Pokročilé algoritmy zpracování obrazu v reálném systému řízení pohybu

Martin Šůstek

Bakalářská práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin Šůstek**
Osobní číslo:	**A18618**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Pokročilé algoritmy zpracování obrazu v reálném systému řízení pohybu**
Téma práce anglicky:	**Advanced Image Processing Algorithms in a Real Motion Control System**

Zásady pro vypracování

1. Vypracujte teoretický základ k metodám zpracování obrazu pro účely strojového vidění.
2. Analyzujte hotovou aplikaci pro reálný model „Kulička na nakloněné rovině“ z pohledu možností implementace doplňkového algoritmu zpracování obrazu.
3. Vyberte alespoň jednu z vhodných metod pokročilé obrazové identifikace, s ohledem na softwarové/hardwarové možnosti modelu.
4. Popište a odůvodněte Vámi vybranou metodu(y).
5. Implementujte zvolený algoritmus(y) jako rozšiřující možnost sledování cíle do stávající aplikace, se zachováním původní techniky obrazového zpracování.
6. Otestujte běh aplikace s Vámi implementovanou metodou, porovnejte ji s původním algoritmem a kriticky zhodnoťte výsledky práce.

Forma zpracování bakalářské práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. BURGER, Wilhelm a Mark James BURGE. Principles of digital image processing: advanced methods. London: Springer, 2013. Undergraduate topics in computer science (Springer). ISBN 978-1848829183.
2. ZHI ENG, L. Qt5 C++ GUI Programming Cookbook, 2nd ed.; Packt Publishing: Birmingham, England, 2019; pp. 428.
3. MALLICK, Satya. Object Tracking using OpenCV (C++/Python) [online]. 13 February 2017, s. 12 [cit. 2020-11-24]. Dostupné z: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/
4. HALFACREE, G. The Official Raspberry Pi Beginner's Guide, 1st ed.; Raspberry Pi Press: Cambridge, UK, 2018; pp. 241.
5. ZÁTOPEK, J. Moderní řízení pohybových stavů mechanické soustavy průmyslového robota prostřednictvím elektromechanických akčních členů. [cit. 2020-11-24]. FAI UTB ve Zlíně, 2018; pp. 57.

Vedoucí bakalářské práce: **Ing. Jiří Zátopek**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **15. ledna 2021**
Termín odevzdání bakalářské práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 6.5.2021

Martin Šůstek v.r.

.....

podpis studenta

ABSTRAKT

Cílem bakalářské práce je implementace pokročilého algoritmu strojového vidění pro sledování objektu v obraze, do řídícího systému, který nedisponuje vysokým výpočetním výkonem. Práce je rozdělena do dvou částí – teoretické a praktické. V teoretické části se zabývá pojmy spojenými s extrakcí příznaků z obrazu, metodami sledování objektu a metodami rozpoznání objektů v obraze. V praktické části se věnuje implementaci algoritmu do reálného modelu řízení „Kulička na nakloněné rovině“ a srovnání výsledků s již implementovanou metodou.

Klíčová slova: zpracování obrazu, strojové vidění, sledování, objekt, OpenCV, Raspberry Pi

ABSTRACT

The aim of this bachelor's thesis an implementation of an advanced computer vision algorithm for object tracking into a motion control system, which does not have high computing power. The thesis is separated into two parts – theoretical and practical. In the theoretical part, it talks about feature extraction, methods of object tracking and methods of object detection in an image. In the practical part it talks about the implementation of an algorithm into the real motion control system “Ball on an inclined plane” and then compares the result with an already implemented method.

Keywords: image processing, computer vision, tracking, object, OpenCV, Raspberry Pi

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Po dlouhou dobu byly počítače k práci s digitálním obrazem využívány pouze malou skupinou specialistů, kteří měli přístup k drahému vybavení. Kombinace specialisty a výkonného vybavení se však obvykle nacházela pouze ve výzkumných ústavech a oblast strojového vidění tam má své základy v akademických oblastech. Není to tak dávno, kdy počítače neměly ani dostatek paměti na to, aby nahrály jednu fotografii z typické moderní kamery. V současné době nám však kombinace výkonného počítače na každém stole a možnosti pořizovat snímky téměř každým zařízením, které najdeme v kapse, umožnila rozšíření této disciplíny jak do běžného života, tak do průmyslových oblastí.

Bakalářská práce je zaměřena na sledování objektu v obraze z kamery, která je součástí reálného modelu „Kulička na nakloněné rovině“. V modelu jsou informace získané z kamery využity pro řízení nakloněné roviny a ve výsledku také samotné kuličky. Tento model byl vytvořen vedoucím této bakalářské práce Ing. Jiřím Zátopkem. Hlavními vlastnostmi zajímavými pro tuto práci, jsou využití počítače Raspberry Pi 3B+, programovacího jazyka C++ a open source knihoven OpenCV.

Cílem této práce je přidání některého z pokročilých algoritmů strojového vidění, do již hotového softwarového i hardwarového řešení, se současným ponecháním původního algoritmu strojového vidění. Nejprve bude potřeba nastudovat problematiku strojového zpracování obrazu, provést literární rešerši a následně se rozhodnout, kterou metodu zvolit pro praktické řešení. V praktické části je třeba tuto metodu implementovat a následně její výsledky porovnat s původním algoritmem vytvořeným panem Ing. Zátopkem.

I. TEORETICKÁ ČÁST

1 EXTRAKCE PŘÍZNAKŮ OBRAZU

Hlavním důvodem k extrahování příznaků z obrazu a nevyužívání pouze čistých dat do algoritmu učení, je snížení rozdílů mezi objekty patřícími do stejné třídy, v porovnání s čistými daty a v následku tak zjednodušení klasifikace objektu. [1]

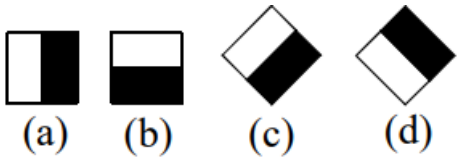
1.1 Haar features

Haarovy příznaky jsou založeny na principu podobnému Haarově vlnce. Hodnota tohoto příznaku se počítá jako suma pixelů odpovídající světlé části, od kterých je odečtena suma pixelů tmavé části. [3] Tato hodnota je dána rovnicí:

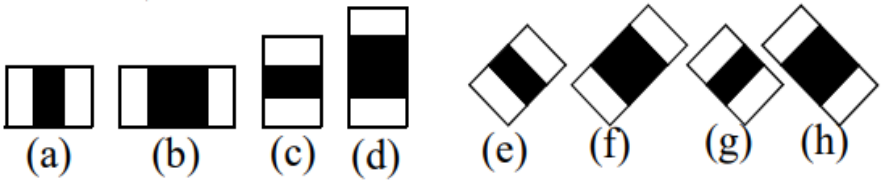
$$f(x) = \omega_0 r_0 + \omega_1 r_1 \quad (1)$$

Kde $f(x)$ je hodnota příznaku ve vstupním snímku x, ω_0 je váha bílé části obdélníku r_0, ω_1 je váha černé obdélníkové části r_1. Váhy mají opačná znaménka, $\omega_0 = 1$ a ω_1 je záporný podíl ploch r_0 a r_1.

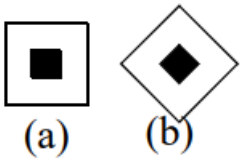
Tyto příznaky se dělí podle toho, jakou informaci mají detekovat na: hranové příznaky (Obrázek 1), čárové příznaky (Obrázek 2) a na příznaky středové (Obrázek 3).



Obrázek 1 Hranové příznaky



Obrázek 2 Čárové příznaky



Obrázek 3 Středové příznaky

Příznaky jsou většinou generovány pro velikost okna 24x24 pixelů. Generování probíhá s různě velkými obdélníky. Obdélníky mění velikost tak, aby postupně pokryly všechny kombinace velikostí. Vždy po změně velikosti dojde k posunu obdélníku celým obrazem.

Tímto postupem se vygeneruje velké množství příznaků pro jeden obraz, jak je vidět v tabulce 1. [1]

Tabulka 1 Počet Haarových příznaků pro obrázek velikosti 24x24 pixelů

Typ příznaku	**Počet příznaků**
Hranový (a), (b)	43 200
Hranový (c), (d)	8 464
Čárový (a), (c)	27 600
Čárový (b), (d)	20 736
Čárový (e), (g)	4 356
Čárový (f), (h)	3 600
Středový (a)	8 464
Středový (b)	1 521
Celkem	117 941

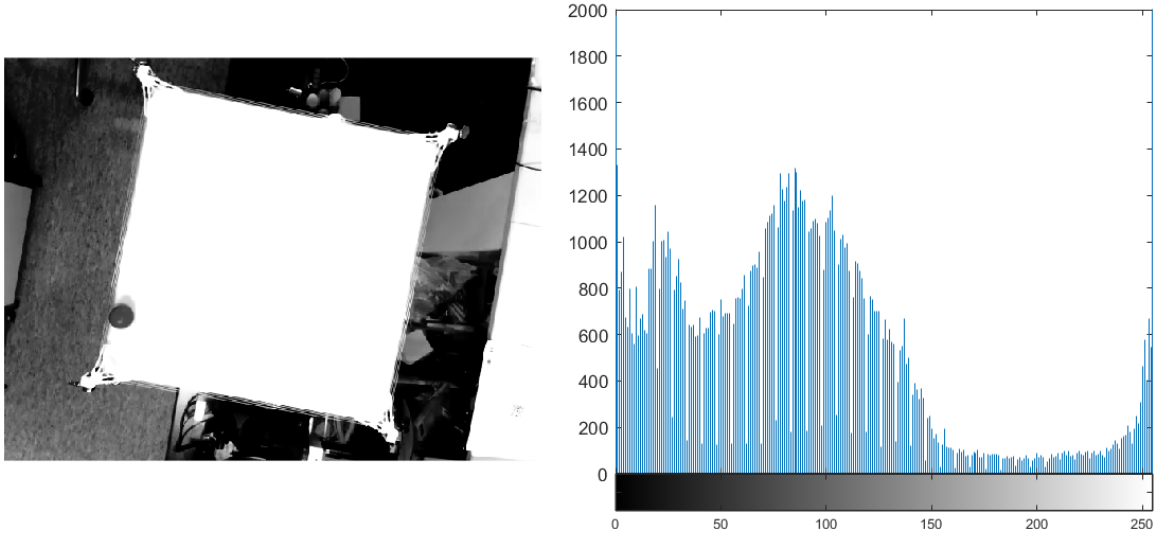
1.2 Histogram of oriented gradients

1.2.1 Histogram v černobílém obraze

Obecně histogramy vyjadřují frekvenční distribuce, histogramy obrazu popisují frekvenci hodnot intenzity nacházejících se v obraze. Histogram h pro černobílý obraz I s hodnotami intenzity v rozsahu $I(u, v) \in [0, K-1]$ obsahuje právě K záznamů, kde pro typický 8-bitový, černobílý obraz, $K = 2^8 = 256$. Každý záznam je definován jako

$$h(i) = počet\ pixelů\ v\ I\ s\ hodnotou\ intenzity\ i$$

pro všechna $0 \leq i \leq K$.[2]



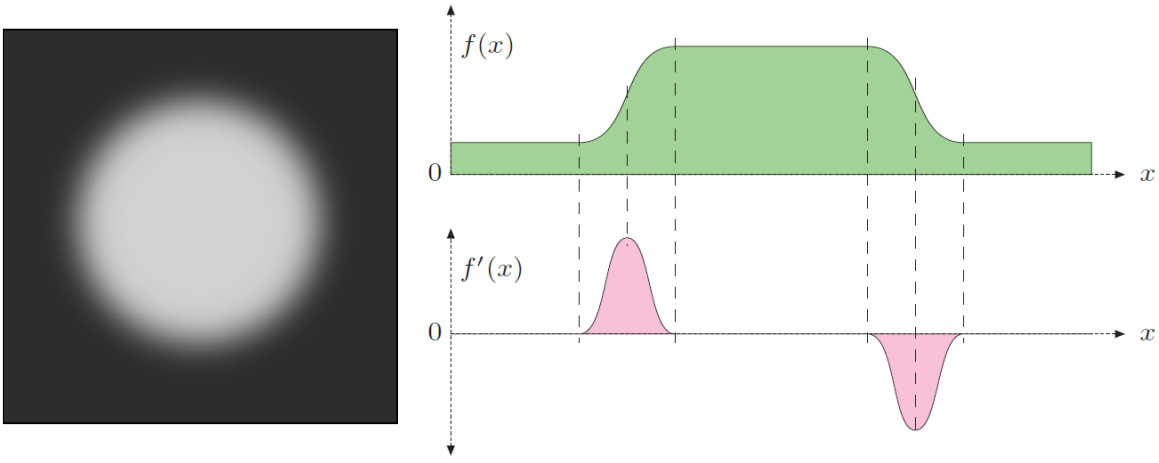
Obrázek 4 Typický obraz v odstínech šedi a jeho histogram

1.2.2 Gradienty v černobílém obraze

Pro zjednodušení uvažme situaci pouze v jednom rozměru. V tomto případě je profil intenzit na jedné lince obrazu jednorozměrnou funkcí $f(x)$. Vypočtením první derivace této funkce f:

$$f'(x) = \frac{df}{dx}(x) \tag{2}$$

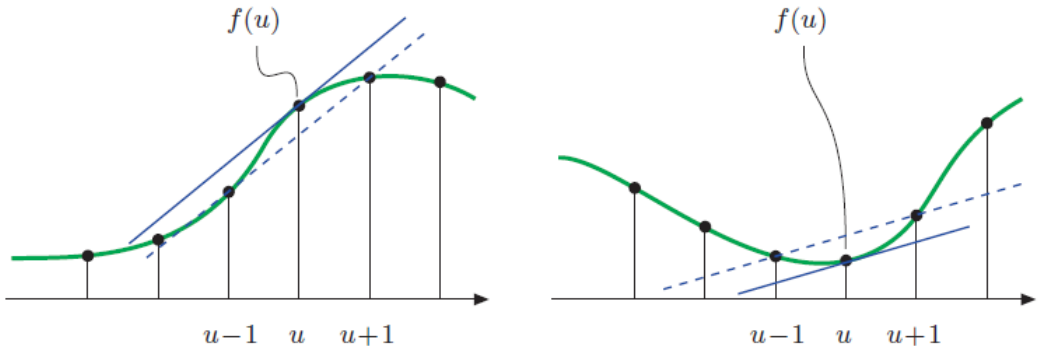
dostaneme pozitivní hodnotu v místech, kde se intenzita obrazu zvyšuje a negativní hodnotu v místech, kde se intenzita snižuje.



Obrázek 5 Funkce intenzity obrazu a její derivace [2]

Obraz je v počítači však reprezentován diskrétně a derivace pro tuto nespojitou funkci $f(u)$ není definována. Je tedy třeba využít metodu aproximace první derivace funkce. První derivace spojité funkce v bodě x reprezentuje směrnici tangenty v tomto bodě. Jednoduchou metodou, jak aproximovat směrnici tangenty pro diskrétní funkci $f(u)$ v bodě u, je proložení přímky sousedními body $f(u-1)$ a $f(u+1)$. [2]

$$\frac{df}{du} \approx \frac{f(u+1)-f(u-1)}{(u+1)-(u-1)} = \frac{f(u+1)-f(u-1)}{2} \tag{3}$$



Obrázek 6 Aproximace směrnice tangenty [2]

Stejná metodika může být aplikována také pro směr osy y, tedy pro jednotlivé sloupce obrazu.

Derivace vícerozměrné funkce podle jedné z os souřadnic se nazývá parciální derivace. Parciální derivace funkce obrazu $I(u,v)$ podle os u a v zapíšeme jako

$$\frac{\partial I}{\partial u}(u,v) \quad a \quad \frac{\partial I}{\partial v}(u,v) \tag{4}$$

Funkce

$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix} \tag{5}$$

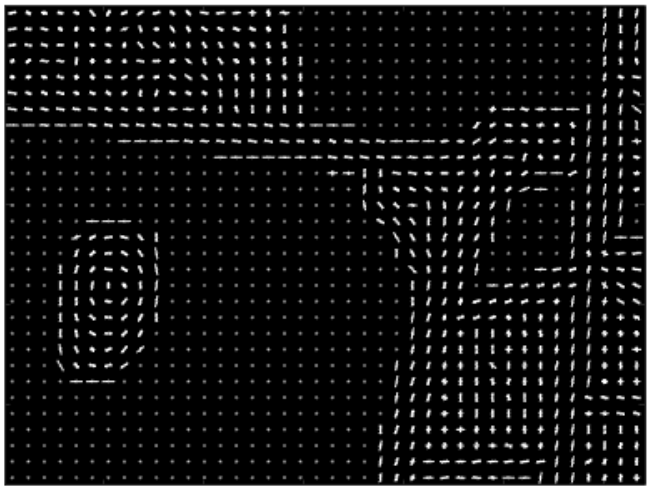
se nazývá gradient funkce I v bodě (u,v). [4] Velikost gradientu je nezávislá na orientaci obrazu a vypočítá se jako

$$|\nabla I|(u,v) = \sqrt{\left(\frac{\partial I}{\partial u}(u,v)\right)^2 + \left(\frac{\partial I}{\partial v}(u,v)\right)^2} \tag{6}$$

1.2.3 Histogram orientovaných gradientů

Histogram orientovaných gradientů (dále jen HOG) je příznak, který jak již ze svého názvu napovídá, využívá k popisu obrazu histogramy a gradienty. Uvažme obraz o velikosti 128 x 64 pixelů. Pro výpočet je obraz první rozdělen do buněk velikosti 8x8. Obecně však velikost buněk může být jakákoli, záleží na velikosti detailů v obraze. Pro každý pixel v buňce, je poté vypočítán gradient a je zaznamenána jeho orientace a velikost. Vypočítané hodnoty jsou využity k vytvoření histogramu, který je reprezentován vektorem. Orientace gradientů je v histogramu kategorizována do 9 rozmezí po 20° orientace. Orientace větší, než 180° jsou do těchto rozmezí také zahrnovány a jsou mapovány na úhly o 180° menší. Do kterého rozmezí jednotlivé pixely spadají je rozhodnuto pomocí Gaussovy funkce a směrodatné odchylky, která je rovna 10° se středem ve středu rozmezí. Každý pixel tedy může přispívat do dvou rozmezí a hodnota příspěvku je dána velikostí gradientu násobenou Gaussovou funkcí. Takto jsou vypočítány histogramy pro všechny buňky v obraze.

