bezpečné vzdálenosti**

Kalibrace vzdálenosti byla provedena v rámci programu, a to uložením střední hodnoty hloubkové mapy po stisknutí klávesy.

Na obr. 30 byly demonstrovány výše uvedené odezvy programu, kde jako pohybující se objekt vystupuje člověk. Výsledný skript lze nalézt v příloze 6.


**Obr. 30: Demonstrace odezvy algoritmu**

### 3.3.4 Zhodnocení úlohy

Přestože OpenCV disponuje dostačujícími funkcemi pro práci se stereovizí, vstupní data byla klíčová pro tento typ úlohy.

Bylo nutno dostatečně přesně provést kalibraci a následnou stereokalibraci kamer. I v případě velmi přesné kalibrace, stabilního osvětlení a nepohyblivých kamer hloubková mapa přesně neodpovídala reálné situaci v záběru.

Při implementaci úlohy v nových prostorech bylo pokaždé nutno volit vhodné parametry $cv2.stereoSGBM()$, proto bylo využito uživatelského rozhraní, kde bylo možno tyto parametry v reálném čase měnit.

Pro dostatečně kvalitní stereovizi konvenčních kamer bylo zapotřebí vysoké rozlišení obrazu spojené s dostatečným výpočetním výkonem. Profesionální stereokamery využívají například infračervenou projekci mapovací sítě do prostoru, díky čemuž získávají další bod pro porovnání.

Jelikož úloha nedokázala rozeznat jednotlivé objekty v záběru a pracovala se střední hodnotou hloubkové mapy, vícero vzdálených předmětů odpovídalo jednomu předmětu, který se nacházel blíže. Toto defenzivní chování algoritmu může být v některých případech žádoucí, v jiných nevhodné. Hloubková mapa měla také obtíže v odhadu vzdálenosti větších ploch jedné barvy, nestálostí díky různým zaostřením kamer při nevhodném osvětlení a s objekty, které měly barvu stejnou nebo podobnou jako pozadí za nimi. V této oblasti by bylo možno provést vylepšení programu, například implementací sledování oblasti změny či přímo detekcí osob a objektů.

Z důvodu výpočetní náročnosti funkce $cv2.StereoSGBM()$ by mohla být použita i méně přesná funkce $cv2.StereoBM()$ s omezenými parametry a jiným způsobem výpočtu, popřípadě zavést časovač, který by hloubkovou mapu počítal jen v daných časových intervalech.

Na zařízeních, která jsem měl v době vypracování úloh k dispozici (notebook a osobní stolní počítač) jsem zaznamenal prodlevu záznamu kamery vůči výpočtu hloubkové mapy, zapsanou v tab. 5. OpenCV s Pythonem využívá pouze jedno jádro procesoru, není-li specifikováno jinak.

| Zařízení | Prodleva [ms] | FPS [snímky/s] |
|---|---|---|
| **Stolní počítač (CPU Intel i5-3570K + 32GB RAM)** | 490 | 2 |
| **Notebook Lenovo ThinkPad E530** | 850 | 1 |

**Tab. 5: Porovnání výpočetní rychlosti algoritmu na různých zařízeních**

Pro pokrytí větší oblasti by bylo nutné implementovat kamery s širším pořizovacím úhlem, použít vícero kamer nebo kamery připevnit na otáčející se součást.

# 4. ZÁVĚR

V předešlém textu byly nejprve vysvětleny základní principy počítačové vize, její využití, použité programové a technologické vybavení, a to formou rešerše v kap. 1. Porozumění těmto principům bylo hlavním předpokladem pro jejich aplikaci v dalších kapitolách.

V této práci vznikly tři vzorové úlohy využitelné v mechatronice, a to pomocí otevřené knihovny počítačového vidění OpenCV. Všechny úlohy byly naprogramovány pomocí programovacího jazyka Python, hlavním vstupem byly obrazové data získaná pomocí kamer Microsoft LifeCam Cinema.

V první úloze, Rozpoznání objektu a jeho lokalizace, jsem se zabýval tréninkem haarových kaskád pro rozeznání dvou objektů na desce stolu. Přestože se jedná o pokročilou metodu počítačového učení, díky solidně zpracované dokumentaci se trénink zdařil a program úspěšně rozeznával zadané objekty. Jako i v následujících úlohách (i díky málemu počtu pozitivních snímků v první úloze) spolehlivost programu úzce souvisela se světelnými podmínkami.

V druhé úloze, Sledování objektu a detekce změny, bylo řešena volba vhodných sledovacích algoritmů použitých pro sledování detekované změny v záběru kamery. Zde bylo nutno zavést prodlevu po detekci první malé změny pro umožnění vstoupení celého objektu do záběru. Jako nejspolehlivější a nejvhodnější se jevil sledovací algoritmus KCF (Kernel Correlation Filter), který byl následně použit pro počítání odchodu a příchodu lidí ze záběru a také pro vlastní interpretaci hry PONG. Úloha byla dále zhodnocena a navrhnuty další rozšíření a úpravy.

V poslední úloze, Hloubková mapa a stanovení bezpečné vzdálenosti, bylo užito dvou kamer v konfiguraci stereovize. Pomocí jejich kalibrace a rektifikace stereo-obrazu bylo možno aplikací triangulace zjistit vzdálenost shodných pixelů od kamery. Tento výstup se nazýval hloubková mapa. Následně, za použití střední hodnoty hloubkové mapy jako reference, byly zjišťovány relativní změny v hloubkové mapě a nakalibrovány nebezpečné a kritické hodnoty hloubkové mapy, vyzkoušené vstupem osoby do záběru kamery. V této úloze bylo klíčové rozlišení kamer, které v mnou použitých zařízeních bylo hlavním limitujícím faktorem.

Z případných budoucích úprav programů bych doporučil rozsáhlejší tréninková data první úlohy, rychlejší obnovovací frekvenci a absenci autofocusu kamer pro úlohu druhou, společně s rozšířením pro sledování více objektů najednou. Pro poslední úlohu by bylo vhodné použít kvalitnějších kamer, i za cenu větší výpočtové náročnosti hloubkových map.

# 5. SEZNAM POUŽITÝCH ZDROJŮ


[1] The 3-Clause BSD License. *Open Source Initiative* [online]. 1998 [cit. 2018-05-02]. Dostupné z: https://opensource.org/licenses/BSD-3-Clause

[2] *Python* [online]. Python Software Foundation, 2001 [cit. 2018-05-02]. Dostupné z: https://www.python.org

[3] *OpenCV library* [online]. Santa Clara: Intel Corporation, Willow Garage, Itseez, 2000 [cit. 2018-05-02]. Dostupné z: https://opencv.org

[4] *NumPy* [online]. 2006 [cit. 2018-05-02]. Dostupné z: http://www.numpy.org

[5] LOWE, David. The Computer Vision Industry. *The University of British Columbia* [online]. Vancouver, 2015 [cit. 2018-05-02]. Dostupné z: https://www.cs.ubc.ca/~lowe/vision.html

[6] *Mobileye: Autonomous Driving & ADAS (Advanced Driver Assistance Systems)* [online]. New York: Mobileye, 1999 [cit. 2018-05-02]. Dostupné z: https://www.mobileye.com

