VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

GAMIFIKACE VIDEOZÁZNAMU ROBOTŮ SLEDUJÍCÍCH ČÁRU
GAMIFICATION OF LINE-FOLLOWER ROBOT VIDEORECORDINGSOUS

BAKALÁŘSKÁ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — Tomáš Soboňa
AUTHOR

VEDOUCÍ PRÁCE — Ing. Ilona Janáková, Ph.D.
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Tomáš Soboňa ***ID:*** 173745

Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Gamifikace vidozáznamu robotů sledujících čáru

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je automatické vyhodnocení videozáznamů pořízených během soutěže robotů v předmětu BPRP, ve které je úkolem projet autonomním robotem předem neznámou soutěžní trať v co nejkratším čase. Vstupem práce jsou záznamy zabírané z ptačí perspektivy, kde roboti jedou po černé čáře na bílém podkladu.

1. Seznamte se s danou úlohou včetně pravidel soutěže.
2. Seznamte se s otevřenou knihovnou pro zpracování obrazu OpenCV ve spojení s programovacím jazykem Python nebo C++.
3. Navrhněte robustní algoritmy pro: detekci hrací plochy (předpokládá se řešení geometrického zkreslení obrazu), detekci namalované čáry včetně případných překážek a detekci a trasování pohybu robotů. Pro detekci je možné využít definované značky umístěné na hrací ploše a na robotech.
4. Na základě předchozího bodu automaticky vyhodnoťte kvalitu sledování čáry jednotlivých robotů – ujetá vzdálenost na čáře od startu, čas jízdy.
5. Zvažte další možné výstupy vyhodnocení – ujetá celková trajektorie, statistiky, kritická místa dráhy atd.
6. Navržené algoritmy implementujte a řádně otestujte.
7. Zhodnoťte výsledky a stanovte omezující podmínky.

DOPORUČENÁ LITERATURA:

HLAVÁČ, V., ŠONKA, M.: Počítačové vidění. Praha: Grada, 1992, ISBN 80-85424-67-3.

SOLEM, Jan Erik. Programming computer vision with Python. Sebastopol, CA: O'Reilly, 2012. ISBN-13: 978-1449316549.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Ilona Janáková, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

Abstrakt

Táto bakalárska práca sa zaoberá spracovaním trajektórie vozidla sledujúceho čiernu čiaru na nemennom bielom pozadí a následným porovnávaním kvality prejdenia tejto dráhy jednotlivými vozidlami. V prvej časti práce je popísaná teória pre rozpoznanie dráhy, jednotlivých grafických tagov a aj nájdenie významných bodov v obraze a ich spárovanie s bodmi v inom obraze, načo sa využíva algoritmus SIFT. Druhá časť sa venuje popisu samotného programu, ktorý bol v tejto práci vytvorený, jeho triedam a metódam. Program je napísaný v jazyku Python a pre spracovanie obrazu sa využíva hlavne „open source-ová " knižnica OpenCV a ale aj v menšej miere knižnica NumPy. Nakoniec sú v práci zhrnuté dosiahnuté výsledky.

Kľúčové slová

Spracovanie obrazu, detekcia pohybu, pozorovanie trajektórie, OpenCV, Python, SIFT, NumPy

Abstract

This bacheor's thesis is concerned with the processing of a vehicle's trajectory following a black line located on an unchanging white background, and subsequently comparing of the quality of each vehicle's covered distance on this trajectory. The first part of the thesis describes the theory for recognizing the track, individual graphic tags, as well as detection of important points in the picture and their correlation with points in another picture, for which an algorithm SIFT is used. The second part deals with the description of the program, which was created as part of this work, its classes and methods. The program is written in Python, for image processing is used mainly an open source library OpenCV and, to a lesser extent, the NumPy library. Finally, the results are concluded at the end of the thesis.

Key words:

Image processing, motion detection, path tracking, OpenCV, Python, SIFT, NumPy

SOBOŇA, T. Gamifikace vidozáznamu robotů sledujících čáru. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. XY s. Vedoucí bakalářské práce Ing. Ilona Janáková, Ph.D..

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na téma Gamifikace vidozáznamu robotů sledujících čáru vypracoval samostatne pod vedením vedúceho bakalárskej práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, hlavne som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovení § 11 a nasledujúcich zo zákona č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovení časti druhej, hlavy VI. diel 4 Trestného zákonníku č. 40/2009 Sb.

V Brne dňa ....

(podpis autora)

Obsah

Zoznam obrázkov

Úvod

Oko je pre človeka jeden z najdôležitejších zmyslových orgánov, cez ktorý získava veľké množstvo informácii. Tak ako človek používa oko pre získanie informácii tak sa aj vo vedeckej disciplíne s názvom počítačové videnie využíva pre získavanie informácii kamera. Obraz z kamery sa pomocou rôznych algoritmov spracuje a interpretuje. Algoritmy pre spracovanie obrazu dnes vďaka rýchlemu rozvoju výpočetnej techniky dokážu riešiť aj veľmi náročné problémy ako napríklad rozpoznanie rukou písaného textu, nájdenie a identifikovanie tváre, sledovanie a predikcia pohybu, vytvorenie 3D modelu objektu, navádzanie autonómneho vozidla na základe počítačového videnia.

Táto bakalárska práca sa zaoberá popisom teórie a aj vytvorením programu pre automatické spracovanie trajektórie pohybujúceho sa vozidla. Následným parametrizovaním trajektórii jednotlivých vozidiel a aj ich vzájomným porovnaním.

V prvej kapitole je popísaná postupne teória pre spracovanie obrazu, prvé sú metódy preprocesingu ako jasové transformácie, filtrácia, detekcia hrán v obraze a rohov, a v ďalšej podkapitole je rozobraná detekcia pohybu pomocou diferenciálnych metód a ďalej aj algoritmus pre rozpoznávanie descriptorov. V druhej kapitole sú popísané nástroje programovací jazyk Python a knižnica OpenCV, ktoré sú použité pri implementácií programu. Ďalšia kapitola sa venuje popisu dôležitých objektov vo videu ako aj tímov ale aj samotného program pre spracovanie trajektórie hlavne jeho triedy a metódy. A nakoniec sú prezentované výsledky a limitujúce podmienky.

1 Spracovanie obrazu

1.1 Predspracovanie digitálneho obrazu

Predspracovanie obrazu je spoločný názov pre operácie, ktoré zjednodušia ďalšiu prácu s obrazom.[3] „Z hľadiska veľkosti okolia prispievajúceho k transformácii vstupného bodu na výstupný poznáme nasledovné typy operácií:

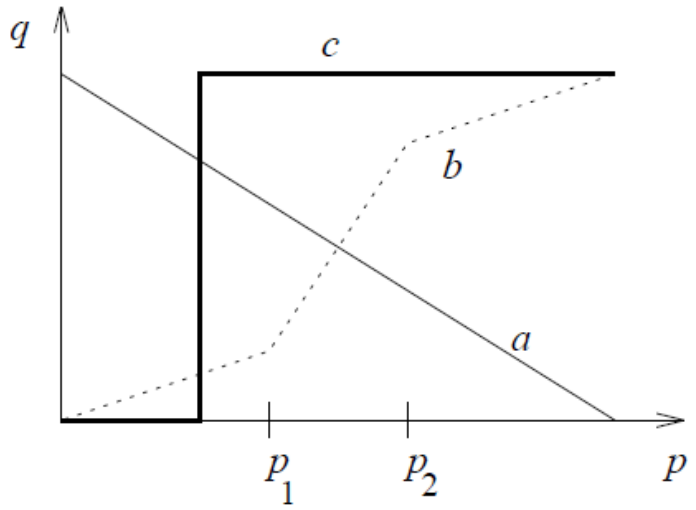
1. bodové (výstupný bod je transformáciu hodnoty jediného vstupného bodu)
2. lokálne (výstupný bod je tvorený hodnotou vstupného bodu a jeho lokálneho okolia)
3. globálne (na hodnote výstupného bodu sa podieľajú všetky body vstupného obrazu).“ [4]

Ďalej sú v podkapitolách rozobrané konkrétne metódy predspracovania, rozobrané budú však len metódy, ktoré sa aj použijú v záverečnej práci.