Aby byl příznak odolnější vůči změnám světelných podmínek, následuje proces normalizace histogramů. Toto se provádí na blocích velikosti 2x2 buněk. Histogramy (vektory) těchto buněk jsou zřetězeny do jednoho vektoru délky 36 a je provedena normalizace vydělením L2-norm vektoru. Postupným posunem bloků po buňkách se vypočítají histogramy pro všech 105 pozic bloků. Výsledný příznak celého obrazu je pak vytvořen zřetězením těchto 105 vektorů. Výsledný vektor má pak pro tuto velikost obrazu s velikostí buňky 8x8 délku 3780.[5][6]



Obrázek 7 HOG deskriptor

Obrázek 8 HOG na vstupním obraze

1.3 Speeded Up Robust Features

Metoda SURF neboli detekce významných bodů, si klade za záměr výpočet rychlého a stabilního deskriptoru schopného pracovat v reálném čase.[7] Deskriptory metoda počítá nezávisle na velikosti obrazu nebo jeho rotaci. Rychlosti metoda dosahuje využitím integrálních obrazů pro konvoluci obrazu.[8]

1.3.1 Integrální obraz

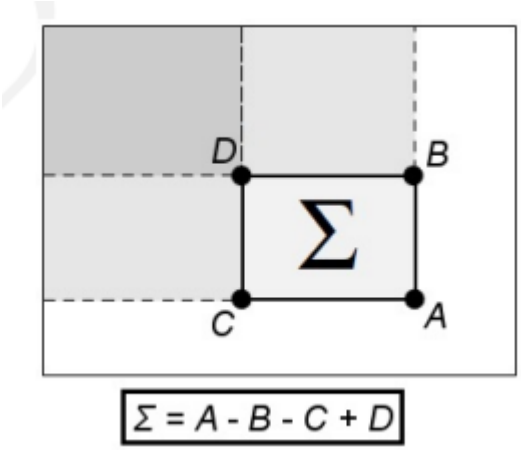
Integrální obraz je sktruktura vycházející ze vstupního obrazu umožňující rychlý výpočet součtu hodnot uvnitř obdélníkové části obrazu. Tuto strukturu můžeme zapsat jako

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \tag{7}$$

kde $I(i,j)$ je vstupní obraz. Součet hodnot tohoto regionu lze pak vyčíslit dosazením do vztahu:

$$\Sigma = A - B - C + D \tag{8}$$

kde Σ značí požadovaný součet a A, B, C, D jsou hodnoty I_Σ v daných souřadnicích.[7]



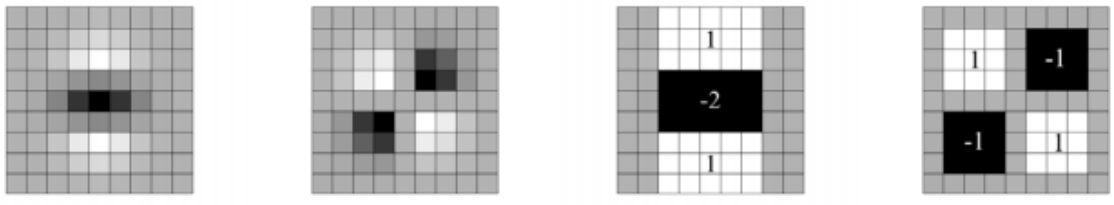
Obrázek 9 Integrální obraz [7]

1.3.2 Hessionova matice

Detektor je založený na Hessianově matici díky její rychlosti výpočtu a přesnosti. Matice je využita jak pro výpočet polohy významného bodu, tak jeho velikosti. Mějme bod $\boldsymbol{x} = (x, y)$ v obraze I, Hessianova matice $H(\boldsymbol{x}, \sigma)$ v bodě $\boldsymbol{x}$ při měřítku σ je definována jako

$$H(\boldsymbol{x}, \sigma) = \begin{bmatrix} L_{xx}(\boldsymbol{x}, \sigma) & L_{xy}(\boldsymbol{x}, \sigma) \\ L_{xy}(\boldsymbol{x}, \sigma) & L_{yy}(\boldsymbol{x}, \sigma) \end{bmatrix} \tag{9}$$

kde $L_{xx}(\boldsymbol{x}, \sigma)$ je hodnota druhé parciální derivace podle $\boldsymbol{x}$ konvoluce vstupního obrazu $I(x, y)$ s Gaussovou funkcí s měřítkem σ. Pro aproximaci druhých derivací Gaussovy funkce jsou využity tzv. obdélníkové funkce.



Obrázek 10 Srovnání druhých derivací a jejich aproximace [8]

Znázorněné filtry jsou velikosti 9×9 a jsou nejmenší možné velikosti pro sestavování těchto deskriptorů. Tyto aproximace jsou označeny jako D_{xx}, D_{yy} a D_{xy} a determinant Hessianu se spočte podle vzorce:

$$det(H_{approx}) = D_{xx}D_{yy} - \omega {D_{xy}}^2 \quad (10)$$

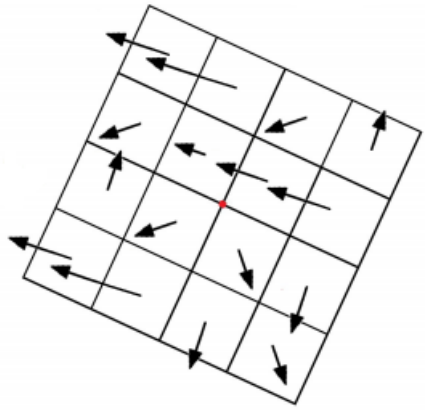
Následně jsou filtry normalizovány vzhledem ke své velikosti, což umožňuje ponechání konstantní váhy ω pro všechny velikosti filtrů. Tato váha byla empiricky stanovena na 0.9.

1.3.3 Sestavení scale-space

Významné body musejí vykazovat stabilní detekci i přes změnu měřítka obrazu, proto je nutné takové body lokalizovat uvnitř měřítkově nezávislé reprezentace. Scale-space je zkonstruován z jednotlivých obrazů vzniklých filtrací. Jak již bylo zmíněno výše, SURF využívá k detekci významných bodů determinant Hessianovy matice. Z výsledných odezev na filtraci je tedy postupně spočtena hodnota tohoto determinantu, čímž je získán obraz $H_{det}(\boldsymbol{x}, \sigma)$. Právě tyto obrazy představují finální měřítkově nezávislou reprezentaci vstupního obrazu, ve které jsou následně lokalizovány významné body. Významné body jsou detekovány jako maxima ve scale-space. Body jsou tedy porovnávány se svým okolím v rámci scale-space (8 sousedů + 18 sousedů z vyšší a nižší vrstvy). Pokud má daný bod ve svém okolí nejvyšší hodnotu, je považován za bod významný.

1.3.4 Sestavení deskriptoru

Prvním krokem ke konstrukci deskriptoru je vytvoření čtvercové oblasti obklopující významný bod. Tato oblast se středem ve zkoumaném bodě je natočena podle dominantní orientace daného bodu. Čtvercová oblast je dále rovnoměrně rozdělena na 4×4 podoblasti. V každé podoblasti je určeno pět pravidelně rozmístěných bodů, pro které je vypočtena jejich odezva na Haarovou vlnku ve směru osy x a y. Odezvy jsou označeny d_x a d_y. V rámci každé podoblasti je poté spočtena Σd_x a Σd_y. Z důvodu zvýšení odolnosti deskriptoru na změny osvětlení v obraze, jsou zjištěny i hodnoty $\Sigma|d_x|$ a $\Sigma\left|d_y\right|$. Tyto čtyři hodnoty pak tvoří vektor v. Výsledný deskriptor se skládá z takto získaných vektorů pro všech 16 podoblastí.[7][8]



Obrázek 11 SURF deskriptor [7]

2 SLEDOVACÍ ALGORITMY

Pro sledování objektu nejsou využívána pouze data ze současného snímku, ale jsou využívána také data z předchozích snímků. Algoritmy jsou tedy založeny na sledování změn kolem vybrané oblasti a odhadu, kam se sledovaná oblast přesunula. Pro sledování není třeba algoritmus trénovat na tvar objektu. Těmto algoritmům je třeba pouze specifikovat oblast sledování v prvním snímku, poté již vše probíhá bez obsluhy.

2.1 Online AdaBoost

Hlavní myšlenkou je formulování problému tak, aby se jednalo o binární klasifikaci sledovaného objektu a pozadí, a aktualizaci současného klasifikátoru sledovaného objektu.

Oblast z prvního snímku je považována za pozitivní vzorek. Dále jsou z nebližšího okolí odebrány další oblasti, které jsou považovány za vzorky negativní. Tyto oblasti jsou využity pro provedení prvních pár iterací algoritmu, za účelem získání prvního modelu. Pro samotné sledování, je v každém snímku prováděno několik kroků. Jako první se vyhodnotí současný klasifikátor na oblasti zájmu a v každém bodě je vypočítána pravděpodobnost, zda se zde objekt nachází. Oblast zájmu je následné přesunuta do bodu s největší pravděpodobností. V tento moment je objekt považován za detekovaný a dochází k aktualizaci klasifikátoru objektu. Zde je opět využita současná oblast jako pozitivní vzorek a okolní oblasti jako negativní. Po aktualizaci klasifikátoru se opět detekuje.

Tento algoritmus umožňuje generování klasifikátoru, který může být efektivně aktualizován inkrementální aplikací vzorků. Pro lepší porozumění definujeme 3 základní pojmy:

Slabý klasifikátor: Slabý klasifikátor musí fungovat pouze o něco lépe než náhodné hádání. Pravděpodobnost h^{weak} generovaná slabým klasifikátorem odpovídá příznakům obrazu.

Selektor: Mějme sadu dat M s pravděpodobnostmi $\varkappa^{weak} = \{h_1^{weak}, \dots, h_M^{weak}\}$. Selektor vybírá právě jednu z těchto pravděpodobností.

$$h^{sel}(x) = h_m^{weak}(x) \tag{10}$$

Kde m je vybráno podle odhadované chyby e_i každého klasifikátoru $h_i^{weak} \in \varkappa^{weak}$ tak, že $m = argmin_i e_i$

Silná klasifikátor: Mějme sadu M slabých klasifikátorů. Silný klasifikátor je vypočítán jako lineární kombinace selektorů. Hodnota $conf(\cdot)$ může být chápána jako míra důvěry silného klasifikátoru.

$$hStrong(x) = sign\big(conf(x)\big) \tag{11}$$

$$conf(x) = \sum_{n=1}^{N} \alpha_n \cdot h_n^{sel}(x) \tag{12}$$

Hlavní myšlenkou online boostingu je zavedení selektorů. Selektory jsou náhodně inicializovány a každý z nich drží sadu slabých klasifikátorů. Když dorazí nová trénovací data slabé klasifikátory každého selektoru jsou aktualizovány. Klasifikátor s nejmenší chybou je vybrán selektorem, kde chyba e_n je odhadována ze všech dosavadních vzorků.

Část vyžadující největší čas je aktualizace všech slabých klasifikátorů. Pro urychlení je proto využívána jedna „globální“ sada slabých klasifikátorů pro všechny selektory [9]. Po aktualizaci všech selektorů dojde k jejich lineární kombinaci do silného klasifikátoru na základě jejich vah α_n[10].

$$\alpha_n = \frac{1}{2} ln\left(\frac{1 - e_n}{e_n}\right) \tag{13}$$

2.2 Multiple instance lerning

V algoritmech učení pro trénování binárního klasifikátorů je třeba sada dat $\{(x_1, y_1), \dots, (x_n, y_n)\}$, kde x_i je instance příznaků a y_i je binární ohodnocení těchto dat. V MIL jsou trénovací data ve formátu $\{(X_1, y_1), \dots, (X_n, y_n)\}$, kde $X_i = \{x_{i1}, \dots, x_{im}\}$ je složka a y_i je ohodnocení složky. Ohodnocení složky je definováno jako: $y_i = max\big(y_{ij}\big)$ kde y_{ij} je ohodnocení instancí ve složce. Složka je tedy považována za pozitivní v případě, že obsahuje alespoň jednu pozitivní instanci.

Pro ohodnocení složek existuje několik metod, jednou z těchto metod je využití urychleného výpočtu gradientů pro maximalizaci log pravděpodobností složek.

$$log\ L = \sum_{i} (\log p(y_i|X_i)) \tag{14}$$

Důležitou vlastností je definování pravděpodobností nad složkami, nikoliv nad instancemi, jelikož jejich ohodnocení nejsou během trénování známy. Přesto však potřebujeme

popisovač, který ohodnocuje instance. Je tedy třeba vyjádřit pravděpodobnost složky ve vztahu k jejím instancím.

$$p(y_i|X_i) = 1 - \prod_j \left(1 - p\left(y_i|x_{ij}\right)\right) \tag{15}$$

Tato rovnice má požadovanou vlastnost a to takovou, že pokud má jedna z instancí ve složce vysokou pravděpodobnost, bude vysoká také pravděpodobnost složky. Takto ohodnocené složky jsou poté využity jako slabé klasifikátory v online AdaBoostingu popsaném v přechozí sekci.[11]

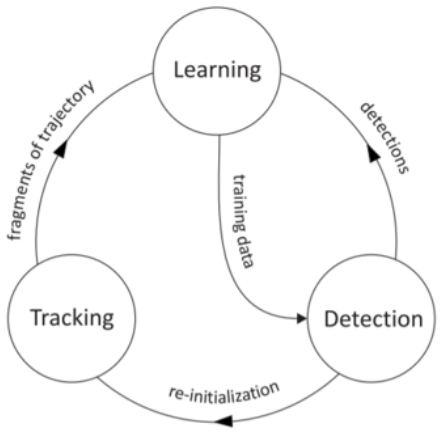
2.3 Tracking-Learning-Detection

TLD je framework navrhnutý pro dlouhodobé sledování neznámého objektu. Komponenty tohoto frameworku jsou následující:

Tracker: Odhaduje pohyb objektu mezi jednotlivými snímky za předpokladu, že pohyb mezi snímky je omezen a objekt zůstává viditelný. Tracker má vysokou šanci na selhání a následné nezotavení v případě, že se objekt ztratí z pohledu kamery.

Detektor: Chová se ke každému snímku nezávisle a provádí skenování celého snímku pro nalezení všech příznaků, které byly v minulosti pozorovány a naučeny. Detektor způsobuje dva typy chyb: chybně pozitivní a chybně negativní.

Learning: Sleduje výkon trackeru a detektoru, odhaduje chyby detektoru a generuje trénovací data pro eliminaci těchto chyb. Komponenta učení předpokládá, že obě komponenty mohou selhat. Na základě učení pak detektor lépe detekuje různé vzhledy objektu a lépe odděluje pozadí.[12]



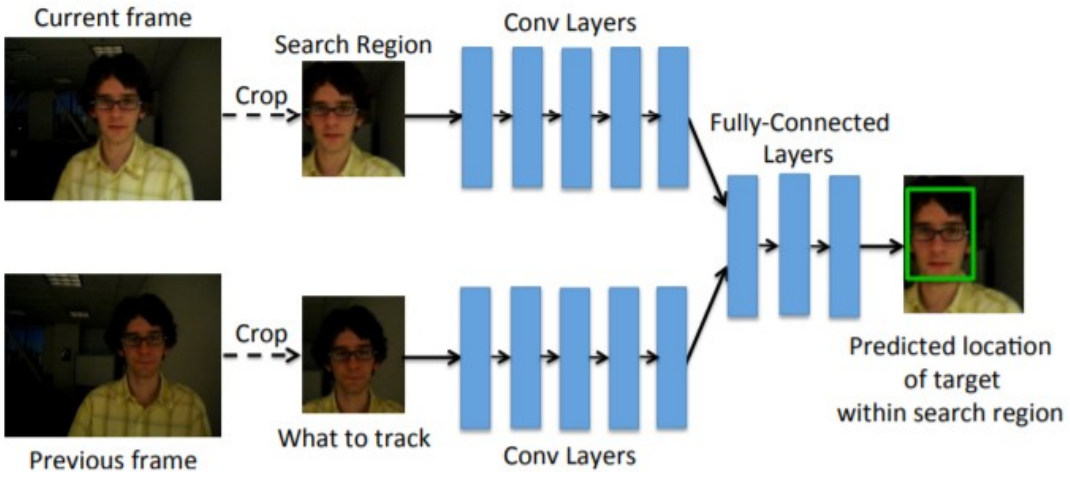
Obrázek 12 Blokový diagram TLD [12]

2.4 Generic Object Tracking Using Regression Networks

Tento přístup využívá pro sledování neuronovou síť. Snímky jsou posílány síti a ta vrací lokaci sledovaného objektu. Síť je trénována kompletně offline, tedy před spuštěním samotného sledování, pomocí obrázků a videí. Offline trénováním se tracker naučí obecný vztah mezi vzhledem a pohybem, čehož je poté využito pro sledování nových objektů bez nutnosti online trénování.

Předchozí snímek je ořezán a vycentrován na hledaný objekt, jak je vidět na obrázku (Obrázek 13). Tento přístup umožňuje sítí sledovat nové objekty, které ještě dříve neviděla. Výřez je následně lehce zvětšen čímž síť dostane více informací o okolí objektu. Pro nalezení objektu v současném snímku je využita znalost jeho předchozí lokace. Jelikož se objekty prostorem většinou pohybují hladce, předchozí lokace je dobrým odhadem pro to, kde by se objekt mohl nacházet v dalším snímku. Současný snímek je ořezán v okolí poslední pozice a předán síti. Cílem sítě je poté nalezení objektu v tomto výřezu. Pokud nedojde k úplnému zakrytí objektu, nebo se objekt nepohybuje příliš rychle, bude objekt nalezen v tomto výřezu. Pro rychle se pohybující objekty se může zvětšit oblast vyhledávání za cenu rychlosti a složitosti sítě.

Při trénování na videích jsou každé dva, po sobě jdoucí, snímky ořezány, jak bylo zmíněno výše. Tyto snímky jsou poté předány síti a provede se pokus o odhadnutí posunu objektu. Tento výsledek je poté využit pro výpočet L1 ztrátové funkce mezi predikcí a pravdou.[13]



Obrázek 13 Architektura GOTURN [13]

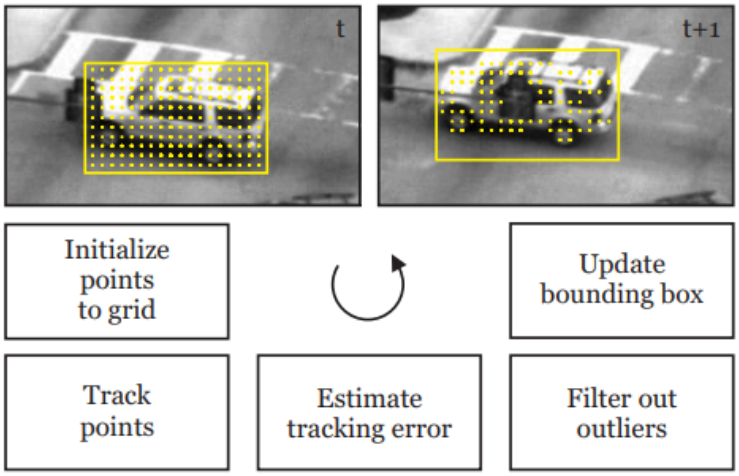
2.5 Median Flow tracker

Tento algoritmus se zakládá na výpočtu rozdílů mezi dopřednou trajektorií pixelů a zpětnou trajektorií pixelů.