[7] Microsoft LifeCam Cinema. In: *Microsoft* [online]. 2011 [cit. 2018-04-13]. Dostupné z: https://compass-ssl.microsoft.com/assets/74/aa/74aa4472-64c4-441a-a455-95f075739946.jpg?n=ic_lcc_large.jpg

[8] *Python Package Index* [online]. Python Software Foundation, 2014 [cit. 2018-05-02]. Dostupné z: https://pypi.org

[9] Free Public Domain CC0 Image: Brown And Black Lighted Flower Bud Picture. Image: 86220326. In: *Dreamstime* [online]. Bucharest, 2011 [cit. 2018-05-02]. Dostupné z: https://www.dreamstime.com/brown-black-lighted-flower-bud-public-domain-image-free-86220326

[10] Welcome to opencv documentation!. *OpenCV 2.4. documentation* [online]. opencv dev team, c2011-2014 [cit. 2018-05-13]. Dostupné z: https://docs.opencv.org/2.4/

[11] Pattern. In: *OpenCV documentation* [online]. opencv dev team, c2011-2014 [cit. 2018-05-13]. Dostupné z: https://docs.opencv.org/2.4/_downloads/pattern.png

[12] Camera calibration With OpenCV. *OpenCV Documentation* [online]. opencv dev team, c2011-2014 [cit. 2018-05-13]. Dostupné z: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[13] Cascade Classification. *OpenCV Documentation* [online]. opencv dev team, c2011-2014 [cit. 2018-05-13]. Dostupné z: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

[14] VIOLA, Paul a Michael J. JONES. Robust Real-Time Face Detection. *International Journal of Computer Vision* [online]. Kluwer Academic Publishers, 2004, **57**(2), 137–154 [cit. 2018-04-13]. Dostupné z: http://www.vision.caltech.edu/html-files/EE148-2005Spring/pprs/viola04ijcv.pdf

[15] GOKHALE, Apoorva. Haar Cascades Explained In 2 Minutes!. In: *Society of Robotics and Automation VJTI* [online]. c2011-2014 [cit. 2018-05-13]. Dostupné z: http://sra.vjti.info/blog/blog-posts/image-processing-haar-cascades-explained-in-2-minutes

[16] PATEL, Savan. Chapter 6: Adaboost Classifier: Machine Learning 101. *Medium* [online]. 2017. Dostupné také z: https://medium.com/machine-learning-101/https-medium-com-savanpatel-chapter-6-adaboost-classifier-b945f330af06
[17] Face Detection using Haar Cascades. *OpenCV* [online]. Santa Clara: OpenCV Team [cit. 2018-04-13]. Dostupné z: https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html
[18] BALL, Thorsten. TRAIN YOUR OWN OPENCV HAAR CLASSIFIER. *Coding Robin* [online]. 2013 [cit. 2018-04-13]. Dostupné z: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html
[19] *BatchCrop: Image cropping made easy!* [online]. Istanbul: Atarca Software, c2018 [cit. 2018-04-15]. Dostupné z: http://www.batchcrop.com
[20] REZAEI, Mahdi. *Creating a Cascade of Haar-Like Classifiers: Step by Step* [online]. Auckland [cit. 2018-04-13]. Dostupné z: https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar-Like_Classifiers_Step_by_Step.pdf. Tutorial. University of Auckland.
[21] HANDAGA. Tutorial-haartraining. In: *GitHub* [online]. San Francisco, 2008 [cit. 2018-05-14]. Dostupné z: https://github.com/handaga/tutorial-haartraining/tree/master/data/negatives
[22] BABENKO, Boris, Ming-Hsuan YANG a Serge BELONGIE, ZHAO, Kelsie, ed. Visual Tracking with Online Multiple Instance Learning. In: *Stanford Vision Lab* [online]. Stanford, 2015 [cit. 2018-04-13]. Dostupné z: http://vision.stanford.edu/teaching/cs231b_spring1415/slides/MIL_kelsie.pdf
[23] MALLICK, Satya. Object Tracking using OpenCV (C++/Python). *Learn OpenCV* [online]. San Diego, 2017 [cit. 2018-04-13]. Dostupné z: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/
[24] HENRIQUES, J. F., R. CASEIRO, P. MARTINS a J. BATISTA. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2015 [cit. 2018-04-13]. DOI: 10.1109/TPAMI.2014.2345390. ISSN 0162-8828. Dostupné z: http://www.robots.ox.ac.uk/~joao/publications/henriques_tpami2015.pdf
[25] CALIN, G. a V. O. RODA. Real-time disparity map extraction in a dual head stereo vision system. In: *Scientific Electronic Library Online*[online]. Bahía Blanca, 2006 [cit. 2018-04-13]. ISSN 0327-0793. Dostupné z: http://www.scielo.org.ar/scielo.php?script=sci_arttext&pid=S0327-07932007000100005
[26] SAVARESE, Silvio. A visual representation of the variables used in image rectification example. In: *Wikipedia* [online]. San Francisco: Wikimedia Foundation, 2014 [cit. 2018-04-13]. Dostupné z: https://en.wikipedia.org/wiki/Image_rectification#/media/File:Lecture_1027_stereo_01.jpg
[27] RAMBHIA, Jay. Disparity Map: Computer Vision. *Jay Rambhia's Blog* [online]. 2013 [cit. 2018-04-13]. Dostupné z: http://www.jayrambhia.com/blog/disparity-mpas
[28] Camera Calibration and 3D Reconstruction. *OpenCV documentation* [online]. Santa Clara: OpenCV Team, 2014 [cit. 2018-04-13]. Dostupné z: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