1.1.1 Transformácia jasovej stupnice

Transformácia jasovej stupnice je rovnaká pre všetky pixely v obraze. Jej realizácia je pomerne jednoduchá pomocou vyhľadávacích tabuliek tzv. look up table. Počet prvkov tabuľky je rovný počtu jasových úrovní v spracovávanom obraze. Transformované hodnoty jasu sú jednotlivými prvkami tabuľky. Tento spôsob je využívaný hlavne pre jeho rýchlosť. Tento princíp sa používa pre šedotónové obrazy, pre farebné obrazy sa využíva podobný princíp. Avšak, vyhľadávacia tabuľka je pre každú z farebných zložiek R, G, B zvlášť. Najbežnejšími transformáciami sú negatív, zmena kontrastu alebo jasu, prahovanie do jedného alebo viacero prahov. [2]

Pri niektorých scénach potrebujeme oddeliť konkrétny objekt od pozadia. Pre vytvorenie extrémne veľkého kontrastu využívame rozdelenie všetkých jasových úrovní obrazu do dvoch úrovní, poprípade viacerých úrovní na základe predom zvoleného prahu. Táto transformácia sa nazýva prahovanie. [3]



Obr. 1.1: Základné jasové úpravy: a)negatív, b)zvýšenie kontrastu medzi p1 a p2, c)prahovanie do 2 úrovní min. a max. možnú hodnotu [2]

Rozloženie jasu v obraze najlepšie popíše charakteristika - Histogram jasových úrovní. V prípade šedotónového 8-bitového obrazu hodnoty 0 až 255 tvoria x-ovú os a počet výskytov jasových hodnôt v obraze tvorí y-ovú os. Pre zvýšenie kontrastu obrazu sa používa operácia ekvalizácia histogramu.

Pre výpočet ekvalizovaného histogramu H_c šedotónového obrazu s rozmermi M x N a počtom jasových úrovní G potrebujeme najprv kumulatívny histogram. Ten vypočítame tak, že od prvého prvku každému ďalšiemu prvku pôvodného histogramu pripočítame aj hodnotu predošlého prvku kumulatívneho histogramu. Až nakoniec posledný prvok je sumou všetkých predošlých prvkov pôvodného histogramu. V ďalšom kroku vynásobíme každý prvok ekvalizovaného histogramu prevodnou funkciou

$$T[p] = round\left(\frac{G-1}{N*M} H_c[p]\right). \quad (1.1)$$

Tento krok vedie k zostrojeniu look up tabuľky. Nakoniec len aplikujeme look up tabuľku na pôvodný obraz a vznikne ekvalizovaný obraz. [5]

1.2 Filtrovanie šumu v obraze

Spracovávaný obraz často obsahuje okrem hľadanej štruktúry, ktorú chceme detegovať aj istú úroveň náhodného šumu, ktorý potrebujeme odstrániť. Hlavná časť energie bežného obrazu býva najčastejšie koncentrovaná v nízkofrekvenčných častiach.

Šum a rôzne nechcené artefakty sú zase často zastúpené vo vysokofrekvenčných častiach obrazu. Zredukovaním vysokofrekvenčných častí pomocou filtru dolná prepusť sa výrazne zníži úroveň šumu v obraze ale zároveň sa môže znehodnotiť aj istá časť signálu, ktorá bola na vysokých frekvenciách, rozmazanie hrán, detailov a podobne.[6]

Tieto dolno-priepustné filtre sa skladajú z jednoduchej konvolučnej masky, ktorá vykonáva operáciu priemeru zvolenej oblasti. Konvolúciou obrazu a masky získame pre každý bod v obraze priemer z konkrétneho bodu a jeho okolia. Takže jednotlivé prvky v maske s rozmerom 3x3 majú hodnotu 1/9. Pre túto masku majú jednotlivé pixely rovnakú váhu na výslednej hodnote. Používajú sa však aj masky, ktoré zdôrazňujú hodnotu stredu alebo stredu a jeho štvôr okolia. Príležitostne môže byť pridaná aj podmienka, že rozdiel medzi vypočítanou a pôvodnou hodnotou musí byť väčší ako predom nastavený prah. Takto je možné odstrániť šum bez prílišného vyhladenia detailov v obraze. [3]

1.2.1 Filtrovanie pomocou mediánu

Filter typu medián patrí do triedy nelineárnych filtrov. Pri tomto filtri sa posúva po obraze okno a postupne sa pre každý pixel stanovuje výstupná hodnota ako medián z hodnôt vnútri okna. Z toho vyplýva, že je pomerne dôležité správne vybratie veľkosti okna. To však závisí vždy od konkrétneho prípadu a nedá sa generalizovať. Pri správne vybratej veľkosti okna tento filter zachováva hrany omnoho lepšie ako predošlí filter priemeru, avšak aj tak poruší ostré rohy. Využíva sa hlavne pre filtrovanie impulzného šumu a šumu soľ a korenie, ktoré môžu vzniknúť napríklad pri náhodnej chybe bitu v komunikačnom kanáli. [6]

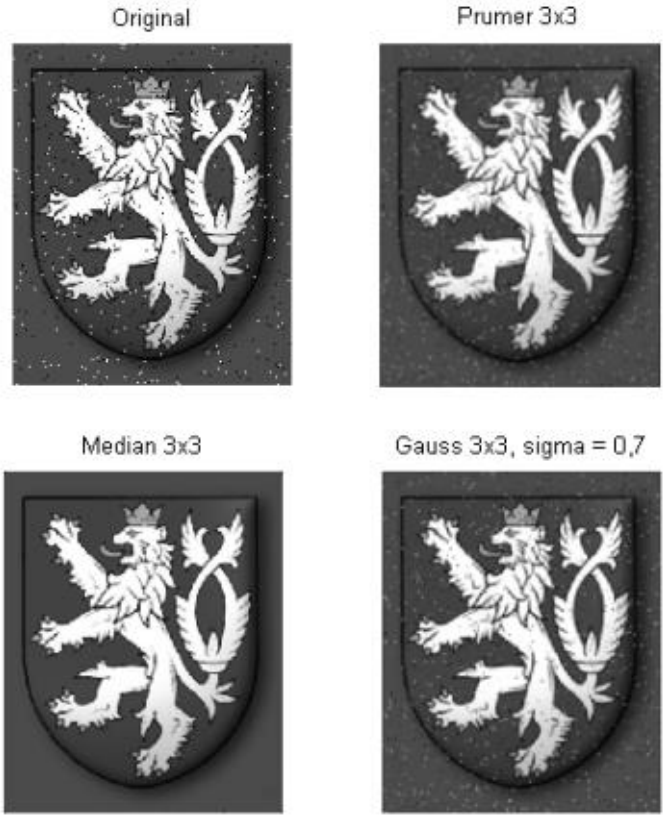
1.2.2 Filtrovanie pomocou filtru s Gaussovým rozložením

Jednou z najčastejšie používaných metód na filtrovanie obrazu je filtrovanie pomocou Gaussiánového filtru. Vzorec pre dvojrozmerné Gaussovo normálne rozloženie je definovaný následne:

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2}). \quad (1.2)$$

Parameter σ sa nazýva smerodajná odchýlka a definuje efektívne rozšírenie funkcie. Pre malú hodnotu σ je funkcia úzka a na druhú stranu pre veľkú hodnotu σ je funkcia pomerne široká.

Dôvodom prečo je tento filter často používaný je, že pri filtrovaní obrazu chceme čo najviac redukovať úroveň šumu, ale zároveň aj čo najlepšie zachovať detaily v obraze. A práve Gaussova funkcia vytvára kompromis medzi týmito dvomi protichodnými požiadavkami. [3]



Obr. 1.2: Porovnanie filtrácie jednotlivých metód, pôvodný obraz má pridaný šum korenie a soľ s hustotou 0,02 [17]

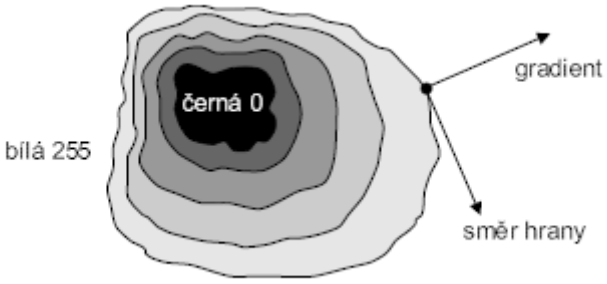
1.3 Detekcia hrán

Jedným z hlavných cieľov spracovania obrazu je v obraze identifikovať a rozpoznať objekty. Hrany tvoria obrys objektu a vďaka ním je možné identifikovať hranicu medzi objektom a pozadím a tak najjednoduchšie odlíšiť objekt od pozadia. Z toho vyplýva, že ak sme schopný presne identifikovať hranu vieme aj presne lokalizovať objekt v obraze. Preto je detekcia hrán jednou z najpoužívanejších operácii pri spracovaní obrazu. [7]
Hrana v obraze je náhla zmena hodnoty obrazovej funkcie f(x,y), ktorú udáva jej gradient ∇. Pre spojitú obrazovú funkciu je smer najväčšieho rastu funkcie smer gradientu ψ a strmosť tohoto rastu je veľkosť alebo modul gradientu $|\nabla f(x,y)|$. Vzorce pre tieto veličiny sú nasledovné:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}, \tag{1.3}$$

$$\psi = arg\left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial x}\right), \tag{1.4}$$

kde arg(x,y) je uhol v radiánoch medzi osou x a radiusvektorom k bodu (x,y).