Mějme $S = (I_t, \dots, I_{t+k})$ sekvenci snímků a x_t jako bod v čase t. Využitím libovolného trackeru je bod x_t sledován k kroků. Výslednou trajektorií je $T_f^k = (x_t, \dots, x_{t+k})$, kde f vyjadřuje směr vpřed. Našim cílem je odhadnutí chyby trajektorie T_f^k ze sekvence S. Pro tento účel si první vytvoříme ověřovací trajektorii. Bod x_{t+k} je sledován zpět až po první snímek, čímž vytvoří trajektorii $T_b^k = (\hat{x}_t, \dots, \hat{x}_{t+k})$, kde $\hat{x}_{t+k} = x_{t+k}$. Rozdíl těchto trajektorií je pak definován jako vzdálenost mezi těmito trajektoriemi. Vzdálenost jednotlivých bodů trajektorie je počítána jako Euklidovská vzdálenost výchozího bodu x_t a konečného bodu zpětné trajektorie $\hat{x}_t$.

$$vzdálenost\left(T_f^k, T_b^k\right) = \|x_t - \hat{x}_t\| \tag{15}$$

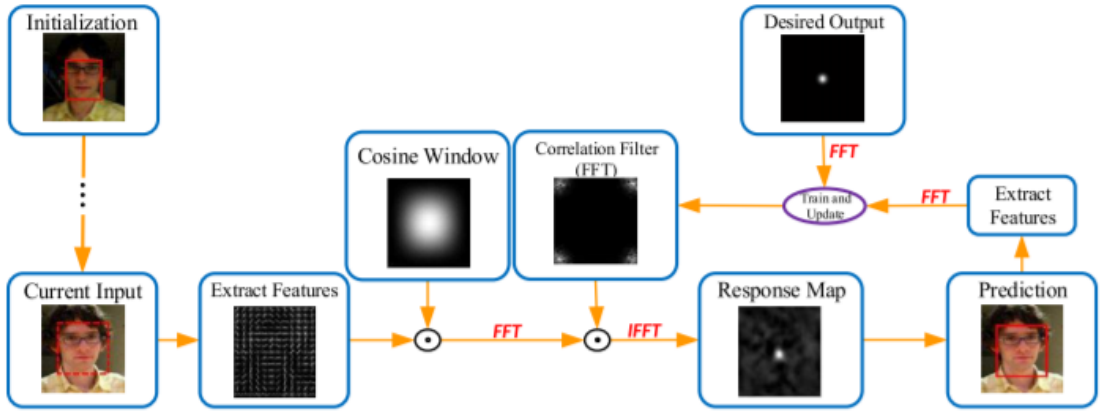
Pro sledování samotné je ve sledované ploše vybráno několik bodů z předchozího snímku. Pozice těchto bodů v současném snímku je odhadnuta pomocí Lucas-Kanadeovy metody. Je odhadnuta kvalita těchto předpovědí a každému bodu je přiřazen jeho rozdíl trajektorie. Polovina nejhorších předpovědí je zahozena. Zbylé předpovědi jsou použity pro výpočet výstupního obdélníku. Přemístění výstupní obdélníku je spočítáno jako medián zbývajících bodů. Změna velikosti obdélníků je počítána jako poměr vzdálenosti dvou bodů v předchozím snímku a jejich vzdálenosti v současném snímku, pro všechny dvojice bodů. Následně je opět vybrán jejich medián.[14]



Obrázek 14 Princip Median flow [14]

2.6 Algoritmy založené na korelačních filtrech

Podle současných algoritmů založených na korelačních filtrech se jejich obecný princip dá popsat následovně. Algoritmus je inicializován na prvním snímku a filtr je vytvářen minimalizací rozdílu mezi jeho výstupem a požadovaným výstupem (Gaussova funkce) nad tímto snímkem. Pro každý další snímek je poté vyřezána oblast kolem předchozí pozice, z oblasti jsou extrahovány příznaky a výsledek je zkombinován s kosinovým oknem. Pro urychlení výpočtů je následně tato oblast převedena do frekvenční oblasti Fourierovou transformací a je provedeno násobení po jednotlivých pixelech. Mapa pravděpodobností je poté získána zpětnou transformací. Pozice s největší pravděpodobností je označena jako pozice sledovaného objektu. Oblast kolem získané pozice je využita pro aktualizaci filtru.[15][16]



Obrázek 15 Princip korelačních algoritmů [16]

2.6.1 Minimum Output Sum of Squared Error

MOSSE je algoritmus pro vytváření filtrů z mála trénovacích snímků. Pro začátek je potřeba sada trénovacích snímků f_i a trénovacích výstupů g_i. g_i obsahují Gaussovu funkci zaměřenou na střed sledovaného objektu v trénovacím snímku f_i. F_i a G_i jsou Fourierovy transformace f_i a g_i, H je transformace filtru.

$$H_i^* = \frac{G_i}{F_i} \tag{16}$$

Pro nalezení filtru, který mapuje trénovací snímky na požadované výstupy MOSSE hledá filtr, který minimalizuje sumu rozdílů mezi výstupem konvoluce a požadovaným výstupem, umocněného na druhou.

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{17}$$

Tento zápis však vychází z ideálních podmínek, kdy je sledovaný objekt vycentrovaný v f_i a výstup g_i je fixní pro všechny snímky. Při sledování však objekt vždy vycentrovaný není a maximum v g_i se pohybuje za objektem v f_i.

Řešení tohoto problému není příliš složité. Každý element filtru H (indexovaný ω a v) je počítán samostatně. Rovnici je přepsána a její první derivace podle $H^*_{\omega v}$ je položena rovno nule:

$$0 = \frac{\partial}{\partial H^*_{\omega v}} \sum_i |F_{i\omega v} H^*_{\omega v} - G_{i\omega v}|^2 \tag{18}$$

Vyřešením pro H^* se dostane:

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*} \tag{19}$$

[17]

2.6.2 Kernelized Correlation Filters

Narozdíl od algoritmů BOOSTING a MIL, představených v kapitolách 2.1 a 2.2 resp., které pro aktualizaci klasifikátoru sbírají pouze několik snímků v okolí objektu, algoritmus KCF využívá pro aktualizaci všechny snímky z celého obrazu. Oproti intuici toto umožní efektivnější trénování. Důvodem tohoto zefektivnění je, že za správných podmínek se matice kernelů stane cirkulární. Cirkulární matice $C(u)\ n \times n$ se z vektoru $u\ n \times 1$ sestaví zřetězením všech posunů vektoru u:

$$C(u) = \begin{bmatrix} u_0 & u_1 & u_2 & \dots & u_n \\ u_{n-1} & u_0 & u_1 & & u_{n-2} \\ u_{n-2} & u_{n-1} & u_0 & \dots & u_{n-3} \\ \dots & \dots & \dots & & \dots \\ u_1 & u_2 & u_3 & \dots & u_0 \end{bmatrix} \tag{20}$$

Motivací za cirkulárními maticemi je to, že vyjadřují konvoluci vektorů, což se koncepčně blíží ohodnocování klasifikátorů na mnoha pozicích. $C(u)v$ reprezentuje konvoluci vektorů a může proto být vypočítán ve Fourierově doméně:

$$C(u)v = F^{-1}\big(F^*(u) \odot \mathrm{F}(v)\big) \tag{21}$$

[18][19]

2.6.3 Discriminative Correlation Filter Tracker with Channel and Spatial Reliability

Tento algoritmus využívá pro sledování několik druhů příznaků, jejichž filtry jsou označovány jako kanály. Mějme sadu N_c kanálů příznaků $f = \{f_d\}_{d=1:N_c}$ a odpovídajících cílových vzorů $h = \{h_d\}_{d=1:N_c}$. Pozice objektu je odhadována jako výsledek korelace $\tilde{g}(h)$:

$$\tilde{g}(h) = \sum_{d=1}^{N_c} f_d * h_d \tag{22}$$

Optimální filtr h je určen minimalizací $\varepsilon(h) = \|\tilde{g}(h) - g\|^2 + \lambda\|h\|^2$, kde g je požadovaný výstup, což se typicky rovná Gaussově funkci. Ne všechny příznaky však nabývají stejných velikostí. Pro vyhnutí se tomuto problému jsou všechny kanály brány samostatně:

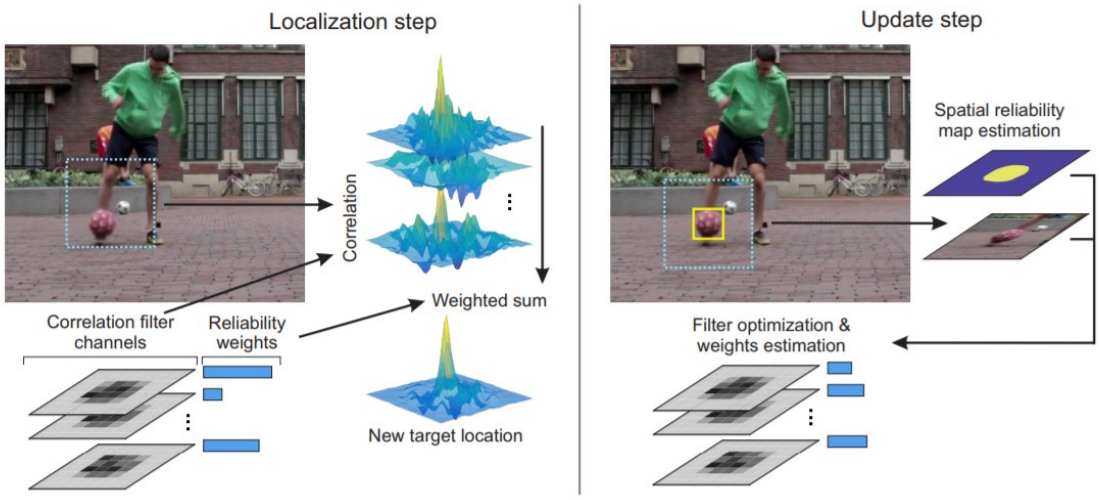
$$\varepsilon(h) = \sum_{d=1}^{N_c} \|f_d * h_d \ - g\|^2 + \lambda\|h\|^2 \tag{23}$$

Dále je zavedena váha kanálu $w = \{\widetilde{w}_d\}_{d=1:N_c}$ která je považována za škálovací faktor založený na síle kanálu. Tato váha se nazývá *váha spolehlivosti kanálu* a je aplikována v momentě výpočtu korelace.

$$\tilde{g} = \sum_{d=1}^{N_c} f_d * h_d \cdot \ \widetilde{w}_d \tag{24}$$

Pro sledování objektu jsou potřeba dva kroky. Lokalizace objektu je provedena podle popsaného algoritmu a poté následuje aktualizace filtru a vah.

Aktualizace začne vycentrováním trénovací plochy na objekt. Jsou spočítány histogramy popředí a pozadí a jsou aktualizovány exponenciálním klouzavým průměrem. Následně je vytvořena *mapa prostorní spolehlivosti* a jsou vytvořeny optimální filtry. Filtry a váhy spolehlivosti kanálů jsou aktualizovány exponenciálním průměrem mezi současným a předchozím snímkem.[20]



Obrázek 16 Princip CSRT [20]

3 DETEKČNÍ ALGORITMY

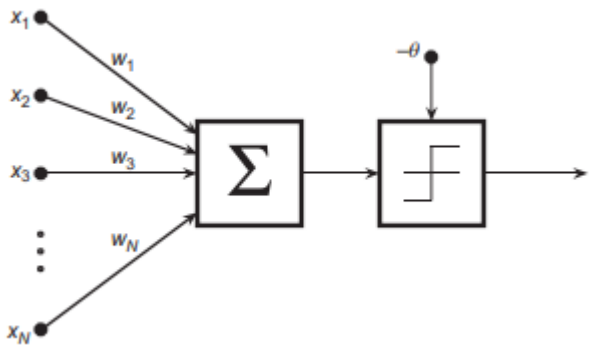
Pro detekci objektu je využíván pouze aktuální snímek. Algoritmy tedy nemají žádnou informaci o tom, kde se objekt nacházel v předchozím snímku. Pro detekování je také třeba algoritmu říct, jak detekovaný objekt vypadá. Tohoto se dá docílit trénováním algoritmů nad positivními a negativními daty. Algoritmy si takto vybudují znalost toho, jak má sledovaný objekt vypadat a s touto znalostí mohou následně pracovat pro detekování naučeného tvaru v obraze.

3.1 Umělá neuronová síť

Koncept umělých neuronových sítí pro využití v systémech rozpoznávání obrazu začal v 50. letech a pokračoval až do 60. let 20. století. Bledsoe and Browning vyvinuli v roce 1959 tzv. „n-tuple“ tyl klasifikátoru, který využíval bitové nahrávání a vyhledávání v binárních datech. Přestože si zmíněny typ klasifikátoru udržel své spřízněnce, největší vliv měl v této oblasti Rossenblattův tzv. „perceptron“.[21]

Neuronové sítě nefungují jako běžný předepsaný program. Umělé neuronové sítě se snaží napodobovat biologickou nervovou soustavu. Jsou založeny na neuronech provádějící výpočty a rozhodování, a synapsích zprostředkujících komunikaci. Tyto struktury jsou během učení aktualizovány, jsou jim přiřazovány různé váhy, čímž se dosahuje adaptace na různé úlohy.

Perceptron je lineární klasifikátor pro klasifikaci objektů do dvou tříd. Jako vstup akceptuje vektor příznaku $x = (x_1, \dots, x_n)$ a produkuje výstup o hodnotě $\sum_{i=1}^{N} w_i x_i$. Klasifikace je pak dokončena aplikací Heavisideovy funkce θ.



Obrázek 17 Perceptron [21]

Matematicky je pak zjednodušeno pomocí zapsání $-\theta$ jako w_0 a položením rovnu vstupu x_0 který je udržován konstantní. Výstup lineární části klasifikátoru je roven:

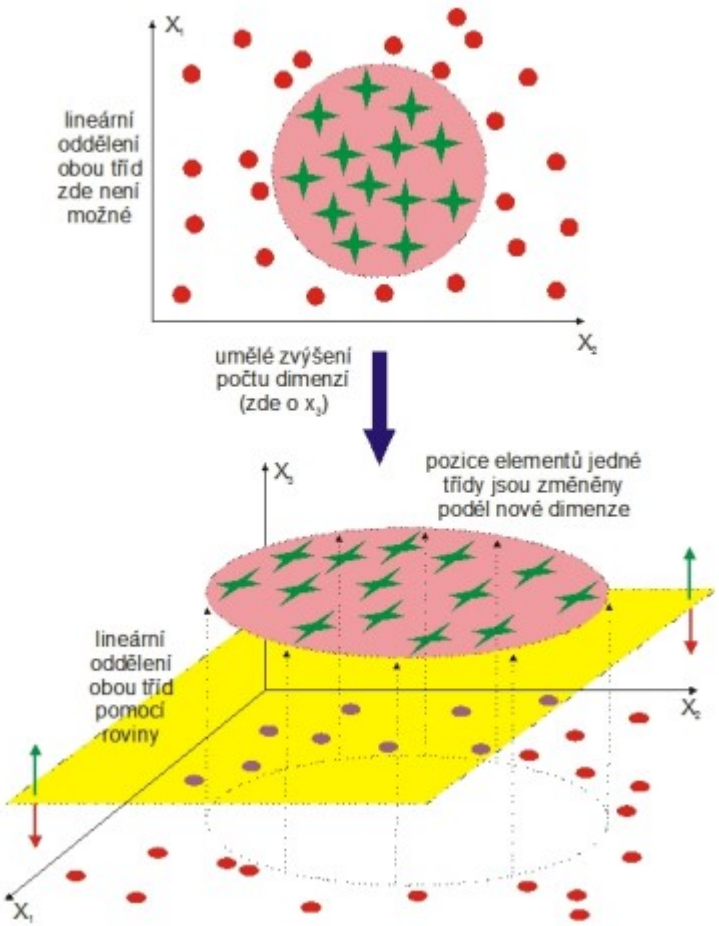
$$d = \sum_{i=1}^{N} w_i x_i - \theta = \sum_{i=1}^{N} w_i x_i + w_0 = \sum_{i=0}^{N} w_i x_i \tag{25}$$

a výsledný výstup celého klasifikátoru je roven [21]:

$$y = f(d) = f\left(\sum_{i=0}^{N} w_i x_i\right) \tag{26}$$

3.2 Support vector machine

Základní myšlenkou SVM je nalezení nadroviny, která separuje n-dimenzionální data do dvou tříd. Ne vždy jsou však data lineárně separovatelná [22]. Aby se tedy třídy daly lineárně rozdělit, je třeba data převést do vícedimenzionálního prostoru, kde již od sebe lineárně rozdělit jdou.



Obrázek 18 Princip SVM [23]

Toto je jednoduchá myšlenka. Na obrázku 15 můžeme vidět, že původní data jsou ve dvojrozměrném prostoru rozdělena kružnicí. Přidáním další dimenze můžeme prvkům přiřadit další souřadnici. Tímto některé prvky posuneme třeba nahoru nad původní rovinu a nyní již existuje rovina rovnoběžná s osami x_1 a x_2, která data rozděluje.[23]

Abychom zamezili chybným klasifikacím, snažíme se také najít takovou nadrovinu, která bude maximalizovat její vzdálenost od nejbližších prvků jednotlivých tříd. Máme-li n trénovacích vzorků $\{x_i, y_i\}$ s ohodnocením $y_i \in \{-1,1\}$, všechny nadroviny jsou charakterizovány vektorem w a konstantou b

$$wx + b = 0 \tag{27}$$

Vzdálenost bodu od této nadroviny je pak:

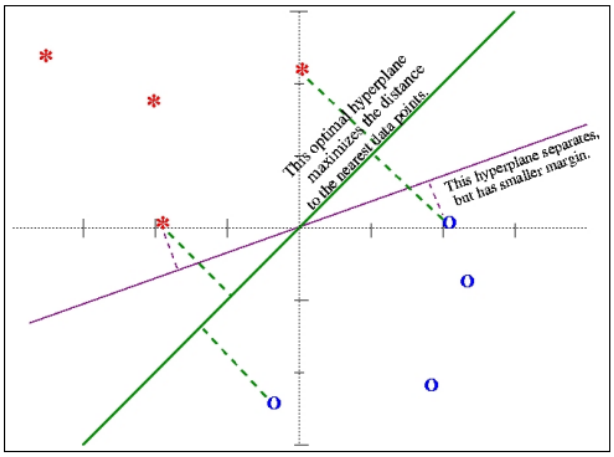
$$d\big((w,b),x_i\big) = \frac{y_i(x_i w + b)}{\|w\|} \geq \frac{1}{\|w\|} \tag{28}$$

Kde pro nalezení nejvýhodnější nadroviny minimalizujeme $\|w\|$. Tohoto dosáhneme metodou Lagrangeových multiplikátorů a problém je tak převeden na:

$$\begin{aligned} &minimalizace: W(\alpha) \\ &\qquad = -\sum_{i=1}^{n} \alpha_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \big(x_i \cdot x_j\big) \\ &za\ podmínek: \sum_{i=1}^{n} \alpha_i y_i = 0\,, 0 \leq \alpha_i \end{aligned} \tag{29}$$

Po získání takovéto roviny je klasifikátor dán rovnicí [22][23]

$$f(x) = sign(wx + b) \tag{30}$$



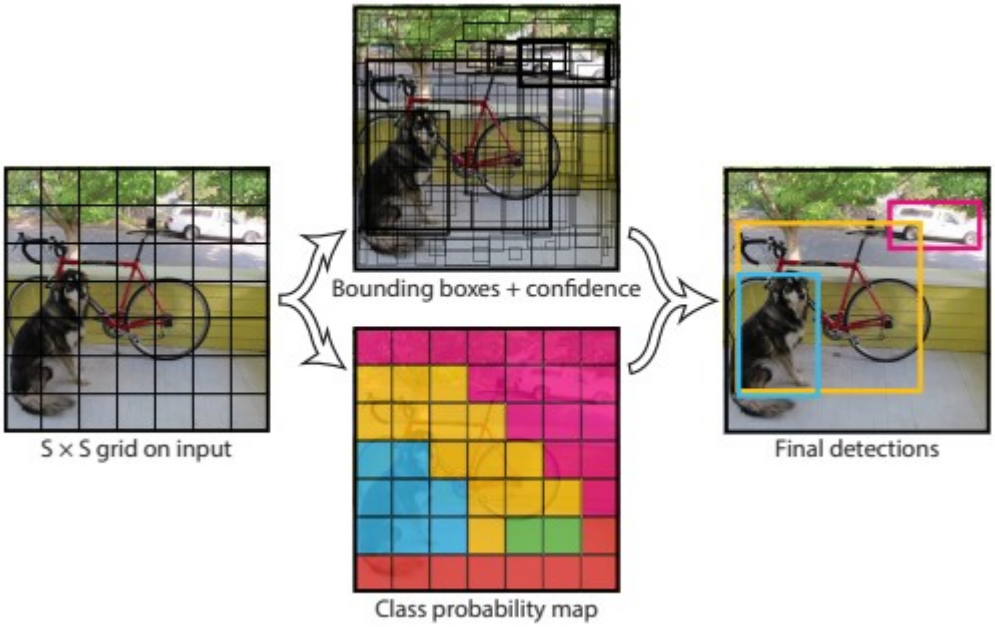
Obrázek 19 Optimalizace nadroviny [22]

3.3 You only look once

YOLO se na problém detekce dívá jako na problém regrese k ohraničujícím obdélníkům a jim přiřazených pravděpodobností tříd. Jedna jediná neuronová síť předpovídá ohraničující obdélníky a pravděpodobnosti tříd přímo z celého obrazu a to za jediného průchodu sítí [24]. YOLO verze 3, vydaný v roce 2018, je jeden z nejrychlejších detekčních algoritmů a je proto vhodný pro aplikace reálného času. V čem však zaostává za jinými algoritmy, jako je například RetinaNet nebo SSD je jeho přesnost [25]. V roce 2020 byla vydána verze 4 tohoto algoritmu, která zvyšuje rychlost o dalších 12 % a přesnost o 10 % [26].