# 6. SEZNAM PŘÍLOH

**Přílohy:**



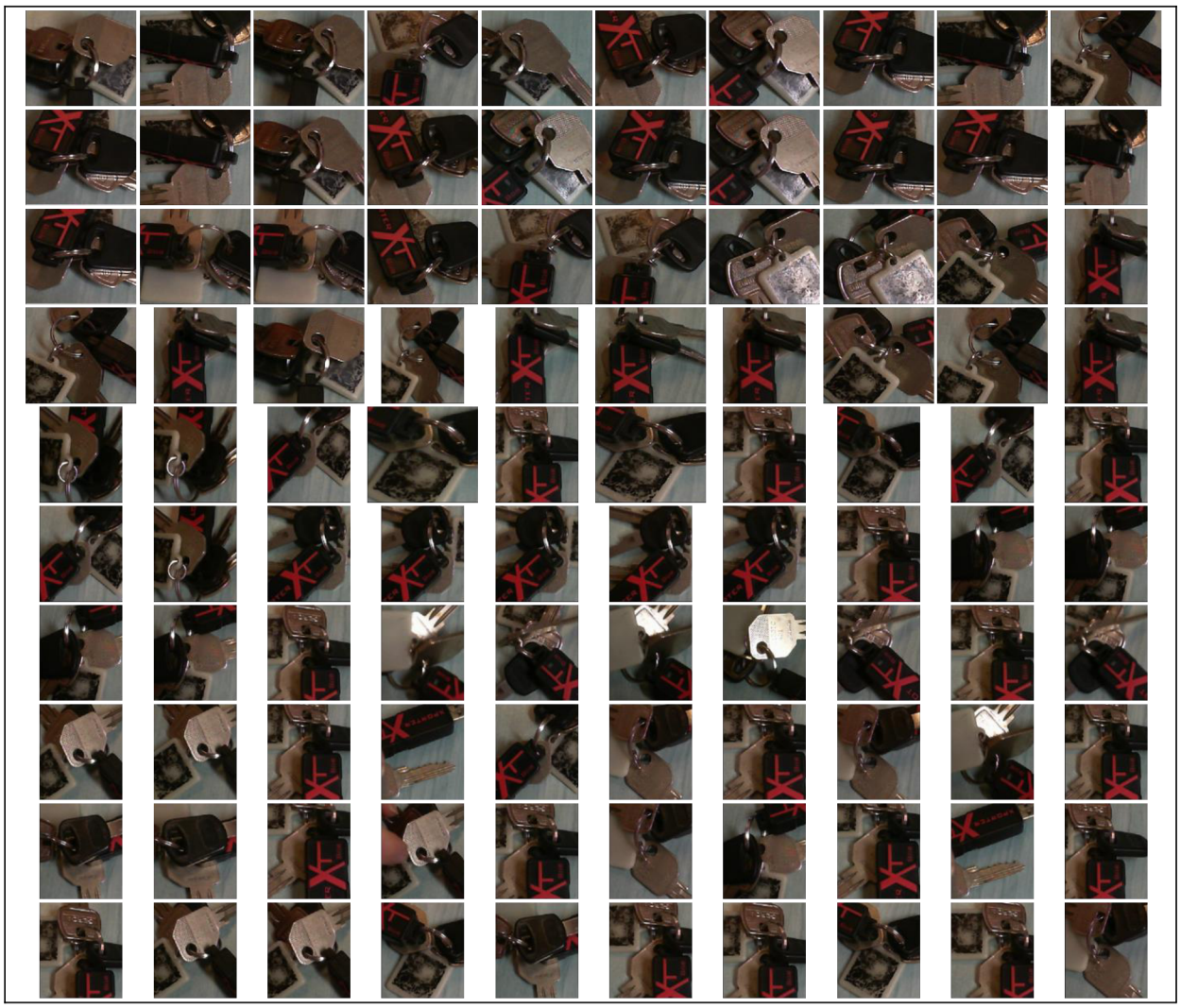
**Přílohy na CD:**

- **haarcascades** - složka obsahuje vytrénované haarovy kaskády
- **Haar Training** - obsahuje vše potřebné k tvorbě haarových kaskád bez obrazového materiálu včetně vlastního českého návodu
- **Obrazový materiál** - složka obsahuje náhled tréninkových dat z přílohy 1 a 2, šachovnicový kalibrační obrazec a obrázek 2x2pixels.png použitý v kapitole 2.2
- **object_recognition_Haar.py** - skript jazyka Python, načte haarovy kaskády ze složky haarcascades a začne objekty detekovat + nahrává výstup
- **motion&object_tracking.py** - skript jazyka Python, zahájí detekci změny a sledování, viz první úloha + nahrává výstup
- **motion&object_PONG_kulicka.py** - skript jazyka Python, zahájí detekci změny a sledování, viz první úloha - modifikace PONG + kalibrace na tmavý objekt + nahrává výstup
- **stereovision_safedistance.py** - skript jazyka Python, zahájí stereokalibraci kamer pomocí šachovnicového vzoru vedoucí k hloubkové mapě

# 7. PŘÍLOHY

## PŘÍLOHA 1 - Tréninková data 1/2

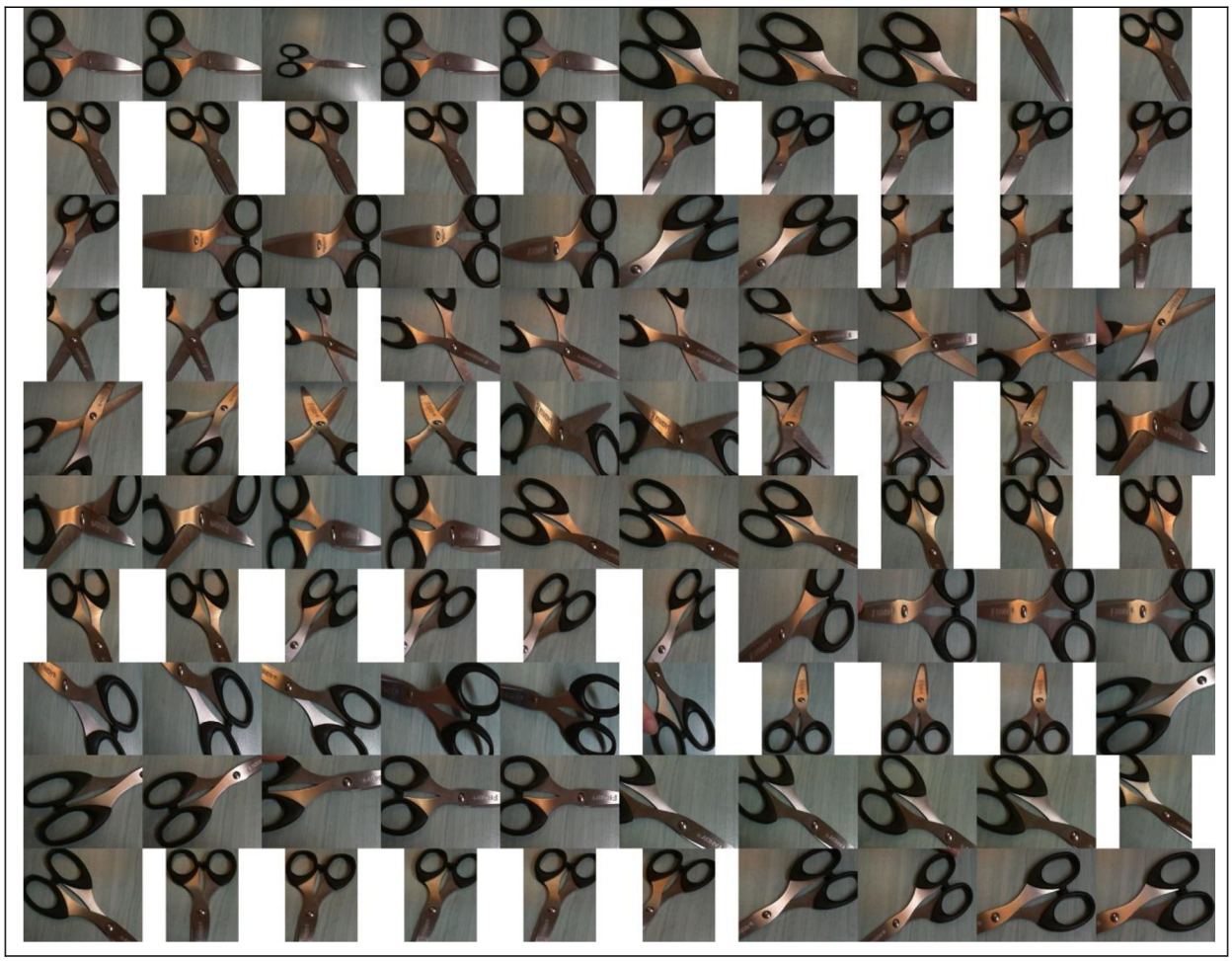
Při pořizování snímků je vhodné aby snímky obsahovaly pouze objekt určený k detekci. V následujícím obrázku 31 je zobrazeno 100 pozitivních snímků užitých pro trénink haarovy kaskády rozpoznávající právě tento objekt.