Obr. 1.3: Naznačená hrana, smer hrany je kolmý na gradient[2]

Ak je potrebná len veľkosť gradientu používa sa vše smerový lineárny Laplaceov operátor Laplace ∇^2, ktorý má nasledovný vzťah:

$$\nabla^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}. \tag{1.5}$$

Gradientné operátory možno rozdeliť do troch kategórií:

1. Operátory aproximujúce deriváciu diferenciami.
2. Operátory hľadajúce prechody druhej derivácie nulou.
3. Operátory lokálne aproximujúce obrazovú funkciu jednoduchým parametrickým modelom. [2]

V najbližších dvoch podkapitolách budú rozobrané prvé dva typy operátorov, tretí nebude v práci použitý, takže ďalej ani nebude popísaný podrobnejšie.

1.3.1 Operátory aproximujúce derivácie diferenciami

V tejto podkapitole budú prebrané detektory, ktoré majú formu jednoduchých konvolučných

masiek. Základné hranové detektory sú: Robertsov, Prewittovej, Sobelov, Robinsonov, Kirschov a Laplaceov operátor, z týchto detektorov budú ďalej rozobrané sú len niektoré, a to hlavne kvôli ich výsledkom. [2]

Prvým je Sobelov hranový detektor, ktorý používa nasledujúcu konvolučnú masku:

$$S_y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \qquad S_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}.$$

Je možné vidieť, že hodnoty na diagonále prispievajú menšou váhou k výsledku ako tie ktoré sú horizontálne a vertikálne. Teda tento operátor dáva najlepšiu odozvu na horizontálne a vertikálne hrany. Hrany v smere osy x deteguje maska S_x a hrany v smere osy y zas maska S_y. Pomocou vzorcov 1.3 a 1.4 je možné dopočítať smer a veľkosť gradientu pre daný pixel. Veľkosť gradientu sa najpresnejšie vypočíta práve podľa vzorcu 1.3, avšak v tom prípade je nutné počítať s odmocninou, ktorej výpočet je pomerne pomalý a zároveň je nutné použitie typov s pohyblivou desatinnou čiarkou, tomuto je najlepšie sa vyhnúť. A to pomocou sumy absolútnych hodnôt S_x a S_y, poprípade použitie vždy len väčšej z oboch hodnôt. Pre správne vyhodnotenie hrán je potrebné ešte prahovanie pre odstránenie falošných hrán. [7]

Ďalším operátorom je Kirschov operátor, ktorý má nasledujúce masky:

$$K_0 = \begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix} \quad K_1 = \begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix} \quad K_2 = \begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix} \quad K_3 = \begin{matrix} 5 & 5 & -3 \\ 5 & 0 & 5 \\ -3 & -3 & -3 \end{matrix}$$

$$K_4 = \begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix} \quad K_5 = \begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix} \quad K_6 = \begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix} \quad K_7 = \begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}.$$

Na prvý pohľad je zrejmé, že tento detektor pracuje na mierne odlišnom princípe ako Sobelov. Tento operátor sa snaží modelovať zmenu jasu v oblasti hrany pomocou rôznych orientácií masky na rozdiel od aproximácie gradientu ako v prípade Sobelovho operátora. Každá z masiek reaguje na hrany v rôznych uhloch, napríklad ak má najväčšiu odozvu maska K_0 tak ide o vertikálnu hranu, horizontálny gradient. Na každý pixel sa aplikuje každá jedna maska, výstup detektora je hodnota jednej z týchto masiek, ktorá je maximálna. Ďalej podľa vzorca:

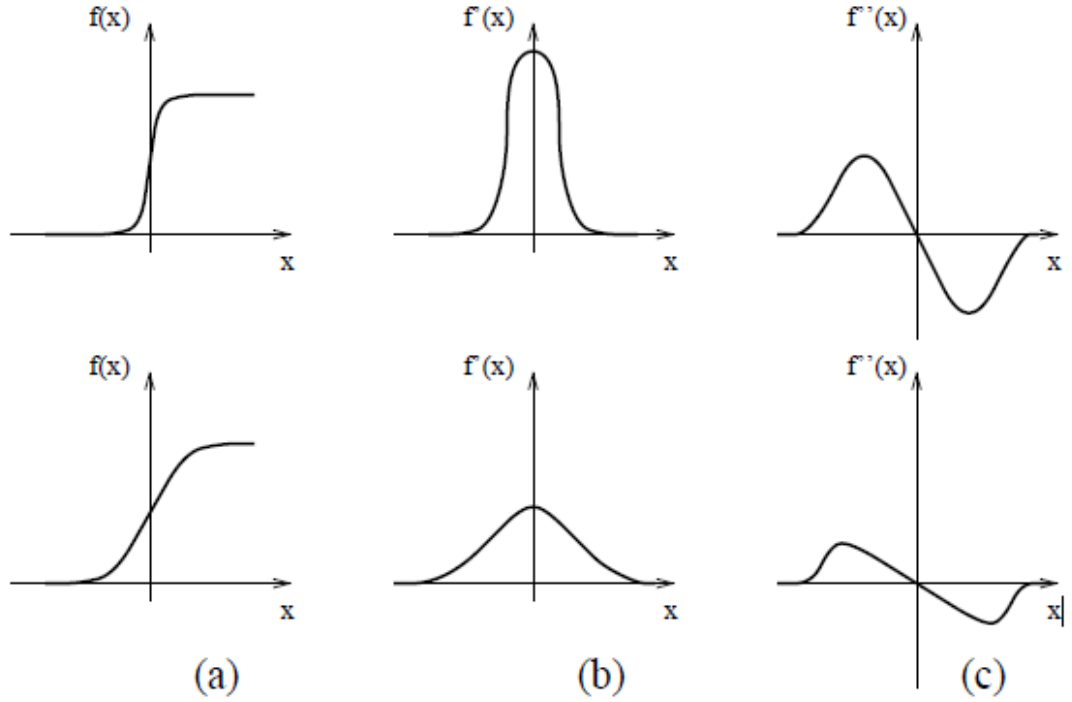
$$\frac{\pi}{4} * i \,, \tag{1.6}$$

kde i je index masky s najväčšou odozvou dostaneme natočenie hrany do jedného z ôsmich základných smerov. [7]

1.3.2 Detektory vyhľadávajúce prechod druhej derivácie nulou

Nevýhodou operátorov z predošlej podkapitole je závislosť veľkosti masky na detailoch v obraze. Takže sú závisle na mierke. Podľa Marrovej teórie sa detegujú hrany na mieste prechodu druhej derivácie obrazovej funkcie nulou. Ako je vidno na obrázku 1.4 prvá

derivácia má maximum v mieste hrany a druhá prechádza v tom istom mieste nulou. Detektormi pracujúcimi na tomto princípe sú napríklad Laplaceov operátor, Marrov-Hildrethovej operátor, poprípade Cannyho hranový detektor. [2]



Obr. 1.4: Model profilu 1D hrany: a)model hrany, b)prvá derivácia, hrana je v maxime c)druhá derivácia, hrana prechádza nulou. [2]

Ďalším rozobraným operátorom je Laplaceov operátor, ktorý používa nasledovné konvolučné masky pre 4-okolie a 8-okolie:

$$H_4 = \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}, \qquad H_8 = \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}.$$

Laplaceov operátor funguje na základe gradientného operátoru ∇^2, podľa vzorca 1.5. Pre detekciu hrán sa používa vždy len jedna z týchto dvoch masiek, operátor je invariantný voči otočeniu a udáva len veľkosť gradientu, nie smer. Nevýhodami sú veľká citlivosť na šum ako aj dvojité odozvy na tenké línie. [2]

Cannyho hranový detektor je jeden z najpoužívanejších detektorov a preto bude ďalej rozobraný. Bol navrhnutý Johnom Cannym v roku 1986 aby splňoval tri základné kritéria:

1. Správna detekcia: je to najjasnejšie kritérium, ktoré hovorí, že každá hrana v obraze by mala byť nájdená a zároveň by sa nemali objaviť žiadne falošné hrany.
2. Správna lokalizácia hranových bodov: vzdialenosť medzi bodmi označenými ako hrana podľa detektora a skutočnými by mala byť čo najmenšia.
3. Jediná odozva na jednu hranu: zabránenie viacnásobným odozvám. [8]

Principiálne vysvetlenie funkcie Cannyho hranového detektoru zhrnuté do štyroch jednoduchých krokov:

1. Vyfiltrovanie šumu v obraze. Na toto je použitý filter s Gaussovým rozložením. Ako príklad môže byť použitá maska z kapitoly 2.3.2.
2. Nájdenie hrán v obraze pomocou Sobelovho hranového operátora ako je popísané v kapitole 2.4.1, teda použitie konvolučných masiek S_x a S_y a spočítanie veľkosti gradientu a smeru podľa vzorcov 1.3, 1.4. Pričom uhol je zaokrúhlený na 0, 45, 90 alebo 135 stupňov.
3. Potlačenie nemaximálnych hrán, v lokálnom okolí hrany sa vždy vyberie len pixel s najväčšou hodnotou gradientu, ostatné sa vymažú a tým vzniknú len veľmi tenké hrany.
4. Aplikovanie hysterézie na hrany, sú zvolené dva prahy horný a dolný, ktoré by mali byť v pomere 2:1 až 3:1. Každý pixel označený ako hranový je testovaný týmito prahmi, ak je jeho hodnota veľkosti gradientu pod dolným prahom tak je vymazaný, ak je nad horným prahom tak je to skutočná hrana. A body, ktoré sú medzi týmito prahmi sú uznané ako hrany len v prípade, že sa dotýkajú už uznaných skutočných hrán. Ak sa vo viacerých po sebe idúcich iteráciách obraz nemení všetky body ktoré sú medzi sú vymazané. [9]

a)Pôvodný obraz b)Rozpoznané hrany v obraze

Obr. 1.5: Použitie Cannyho hranového detektoru pre rozpoznanie hrán v obraze [18]

1.4 Detegovanie rohov

Nájdenie dvoch korešpondujúcich bodov v dvoch rovnakých obrazoch je výhodné pre ďalšie operácie spracovania obrazu, ako napríklad geometrické transformácie alebo analýza pohybu. Počet možných korešpondujúcich bodov v dvoch obrazoch je veľmi veľký a preto sa vyšetrujú len dôležité body v obraze, ktorých je omnoho menej. Ak sú v obraze hranaté objekty, ako dôležité body v obraze sa označujú rohy. Roh v obraze môže byť definovaný ako bod, v ktorom sú dominantné dve hrany v rôznych smeroch. Ak sú v obraze hľadané rohy len podľa tejto definície môžu byť nájdené aj body, ktoré nie sú rohy ako napríklad koniec čiary alebo izolovaný bod lokálneho extrému. Z toho dôvodu sú pre správnu detekciu rohov potrebné ďalšie obmedzenia.

Jeden z prvých a najjednoduchších rohových detektorov sa nazýva Moravcov rohový detektor, ktorý deteguje rohy a ostré hrany ako body s maximálnym kontrastom voči okoliu podľa vzorca: [5]

$$MO(i,j) = \frac{1}{8}\sum_{k=i-1}^{i+1}\sum_{l=j-1}^{j+1}|f(k,l) - f(i,j)|\,. \tag{1.7}$$

1.4.1 Harrisov rohový detektor

Harrisov rohový detektor využíva princíp Moravcovho rohovho detektoru, v ktorom je vylepšené vyhodnocovanie diferencií. Je daný vzorcom:

$$S_W(\Delta x, \Delta y) = \sum_{x_i \in W}\sum_{y_i \in W}(f(x_i, y_i) - f(x_i - \Delta x, y_i - \Delta y))^2\,. \tag{1.8}$$

Z obrazu f je vždy vybrané pre každý pixel okienko W, ktoré je prechádzané pomocou Δx, Δy. Diferencie sa v tomto detektore aproximujú pomocou Taylorovej rady:

$$f(x_i - \Delta x, y_i - \Delta y) \approx f(x_i, y_i) + \left[\frac{\partial f(x_i,y_i)}{\partial x}, \frac{\partial f(x_i,y_i)}{\partial y}\right]\begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}, \tag{1.9}$$

vďaka tomu môže byť minimálne $S_W(\Delta x, \Delta y)$ získané analyticky. [5]

Po dosadení aproximácie do pôvodného vzorca a niekoľkých úpravách dostávame:

$$S_W(\Delta x, \Delta y) = [\Delta x, \Delta y] A_W(x,y) \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}. \tag{1.10}$$

Kde $A_W(x,y)$ je Harrisova matica je pozitívne semi-definitná, a symetrická, je v tvare:

$$A_W(x,y) = \begin{bmatrix} \sum_{x_i \in W}\sum_{y_i \in W}\frac{\partial^2 f(x_i,y_i)}{\partial x^2} & \sum_{x_i \in W}\sum_{y_i \in W}\frac{\partial f(x_i,y_i)}{\partial x}\frac{\partial f(x_i,y_i)}{\partial y} \\ \sum_{x_i \in W}\sum_{y_i \in W}\frac{\partial f(x_i,y_i)}{\partial x}\frac{\partial f(x_i,y_i)}{\partial y} & \sum_{x_i \in W}\sum_{y_i \in W}\frac{\partial^2 f(x_i,y_i)}{\partial y^2} \end{bmatrix} \tag{1.11}$$

Pomocou vlastných čísel tejto matice λ_1, λ_2 je možné zistiť aký typ pixelu je momentálne vyšetrovaný :

- Ak sú majú λ_1, λ_2 veľké pozitívne hodnoty, daný bod je roh.
- Ak λ_1 má veľkú hodnotu a $\lambda_2 \approx 0$, daný bod je hrana.
- Ak $\lambda_1 \approx \lambda_2 \approx 0$, obraz je v tejto oblasti plochý a pixel nie je ani hrana a ani roh.

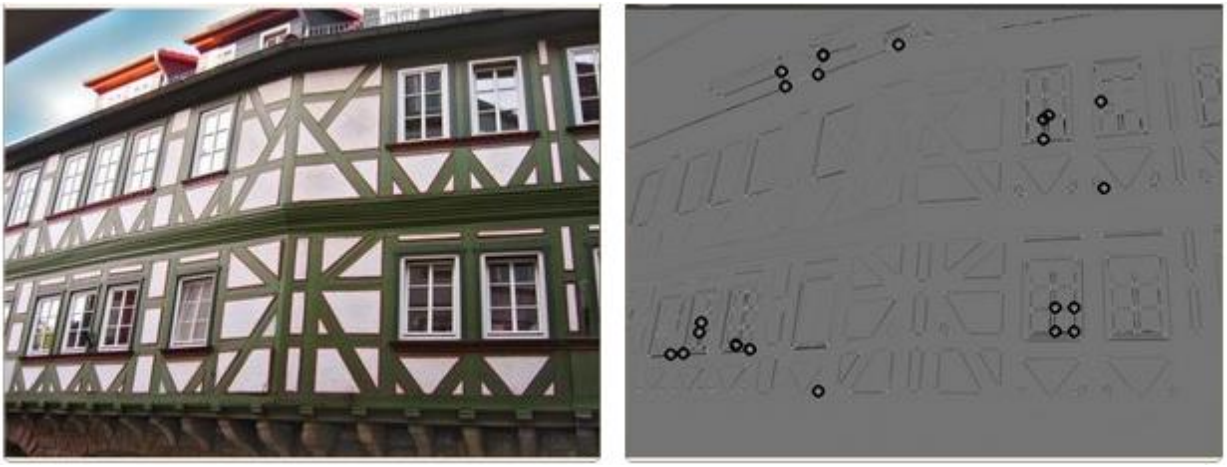
Vypočítanie vlastných čísel matice A_W môže byť výpočetne náročné a preto Harris a Stephens vyvinuli metódu bez týchto čísel:

$$\mathrm{R(A)} = \det(A_W) - \kappa\, trace(A_W)^2. \qquad (1.12)$$

Často býva používaný pre výpočet R namiesto rozdielu podiel, v ktorom nie je potrebné použiť konštantu κ.
V tomto prípade stačí nájsť hodnoty R nad nami zvoleným prahom.

Principiálna funkcia Harrisovho rohového detektora:

- Filtrovanie obrazu filtrom s Gaussovým rozložením.
- Stanovenie Harrisovej matice pre každý bod v obraze.
- Vypočítanie hodnoty funkcie $\mathrm{R(A)}$ z každej matice.
- Pomocou vhodne zvoleného prahu vybrať rohové body. [10]



a)Pôvodný obraz

b)Nájdené najsilnejšie rohy v čiernych krúžkoch

Obr. 1.6: Použitie Harrisovho rohového detektora pre nájdenie rohov v obraze [19]

1.5 Analýza pohybu v obraze

Analýza pohybu v obraze je možné rozdeliť na tri základné typy úloh:

- Detekcia samostatného pohybu – v tomto type úlohy je statická kamera, ktorá deteguje len pohyb na scéne, ktorý nijak bližšie nerozpoznáva, musí však odfiltrovať pohyby v pozadí ako napríklad pohyb lístia a podobne.
- Detekcia a lokalizácia pohybujúceho sa objektu – tento typ úloh je zložitejší ako tie predošlé v tomto prípade môže ako statická tak aj pohyblivá kamera sledovať pohybujúci sa objekt. Pričom je potrebné okrem detekcie objektu aj určiť jeho trajektóriu a v niektorých prípadoch ju aj predpovedať. Využíva sa hlavne pri dopravných úlohách.
- Popísanie vlastností 3D objektu – najzložitejší typ úloh, pri ktorých sa pomocou pohyblivej kamery získavajú 2D snímky pohybujúceho sa objektu. Z týchto snímok sa potom vytvorí model 3D objektu. [4]

Pre zjednodušenie detekcie a spracovania pohybu pevných telies je možné zaviesť nasledujúce predpoklady:

- Obmedzenie maximálnej rýchlosti – jeden konkrétny pixel sa môže medzi dvomi po sebe nasledujúcimi obrazmi posunúť len do kruhu s polomerom v dt od pôvodnej polohy.
- Obmedzenie akcelerácie – maximálna zmena polohy medzi časom t1 a t2 je daná zmenou polohy medzi časom t0 a t1.
- Pohybová korešpondencia – pri pohybe tuhých telies zostáva konfigurácia bodov rovnaká.