Obraz je první rozdělen do $S \times S$ mřížky. Pokud střed objektu spadá do jedné z těchto buněk, je tato buňka zodpovědná za detekci objektu. Každá buňka prochází sítí a predikuje se B ohraničujících obdélníků a míru důvěry pro tyto obdélníky. Tyto míry důvěry reflektují, jak sebevědomý model je, že se v obdélníku nachází objekt a s jakou přesností. Každý obdélník se skládá z 5 predikcí: x, y, w, h a míry důvěry. Každá buňka vrací jednu sadu predikcí nehledě na to, kolik se v ní nacházelo obdélníků. Při testování se provádí násobení podmíněných pravděpodobností tříd s individuálními mírami důvěry obdélníků.[24]

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class) * IOU_{pred}^{truth} \quad (31)$$



Obrázek 20 Princip YOLO [24]

II. PRAKTICKÁ ČÁST

4 TESTOVÁNÍ RŮZNÝCH METOD

Výstupy zpracování obrazu jsou využívány k řízení polohy nakloněné roviny. Důležitou vlastností zpracování obrazu je tedy rychlost, s jakou jsme schopni systému řízení předat informaci o poloze kuličky. Rychlost zpracování musí pro zachování správné funkčnosti sytému tedy musí dosahovat minima 15 FPS a ideálně alespoň 20 FPS.

Abych tedy mohl správně rozhodnout, který algoritmus je vhodný pro následnou implementaci, vyzkouším tyto algoritmy v umělém prostředí na stolním PC s procesorem AMD Ryzen 2700X, který je dle webu cpubenchmark.net asi 5x výkonnější při práci s jedním vláknem. Tímto získám alespoň základní představu o tom, v jakých rychlostních oblastech se algoritmy pohybují a nevhodné algoritmy tak budu moci vyřadit.

4.1 YOLO

Pro vyzkoušení této metody jsem využil tutoriálu dostupného z [30]. Tento tutoriál zahrnuje všechno, od vytvoření neuronové sítě, až po zpracování všech ohraničujících obdélníků. Pro testování rychlosti jsem nepovažoval za důležité vlastní trénování celé sítě, využil jsem proto předem natrénovaných modelů dostupných přímo od autorů algoritmu [27]. Autoři nabízejí hned několik verzí. Jednotlivé modely se liší jejich velikostí a v důsledku jejich výpočetní rychlostí. Z těchto typů jsem zvolil možnost YOLOv3-tiny, která je několikanásobně rychlejší než ostatní modely.

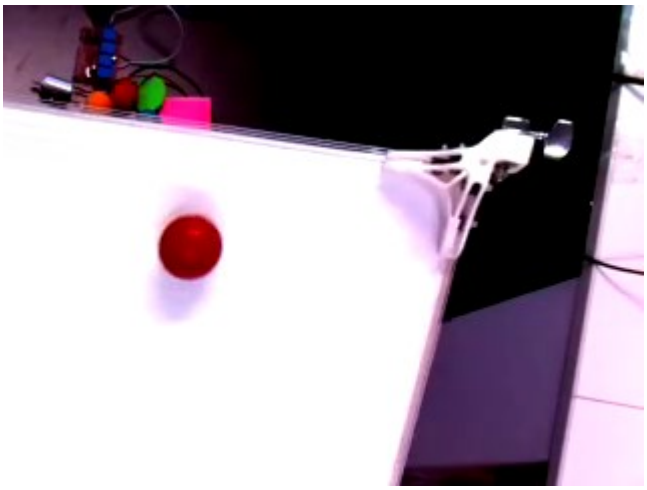
Tyto modely jsou trénovány nad datovou sadou COCO. Tato datová sada obsahuje 80 různých tříd, bohužel kulička není jednou z nich. Pro testování jsem tedy využil videa s projíždějícími auty, jelikož auto je jednou z natrénovaných tříd.

Z obrázku (Obrázek 21) můžeme vidět, že detekce není perfektní. Rychlostně se pohybujeme na úrovni 30-40 FPS, což není vůbec slibné. Při úvaze, že je náš procesor asi 5x výkonnější než procesor RP, můžeme velice zhruba odhadnout, že rychlost na RP by se pohybovala pod 10 FPS. Dále se zde vyskytuje problém s několikanásobnou detekcí jednoho objektu.

Obrázek 21 Testování YOLO [33]

4.2 SVM s využitím HOG

Součástí dokumentace OpenCV je vzorová aplikace, která umožnuje trénování SVM s využitím příznaků HOG a následné detekování. Implementace je dostupná na [31]. Pro trénovaní jsou potřeba dvě sady dat. Jedna sada dat, takzvaná pozitivní, obsahuje snímky obsahující kuličku. Druhá, negativní sada, obsahuje snímky, které neobsahují žádnou část kuličky. Tato data jsem pomocí OpenCV vygeneroval z videa pořízeného kamerou systému. Video jsem rozdělil na jednotlivé snímky a každý snímek rozdělil na 4 menší. Z takto vytvořených snímků jsem vybral 221 pozitivních snímků



Obrázek 22 SVM pozitivní data

a 1503 negativních snímků.

Obrázek 23 SVM negativní data

S takto připravenými daty jsem spustil zkompilovanou aplikaci pro natrénování SVM:

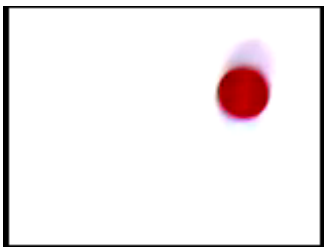
```
SVM.exe -pd=positive_path -nd=negative_path -fn=output_path
```

Výstupem tohoto příkazu je soubor obsahující definici nadroviny, která separuje vstupní data. Pro testování aplikaci spustím příkazem:

```
SVM.exe -t -td=testing_path -fn=definition_path
```

Aplikace zobrazuje jednotlivé snímky z testovacích dat s ohraničujícími obdélníky kolem nalezených objektů.

S těmito daty se aplikaci bohužel nepodařilo detekovat kuličku ani v jednom snímku. Vytvořil jsem proto další sadu pozitivních dat, ve které jsem ořezal snímek tak, aby obsahoval pouze kuličku a bílé pozadí nakloněné roviny. Ze snímků tak zmizely všechny části pozadí, které se nacházejí mimo nakloněnou rovinu. Takto jsem dostal 34 snímků kuličky (černý okraj je pouze ilustrační):



Obrázek 24 Čisté pozadí

Opět jsem provedl trénování SVM a obdržel novou definici. S touto definicí jsem spustil detekování kuličky. Výsledek byl úplně stejný. Kulička opět nebyla detekována ani v jednom ze snímků.

Jako poslední jsem pozitivní data upřesnil ještě více a z použitých 34 snímků jsem vyřízl oblasti o velikosti 32×32 pixelů zaměřených přímo na kuličku.

Obrázek 25 Výřez kuličky

Následně jsem opakoval stejný postup pro trénování. Pro detekci jsem však využil dalších parametrů aplikace.

```
SVM.exe -pd=positive_path -nd=negative_path -fn=output_path
-dw=32 -dh=32
```

Parametry `dw` a `dh` specifikují velikost detekčního okna. Nyní, když se detekční okno pohybuje po obraze, odpovídá jeho velikost přímo velikosti kuličky v obraze.

Ani tato úprava bohužel nepřinesla úspěch při detekování. Důvod, proč tomu to tak je mi není jasný. Předpokládám, že by se mohlo jednat o problém s velikostí kuličky v obraze a ve tvaru kuličky jako takovém. Kulička je ve vstupním obraze velice malá, její tvar je příliš jednoduchý a velice nevýrazný v porovnání třeba se siluetou vozidla.

4.3 Sledovací algoritmy

Implementace těchto algoritmů je velice jednoduchá. Prakticky se jedná pouze o inicializaci algoritmu prvním ohraničujícím obdélníkem, který obsahuje náš sledovaný objekt:

```
tracker->init(prvniSnimek, obdelnik)
```

a poté stačí při zpracovávání snímku tento celý snímek předat algoritmu:

```
tracker->update(dalsiSnimek, vystupniPozice)
```

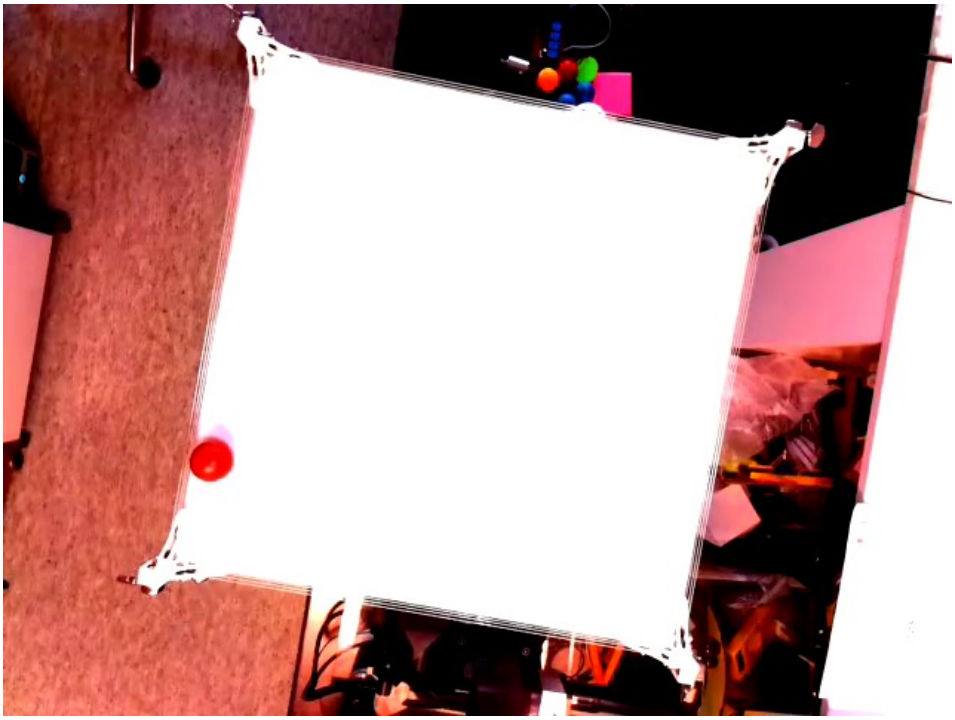
Algoritmus nám poté vrátí souřadnice bodu, kde si myslí, že se kulička nachází.

Verze knihoven OpenCV 3.2.0. obsahuje 6 sledovacích algoritmů, oproti tomu novější verze 4.5.0 implementuje další 2 sledovací algoritmy. Konkrétně nová verze přidává algoritmy CSRT a MOSSE. Při testování těchto algoritmů se výkon a přesnost pohybovaly na různých úrovních. Například algoritmy BOOSTING a TLD ukazovaly dostatečnou přesnost, jejich rychlost byla však slabých 20 FPS. CSRT se vykazoval skvělou přesností, ale také nedosahoval výše než 25 FPS. Přestože KCF běžel rychlostí až 60 FPS jeho přesnost byla mizivá a sledovaný objekt ztrácel více méně okamžitě. Nejslibnějšími byly algoritmy

MedianFlow a MOSSE. MedianFlow dosahoval rychlosti až 400 FPS a více s velice dobrou přesností a MOSSE se pohyboval na 150 FPS rovněž s velice dobrou přesností.

5 VÝBĚR VHODNÉHO ALGORITMU

Obrazová data, která získáváme z kamery mají následující podobu:



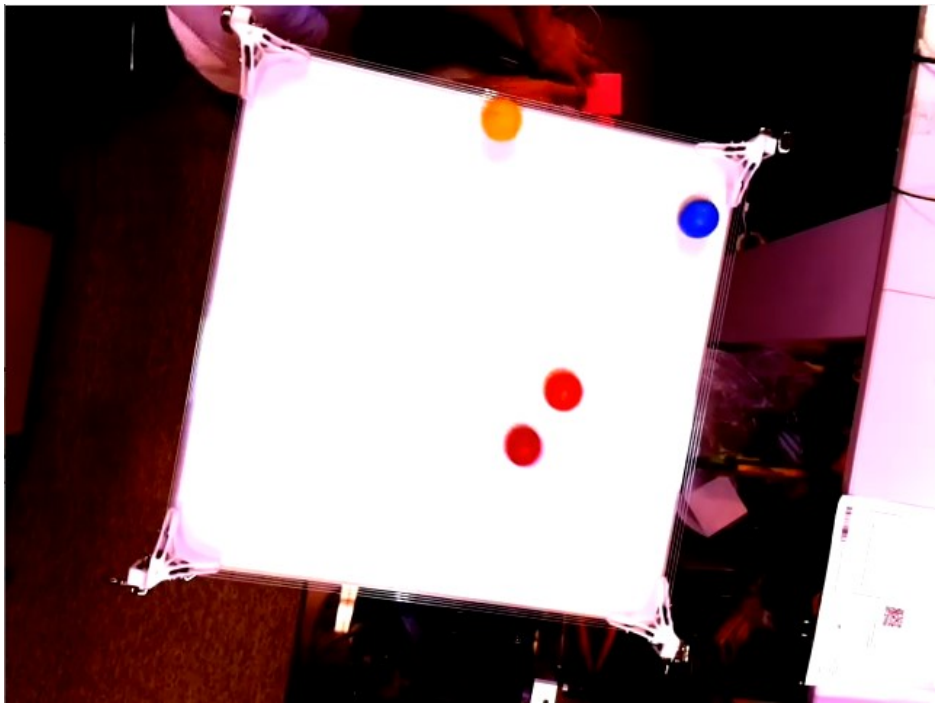
Obrázek 26 Originální obraz z kamery

Úkolem algoritmu je vrácení pozice kuličky v tomto obraze. Hlavními rysy kuličky, jsou její tvar a barva. Hledat kuličku tedy můžeme podle těchto dvou parametrů. Původní algoritmus využívá pro hledání výhradně barvu kuličky a tvarem jako takovým se nezabývá.

5.1 Detekce objektu

Pro tyto algoritmy se často využívá specializovaný HW jako je například řada produktů NVIDIA Jetson. Jak bylo uvedeno výše, i při využití algoritmu YOLO, který je jedním z nejrychlejších algoritmů, nedosahují na běžném HW dobrých rychlostí. Jelikož je rychlost jedním z požadavků pro správnou regulační funkci celého modelu, není tento přístup vhodný.

Dalším problémem tohoto přístupu je několikanásobná detekce. Na nakloněné rovině se může nacházet několik kuliček různé barvy. Sledovat však chceme pouze jednu, kterou si můžeme volně vybrat.



Obrázek 27 Rovina s více kuličkami

I v případě, kdyby nám detekce vrátila seznam těchto kuliček, bylo by ještě třeba z tohoto seznamu pomocí barvy vybírat zvolenou kuličku, což přináší další zpomalení.

Výhodou tohoto přístupu může být právě jeho nezávislost na předchozích snímcích. Jelikož nedochází ke vnitřním změnám dat na jejichž základě detekujeme, máme jistotu, že vždy detekujeme kuličku tak, jak jsme ji dříve do algoritmu natrénovali.

5.2 Sledování objektu

Rychlostně se algoritmy pohybují na různých úrovních. Některé se na RP nevyšplhají ani na 10 FPS, některé běží rychlostí několik stovek FPS.

Problém vícenásobné detekce se zde nevyskytuje. Algoritmy vždy vrací pouze jeden výsledek, což je velkou výhodou.

Nevýhodou je však přesnost některých algoritmů. Jelikož objekt specifikujeme pouze na začátku, nemáme žádnou kontrolu nad tím, kam se sledovaná oblast posouvá. Může se tedy stát, že se za běhu sledovaná oblast odpoutá od kuličky. V případě, že toto nezpůsobí chybu při sledování, o které bychom byli algoritmem informováni, nemusíme se o tomto problému dozvědět, což může vést k chybnému řízení.

5.3 Volba algoritmu

Na základě předcházejících informací jsem se rozhodl využít sledovací algoritmy. Knihovny OpenCV obsahují několik implementací různých sledovacích algoritmů. Do aplikace implementuji všechny tyto algoritmy s jednoduchou možností jejich přepínání za běhu aplikace.

6 ROZBOR SOUČASNÉ APLIKACE

Jedním z hlavních prvků práce je to, že algoritmus implementuji do již vytvořené aplikace pro řízení reálného modelu. Tato skutečnost s sebou přináší také několik omezení. Omezení se primárně týkají volby programovacího jazyka aplikace a využitých softwarových balíčků. Aplikace je psána v jazyce C++ s využitím knihoven Qt pro vytvoření grafického uživatelského prostředí a pro zpracování obrazu jsou využity knihovny OpenCV ve verzi 3.2.0.

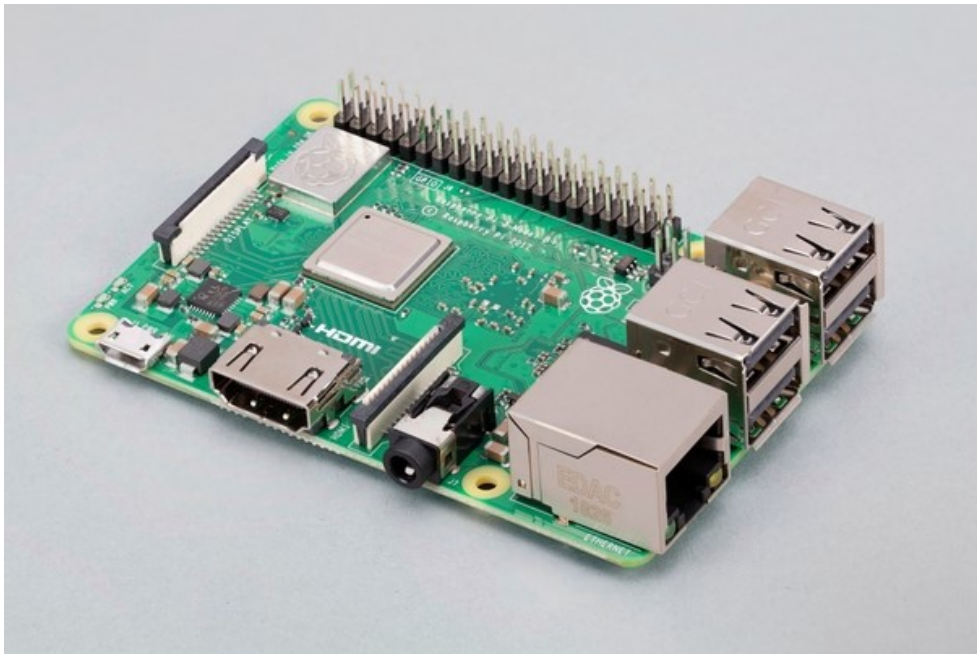
Celý model je řízen počítačem Raspberry Pi 3B+ a obrazová data jsou pořizována pomocí modulu Raspi Cam v rozlišení 640x480.

6.1 Raspberry Pi 3B+

Vlastnosti počítače:

- 64-bit architektura ARM
- Cortex-A53 @ 1,4 GHz, 4 jádra
- Bezdrátová rozhraní Wi-Fi a Bluetooth
- 40 pinová GPIO sběrnice
- 4 USB 2.0 porty
- RJ45 Ethernet port
- CSI port pro připojení Raspi Cam modulu
- Podpora PoE

Jako operační systém je zvolena unixová distribuce Raspberry Pi OS vycházející z Debianu.