**Obr. 31: Tréninková data objektu klíče**

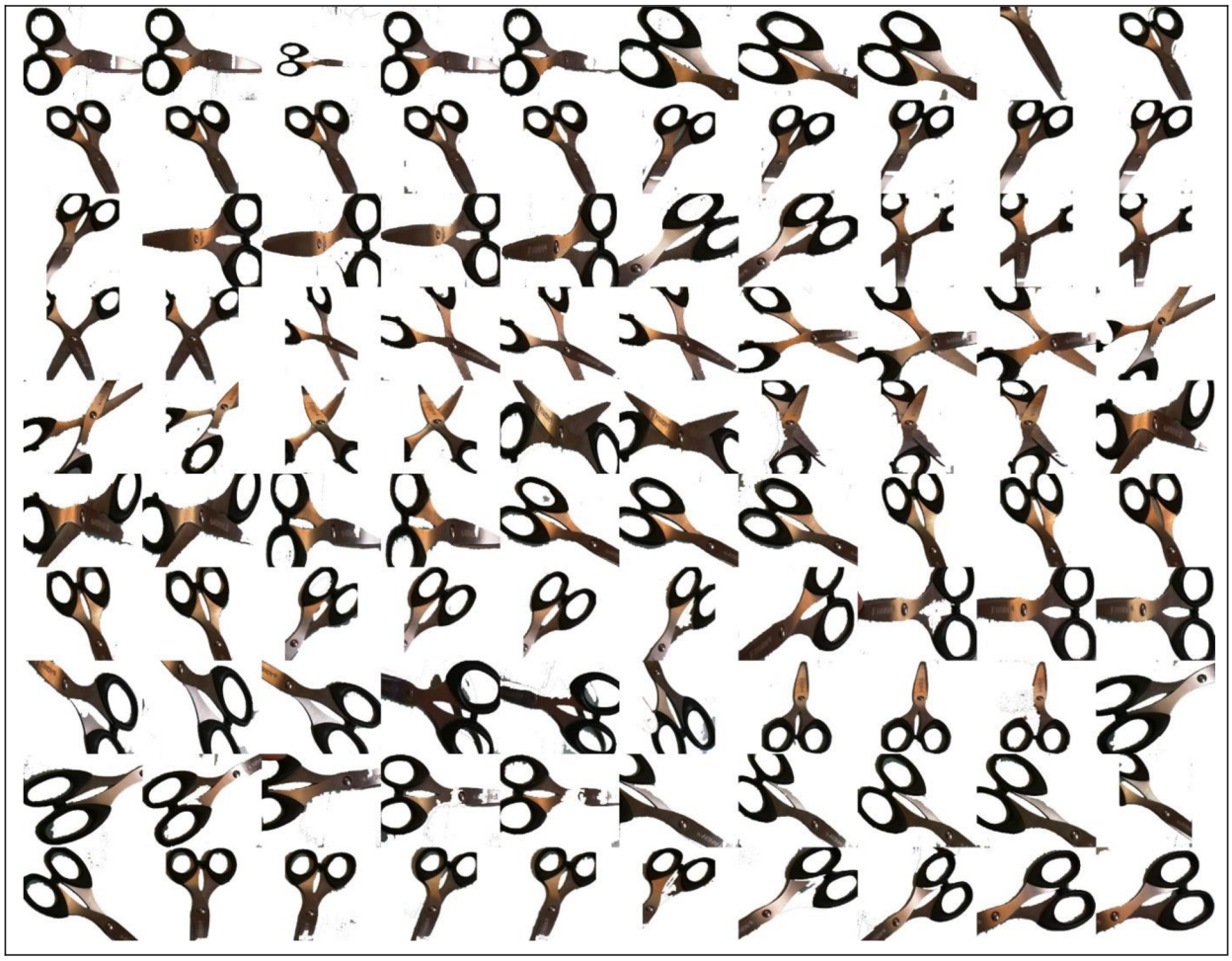
# PŘÍLOHA 2 - Tréninková data 2/2

Při pořizování snímků je vhodné aby snímky obsahovaly pouze objekt určený k detekci. V následujícím obrázku 32 je zobrazeno 100 pozitivních snímků užitých pro trénink haarovy kaskády rozpoznávající právě tento objekt.



**Obr. 32: Trénink objektu nůžky - základní**

Z důvodu nedostatečné přesnosti haarovy kaskády bylo nutno eliminovat možnost ovlinění tréninku pozadím objektu (barva a textura stolu), kdy barva pozadí mohla být považováná jako součást objektu. V rámci tréninku pak algoritmus ignoroval bílou barvu. Upravená data byla zobrazena v obr. 33.



**Obr. 33: Trénink objektu nůžky - upravené**

# PŘÍLOHA 3 - Skript: Rozpoznání objektu a jeho lokalizace

```
"""
Tento skript rozpoznává objekty, jež byly natrénovany předem na pozitivních
a negativních snímcích. Skript využívá inventářní systém, detekuje který objekt
chybí a na tuto skutečnost uživatele upozorňuje.

PSEUDOKÓD
0.HAAR trénink detekovaných objektů
1.Načti kameru
2.Rozpoznání objektů
3.Označení objektů na obraze
4.Učení polohy
5.Uživatelský feedback (chybí/nemůže najít, špatná detekce atp.)
"""
import cv2, time
import numpy as np
"""Vstupní parametry + načtení kaskád"""
klice = cv2.CascadeClassifier("haarcascades/klice_17.xml"); # objekt 1
nuzky = cv2.CascadeClassifier("haarcascades/nuzky.xml"); # objekt 2
cap = cv2.VideoCapture(0);
truth_1= 1; truth_2= 1; #
timer1 = time.time()+100000; timer2 = time.time()+100000 # časovače

"""Detekce objektu"""
def detect(frame, minSize, minNeighbors, groupRectangleMin, esq, cascade, truth,
timer):
#minSize = nejmenší možný obdélník
#minNeighbors = nejmenší možný počet sousedících obdélníků
#groupRectangleMin = nejmenší možný počet sjednocených obdélníků
#esq = 1-esq procentuální překrytí sousedících obdélníků pro jejich sjednocení
#cascade = vybraná haarova kaskáda pro detekci

    item = cascade.detectMultiScale(frame, minSize=minSize,
                                        minNeighbors=minNeighbors);
    item, weights_item = cv2.groupRectangles(np.array(item).tolist(),
                                        groupRectangleMin, esq)
"""Spojení nejvíce možných obdélníků, potlačení ostatních"""
try:
        weights_item = weights_item[0];
if weights_item[0]>=truth:
            ok = 1;
            truth = weights_item[0];
            marker = item[0];
            timer = time.time()
if time.time()>timer+3:#po třech sekundách reset min. počet obdélníků
                truth = 0;
return marker, truth, ok, timer
else:
return 0,0,0,timer
except:
        ok = 0;
        marker = 0;
        truth = 0;
return marker, truth, ok, timer

while (True):
    ret, frame = cap.read();
```

```
"""Detekce objektu 1"""
    marker_1, truth_1, ok_1, timer1 = detect(frame, (30,30), 1, 1, 0.99, klice,
                                             truth_1, timer1);
if ok_1 == 1:
        kx1,ky1,kw1,kh1 = marker_1;
        cv2.rectangle(frame, (kx1,ky1), (kx1+kw1,ky1+kh1), (0,0,0), 8);
                        #vykreslení obdélníku
        xt1=int(kx1+kw1/2) #x-ová souřadnice těžiště obrazce
        yt1=int(ky1+kh1/2) #y-ová souřadnice těžiště obrazce
        cv2.circle(frame, (xt1,yt1), 10, (0,0,255), -1); #vykreslení středu
        cv2.putText(frame, "KLICE", (kx1,ky1),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,250,250),2);
elif time.time()>timer1+2:
        cv2.putText(frame, "Detection 1 failed ", (100,20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,50,250),2); #selhání
"""Inventář objektu 1"""
try:
if xt1 < (np.shape(frame)[1]*0.33): #specifikace místa těžiště
            cv2.putText(frame, "Objekt 1 poloha OK", (100,110),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
else:
            cv2.putText(frame, "Objekt 1 poloha notOK", (100,110),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
except:
pass
"""Detekce objektu 2"""
    marker_2, truth_2, ok_2, timer2 = detect(frame, (50,50), 2, 1, 0.99, nuzky,
                                             truth_2, timer2);
if ok_2 == 1:
        kx2,ky2,kw2,kh2 = marker_2;
        cv2.rectangle(frame, (kx2,ky2), (kx2+kw2,ky2+kh2), (255,0,0), 8);
                        #vykreslení obdélníku
        xt2=int(kx2+kw2/2) #x-ová souřadnice těžiště obrazce
        yt2=int(ky2+kh2/2) #y-ová souřadnice těžiště obrazce
        cv2.circle(frame, (xt2,yt2), 10, (0,0,255), -1); #vykreslení středu
        cv2.putText(frame, "NUZKY", (kx2,ky2),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,250,250),2);
elif time.time()>timer2+0.5:
        cv2.putText(frame, "Detection 2 failed", (100,50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,50,250),2); #selhání
"""Inventář objektu 2"""
try:
if xt2 > (np.shape(frame)[1]*0.66): #specifikace místa těžiště
            cv2.putText(frame, "Objekt 2 poloha OK", (100,140),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
else:
            cv2.putText(frame, "Objekt 2 poloha notOK", (100,140),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
except:
pass

    cv2.imshow("img", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release();
cv2.destroyAllWindows();
```