Ďalej môže byť analýza pohybu rozdelená na nezávislú, ktorá je rovnaká pre celý obraz bez ohľadu na pozíciu hľadaného objektu a je bodovo alebo blokovo orientovaná. Druhá možnosť je závislá analýza, ktorá je na základe detekcie pozície objektov a využívajú sa pri nej predpoklady pre pohyb tuhých telies, pri nej sa používajú rozdielové metódy alebo rýchlostné pole. [13]

1.5.1 Rozdielové metódy

Rozdielové metódy sú súbor metód na detekciu pohybu v obraze na základe porovnania dvoch obrazov na úrovni pixelov, z ktorých vznikne rozdielový obraz d podľa nasledujúceho vzorca:

$$d(i,j) = \begin{cases} 0 & |f_1(i,j) - f_2(i,j)| < \varepsilon \\ 1 & |f_1(i,j) - f_2(i,j)| \geq \varepsilon \end{cases}, \tag{1.18}$$

kde f_1 a f_2 sú porovnávané obrazy a ε je prahová hodnota jasu. [5]
Pre $d(i,j) = 1$ môžu nastať nasledujúce prípady:

- „$f_1(i,j)$ je pixel na pohybujúcom sa objekte, $f_2(i,j)$ je pixel na statickom pozadí, (alebo vice versa).

- $f_1(i,j)$ je pixel na pohybujúcom sa objekte, $f_2(i,j)$ je pixel na ďalšom pohybujúcom sa objekte.
- $f_1(i,j)$ je pixel na pohybujúcom sa objekte , $f_2(i,j)$ je pixel na inej časti rovnakého pohybujúceho sa objektu.
- Šum, nepresnosti v pozícii stacionárnej kamery, a pod.“ [5]

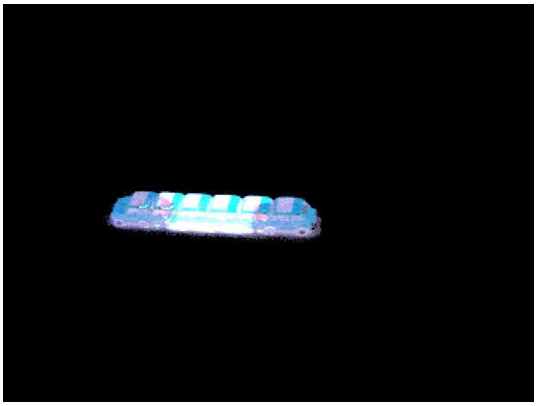
Rozdielový obraz d je možné vypočítať ako jednosmerný alebo obojsmerný. Jednosmerný pracuje na základe vzorcu 1.18 avšak bez absolútnej hodnoty a záporné hodnoty sú nulované. Nevýhodou je, že časť objektu nie je detegovaná, obojsmerný rieši tento problém, pretože je počítaný presne podľa vzorca 1.18, bez vynechanie absolútnej hodnoty a tak sú aj časti, ktoré vyjdú záporné brané v úvahu. Avšak smer pohybu nie je zrejmý ani z jednej z týchto metód.

Pomocou kumulovaného rozdielového snímku je tento problém čiastočne odstránený. A to tak, že od prvého obrazu, ktorý je referenčný sa odčítajú všetky nasledujúce a potom sa tieto rozdielové snímky sčítajú do jedného obrazu. Kumulatívny rozdielový snímok d_{kum} pre N obrazov sa počíta podľa nasledujúceho vzorca:

$$d_{kum} = \sum_{i=2}^{N} w_i \cdot d_i = \sum_{i=2}^{N} w_i \cdot |f_1(x,y) - f_i(x,y)|. \qquad (1.18)$$

Ak je parameter w_i, ktorý určuje váhu jednotlivých obrazov vždy rovnaký tak je zrejmé ako moc sa odlišujú jednotlivé snímky od referenčného a tým je zrejmý aj smer pohybu objektov, ako je vidno na obr. 1.10. [13]

Obr. 1.7: Pôvodné snímky z videa. [13]



Obr. 1.8: Kumulatívny rozdielový snímok vytvorený z predošlého obrázku. [13]

1.5.2 Model pozadia

Rozdielové metódy spomínané v predošlej podkapitole je možné použiť len pri jednoduchých prípadoch ak sa pozadie nemení. V zložitejších prípadoch keď sa mení, ako napríklad pohyb

tieňov a podobne je potrebné vytvoriť vhodný model prostredia, ktorý je potrebné aj aktualizovať. Tento model je možné vytvoriť ako kĺzavý aritmetický priemer z X posledných snímok. A aktuálny snímok sa potom porovnáva s modelom pozadia rozhodnutie či je to pozadie alebo nie sa znovu vykonáva na základe dopredu zvoleného prahu. Pre rýchlejšie zmeny pozadia je možné použiť aj viac ako jeden prah a to horný a dolný. [4]

1.6 Lokálne príznaky

Pri porovnávaní dvoch obrazov je málokedy potrebné porovnávať všetky pixely. A to hlavne pri sledovaní pevných telies a to z dôvodu stálej konštelácie bodov na telese. V tomto prípade je oveľa výhodnejšie hľadanie a porovnávanie významných bodov alebo inak lokálnych príznakov. Jednou z najúspešnejších metód je metóda s názvom Scale Invariant Feature Transform (SIFT).

1.6.1 Algoritmus SIFT

Algoritmus SIFT je škálovo a rotačne nemenný detektor a deskriptor zaujímavých bodov vytvorený Davidom G. Lowe.
Algoritmus je možné rozdeliť do štyroch krokov, ktorými sú:

- nájdenie zaujímavých bodov v obraze,
- určenie presnej polohy významných bodov,
- určenie smeru významných bodov,
- a vytvorenie deskriptora pre každý z nájdených bodov.

Detektor významných bodov – SIFT využíva takzvanú škálovú pyramídu, ktorá sa vytvorí pomocou rozdielu Gaussiánov (DOG). Obraz je podvzorkovaný do k oktáv, v každej je ďalších n obrazov vytvorených Gaussiánovým filtrovaním. Príklad: pri vyhladzovaní v prvej oktáve z pôvodného obrazu A dostávame obraz B pomocou Gausiánového 1D filtru. Nasledujúci obraz C dostaneme pomocou ďalšieho vyhladenia obrazu B rovnakým filtrom. V každej oktáve sa všetky susedné obrazy odčítajú a vznikne n-1 obrazov v každej oktáve. Gaussián sa v tomto prípade využíva hlavne pre možnosť použitia dvoch 1D filtrov namiesto jedného 2D filtra, čím sa zjednodušuje výpočet. Vo vytvorenom priestore sa hľadajú lokálne maximá v okolí 3x3x3, čo znamená, že sa hľadá v priestore 3x3 v obraze kde je skúmaný bod ale aj v obraze pred týmto a v obraze za týmto obrazom v škálovej pyramíde. Tieto extrémy sú potencionálnymi významnými bodmi. [12]

Obr. 1.9: Škálovacia pyramída so 4 oktávami a 5 obrazmi v každej oktáve. Obrazy sú vkaždej oktáve filtrované Gaussiánom. [12]