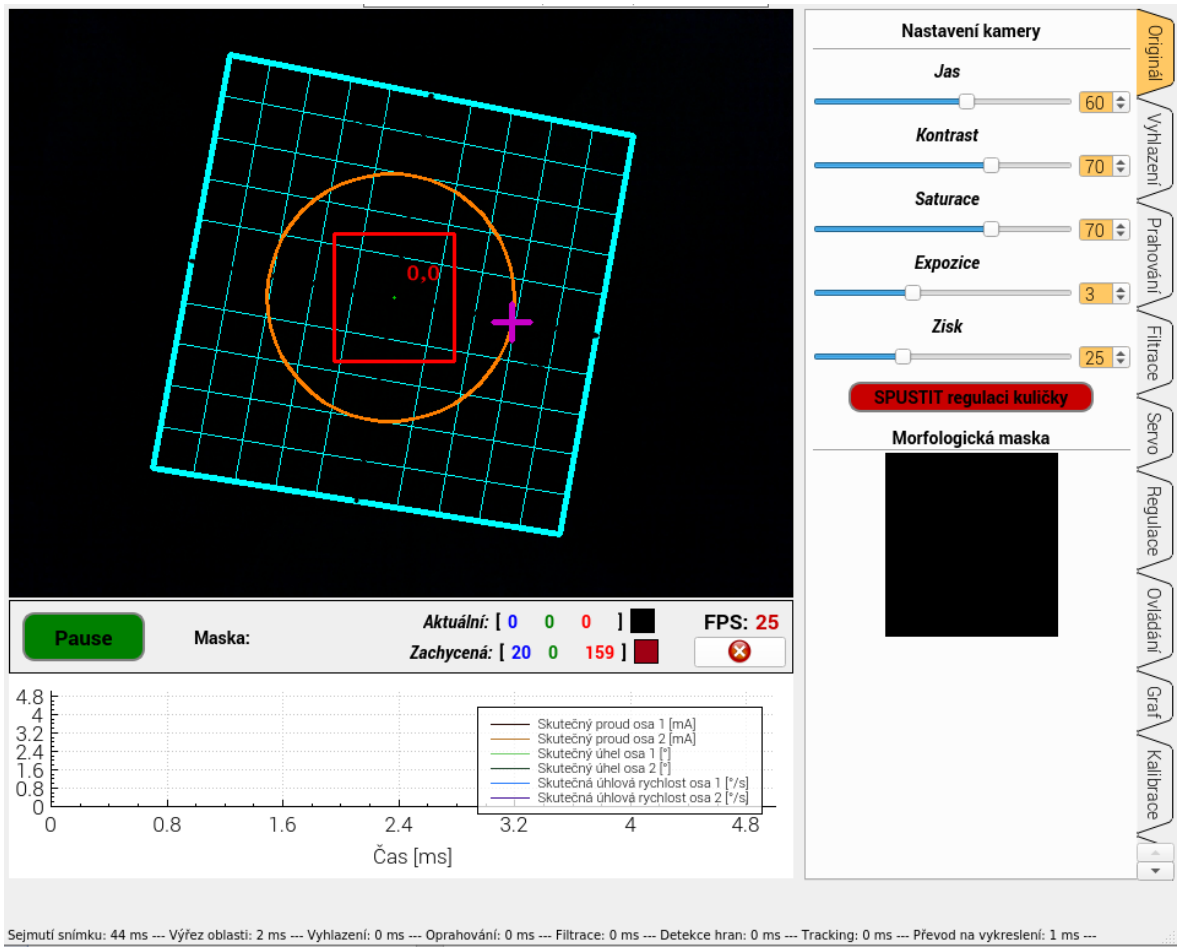
Obrázek 28 Raspberry Pi 3B+ [28]

6.2 OpenCV

OpenCV (Open Source Computer Vision Library) je open-source knihovna obsahující mnoho vysoce optimalizovaných algoritmů pro práci s obrazem. Tuto knihovnu lze využít v programovacích jazycích JAVA, Python a C++.

6.3 Grafické rozhraní

Grafické rozhraní je rozděleno do 3 hlavních částí:



Obrázek 29 Grafické rozhraní aplikace

V pravé části okna aplikace se nachází seznam oken. Přepínání v tomto seznamu určuje, jaké prvky se zobrazují ve zbytku aplikace. Pro sledování zde přidám další položku do seznamu.

V levé části okna se nachází náhled z kamery, grafy ovládaných fyzikálních veličin a pár dalších ovládacích prvků. Zajímavým prvkem je zde maska. Tady se ukládá obdélníková část obrazu, kterou aplikace umožňuje vybrat podržením kolečka myši. Tato funkcionalita se skvěle hodí pro výběr inicializační oblasti sledovacích algoritmů. Do masky můžu také ukládat výstup sledovacího algoritmu, který vrací souřadnice obdélníku obklopujícího sledovanou plochu. Takto bude dobře vidět, co zrovna algoritmus sleduje.

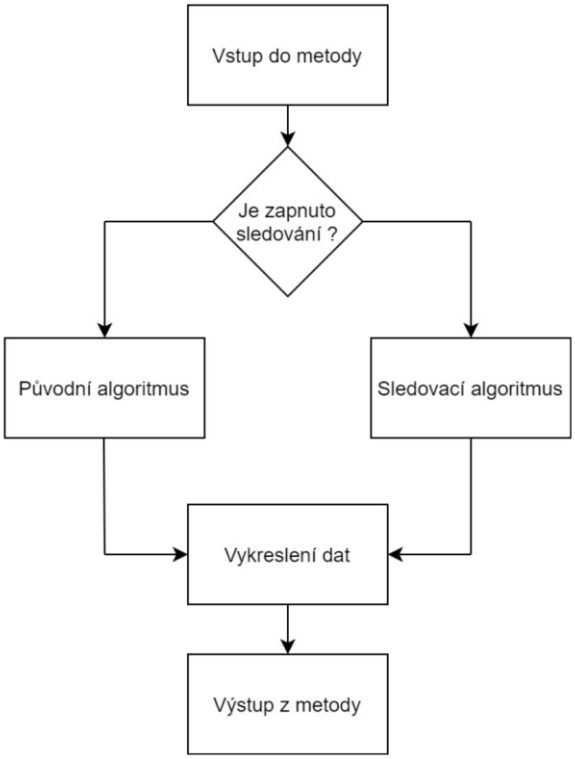
Napravo od náhledu kamery se nachází oblast s ovládacími prvky pro dané okno. Zde se bude nacházet přepínání mezi původním algoritmem a novými algoritmy. Také se zde bude nacházet přepínání mezi jednotlivými algoritmy sledování.

6.4 Kódová základna

Aplikace je napsána tak, aby maximálně využívala HW prostředky RP. Za běhu jsou vytvořena 4 vlákna, kde každé vykonává svou roli. Jedno z těchto vláken je využíváno ke zpracování obrazu. V rámci tohoto vlákna je volána metoda, která se postará jak o detekci, tak o výsledné zapsání výsledků a zobrazení dat.

Obrázek 30 Současná struktura metody

Této struktury využiji a přidám před zpracování obrazu přepínač, dle kterého se aplikace rozhodne, jakou metodou obraz zpracuje.



Obrázek 31 Upravená struktura metody

7 IMPLEMENTACE ALGORITMŮ

7.1 Upgrade verze OpenCV

Jak jsem již zmínil v kapitole 4.3, verze 4.5.0 obsahuje dva nové sledovací algoritmy. Verze 3.2.0 je také verze starší a její upgrade je tak žádaný. Jedním z problémů je to, že tato verze není součástí žádného balíčku, který by se dal jednoduše nainstalovat. Druhým problémem je, že sledovací knihovny nejsou součástí hlavního repozitáře OpenCV. Tyto knihovny jsou součástí repozitáře *OpenCV_contib.* Z těchto důvodů je třeba si OpenCV 4.5.0 zkompilovat samostatně přímo ze zdrojových kódů.

Celý proces instalace včetně prerekvizit je popsán na [32]. Jediným odkloněním od tohoto postupu je povolení podpory Qt v kroku konfigurace příkazu *cmake*. Pro tento příkaz je třeba změnit parametr *WITH_QT* na hodnotu *ON*. Důležité je také zaznamenání parametru *CMAKE_INSTALL_PREFIX.* Hodnota tohoto parametru je součástí nastavení aplikace a je zapisována do konfiguračního souboru aplikace.

Upgrade knihoven zasáhl také do původní aplikace. Původní aplikace nelze spustit ihned po instalaci. Rozdíly mezi verzemi naštěstí nejsou příliš velké a oprava původní aplikace zahrnuje pouze úpravy názvů některých konstant. Konkrétně se jedná o změny u nastavení parametrů kamery:

```
CV_CAP_PROP_CONTRAST => CAP_PROP_CONTRAST

CV_CAP_PROP_EXPOSURE => CAP_PROP_EXPOSURE

CV_ CAP_PROP_BRIGHTNESS => CAP_PROP_BRIGHTNESS

CV_ CAP_PROP_SATURATION => CAP_PROP_SATURATION

CV_ CAP_PROP_GAIN => CAP_PROP_GAIN
```

Reprezentace barev:

```
CV_COLOR_BGR2RGB => COLOR_BGR2RGB
```

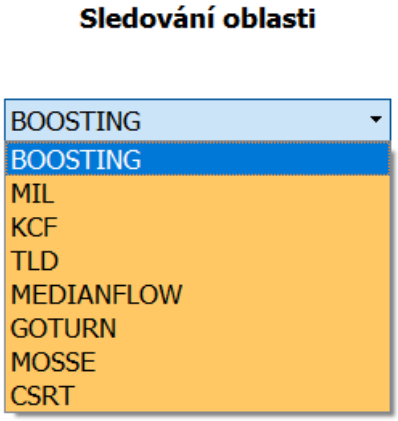
Typ vykreslování čar:

```
CV_AA => LINE_AA
```

Po provedení těchto změn funguje původní aplikace bezchybně na nových knihovnách.

7.2 Grafické prostředí

Hlavní změnou je přidání další záložky do seznamu v pravé části obrazovky. Tato záložka obsahuje dva ovládací prvky. Prvním ovládacím prvkem je seznam obsahující názvy všech 8 sledovacích algoritmů.

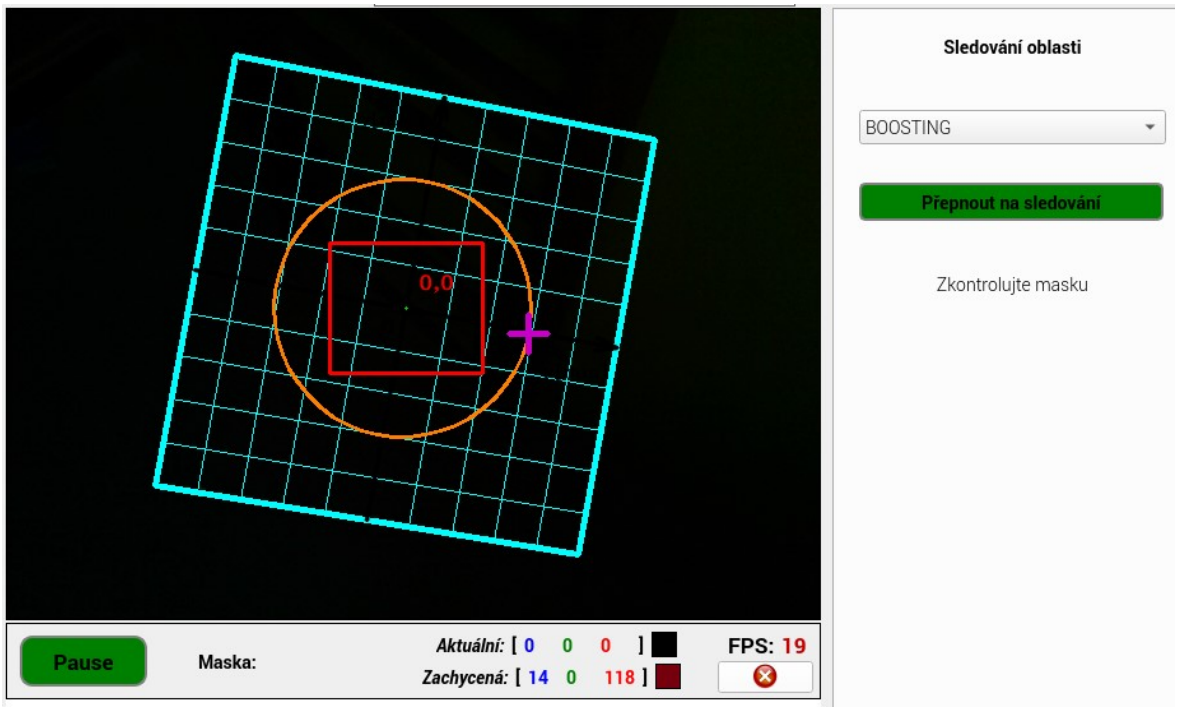


Obrázek 32 Přepínání algoritmů

Tento seznam umožňuje vybrat, který algoritmus bude spuštěn při kliknutí na tlačítko *Přepnout na sledování*, které je druhým ovládacím prvkem této záložky. Tlačítko mění svou podobu podle toho, zda právě běží sledování nebo ne.

Obrázek 33 Tlačítko spuštění sledování

Pro vybírání prvního ohraničujícího obdélníku je využita exitující funkcionalita výběru masky. V případě, že by nebyla vybrána maska a došlo ke kliknutí na tlačítko *Přepnout na sledování*, dojde k vypsání upozornění o neexistující masce.



Obrázek 34 Prázdná maska

7.3 Implementace

7.3.1 Spuštění aplikace

Při spuštění aplikace dojde k nastavení vzhledu tlačítka a skrytí chybové hlášky:

```
1.  SetTrackingSwitchText(worker->GetTrackingSwitch());

2.  ui->TrackingErrLbl->setVisible(false);
```

Getter a Setter přepínače sledování:

```
1.  bool CameraWorker::GetTrackingSwitch()
2.  {
3.      return TrackingEnabled;
4.  }
5.  void CameraWorker::SetTrackingSwitch(bool clicked)
6.  {
7.      TrackingEnabled = clicked;
8.  }
```

Metoda nastavující vzhled tlačítka:

```
1.  void MainWindow::SetTrackingSwitchText(bool val)
2.  {
3.      if(val) {
4.          ui->TrackingBtn->setText("Vypnout sledování");
5.          ui->TrackingBtn->setStyleSheet("background-color: rgb(200, 0, 0); color:
    rgb(0, 0, 0); border:2px solid rgb(128, 128, 128); border-radius: 5px");
6.      }
7.      else {
8.          ui->TrackingBtn->setText("Přepnout na sledování");
9.          ui->TrackingBtn->setStyleSheet("background-color: rgb(0, 128, 0); color:
    rgb(0, 0, 0); border:2px solid rgb(128, 128, 128); border-radius: 5px");
10.     }
11. }
```

7.3.2 Přepnutí metod

Při kliknutí na tlačítko pro přepnutí sledování je volána metoda:

```
1.  void MainWindow::on_TrackingBtn_clicked()
2.  {
3.      {
4.          ui->TrackingErrLbl->setVisible(false);
5.          tmp1b = !worker->GetTrackingSwitch();
6.          if(tmp1b){
7.              //pokud jsem nesledoval tak inicializuji
8.              bool ok = worker->InitializeTracker(ui->TrackingDDL->currentText());
9.              if(ok){
10.                 worker->SetTrackingSwitch(tmp1b);
11.                 SetTrackingSwitchText(tmp1b);
12.
13.                 ui->tabWidget->setTabEnabled(1,false);
14.                 ui->tabWidget->setTabEnabled(2,false);
15.                 ui->tabWidget->setTabEnabled(3,false);
16.             }else{
17.                 ui->TrackingErrLbl->setVisible(true);
18.             }
19.         }else{
20.             //pokud sleduji tak sledovani vypnu
21.             worker->SetTrackingSwitch(tmp1b);
22.             SetTrackingSwitchText(tmp1b);
23.             ui->tabWidget->setTabEnabled(1,true);
24.             ui->tabWidget->setTabEnabled(2,true);
25.             ui->tabWidget->setTabEnabled(3,true);
26.         }
27.     }
28. }
```

Metoda se podle toho, zda je zapnuto sledování rozhoduje, zda provede inicializaci nového trackeru, nebo vypnutí sledování. Také vypíná/zapíná některé jiné záložky. Tyto záložky zobrazují data, která jsou generována původním algoritmem. Jelikož je ovšem tento algoritmus vypnut, při přepnutí na tyto záložky by došlo k chybě.

Inicializace trackeru:

```
1.  bool CameraWorker::InitializeTracker(QString trackerName){
2.      if (trackerName == "BOOSTING")
3.          tracker = TrackerBoosting::create();
4.      if (trackerName == "MIL")
5.          tracker = TrackerMIL::create();
6.      if (trackerName == "KCF")
7.          tracker = TrackerKCF::create();
8.      if (trackerName == "TLD")
9.          tracker = TrackerTLD::create();
10.     if (trackerName == "MEDIANFLOW")
11.         tracker = TrackerMedianFlow::create();
12.     if (trackerName == "GOTURN")
13.         tracker = TrackerGOTURN::create();
14.     if (trackerName == "MOSSE")
15.         tracker = TrackerMOSSE::create();
16.     if (trackerName == "CSRT")
17.         tracker = TrackerCSRT::create();
18.
19.     Rect2d initRect(maskRect1.x,maskRect1.y, maskRect2.x-maskRect1.x,maskRect2.y-
    maskRect1.y);
20.     //ověření nenulovosti masky
21.     if(maskRect1.x == maskRect2.x || maskRect1.y == maskRect2.y ){
22.         return false;
23.     }
24.
25.     if((tracker->init(originalFrame, initRect)) == 1 ){
26.         return true;
27.     }else{
28.         return false;
29.     }
30. }
```

Metoda podle vstupního parametru vytvoří odpovídající tracker, který inicializuje obdélníkem uloženým v masce a současným snímkem. V případě, že by maska byla nulové velikosti, nebo došlo k jiné chybě, vrátí metoda chybu inicializace.

7.4 Zpracování snímků

Metoda *Process* je zodpovědná za kompletní zpracování obrazu. Vložil jsem do ní jednu podmínku, která podle přepínače rozhoduje, jaký algoritmus bude obraz zpracovávat.

```
1.    void CameraWorker::Process()
2.  {
3.    …
4.      if(TrackingEnabled){
5.          OCVTrancking();
6.      }else{
7.       //puvodni algoritmus
8.       …
9.      }
10. }
```

Samotné sledování poté probíhá v metodě *OCVTracking*, která provedete krok sledování a podle výsledku buď aktualizuje nalezenou pozici kuličky a aktualizuje masku, nebo zahlásí chybu detekce.