# PŘÍLOHA 4 - Skript: Sledování objektu a detekce pohybu

```
"""
Tento script pozoruje změny, následně tyto změny zaznamená a sleduje je pohyb-
ující se po obraze. Využito sledovacích algoritmů KCF a MIL (dostupných více).

PSEUDOKÓD
1.Načti kameru + kalibrace
2.Snímek pozadí
3.Porovnej s aktuálním snímkem
4.Při první změně počkej daný čas, následně diferenci označ obdélníkem
5.Daný obdélník sleduj algoritmy KCF/MIL
6.Uživatelský feedback
7.Zaznamenej dráhu obdélníku v čase
8.Pracuj s touto hodnotou pro počítání objektů (počet, pozice,...)
"""
import cv2,time
import numpy as np

def target(cap):
while (True):
        ____, frame = cap.read();
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY);
        gray = cv2.GaussianBlur(gray, (21, 21), 0); # pro snížení noise
"""Vytvoření snímku pro porovnání s aktuálním feedem"""
try:
            firstFrame
except(NameError):
            firstFrame = gray;
      #vytvoření pozadí, které se nebude přepisovat při každém cyklu
"""Porovnání s prvním snímkem + uprava"""
        frameDelta = cv2.absdiff(firstFrame, gray);
      #diference mezi pozadím a aktuálním obrazem
        thresh = cv2.threshold(frameDelta, 55, 255, cv2.THRESH_BINARY)[1];
      #přes threshold zvýrazním změny obrazu a převedu je na "data" pro
      findContours
        thresh = cv2.dilate(thresh, None, iterations=2);
"""Hledání souvislých změn pro označení"""
        (__,cnts,___) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
                                         cv2.CHAIN_APPROX_SIMPLE);
for c in cnts:
      # pokud je contourArea moc malá nebo velká, nevytvoří se obdélník
if cv2.contourArea(c) < 15000:
continue
if cv2.contourArea(c) > 25000:
continue
            x,y,w,h = cv2.boundingRect(c)
             # kolem kontury se vytvoří nejmenší čtverec
            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

        cv2.imshow("tracking-prepare", frame);
if cv2.waitKey(2) & 0xFF == ord("q"):
# stisknutím klávesy "q" se hledání změny zastaví
break
"""Detekce změny + spuštění časovače"""
try:
            timerDiff
except(NameError):
```

```
            timer = time.time()+10000;
try:
if x!=0:
                    timer = time.time();
except:
continue
"""Přerušení po zadaném čase"""
if time.time()>timer+0.5:
break
return x,y,w,h
"""Výběr a inicializace trackeru"""
def createTracker(trackerType):
    tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW', 'GOTURN']
    tracker_type = tracker_types[trackerType]
if tracker_type == 'BOOSTING':
        tracker = cv2.TrackerBoosting_create()
if tracker_type == 'MIL':
        tracker = cv2.TrackerMIL_create()
if tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
if tracker_type == 'TLD':
        tracker = cv2.TrackerTLD_create()
if tracker_type == 'MEDIANFLOW':
        tracker = cv2.TrackerMedianFlow_create()
if tracker_type == 'GOTURN':
        tracker = cv2.TrackerGOTURN_create()
return tracker, tracker_type
"""Sledování objektu"""
def Track(tracker,frame):
    ok, bbox = tracker.update(frame)
    p1, p2 = False, False
if ok:
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)
else:
        cv2.putText(frame, "Trackovani neuspesne", (100,80),
cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
return frame, ok, p1, p2
"""Vstupní parametrya proměnné"""
pocet = 0;chyba = 0;
cap = cv2.VideoCapture(0); # video vstup
____, frame = cap.read();
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc(*'XVID'),
                          30.0, (640,480))# nahrávání skriptu
"""Volání funkcí"""
x,y,w,h = target(cap);
bbox = (x,y,w,h);
tracker, tracker_type = createTracker(2);
tracker.init(frame, bbox);
"""Kontinuální sledování"""
while (True):
    ____, frame = cap.read();
    timer = cv2.getTickCount(); # funkce pro výpočet FPS
    frame, ok, p1, p2 = Track(tracker, frame);
if ok == False:
        cv2.putText(frame, "Trackovani neuspesne", (100,80),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
        cv2.imshow("output", frame);
```

```
"""Detekce chyby ve střední části obrazu"""
try:
if xt > (np.shape(frame)[1]*0.25):
if xt < (np.shape(frame)[1]*0.75):
                    chyba += 1
except:
pass
        first = time.time();cap.release();
        time.sleep(1); # pozastavení skriptucap = cv2.VideoCapture(1);
"""Reinicializace sledování"""
        x,y,w,h = target(cap);bbox = (x,y,w,h);
        tracker, tracker_type = createTracker(2);
        ____, frame = cap.read();
        tracker.init(frame, bbox);
continue
    xt = int((p1[0]+p2[0])/2) # x-ová souřadnice těžiště sledovaného objektu
    yt = int((p1[1]+p2[1])/2) # y-ová souřadnice těžiště sledovaného objektu
fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer); #výpočet FPS
"""Zapamatování si počáteční pozice objektu"""
try:
        pozice1
except(NameError):
        pozice1 = xt
"""Rozhodnutí mezi pohybem z jedné či druhé strany"""
if pozice1 < (np.shape(frame)[1]/2):
        cv2.putText(frame, "Prichod", (100,110), cv2.FONT_HERSHEY_SIMPLEX,
                    0.75, (50,170,50), 2);
if xt > (np.shape(frame)[1]*0.75):
            pocet += 1;
del pozice1
elif xt < (np.shape(frame)[1]*0.25):
del pozice1
else:
        cv2.putText(frame, "Odchod", (100,110), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                    (50,170,50), 2);
if xt < (np.shape(frame)[1]*0.25):
            pocet -= 1;
del pozice1
elif xt > (np.shape(frame)[1]*0.75):
del pozice1
    cv2.putText(frame, tracker_type + " Tracker", (100,20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
                    # Zobrazení typu trackeru
    cv2.putText(frame, "FPS : " + str(int(fps)), (100,50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
                    # Zobrazení FPS
    cv2.putText(frame, "Chyby :" + str(chyba), (100,80),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
                    # Zobrazení neúspěšnosti
    cv2.putText(frame, str(pocet), (int((np.shape(frame)[1]/2)),50),
                    cv2.FONT_HERSHEY_SIMPLEX, 2, (50,170,50),2);
                    # Zobrazení počtu objektů
    out.write(frame)            # výstup trackingu do .avi
    cv2.imshow("output", frame) # výstupní obraz
if cv2.waitKey(2) & 0xFF == ord("q"):
# konec programu po stisknutí klávesy "q"
        cap.release()
break
```