Ďalším krokom je určenie presnej polohy významných bodov. A to tak, že škálovací priestor je rozšírený pomocou Taylorovej rady a ak je intenzita extrému nižšia ako je nastavený prah bod je odmietnutý. Ďalej pre odstránenie hranových extrémov, sa využíva 2x2 Hessova matica.
Ďalej je určovanie orientácie významných bodov pre dosiahnutie invariantnosti voči rotácii. Pre každý významný bod je vypočítaná jeho amplitúda a smer gradientu. Vytvorí sa histogram s 36 jednotlivými úrovňami, každý prvok je vážení amplitúdou a Gaussovým oknom s hodnotou σ rovnou 1,5 krát škále daného významného bodu. Najvyšší vrchol v histograme je vzatý ako orientácia daného významného bodu. Avšak každý vrchol s hodnotou minimálne 80% z maxima je započítaný tiež. Vytvorí sa tým viac deskriptorov s rovnakou polohou a škálou ale odlišným smerom, čím sa zvýši stabilita.
Posledným krokom je výpočet descriptoru, okolo každého významného bodu sa zobraté okolie 16x16 pixelov. Toto okolie je rozdelené na 16 podblokov o veľkosti 4x4, pre každý podblok je vytvorený osem úrovňový histogram orientácie. Dokopy je vytvorených 128 úrovní. Prvky histogramu sú zoradené do vektoru, ktorý tvorí výsledný descriptor.[11]

2 Použité nástroje

2.1 Python

Ako programovací jazyk pre túto a následnú prácu som si vybral Python 3, ktorý je vysoko úrovňový, objektovo orientovaný programovací jazyk s dynamickou sémantikou. Vďaka tomu že tento jazyk je vysoko úrovňový a má dynamickému typovanie a dynamické viazanie má veľký rozsah použitia. Je to ľahko naučiteľný jazyk s prirodzenou a ľahko čitateľnou syntaxou, vďaka čomu je aj časovo nenáročný na vytvorenie ale aj neskoršiu údržbu programov. Python podporuje aj mnoho modulov, ktoré dodávajú veľkú modularitu a znovu využitie programov.[14]

Pri porovnaní s druhým jazykom, ktorý bol na výber v zadaní práce a to C++, je zrejmé, že chod programu písaného v C++ je oveľa rýchlejší ako ten písaný v Pythone. Avšak program v C++ môže byť aj viac ako 5-krát dlhší a tým sa aj predlžuje čas, ktorý je potrebný na vytvorenie tohto programu a aj jeho následná údržba je zložitejšia. Tieto rozdiely sú práve vďaka vyššie spomínaným vlastnostiam ako vysoko-úrovňovosti a dynamickému typovaniu Pythonu. [15]

Ďalším z dôvodov je aj veľký počet užívateľov a vyplývajúca veľká komunita a množstvo voľne dostupných materiálov na internete ako napríklad rôzne návody, z ktorých je možné sa tento jazyk naučiť od základov až na vysokú úroveň.

2.2 OpenCV

Pre prácu s obrazom bude v následnej bakalárskej práci použitá knižnica OpenCV, vo verzií 3.3.1, ktorá bola počas inštalovania na môj stroj aj najnovšou dostupnou verziou. Táto knižnica obsahuje viac ako 2500 algoritmov pre úlohy počítačového videnia a strojového učenia ako napríklad detegovanie a rozoznanie tvárí, sledovanie dráhy pohybujúcich sa objektov, vytvorenie 3D modelov rôznych objektov.

Knižnica je využívaná ako v komerčnej sfére, kde ju využívajú veľké spoločnosti ako Google alebo Intel ale aj mnoho malých „startup-ov“ tak aj vo výskumných tímoch a tak má veľmi veľké množstvo používateľov, čo uľahčuje nájdenie a porozumenie nejasností pri učení sa tejto knižnice.

OpenCV podporuje operačné systémy Windows, Linux, Android, Mac OS a podporuje viaceré programovacie jazyky ako C, C++, Python, Java, MATLAB, takže aj z tohto dôvodu je naučenie sa pracovať s touto knižnicou do budúcnosti veľmi výhodné. [16]

3 Popis programu pre spracovanie súťažných záznamov

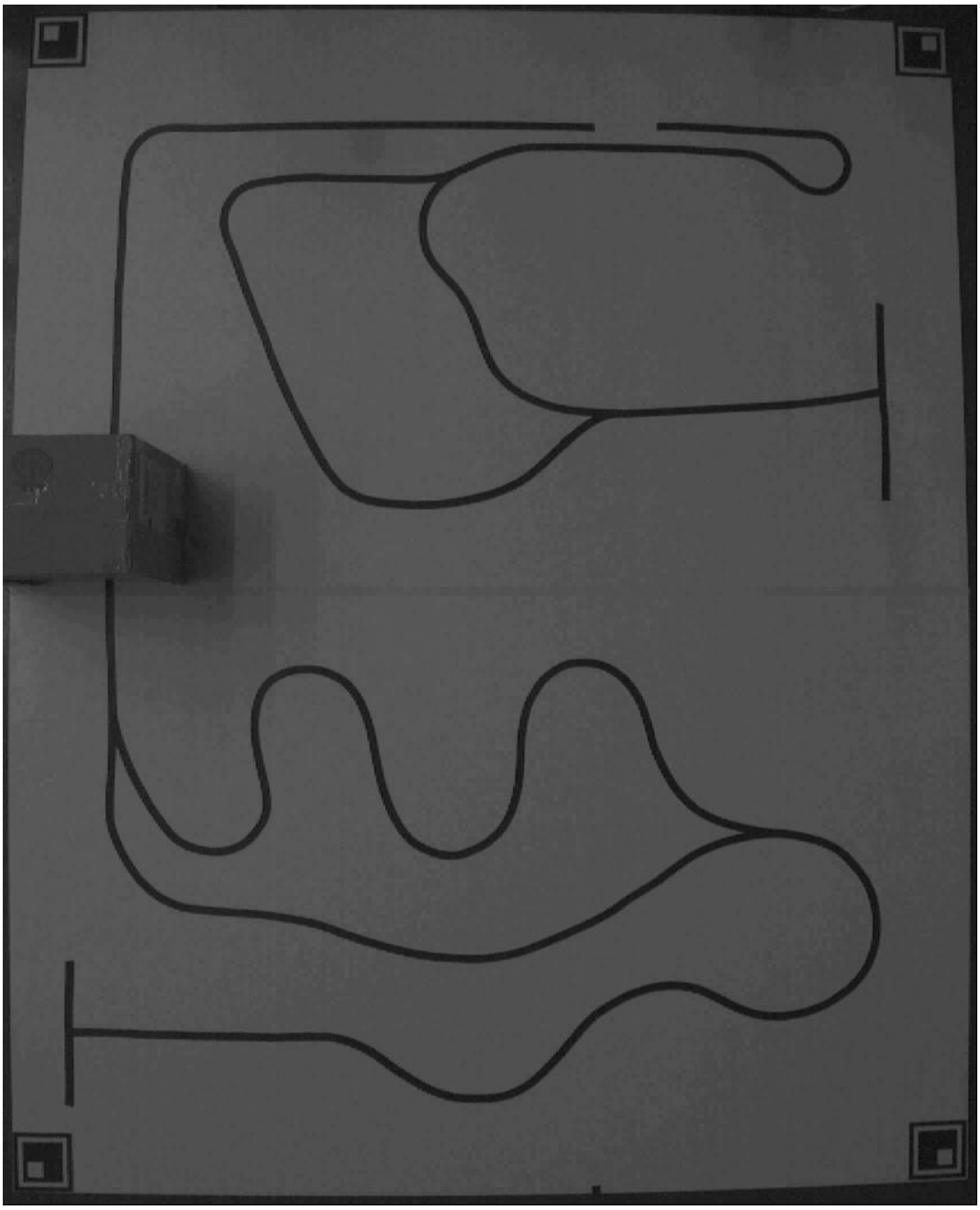
V nasledujúcej kapitole je rozobraný mnou vytvorený program pre spracovanie videí. V prvej časti je vysvetlená trieda Track a jej metódy, ktorá je použitá pre spracovanie prázdneho obrazu trate. Ako z tohto obrazu získava informácie o trati ale aj pre vyhodnotenie pohybu po trati. Ďalej je možné nájsť popis triedy Team a jej metódy, ktoré sú priamo zodpovedné za presné nájdenie súťažného robota na trati ako aj za vyhodnotenie jednotlivých pokusov.
A nakoniec triedu Result, ktorá vytvára konečné poradie tímov v súťaži na základe najlepších pokusov všetkých tímov.
Druhá časť je venovaná úspešnosti vyhodnotenia súťažných pokusov ako aj stanovenie medzných podmienok pre spracovanie.

3.1 Spracovanie súťažnej trate

Dráha, po ktorej jazdia vozidlá, je vytvorená z čiernej pásky na bielom podklade. Dráha je ohraničená štyrmi rohovými grafickými tagmi. Rozmery týchto tagov 10x10 centimetrov, ako aj ich podoba a rozmiestnenie je dopredu dané. Na trati sú prekážky ako je rozdelenie trate a tunel, tieto prekážky môžu byť na trati len v rovných úsekoch. Cieľ je označený v tvare písmena T. Toto sú všetko veci, ktoré by trieda Track mala spracovať. Preto však potrebuje obraz prázdnej trate. Tento obraz je možné vidieť na Obr. číslo 3.1. V tomto prípade je už aj šedotónový. V každom rohu je možné vidieť rohový tag.