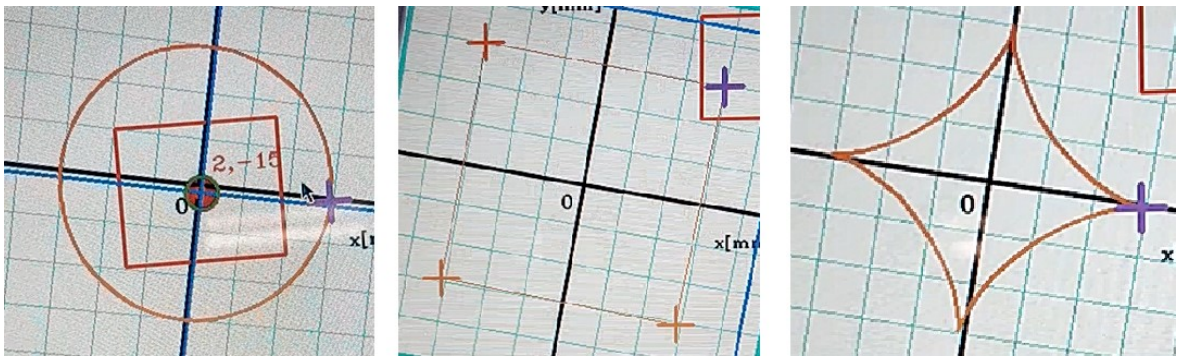
```
1.  void CameraWorker::OCVTrancking()
2.  {
3.      //provedení kroku sledování
4.      bool ok = tracker->update(originalFrame, lastLocation);
5.      if(ok){
6.          objectDetected = true;
7.          //nastavení nalezené pozice, jako stredu obdelniku
8.          Point2f(centerOfMass.x + rect1Act.x, centerOfMass.y + rect1Act.y);
9.          centerOfMass = Point(round((2*lastLocation.x +
    lastLocation.width)/2),round((2*lastLocation.y + lastLocation.height)/2));
10.         //aktualizace masky GUI
11.         SetMaskRect1(QPoint(lastLocation.x,lastLocation.y));
12.
    SetMaskRect2(QPoint(lastLocation.x+lastLocation.width,lastLocation.y+lastLocation.heig
    ht));
13.
14.     }else{
15.         NoObjectDetected();
16.     }
17. }
18.
```

Pracovní rozdělení obrazu je v aplikaci rozděleno na *Plocha roviny* > *Plocha zájmu* > *Maska. Plocha zájmu* nesmí opustit *Plochu roviny* a *Maska* nesmí opustit *Plochu zájmu*. Tato omezení jsou stanovená pro původní algoritmus u kterého nevadí, když se kulička nenachází uprostřed plochy zájmu a maska není využívána vůbec. Tato omezení však jsou nežádoucí a přináší problémy při využití sledování a to takové, že při přiblížení kuličky k hraně plošiny dochází k posunům masky. Tato omezení jsou v případě, že je zapnuto sledování, vypnuta.

```
1.  void CameraWorker::Tracking()
2.  {
3.    …
4.              if(!TrackingEnabled){
5.                  CheckAreaBorderCornerEnable();
6.              }
7.    …
8.  }
9.
```

8 VYHODNOCENÍ VÝSLEDKŮ

Pro srovnávání výkonu algoritmů jsem využil vestavěných funkcí systému. Součástí systému je program, který umožňuje řízení kuličky po požadované dráze. Na výběr je ze 3 drah ve tvaru kružnice, čtverce a asteroidy se 4 vrcholy.



Obrázek 35 Dráhy kuličky

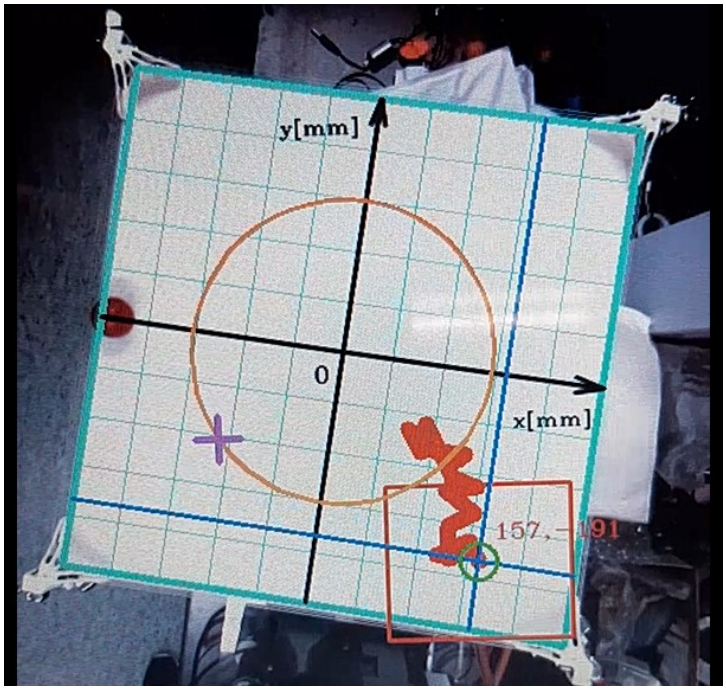
Na obrázku je vidět fialový kříž, který vyznačuje žádanou polohu kuličky a světlejší červenou barvou je vyznačena dráha, kterou bude bod periodicky opisovat. Úkolem programu je ovládání roviny tak, aby kulička sledovala fialový kříž. V případě že je dráha čtvercová dochází k okamžitému přepínání mezi vrcholy čtverce, nikoliv k postupnému posunu.

Jako srovnávací kritérium jsem zvolil směrodatnou odchylku vzdálenosti bodu, kde je detekována kulička od požadovaného bodu. Tato hodnota mi ukáže kvalitu regulace roviny, což přímo odpovídá kvalitě zpracování obrazu. Jelikož je implementováno všech 8 sledovacích algoritmů, je vyhodnocení rozděleno do dvou částí. V první části diskutuji, zda je algoritmus vůbec použitelný, a v druhé části poté provádím samotné srovnání zbylých algoritmů.

8.1 První sada testů

V této části je testováno všech 8 algoritmů. Jelikož ne všechny algoritmy dosahují požadované rychlosti a přesnosti potřebné pro správnou funkci systému, dojde k následnému vyřazení těch nedostačujících, ať už z jednoho, nebo druhého důvodu. V následujících obrázcích je červeně značena trajektorie sledovaného bodu. Testování je prováděno pouze na kružnici.

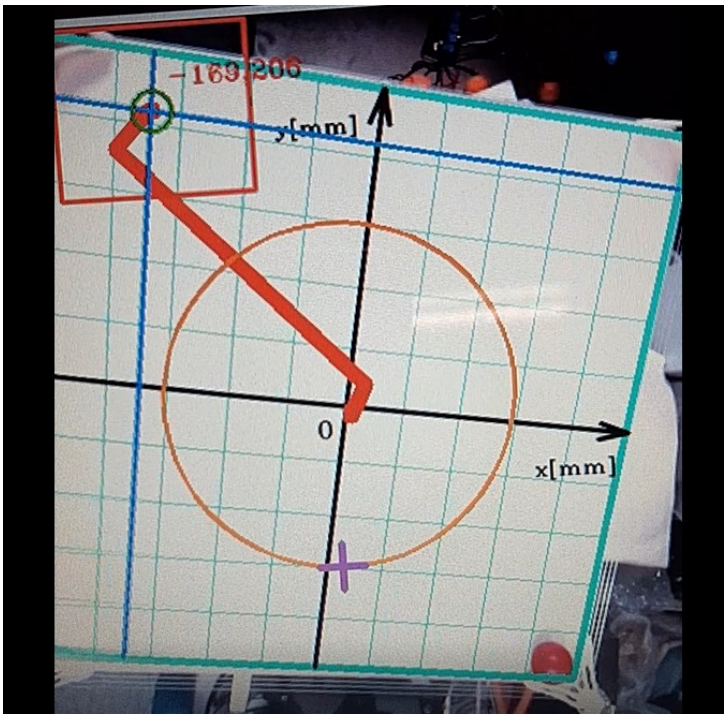
8.1.1 Boosting



Obrázek 36 Test Boosting

Algoritmus okamžitě ztrácí kuličku a sleduje náhodný bod v pozadí. Chyba detekce hlášena není. Rychlost se pohybuje na ~8 FPS. Tento algoritmus vhodný není a je proto vyřazen.

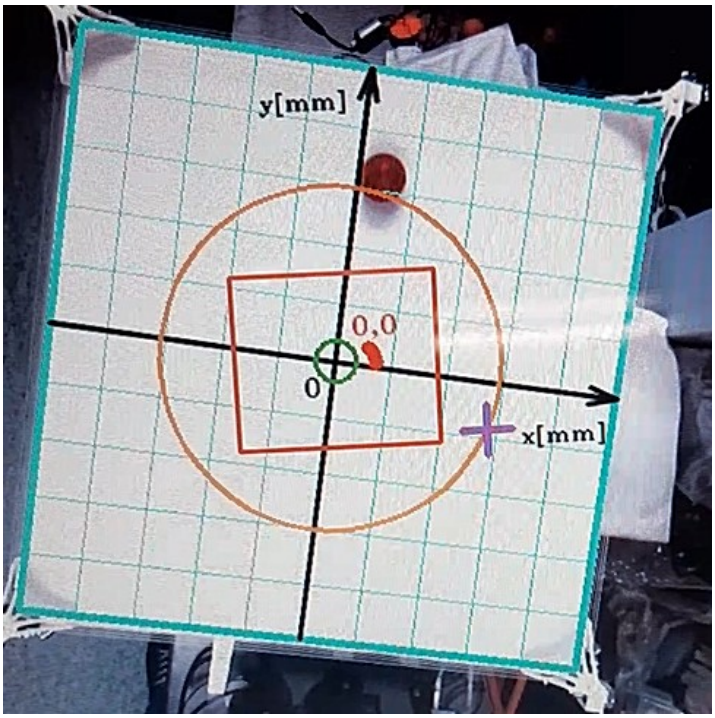
8.1.2 MIL



Obrázek 37 Test MIL

Algoritmus okamžitě ztrácí kuličku a sleduje náhodný bod pozadí. Chyba detekce hlášena není. Rychlost se pohybuje na ~4 FPS. Tento algoritmus vhodný není a je proto vyřazen.

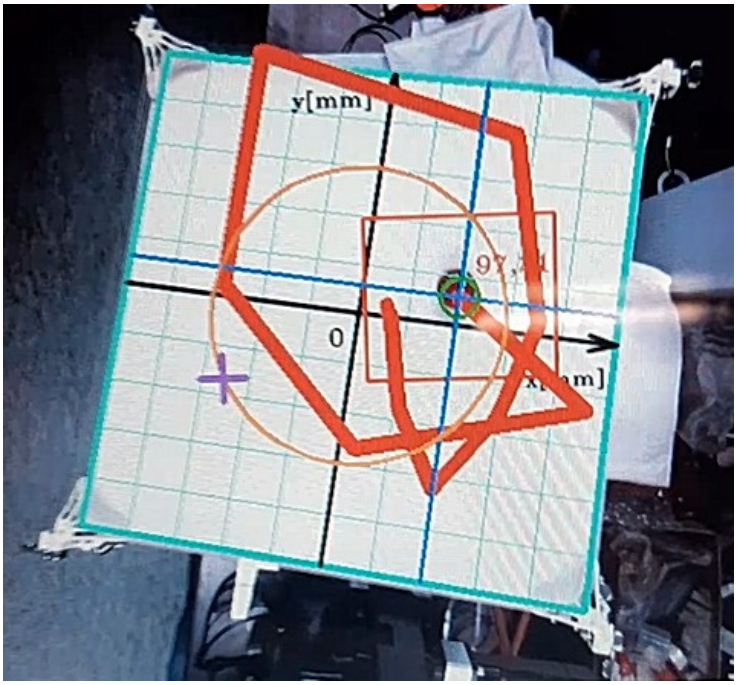
8.1.3 KCF



Obrázek 38 Test KCF

Algoritmus kuličku sleduje pouze pár snímků a poté ji ztrácí. Výhodou KCF, oproti předchozím algoritmům, je jeho skvělé hlášení chyb. Ztráta kuličky je hlášena okamžitě a předchází se tak chybnému řízení roviny. Rychlost se pohybuje na ~15 FPS. Přesto že by algoritmus dosahoval minima rychlosti pro potenciální fungování, neschopnost udržet kuličku jej vyřazuje.

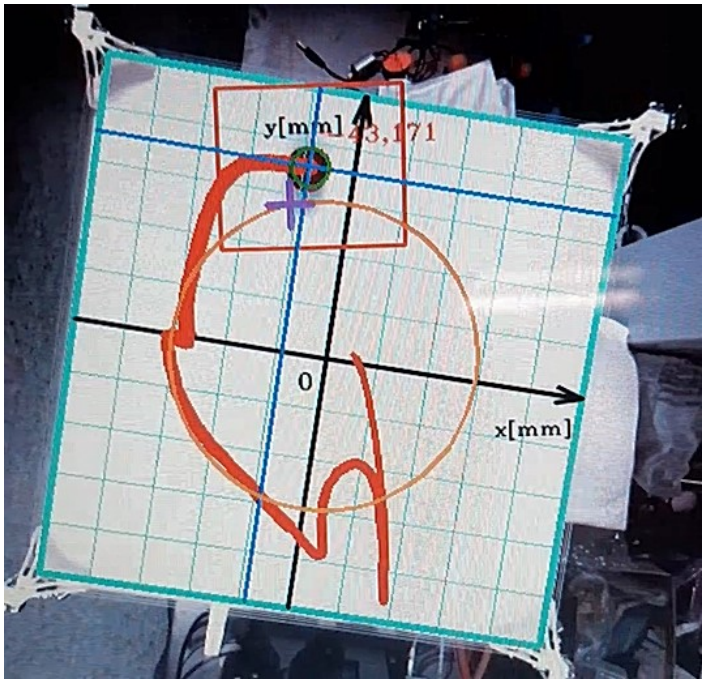
8.1.4 TLD



Obrázek 39 Test TLD

Algoritmus prokázal velkou robustnost vůči změně pozice kuličky. Přestože mezi snímky kulička změnila svou pozici až o půl roviny, algoritmus ji dokázal stále sledovat. Tyto velké změny byly však zapříčiněny velice nízkou rychlostí, která se pohybovala ~2 FPS. Algoritmus je tedy nevhodný.

8.1.5 Median Flow



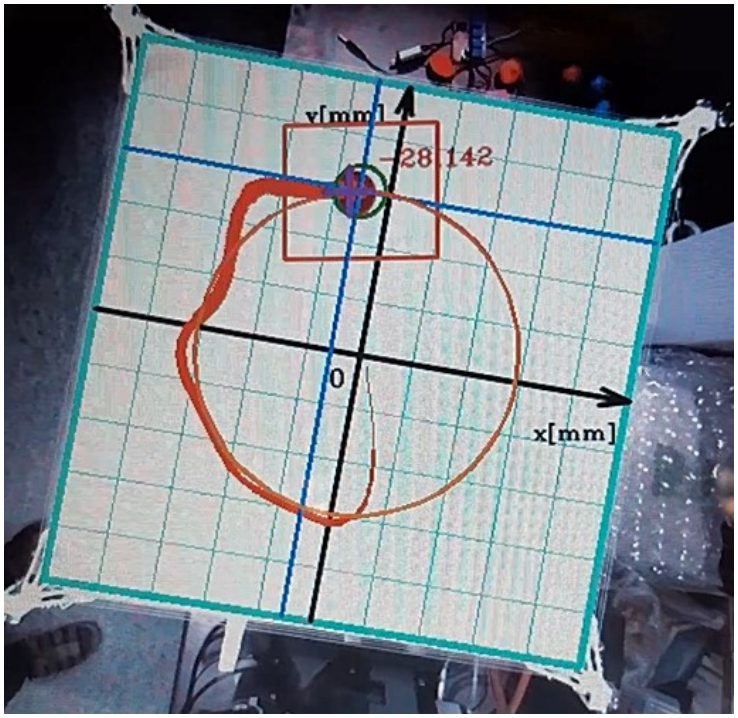
Obrázek 40 Test Median Flow

Algoritmus dokázal sledovat kuličku skvěle. Jeho rychlost se pohybovala na ~20 FPS. Tento algoritmus bude podrobněji otestován v druhém kole.

8.1.6 Goturn

Jak jsem již zmínil v kapitole 2.4, jako jediný využívá tento algoritmus offline trénování. To znamená, že při jeho spuštění, je třeba nahrát tato natrénovaná data do paměti, aby s nimi mohl algoritmus pracovat. Tato data mají velikost asi 500 MB a to se zdá být pro 1GB RAM RP příliš. Při pokusu o spuštění algoritmu tak dojde k pádu aplikace. Algoritmus je proto vyřazen.

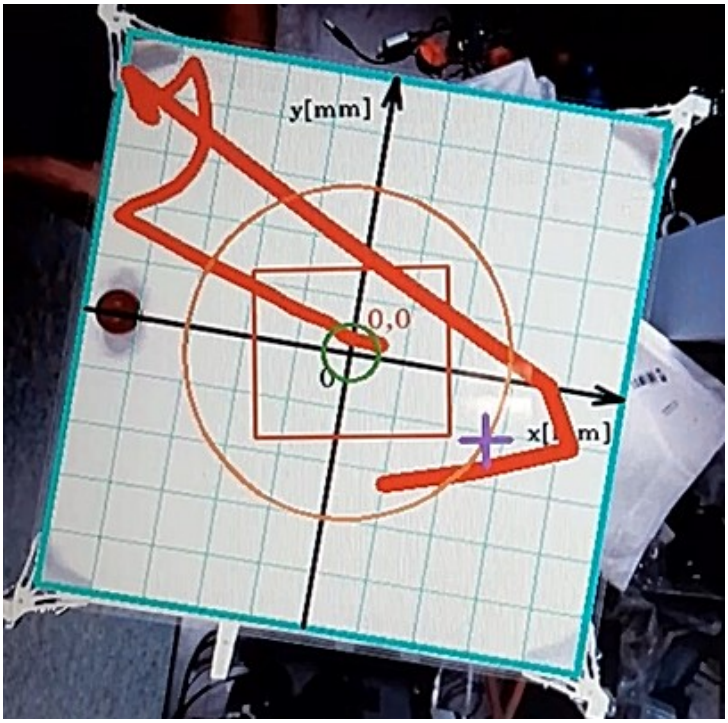
8.1.7 Mosse



Obrázek 41 Test Mosse

Algoritmus dokázal sledovat kuličku skvěle, i tak však docházelo k lehkému odsunu kuličky ze středu sledované oblasti. Jeho rychlost se pohybovala na ~30 FPS. Tento algoritmus bude podrobněji otestován v druhém kole.

8.1.8 CSRT



Obrázek 42 Test CSRT

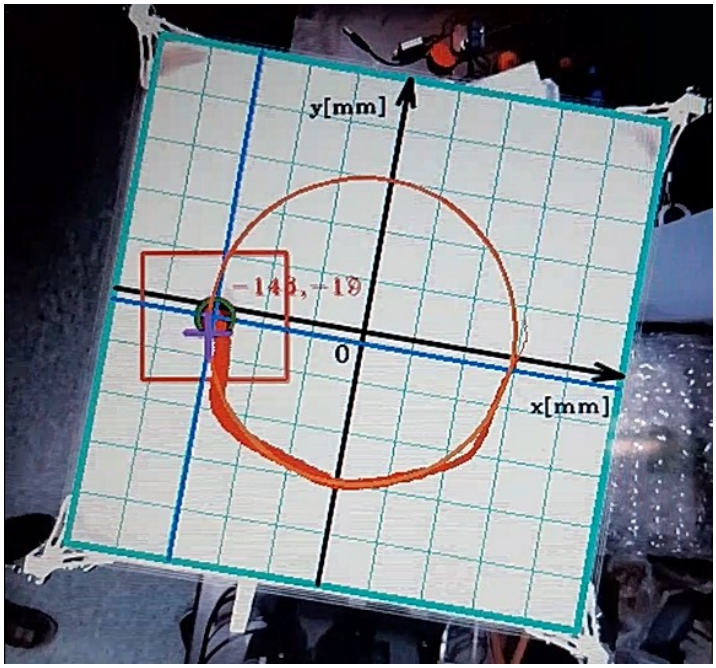
Problémem tohoto algoritmu je opět jeho rychlost, která se pohybuje na ~3 FPS. Přestože je jeho přesnost velice dobrá, při takovéto rychlosti dojde po pár větších pohybech ke ztrátě kuličky a k nahlášení chyby sledování. Algoritmus je tedy vyřazen.