# PŘÍLOHA 5 - Skript: PONG, Sledování objektu

```
"""
Tento script pozoruje změny, následně tyto změny zaznamená a sleduje je pohyb-
ující se po obraze a používá je k realizaci hy PONG. Užito trackování KCF.

PSEUDOKÓD
1.Načti kameru
2.Snímek pozadí
3.Porovnej s aktuálním snímkem
4.Při první změně počkej daný čas, následně diferenci označ obdélníkem
5.Daný obdélník sleduj algoritmy KCF/MIL
6.Uživatelský feedback
7.Zaznamenej dráhu obdélníku v čase
8.Realizuj sledování, počítání, skóre
"""
import cv2,time
import numpy as np

def target(cap,lower,upper):
while (True):
        ____, frame = cap.read();
"""Vytvoření barevné masky"""
        mask = cv2.inRange(frame, lower, upper);
        mask = cv2.erode(mask,None, iterations = 2)
        mask = cv2.dilate(mask, None, iterations=5);
      try:
            firstMask
            mask = cv2.bitwise_and(mask,firstMask)
      except(NameError):
            firstMask = cv2.bitwise_not(mask)
"""Převod do černobíla"""
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY);
        gray = cv2.GaussianBlur(gray, (21, 21), 0); # pro snížení noise
"""Vytvoření snímku pro porovnání s aktuálním feedem"""
      try:
            firstFrame
      except(NameError):
            firstFrame = gray;
#vytvoření pozadí, které se nebude přepisovat při každém cyklu
"""Porovnání s prvním snímkem + uprava"""
        frameDelta = cv2.absdiff(firstFrame, gray);
#diference mezi pozadím a aktuálním obrazem
        thresh = cv2.threshold(frameDelta, 40, 255, cv2.THRESH_BINARY)[1];
#přes threshold zvýrazním změny obrazu a převedu je na "data" pro
      findContours
        thresh = cv2.dilate(thresh, None, iterations=3);
"""Maska + pohyb"""
        search = cv2.bitwise_and(mask,firstFrame) # binární sjednocení
"""Hledání souvislých změn pro označení"""
        (__,cnts,___) = cv2.findContours(search.copy(), cv2.RETR_EXTERNAL,
                                          cv2.CHAIN_APPROX_SIMPLE);
      for c in cnts:
      # pokud je contourArea moc malá, nevytvoří se obdélník
      if cv2.contourArea(c) < 1500:
            continue
      if cv2.contourArea(c) > 25000:
            continue
```

```
            x,y,w,h = cv2.boundingRect(c) # kolem kontury nejmenší čtverec
            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
        cv2.imshow("tracking-prepare", frame);
      if cv2.waitKey(2) & 0xFF == ord("q"):
            # stisknutím klávesy "q" se hledání změny zastaví
            break
"""Detekce změny + spuštění časovače"""
      try:
            timerDiff
      except(NameError):
            timer = time.time()+10000;
      try:
      if x!=0:
                  timerDiff = time.time();
                  timer = timerDiff;
      except:
            continue
"""Přerušení po zadaném čase"""
      if time.time()>timer+0.5:
            break
      return x,y,w,h
"""Výběr a inicializace trackeru"""
def createTracker(trackerType):
    tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW', 'GOTURN']
    tracker_type = tracker_types[trackerType]
      if tracker_type == 'BOOSTING':
        tracker = cv2.TrackerBoosting_create()
      if tracker_type == 'MIL':
        tracker = cv2.TrackerMIL_create()
      if tracker_type == 'KCF':
        tracker = cv2.TrackerKCF_create()
      if tracker_type == 'TLD':
        tracker = cv2.TrackerTLD_create()
      if tracker_type == 'MEDIANFLOW':
        tracker = cv2.TrackerMedianFlow_create()
      if tracker_type == 'GOTURN':
        tracker = cv2.TrackerGOTURN_create()
return tracker, tracker_type
"""Sledování objektu"""
def Track(tracker,frame):
    ok, bbox = tracker.update(frame)
    p1, p2 = False, False
      if ok:
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)
      else:
        cv2.putText(frame, "Trackovani neuspesne", (100,80),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
      return frame, ok, p1, p2
"""Vstupní parametry a proměnné"""
lower = np.array((0,0,0), dtype="uint8");
upper = np.array((50,50,50), dtype="uint8");counter_left = 0;counter_right = 0;
chyba = 0; state = 0; cap = cv2.VideoCapture(0);
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc(*'XVID'), 30.0,
                          (640,480))
"""Volání funkcí"""
x,y,w,h = target(cap,lower,upper);
bbox = (x,y,w,h);
```

```
tracker, tracker_type = createTracker(2);
____, frame = cap.read();tracker.init(frame, bbox);
"""Sledování objektu"""
while (True):
    ____, frame = cap.read();
    timer = cv2.getTickCount();
    frame, ok, p1, p2 = Track(tracker, frame);
      if ok == False:
        cv2.putText(frame, "Trackovani neuspesne", (100,80),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
        cv2.imshow("output", frame);
      try:
            if xt > (np.shape(frame)[1]*0.15):
            if xt < (np.shape(frame)[1]*0.85):
                  chyba += 1
      except:
            pass
        first = time.time(); cap.release(); time.sleep(1); # pozastavení skriptu
        cap = cv2.VideoCapture(0);
        x,y,w,h = target(cap,lower,upper);
        bbox = (x,y,w,h);
        tracker, tracker_type = createTracker(2);
        ____, frame = cap.read();
        tracker.init(frame, bbox);
            continue
    xt = int((p1[0]+p2[0])/2) # x-ová souřadnice těžiště sledovaného objektu
    yt = int((p1[1]+p2[1])/2) # y-ová souřadnice těžiště sledovaného objektu
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer); #výpočet FPS
    cv2.line(frame,(int(np.shape(frame)[1]*0.85),0),(int(np.shape(frame)[1]*0.85),
      int(np.shape(frame)[0])), (255,255,255), 10);
    cv2.line(frame,(int(np.shape(frame)[1]*0.15),0),(int(np.shape(frame)[1]*0.15),
      int(np.shape(frame)[0])), (255,255,255), 10); #vykreslení čar započtení bodů
      if xt > np.shape(frame)[1]*0.85:
      if state == 0:
            counter_left +=1; state = 1;
      elif xt < np.shape(frame)[1]*0.15:
      if state == 0:
            counter_right +=1; state = 1;
      elif xt < np.shape(frame)[1]*0.75:
      if xt > np.shape(frame)[1]*0.25:
            state = 0;
    cv2.putText(frame, tracker_type + " Tracker", (100,20),
                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
      # Zobrazení typu trackeru
    cv2.putText(frame, "FPS : " + str(int(fps)), (100,50),
                cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);
      # Zobrazení FPS
    cv2.putText(frame, "Chyby :" + str(chyba), (100,80),
                cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2);
      # Zobrazení neúspěšnosti
    cv2.putText(frame, str(counter_left) + " : " + str(counter_right) ,
(int((np.shape(frame)[1]/2)),50), cv2.FONT_HERSHEY_SIMPLEX, 2, (50,170,50),2);
      # Zobrazení skóre
    out.write(frame)        #vystup trackingu do .avi
    cv2.imshow("output", frame)# vysledny obrazový výstup
      if cv2.waitKey(2) & 0xFF == ord("q"):
      # konec programu po stisknutí klávesy "q"
        cap.release()
      break
```