3.1.1 Trieda Track

Objekt triedy Track nesie dáta o trati ako sú pozícia tunela, pozícia cieľu, pozícia prerušenia trate, ale napríklad aj pozíciu kamery. Metódy tejto triedy hľadajú tagy v rohoch obrazu, aby našli pozíciu trate, ďalej pomocou týchto tagov spočítajú konštantu na prepočet medzi milimetrami a pixelmi. Objekt triedy Track je v programe vždy len jeden a vytvára sa hneď na začiatku programu.



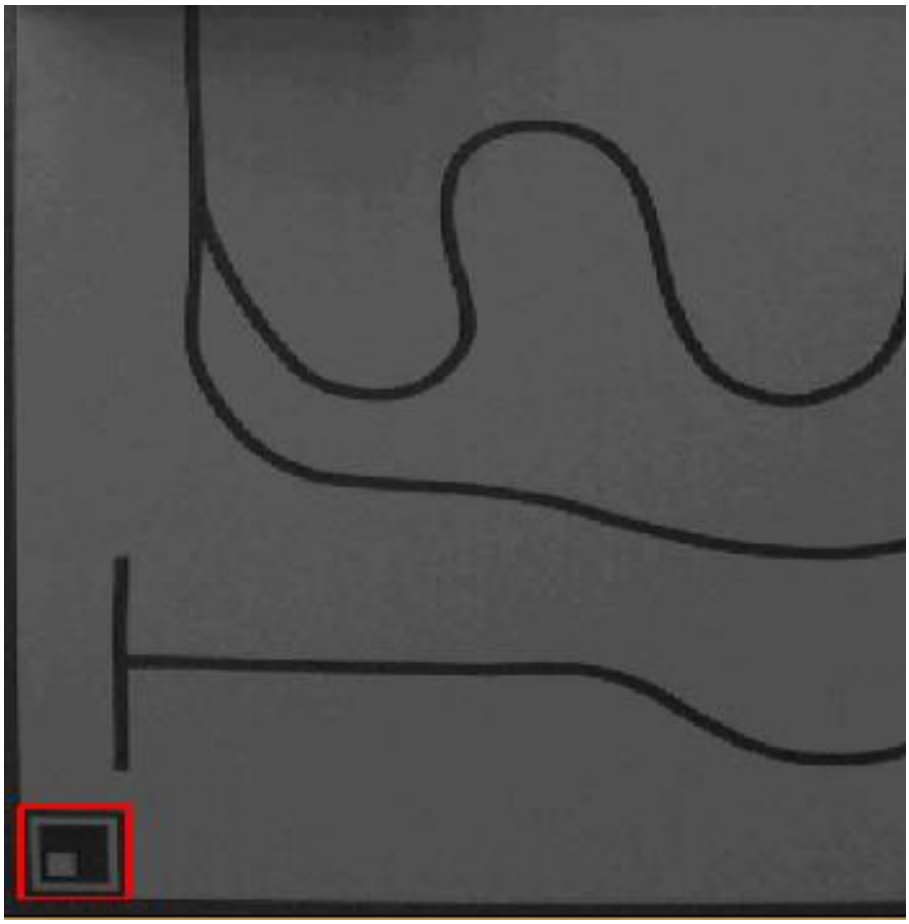
Obr. 3.1: Obrázok prázdnej trate

3.1.2 Metóda __init__

Metóda *__init__* má dva vstupné parametre a tými sú pozícia kamery v milimetroch od stredu podložky na ktorej je závodná trať, a druhý parameter je názov obrazu trate z tejto kamery bez akéhokoľvek predmetu mimo tunela na nej. Ak sú zadané typovo správne parametre metóda načíta obraz a inicializuje atribúty objektu pozíciu kamery a prázdnu snímku trate.

3.1.3 Metóda find_tags

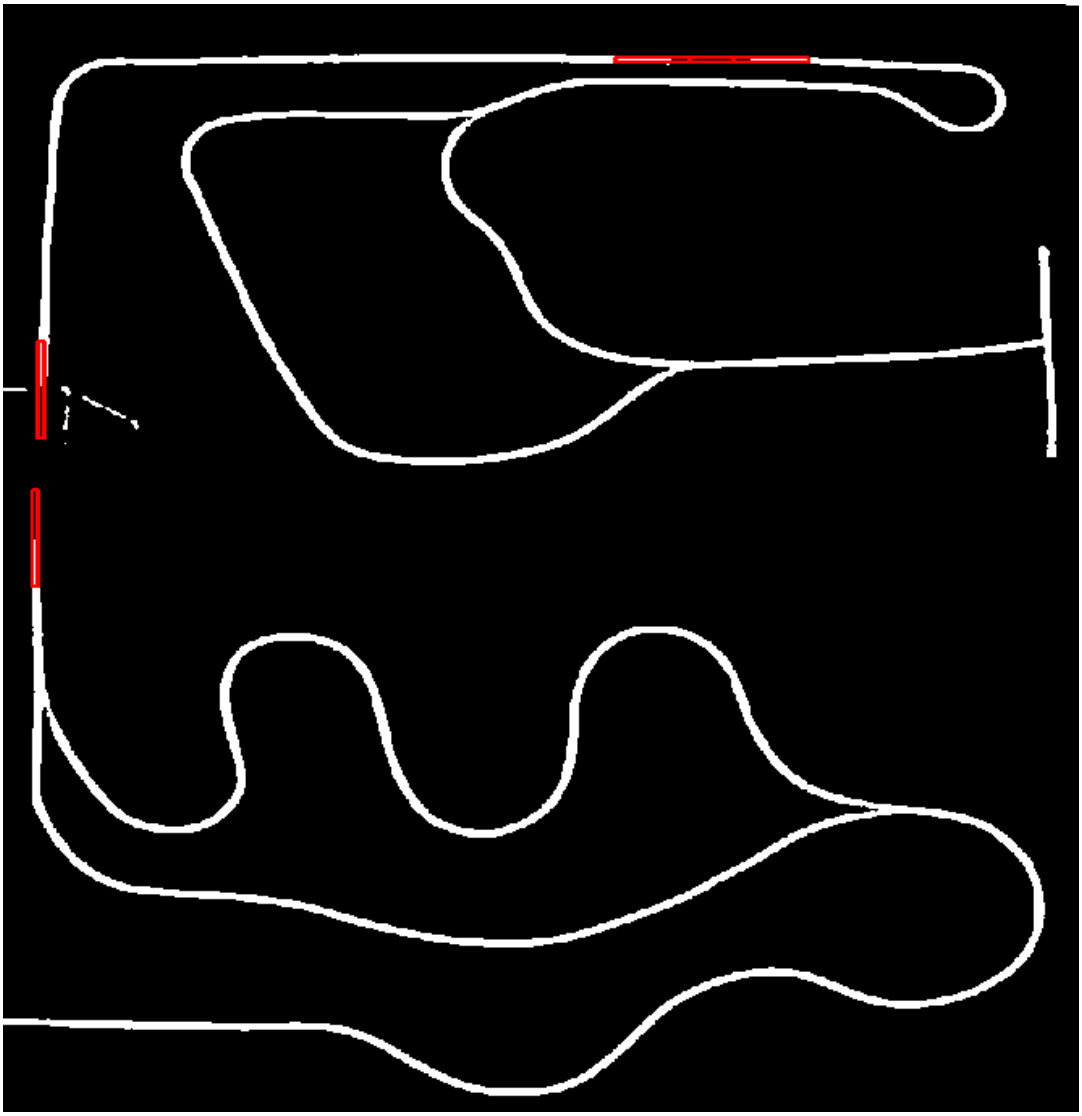
Táto metóda je bez vstupných parametrov a má za úlohu nájsť tagy v rohoch trate a zároveň spočítať prevodnú konštantu medzi milimetrami a pixelmi. Keďže metóda nemá vstupné parametre na začiatku si skopíruje do svojej premennej obraz prázdnej trate a s tým ďalej pracuje. Hľadanie rohových tagov prebieha postupne po jednom načítava a vyhľadáva predom pripravené vzory. Po načítaní vzoru sa v ňom nájdu najvýraznejšie hrany pomocou funkcie *Canny*, ktorá naň aplikuje Cannyho hranový detektor. A následne, podľa toho v ktorom rohu sa tag vyhľadáva sa vyreže konkrétna štvrtina obrazu. Pre čo najpresnejšie nájdenie rohových tagov sa obraz, v ktorom sa tagy hľadajú postupne zmenšuje pomocou funkcie *resize* z knižnice *imutils*. Na rozmerovo zmenený obraz sa tiež aplikuje funkcia Canny s rovnakými parametrami ako pre vzor. Takto upravený obraz a vzor sa posiela do funkcie *matchTemplate*, ktorá posúva vzor postupne po obraze a hľadá čo najväčšiu zhodu obrazu so vzorom, v tomto prípade najlepšie výsledky podáva s metódou TM_CCORR. Jednotlivé výsledky sa porovnávajú a najlepší sa vždy ukladá. Keď je obraz zmenšený na toľko, že jeho rozmery sú menšie ako rozmery vzoru hľadanie pre tento roh sa ukončí. Najlepšie nájdená pozícia sa uloží do atribútov objektu a pokračuje sa hľadaním v ďalšom rohu. Keď sú nájdené všetky tagy metóda začne počítať prevodnú konštantu. Na základe známych rozmerov rohových tagov.