8.2 Druhá sada testů

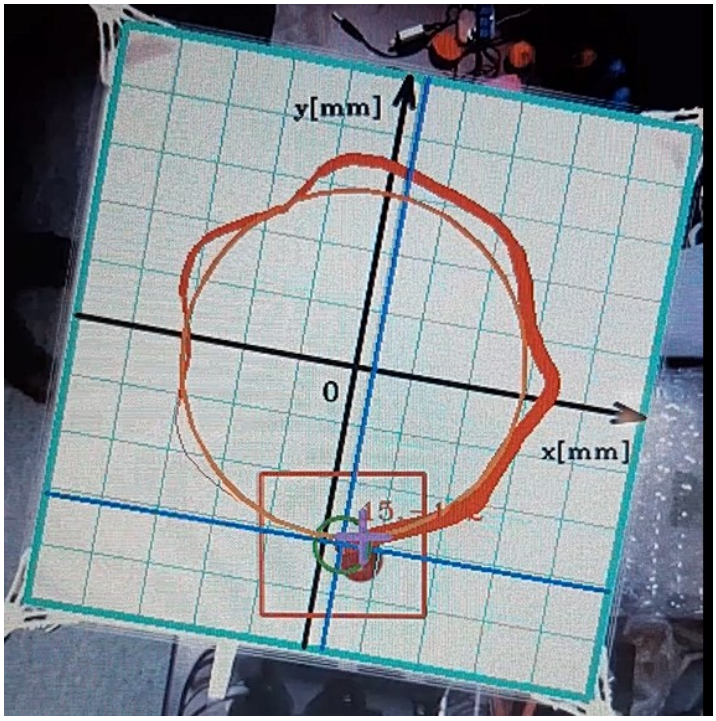
Na základě výsledků z prvního kola, bylo vyřazeno 6 z 8 implementovaných algoritmů. Zbylé algoritmy Median Flow a MOSSE srovnám s originálním algoritmem (dále jen OG) na základě směrodatné odchylky vzdáleností od požadované dráhy. Jelikož světelné podmínky nejsou v reálném světe vždy naprosto konzistentní, porovnám také vlivy změny světelných podmínek na jednotlivé algoritmy. Jako posledním se podívám na vliv kuliček jiných barev.

8.2.1 Kružnice

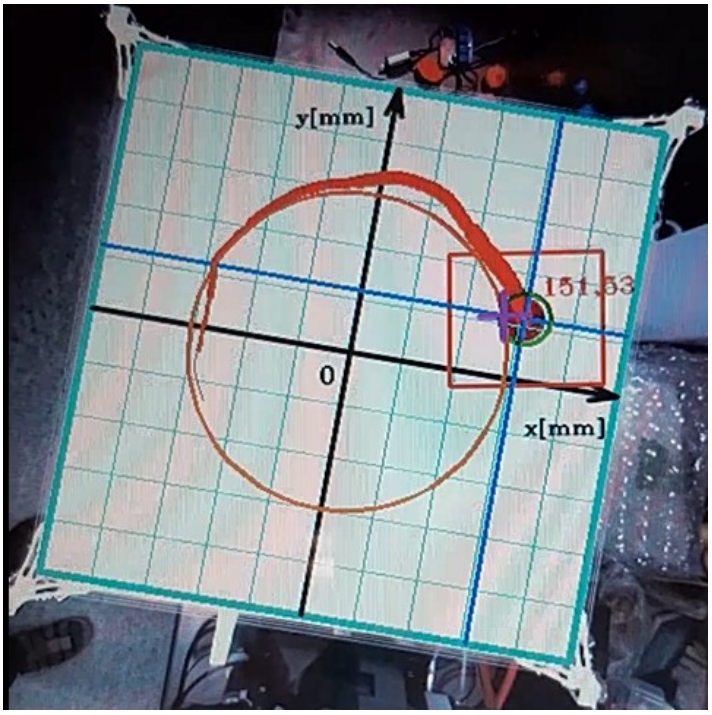
Nejprve se podívám na dráhu kružnice. Měření probíhalo u všech algoritmů po dobu 30 sekund. Za tuto dobu stihla kulička objet celou kružnici asi 2krát.



Obrázek 43 OG na kružnici



Obrázek 44 Median Flow na kružici



Obrázek 45 MOSSE na kružnici

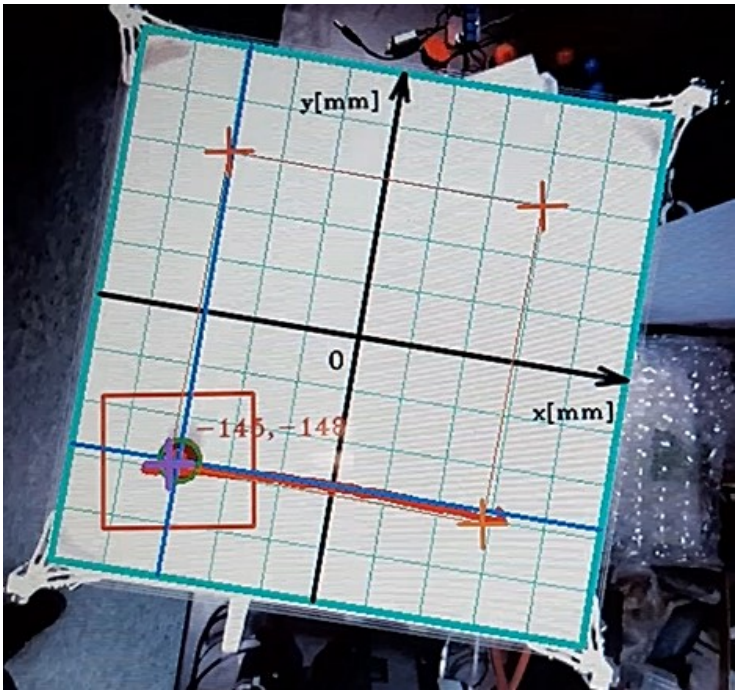
Tabulka 2 Odchylka na kružnici

	OG	Median Flow	MOSSE
σ (pixely)	7,07	13,94	10,71

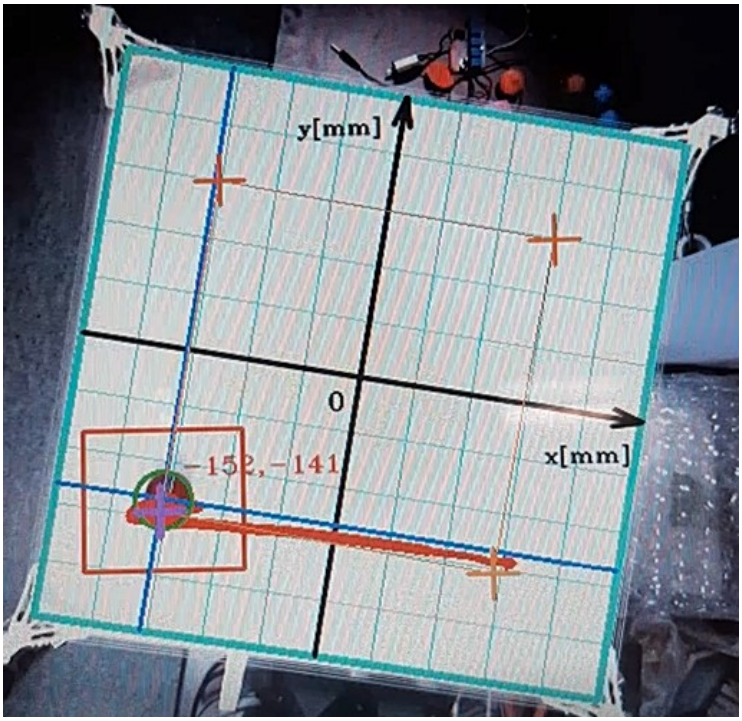
Všechny tři algoritmy dokázaly kuličku udržet na kružnici. Nejlépe na tom byl OG. U Median Flow je vidět téměř dvojnásobná odchylka v porovnání s OG. Důvodem je posun kuličky ze středu sledované plochy, jak je vidět na obrázku 41. Nevycentrovanost kuličky způsobí, že systém nemá přesnou pozici kuličky tak, jak by potřeboval a dochází proto k nepřesnému řízení. MOSSE také neměl kuličku ideálně vycentrovanou, každopádně podal lepší výkon než Median Flow.

8.2.2 Čtverec

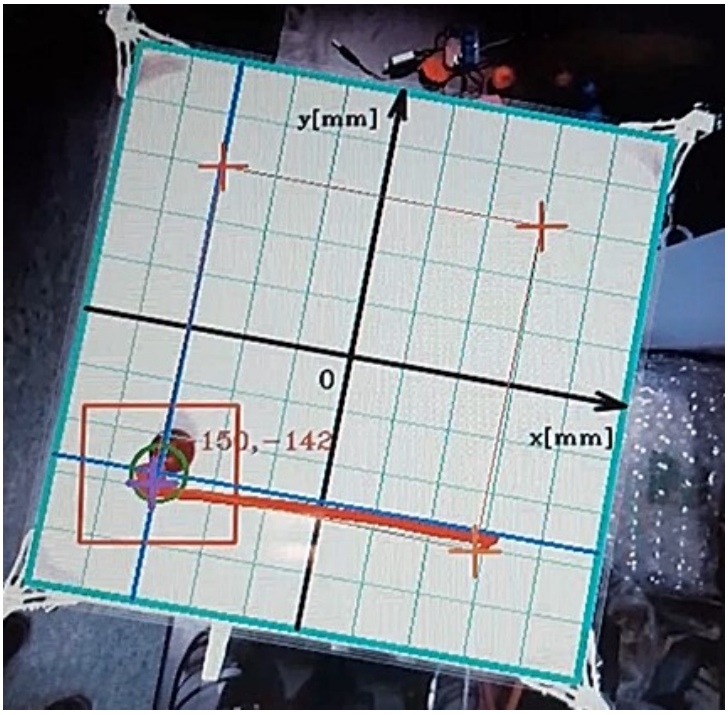
Měření probíhalo po dobu 1 minuty. Kulička začala v pravém horním rohu a na konci času objela zbylé 3 vrcholy a vrátila se do původní pozice.



Obrázek 46 OG na čtverci



Obrázek 47 Mediann Flow na čtverci



Obrázek 48 MOSSE na čtverci

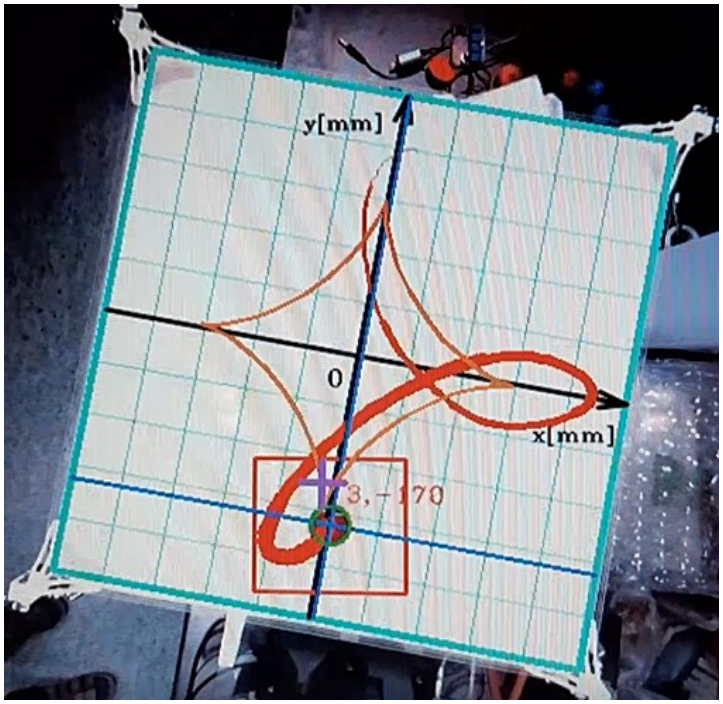
Tabulka 3 Odchylka na čtverci

	OG	Median Flow	MOSSE
σ (pixely)	39,94	42,98	49,15

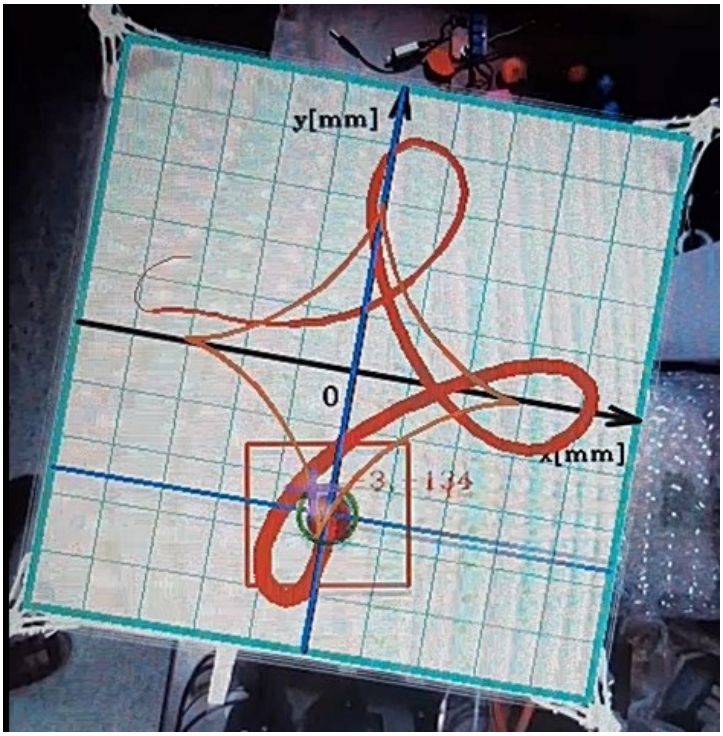
Odchylka je v této situaci mnohem větší, jelikož dochází ke skokovým změnám požadované polohy a kulička poté rychle přejíždí do druhého bodu, kde se pomalu usadí. Všechny algoritmy opět dokázaly kuličku správně navigovat. OG a Median Flow podaly téměř totožný výkon. MOSSE bohužel ztratil přesnou polohu kuličky a v důsledku toho oproti zbylým algoritmům trochu ztrácí. V této situaci však není chyba tak výrazná, jelikož je většina odchylky nasbírána na cestě mezi jednotlivými body.

8.2.3 Asteroida

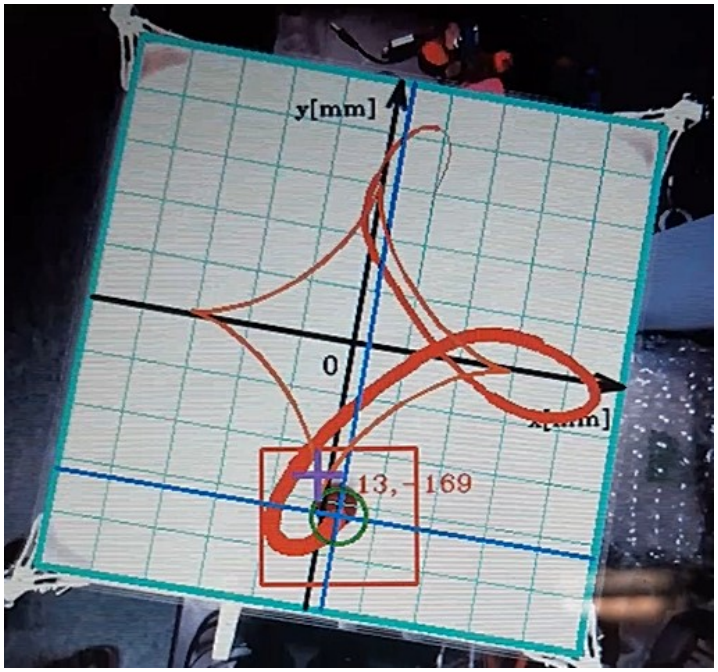
U této dráhy se výrazně projevuje hybnost kuličky. Přestože má dráha vrcholy zakončeny ostrou špicí, kulička kolem těchto bodů provádí velké smyčky a samotné požadované dráhy se téměř nedotkne. Měření probíhalo po dobu jedné minuty, kdy za tuto dobu kulička objede dráhu asi 5krát.



Obrázek 49 OG na asteroidě



Obrázek 50 Median Flow na asteroidě



Obrázek 51 MOSSE na asteroidě

Tabulka 4 Odchylka na asteroidě

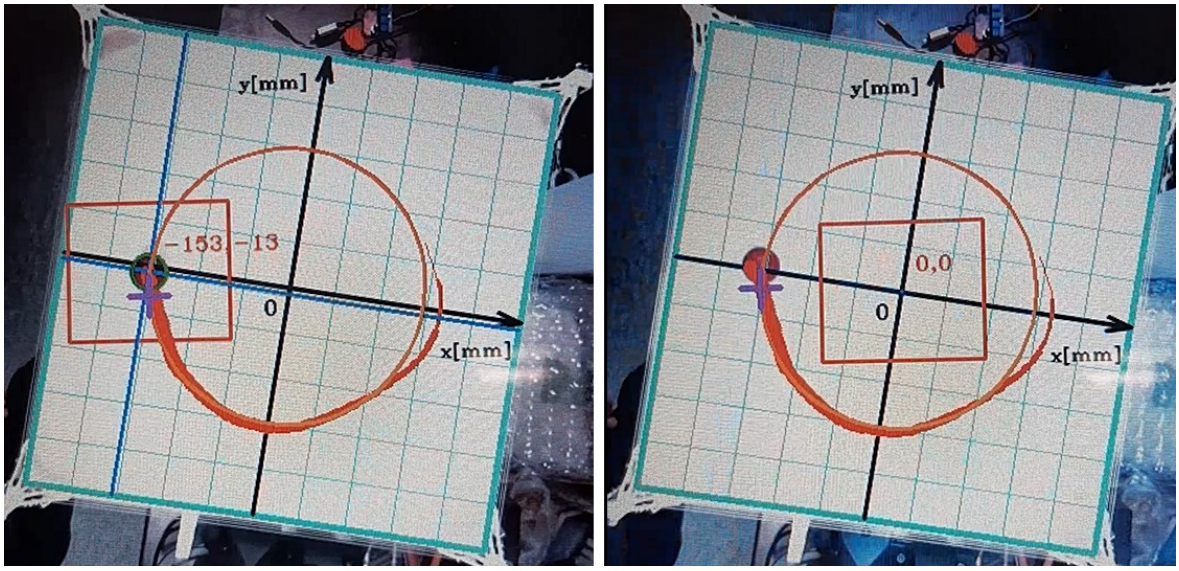
	OG	Median Flow	MOSSE
σ (pixely)	20,12	19,97	21,31

U této dráhy podaly všechny algoritmy téměř totožný výkon. Jelikož se zde kulička pohybuje mimo požadovanou dráhu prakticky pořád, dochází ke smazání jakýchkoliv nedostatků způsobených odchýlením kuličky ze středu sledované oblasti.

8.2.4 Různé světelné podmínky

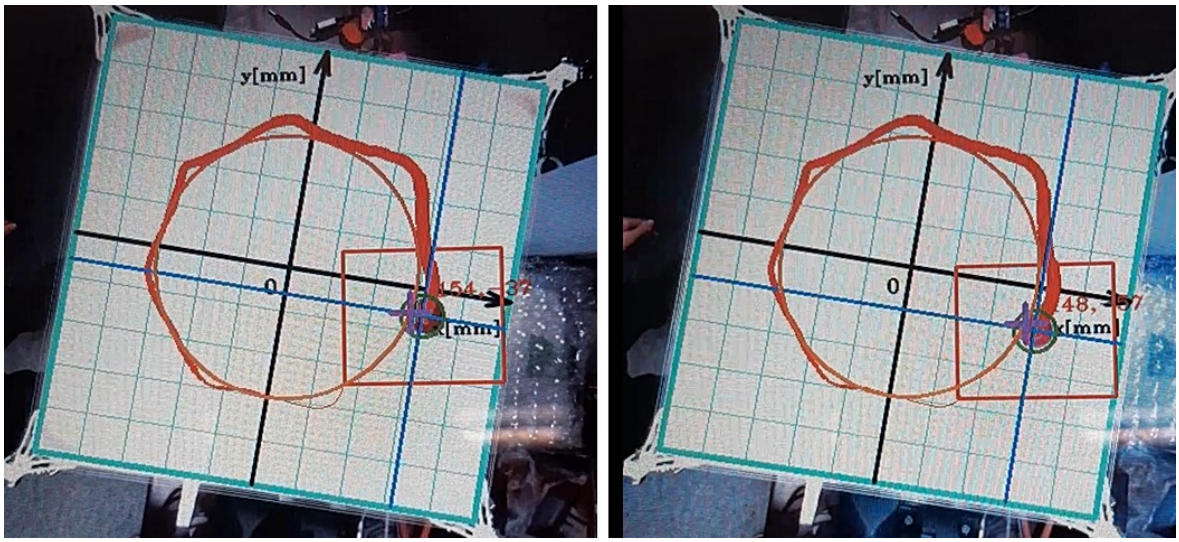
Změna světelných podmínek je něco, co je součástí každodenní reality a algoritmus zpracování obrazu by si s tímto měl umět poradit. Někdy jsou tyto změny předvídatelné, ale ve většině případů jsou to změny naprosto neznámé. Tato změna pak může mít velký vliv na OG, který se spoléhá na barvu kuličky.