# PŘÍLOHA 6 - Script: Stanovení bezpečné vzdálenosti

```
"""
Tento skript po zkalibrování páru kamer nabídne k nahlédnutí disparity mapu,
pomocí které se za pomocí kalibračního obrazce nakonfiguruje nebezpečná a
kritická vzdálenost. Tyto stavy jsou následně zobrazeny pro uživatelský feedback.

PSEUDOKÓD
1.Načti 2 kamery + stereokalibrace
2.Výpočet a kalibrace disparity mapy
3.Kalibrace bezpečné vzdálenosti pomocí specifického objektu
4.Uživatelský feedback (stav vzdálenosti)
5.Output (Objekt v bezpečné/nebezpečné oblasti, FPS atd.)
5.Zaznamenání času prostoje
"""
import numpy as np
import cv2, time

"""Kalibrace kamer"""
def Calibrate2Cameras(camera1,camera2):
                    #vrátí feed, matici, přetvoření a další věci, celkem 7

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
    objp = np.zeros((6*9,3), np.float32)
    objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
    objp2 = np.zeros((6*9,3), np.float32)
    objp2[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
    objpoints = []; imgpoints = []; objpoints2 = []; imgpoints2 = [];

    #Nastavení kamery
    CAMERA_WIDTH = 720
    CAMERA_HEIGHT = 640
    CAMERA_BRIGHTNESS = 5

    cap = cv2.VideoCapture(camera1); cap2 = cv2.VideoCapture(camera2);
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH);
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT);
    cap2.set(cv2.CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH);
    cap2.set(cv2.CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT);
    cap.set(cv2.CAP_PROP_BRIGHTNESS, CAMERA_BRIGHTNESS);
    cap2.set(cv2.CAP_PROP_BRIGHTNESS, CAMERA_BRIGHTNESS);
    i = 0

    while i<=30:
        __, frame = cap.read(); __, frame2 = cap2.read();

        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY);
        gray2 = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY);
        ret, corners = cv2.findChessboardCorners(gray, (9,6),None);
        ret2, corners2 = cv2.findChessboardCorners(gray2, (9,6),None);
        #kontrola funkce obou kamer
        both = ret + ret2;

        if both == 2:
            objpoints.append(objp);
            objpoints2.append(objp2);
            corners_draw = cv2.cornerSubPix(gray,corners,(11,11),
                                            (-1,-1),criteria);
```

```
            corners2_draw2 = cv2.cornerSubPix(gray2,corners2,(11,11),
                                              (-1,-1),criteria);
            imgpoints.append(corners_draw); imgpoints2.append(corners2_draw2);
            frame = cv2.drawChessboardCorners(frame, (9,6), corners_draw,ret);
            frame2 = cv2.drawChessboardCorners(frame2, (9,6),corners2_draw2,ret2);
            i += 1; time.sleep(0.5);
        cv2.imshow("cap",frame)
        cv2.imshow("cap2",frame2)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

#Volání kalibrace kamer
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
                    objpoints, imgpoints, gray.shape[::-1],None,None);
    ret2,mtx2,dist2,rvecs2,tvecs2 = cv2.calibrateCamera(
                    objpoints2,imgpoints2,gray2.shape[::-1],None,None);
#Stereokalibrace
    (_,_,_,_,_,rotationMatrix, translationVector,_,_) = cv2.stereoCalibrate(
objpoints, imgpoints,imgpoints2,mtx,dist,mtx2,dist2,gray.shape[::-1],None,None,
cv2.CALIB_FIX_INTRINSIC, criteria)
#Stereorektifikace
(leftRectification, rightRectification, leftProjection, rightProjection,
dispartityToDepthMap, leftROI, rightROI) = cv2.stereoRectify(mtx,dist,mtx2, dist2,
gray.shape[::-1],rotationMatrix,translationVector,cv2.CALIB_ZERO_DISPARITY,0,(0,0))
    leftMapX, leftMapY = cv2.initUndistortRectifyMap(mtx, dist,
                    leftRectification, leftProjection, imgsize, cv2.CV_32FC1)
    rightMapX, rightMapY = cv2.initUndistortRectifyMap(mtx2, dist2,
                    rightRectification, rightProjection, imgsize, cv2.CV_32FC1)
    return imgsize, leftMapX, leftMapY, leftROI, rightMapX,
           rightMapY, rightROI, frame, frame2

"""Funkce výpočtu nové hloubkové mapy"""
def new(imgL,imgR):
    #výpočet depthmap
    disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0;

    cv2.imshow('left', imgL);
    print ('computing disparity...');

    disparity = ((disp-min_disp)/num_disp);
    erosion = cv2.erode(disparity,kernel_erode,iterations = 1);
    dilatation = cv2.dilate(erosion,kernel_dilate,iterations = 1);
    blur = cv2.GaussianBlur(dilatation,(3,3),0);
    cv2.imshow("disparity",disparity); cv2.imshow('tweaked', blur);

    return dilatation
    time.sleep(0.05)

"""Funkce dodávající již nezkreslený video-sekvenci"""
def feed(cap1,cap2):
    ret1, frame1 = cap1.read();
    ret2, frame2 = cap2.read();
    #Transformace kamer skrz 3D kalibraci
    frame1 = cv2.remap(frame1, leftMapX, leftMapY, cv2.INTER_LINEAR);
    frame2 = cv2.remap(frame2, rightMapX, rightMapY, cv2.INTER_LINEAR );
    return frame1,frame2

"""Funkce dodávající změny parametrů vyvolané uživatelem na posuvnících"""
def update(val = 0):
```

```
    stereo.setBlockSize(cv2.getTrackbarPos('window_size', 'disparity'))
    stereo.setUniquenessRatio(cv2.getTrackbarPos('uniquenessRatio', 'disparity'))
    stereo.setSpeckleWindowSize(cv2.getTrackbarPos('speckleWindowSize',
                                                   'disparity'))
    stereo.setSpeckleRange(cv2.getTrackbarPos('speckleRange', 'disparity'))
    stereo.setDisp12MaxDiff(cv2.getTrackbarPos('disp12MaxDiff', 'disparity'))
    stereo.setP1(cv2.getTrackbarPos('P1', 'disparity'))
    stereo.setP2(cv2.getTrackbarPos('P2', 'disparity'))

"""Porovnání s prvním snímkem"""
def diff(firstFrame,dispmap,roi):

    frameDelta = cv2.absdiff(firstFrame, dispmap);
    frameDelta = np.uint8(frameDelta*255)