Původní algoritmus je v tomto testu nastaven na barvu kuličky za výchozích světelných podmínek. Poté je po chvíli běhu řízení zapnuto světlo, které změní světelné podmínky natolik, že OG kuličku okamžitě ztrácí (Obrázek 52).



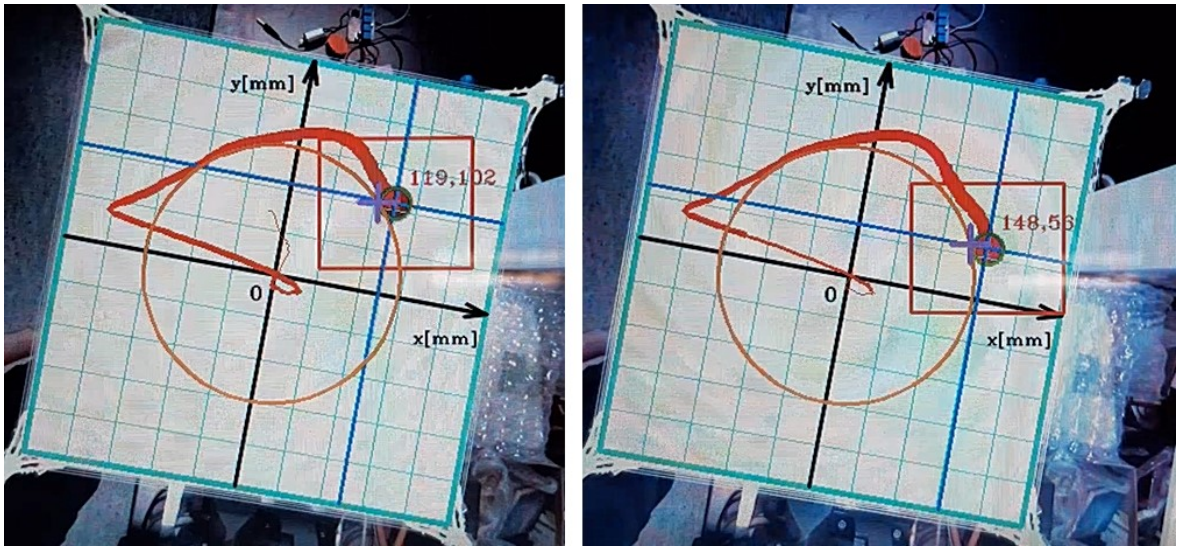
Obrázek 52 OG-ztráta kuličky při změně světla

Při využití algoritmů Median Flow nebo MOSSE se tento problém nevyskytuje. Oba algoritmy si dokážou se změnou světelných podmínek hravě poradit a pokračují dále ve sledování kuličky (Obr. 50).



Obrázek 53 Median Flow při změně světla

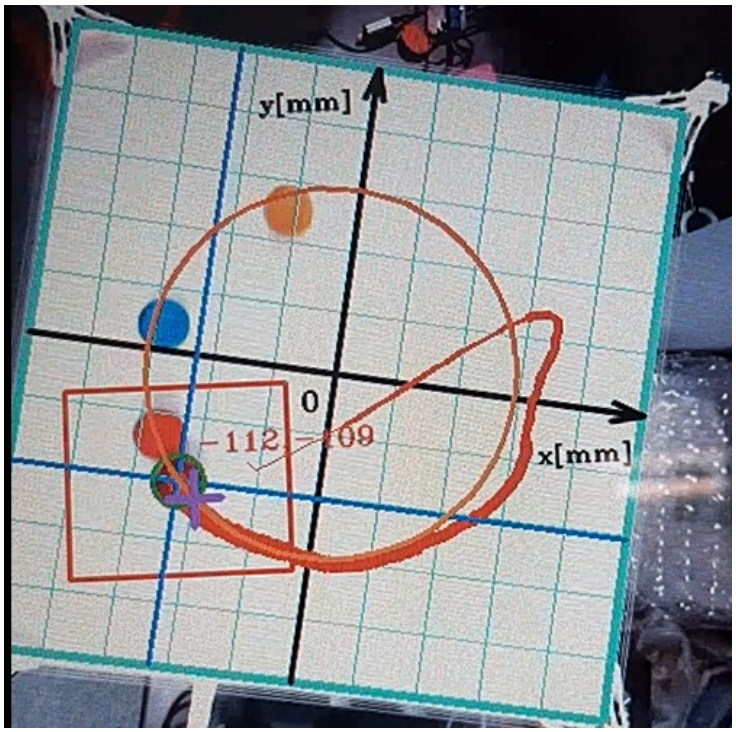
V situaci, kdy je změna světelných podmínek známá, je možné OG nastavit tak, aby si dokázal poradit i s takovouto změnou (Obrázek 54 OG-Nastavení pro známé podmínkyObrázek 54). V tento moment se ovšem stává problematická jiná situace, která bude popsána v kapitole 8.2.5.



Obrázek 54 OG-Nastavení pro známé podmínky

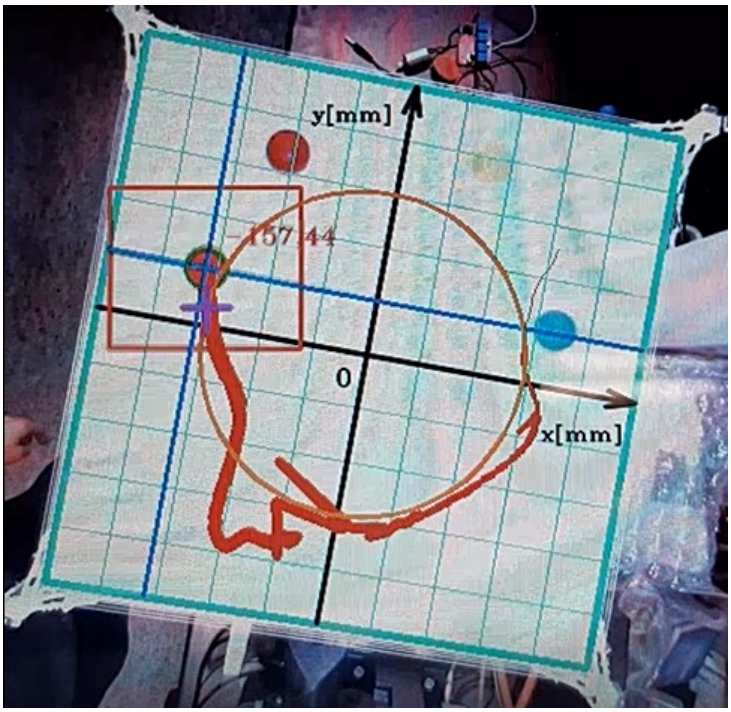
8.2.5 Další kuličky

Přidání dalších kuliček se může zdát jako očividný problém pro všechny algoritmy. V případě nedostatečného nastavení barvy pro OG může dojít ke sledování jiné kuličky podobné barvy. U sledovacích algoritmů by tvarová podoba mohla způsobit přeskočení na jinou kuličku v případě, že se kuličky přiblíží.



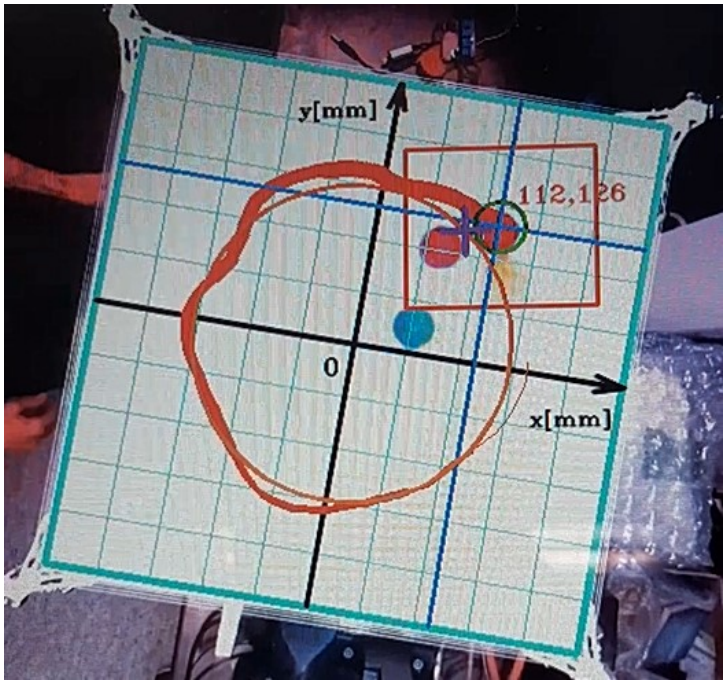
Obrázek 55 Přidání dalších kuliček

U OG tento problém opravdu spočívá ve správném nastavení barvy kuličky. V případě, že je toto nastavení správné i kulička velice podobné barvy algoritmus neovlivní, jak je vidět na obrázku (Obrázek 55). Problematickou se však stává situace, kdy se přidají také změny světelných podmínek. Je-li barva nastavena tak, aby byla kulička detekována za obou světelných podmínek, může nyní dojít k situaci, kdy toto nastavení zahrnuje i kuličku velice podobné barvy. Kulička, která tedy dříve detekována nebyla, je nyní detekována a kterou z těchto dvou kuliček bude algoritmus sledovat je otázkou náhody (Obrázek 56).



Obrázek 56 Sledování kuličky špatné barvy

Oba sledovací algoritmy jsou však proti tomuto odolné. Přidání více kuliček je nijak neovlivní a kuličky podobné barvy na ně také nemají vliv (Obrázek 57).



Obrázek 57 Median Flow s více kuličkami

8.3 Zhodnocení

Celkově bych neřekl, že ze srovnání vychází jeden z algoritmů jako jasný vítěz. Přesnost OG je nepopíratelná. Sledovací algoritmy v této oblasti trpí závislostí na online trénovaní, které může ve výsledku kuličku vynést ze středu sledované oblasti. Nepřesnost však není natolik velká, aby znemožňovala využití těchto algoritmů. Rychlost všech algoritmů je dostačující pro kvalitní řízení, i když Medin Flow zaostává asi o 10 FPS za ostatními. Situace, ve které mají sledovací algoritmy výrazně navrch je však taková, ve které dochází ke změnám světelných podmínek. V této situaci často OG havaruje úplně, kdežto sledovací algoritmy pokračují bez jakéhokoliv problému. Ideálním řešením by pak mohla být kombinace obou těchto přístupů, kdyby byl například OG občas využit pro vycentrování kuličky ve sledované oblasti.

ZÁVĚR

V této práci bylo mým úkolem vybrat alespoň jednu pokročilou metodu strojového vidění a vybranou metodu implementovat do již hotové aplikace reálného systému řízení. Po nastudování dokumentace a zjištění jaké možnosti nabízí využitá knihovna OpenCV, jsem tyto možnosti nejprve vyzkoušel na stolním počítači. Tímto jsem zjistil, které algoritmy nabízejí vlastnosti požadované reálným systémem řízení.

Z tohoto výběru vyšly nejlépe sledovací algoritmy, které jsem následně implementoval do aplikace s jednoduchou možností přepínání a se zachováním možnosti využití původního způsobu detekce. Při testování na reálném modelu se poté většina těchto algoritmů projevila jako nevhodná a do závěrečného srovnání byly zařazeny pouze implementace MOSSE a Median Flow.

Tyto algoritmy jsem srovnal s původním způsobem detekce a sledovací metody se ukázaly jako vhodná volba pro toto využití. V oblasti přesnosti sice zaostávaly za původním přístupem, v oblasti robustnosti se však ukázaly jako lepší volba. Tyto algoritmy by se daly nadále vylepšit s využitím jejich parametrizace. OpenCV umožňuje tyto metody parametrizovat a zajímavou možností by pak mohlo být hledání parametrů pro nejlepší fungování s daným problémem. Pozoruhodné by bylo také spojení původní metody s novými algoritmy, čímž by mohlo dojít ke vzájemnému vynulování se jejich nedostatků a dosažení ještě lepších výsledků.

SEZNAM POUŽITÉ LITERATURY

[1] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," Proceedings. International Conference on Image Processing, Rochester, NY, USA, 2002, pp. I-I, doi: 10.1109/ICIP.2002.1038171.

[2] BURGER, Wilhelm a Mark James BURGE. Principles of digital image processing: fundamental techniques. London: Springer, c2009. Undergraduate topics in computer science, 7592. ISBN 978-1-84800-191-9.

[3] PŘINOSIL, Jiří a Martin KROLIKOWSKI. Využití detektoru Viola-Jones pro lokalizaci obličeje a očí v barevných obrazech. Elekrorevue. 2008, 2008(31).

[4] RUSS, John C. a F. Brent NEAL. The image processing handbook. Seventh edition. Boca Raton, [2016]. ISBN 978-1-4987-4028-9.

[5] Histogram of Oriented Gradients explained using OpenCV [online]. [cit. 2021-04-09]. Dostupné z: https://learnopencv.com/histogram-of-oriented-gradients/

[6] CHRÁPEK, David. UČENÍ A DETEKCE OBJEKTŮ RŮZNÝCH TŘÍD V OBRAZE. Brno, 2012. Diplomová práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

[7] SIFT, SURF, MSER [online]. Plzeň, 2015 [cit. 2021-4-24]. Dostupné z: http://www.kky.zcu.cz/uploads/courses/mpv/04/materialy04.pdf

[8] BAY, Herbert, Tinne TUYTELAARS a Luc VAN GOOL. SURF: Speeded Up Robust Features [online]. Zurich [cit. 2021-4-24]. Dostupné z: https://people.ee.ethz.ch/~surf/eccv06.pdf

[9] GRABNER, Michael a Horst BISCHOF. Real-Time Tracking via On-line Boosting. Graz, 2006. Graz University of Technology.

[10] ROJAS, Raúl. AdaBoost and the Super Bowl of Classifiers A Tutorial Introduction to Adaptive Boosting. Berlin, 2009. Freie Universität Berlin.

[11] BABENKO, Boris, Ming-Hsuan YANG a Serge BELONGIE. Visual tracking with online Multiple Instance Learning. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition [online]. IEEE, 2009, 2009, s. 983-990 [cit. 2021-4-25]. ISBN 978-1-4244-3992-8. Dostupné z: doi:10.1109/CVPR.2009.5206737

[12] KALAL, Zdenek, Krystian MIKOLAJCZYK a Jiri MATAS. Tracking-Learning-Detection. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. 2010, 6(1).

[13] HELD, David, Sebastian THRUN a Silvio SAVARESE. Learning to Track at 100 FPS with Deep Regression Networks. Department of Computer Science Stanford University, 2016. Stanford University.

[14] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In Pattern Recognition (ICPR), 2010 20th International Conference on, pages 2756–2759. IEEE, 2010.

[15] VOJÍŘ, Tomáš. Short-Term Visual Object Tracking in Real-Time. Prague, 2017. Disertační práce. Faculty of the Electrical Engineering of the Czech Technical University in Prague. Vedoucí práce Prof. Ing. Jiří Matas, Ph.D.

[16] CHEN, Zhe, Zhibin HONG a Dacheng TAO. An Experimental Survey on Correlation Filter-based Tracking. CoRR, abs/1509.05520, 2015.

[17] BOLME, David S., J. Ross BEVERIDGE, Bruce A. DRAPER a Yui MAN LUI. Visual Object Tracking using Adaptive Correlation Filters. Fort Collins, 2010. Computer Science Department Colorado State University.

[18] HENRIQUES, Joao F., Rui CASEIRO, Pedro MARTINS a Jorge BATISTA. Exploiting the Circulant Structure of Tracking-by-detection with Kernels. 2012. Institute of Systems and Robotics, University of Coimbra.

[19] HENRIQUES, Joao F., Rui CASEIRO, Pedro MARTINS a Jorge BATISTA. High-Speed Tracking with Kernelized Correlation Filters. High-Speed Tracking with Kernelized Correlation Filters, arXiv:1404.7584v3.

[20] LUKEŽIČ, Alan, Tomáš VOJÍŘ, Luka ČEHOVIN ZAJC, Jiří MATAS a Matej KRISTAN. Discriminative Correlation Filter Tracker with Channel and Spatial Reliability. International Journal of Computer Vision [online]. 2018, 126(7), 671-688 [cit. 2021-4-24]. ISSN 0920-5691. Dostupné z: doi:10.1007/s11263-017-1061-3

[21] DAVIES, E. R. Computer and machine vision: theory, algorithms, practicalities. 4th ed. Waltham: Academic Press, 2012. ISBN 978-0-12-386908-1.

[22] BOSWELL, Dusti. Introduction to Support Vector Machines. 2002.

[23] Support vector machines (SVM) [online]. [cit. 2021-4-24]. Dostupné z: https://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf

[24] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. You Only Look Once: Unified, Real-Time Object Detection [online]. 2015 [cit. 2021-4-24]. Dostupné z: https://arxiv.org/abs/1506.02640v5

[25] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement [online]. 2018 [cit. 2021-4-24]. Dostupné z: https://arxiv.org/abs/1804.02767v1. University of Washington.

[26] BOCHKOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection [online]. 2020 [cit. 2021-4-24]. Dostupné z: https://arxiv.org/abs/2004.10934

[27] YOLO: Real-Time Object Detection. YOLO: Real-Time Object Detection [online]. [cit. 2021-04-05]. Dostupné z: https://pjreddie.com/darknet/yolo/

[28] Buy a Raspberry Pi 3 Model B+ – Raspberry Pi. Teach, Learn, and Make with Raspberry Pi [online]. [cit. 2021-04-05]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[29] OpenCV: cv::Tracker Class Reference. OpenCV documentation index [online]. [cit. 2021-04-05]. Dostupné z: https://docs.opencv.org/4.5.0/d0/d0a/classcv_1_1Tracker.html

[30] Simple Opencv tutorial for yolo darknet object detection in DNN module [online]. [cit. 2021-4-25]. Dostupné z: https://funvision.blogspot.com/2020/04/simple-opencv-tutorial-for-yolo-darknet.html

[31] OpenCV: samples/cpp/train_HOG.cpp [online]. [cit. 2021-4-25]. Dostupné z: https://docs.opencv.org/4.5.0/d0/df8/samples_2cpp_2train_HOG_8cpp-example.html

[32] Install OpenCV 4.5 on Raspberry Pi 4 - Q-engineering [online]. [cit. 2021-4-25]. Dostupné z: https://qengineering.eu/install-opencv-4.5-on-raspberry-pi-4.html

[33] 30 Minutes of Cars Driving By in 2009. YouTube [online]. [cit. 2021-5-6]. Dostupné z: https://www.youtube.com/watch?v=e_WBuBqS9h8&ab_channel=OnlyHDVideos

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

OpenCV	Open Source Computer Vision
GPIO	General-purpose input/output
CSI	Camera serial interface
RP	Raspberry Pie
HW	Hardware
YOLO	You only look once
FPS	Frames per second
GOTURN	Generic Object Tracking Using Regression Networks
TLD	Tracking Learning Detection
MOSSE	Minimum Output Sum of Squared Error
KCF	Kernelized Correlation Filters
MIL	Multiple Instance Learning
CSRT	Channel and Spatial reliability tracker
SVM	Support vector machine
SURF	Speeded Up Robust Features
OG	Originální algoritmus

SEZNAM OBRÁZKŮ

SEZNAM TABULEK