    thresh = cv2.threshold(frameDelta, 50, 245, cv2.THRESH_BINARY)[1];
    frameDeltaROI = frameDelta[int(roi[1]):int(roi[1]+roi[3]),
                               int(roi[0]):int(roi[0]+roi[2])]
    dispmapROI = np.uint8(dispmap*255)
    dispmapROI = dispmapROI[int(roi[1]):int(roi[1]+roi[3]),
                               int(roi[0]):int(roi[0]+roi[2])]
    return frameDeltaROI, dispmapROI

"""Kalibrace bezpečné vzdálenosti"""
def calibrate2safedistance(frameDeltaROI,dispmapROI):
    print(cv2.mean(frameDeltaROI))
    print(cv2.mean(dispmapROI))
    if cv2.waitKey(200) & 0xFF == ord("s"):
        savedMeanDelta = cv2.mean(frameDeltaROI)
        savedMeanDispmap = cv2.mean(dispmapROI)
        return savedMeanDelta, savedMeanDispmap
    else:
        return 0,0

if __name__ == "__main__":
    #Nastavení depthmap
    window_size = 5
    min_disp = -64
    num_disp = 192
    uniquenessRatio = 1
    speckleRange = 2
    speckleWindowSize = 75
    disp12MaxDiff = 50
    P1 = 600
    P2 = 2400

    #Vytvoření měnitelných sliderů
    cv2.namedWindow('disparity')
    cv2.createTrackbar('speckleRange', 'disparity', speckleRange, 50, update)
    cv2.createTrackbar('window_size', 'disparity', window_size, 21, update)
    cv2.createTrackbar('speckleWindowSize', 'disparity', speckleWindowSize,
                                                    200, update)
    cv2.createTrackbar('uniquenessRatio', 'disparity', uniquenessRatio, 50,
                                                           update)
    cv2.createTrackbar('disp12MaxDiff', 'disparity', disp12MaxDiff, 250, update)
    cv2.createTrackbar('P1', 'disparity', P1, 600, update)
    cv2.createTrackbar('P2', 'disparity', P2, 2400, update)
```

```
    stereo = cv2.StereoSGBM_create(
        minDisparity = min_disp,        #minimální možná disparita(nesourodost)
        numDisparities = num_disp,      #maximální možná disparita(nesourodost)
        blockSize = window_size,        #velikost oblasti ke spárování
        uniquenessRatio = uniquenessRatio, #procentuální rozpětí pro potvrzení
                                            správnosti párování
        speckleRange = speckleRange,    #maximální variace disparity v každém
                                         spojitém prvku
        speckleWindowSize = speckleWindowSize, #maximální velikost spojité
                                         oblasti, kdy ještě šum v této oblasti
                                         bude anulován
        disp12MaxDiff = disp12MaxDiff,  #maximální diference mezi pixely obou
                                         snímků
        P1 = P1,                        #trest změny sousedících pixelů o 1
                                         hodnotu
        P2 = P2                         #trest změny sousedících pixelů o více než
                                         1 hodnotu

        )

#Volání funkcí
(imgsize, leftMapX, leftMapY, leftROI, rightMapX,
rightMapY, rightROI, frame, frame2 )= Calibrate2Cameras(0,2);
cap1 = cv2.VideoCapture(0); cap2 = cv2.VideoCapture(2);
kernel_erode = np.ones((7,7),np.uint8)
kernel_dilate = np.ones((5,5),np.uint8)
savedMeanDelta_danger = 0
savedMeanDelta_stop = 0

while True:
    timer = cv2.getTickCount();
    img = np.zeros((512,512,3), np.uint8)
    imgL,imgR = feed(cap1,cap2);
    dispmap = new(imgL, imgR);
    try:
        firstFrame
    except(NameError):
        firstFrame = dispmap;
    try:
        roi
    except(NameError):
        roi = cv2.selectROI(dispmap)

    frameDeltaROI, dispmapROI = diff(firstFrame, dispmap,roi);

#Dokud ještě není nastavena bezpečná vzdálenost, program volá funkci pro její
nastavení
    while savedMeanDelta_danger == 0:
        imgL,imgR = feed(cap1,cap2);
        dispmap = new(imgL, imgR);
        frameDeltaROI, dispmapROI = diff(firstFrame, dispmap,roi);
        cv2.imshow("dispmap-calibrate", dispmapROI)
        cv2.putText(dispmapROI, "Kalibrace DANGER", (100,20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
        savedMeanDelta_danger, savedMeanDispmap_danger = calibrate2safedistance(
                                                        frameDeltaROI, dispmapROI);
        savedMeanDelta_danger = savedMeanDelta_danger;

    while savedMeanDelta_stop == 0:
        imgL,imgR = feed(cap1,cap2);
```

```
            dispmap = new(imgL, imgR);
            frameDeltaROI, dispmapROI = diff(firstFrame, dispmap,roi);
            cv2.imshow("dispmap-calibrate", dispmapROI)
            cv2.putText(dispmapROI, "Kalibrace STOP", (100,20),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);
            savedMeanDelta_stop, savedMeanDispmap_danger = calibrate2safedistance(
                                                            frameDeltaROI, dispmapROI);
            savedMeanDelta_stop = savedMeanDelta_stop;

         #Porovnání aktuální střední hodnoty s kalibrovanými
        if savedMeanDelta_danger <  cv2.mean(frameDeltaROI):
            cv2.putText(img, "DANGER oblast", (100,80),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,255),2);
            if savedMeanDelta_stop < cv2.mean(frameDeltaROI):
                cv2.putText(img, "STOP oblast", (100,100),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,255),2);
         #Výpočet FPS
        fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);
        cv2.putText(dispmapROI, "FPS : " + str(int(fps)), (100,50),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);

        #Výstup
        cv2.imshow("delta", frameDeltaROI)
        cv2.imshow("dispmap-small", dispmapROI);
        cv2.imshow("info", img)
        if cv2.waitKey(2) & 0xFF == ord("q"):
            break

#Vypnutí kamer, zavření oken
cv2.destroyAllWindows();
     cap1.release(); cap2.release()
```