Obr. 3.2 Postupné nachádzanie jednotlivých rohových tagov.

3.1.4 Metóda find_track

Táto metóda je bez vstupných parametrov a mala by z obrazu prázdnej trate získať pozíciu tunela, medzery v trati, cieľu, kostru trate a obraz modelu trate. Na začiatku si načíta obraz prázdnej trate, ten prevedie na šedotónový obraz a vhodne sa prahuje adaptívnym prahom, čím získam len binárny obraz, kde trať je biela, okolie je čiene. Tým je vytvorený model trate. Následne pomocou morfologických operácii erózie, dilatácie, odčítania obrazov a bitovej operácie OR je postupne získaná kostra trate. Ďalej pomocou morfologickej operácie erózie a pomocou elementov v tvare písmena T a kríža 3x3 je získaná aj pozícia cieľu.
Nakoniec sa postupne volajú jednotlivé vzory pre medzery v trati a postupne sa nachádzajú pomocou funkcie *matchTemplat* jednotlivé medzery v trati.



Obr. 3.3 Nájdený model trate, na ktorom je vyznačená pozícia tunelu a prerušenej trate.

3.2 Nájdenie robota po trati

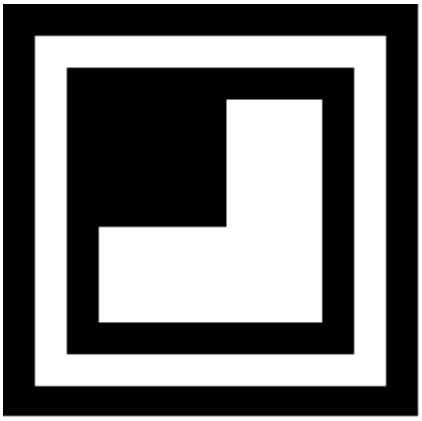
Pre správne určenie pozície robota v obraze, je potrebné rozhodnúť o bode, ktorý je pre všetky roboty na rovnakom mieste. To môže byť problém a to z dôvodu, že jednotlivé tímy postavili robotov s rôznymi fyzickými rozmermi, ako aj rôznymi koncepciami jazdy. Niektoré idú dopredu, iné zas cúvajú. Z toho dôvodu bol ako stred vozidiel vybraný stred hnanej nápravy robota. Tento bod má každé vozidlo na inom mieste a tak je na vozidlách nalepený grafický tag o rozmeroch 5x5 centimetrov. Z toho vyplýva, že hľadanie pozície robota sa zameriava hlavne na nájdenie tohto grafického tagu. Nato je využívaný algoritmus SIFT.
V nasledujúcej tabuľke je popísaná pozícia stredu hnanej nápravy robota na základe pozície grafického tagu v x-ovom a y-ovom smere. V tabuľke chýba tím číslo jedna. Tento tím som vyradil zo súťaže pretože má len jeden súťažný pokus a aj pri tom sa robot nerozbehne sám ale do videa zasahujú ruky, ktoré ho zapínajú a zakrývajú grafický tag, čím je znemožnené nájdenie pozície robota.

Tím č.	Názov tímu	x[mm]	y[mm]
2	PHGP	74	-29
3	INTERNATIONAL	115	-14
4	PECKA POWER	-46	-28
5	DEFEKT	-87	-23
6	MAD PI	17	23

Tab č. 1 Tabuľka súťažných tímov

3.2.1 Trieda Team

Objekty triedy *Team* nesú dáta každého tímu o polohe rozpoznávacieho tagu na robotovi, ale aj dáta z jeho súťažných jázd. Metódy tejto triedy spracovávajú videá z jázd a hľadajú v nich robota. Na robotovi nachádzajú dôležité body a následne pomocou nich dokážu spracovať pohyb robota po trati.
Po spracovaní snímku trate je možné pridať po jednom súťažné tímy. Vtedy sa vytvára inštancia triedy *Team* a volá sa metóda __init__.



Obr. 3.4 Vzor grafického tagu, ktorý je na robotovi.

3.2.2 Metóda __init__

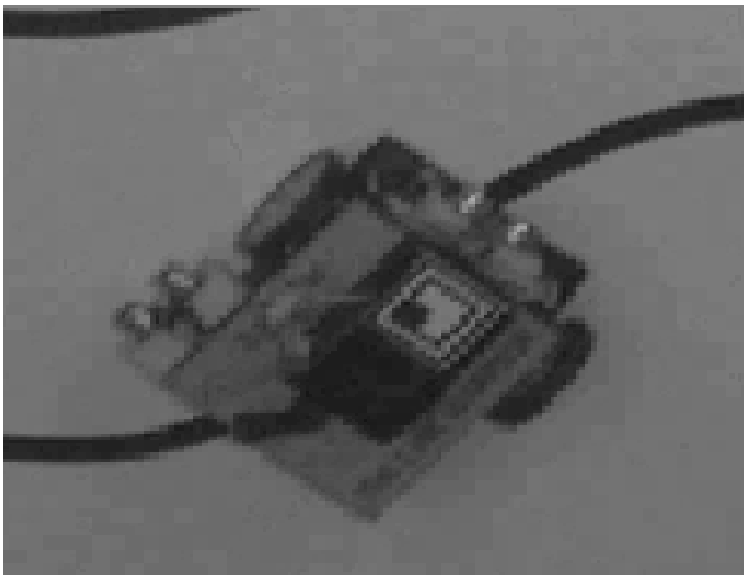
Metóda __init__ má tri parametre, ktorými inicializuje atribúty objektu a to názov tímu, vzdialenosť v x-ovom a v y-ovom smere od stredu hnaného podvozku robotu po ľavý horný rohu grafického tagu umiestného na robotovi, ktorý zobrazený je na Obr.3.4 .

3.2.3 Metóda find_robot

Metóda find_robot je zodpovedná, za nájdenie robota v obraze pomocou rozdielových obrazov. Táto metóda má šesť vstupných parametrov a to param, ktorý rozhoduje o tom kde sa bude robot hľadať, ďalej dva profilové histogramy rozdielového obrazu a to pre riadky a aj stĺpce, predošlú lokáciu robota v stĺpcoch a aj riadkoch a nakoniec objekt trať. Ako bolo už povedané param rozhoduje o tom kde sa hľadá robot, ak má hodnotu nula a jedna robot sa hľadá v predošlej lokácii a to tak, že profilové histogramy sa orežú podľa predošlej rozšírenej lokácie robota. Hľadanie robota sa tak zameriava na oblasť v ktorej by sa mal najpravdepodobnejšie nachádzať, čím sa aj odstraňujú rušivé vplyvy v iných častiach trate ako napríklad vložený nežiadúci predmet. Takto nájdená oblasť sa už len prahuje aby sa odstránili riadky a stĺpce na ktorých sa robot nenachádza. A výsledná poloha je výstup z tejto metódy.

Ak má param hodnotu dva až tri, vozidlo vošlo do tunelu. Hľadanie robota prebieha rovnako ako v predošlom prípade avšak v histogramoch rozdielového obrazu sa nehľadá na predošlej pozícii ale na opačnej strane tunelu. To ktorou stranou do tunelu vošiel hovorí hodnota premennej param či je dva alebo tri.

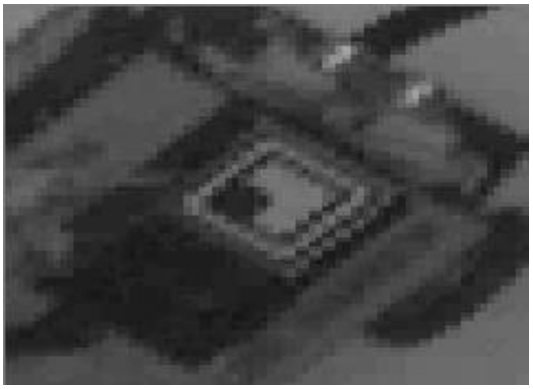
V prípade param hodnoty štyri až päť je vozidlo v medzere na trati. Hľadanie prebieha znovu rovnako s tým rozdielom, že je len hľadané na opačnej strane medzery. Ak je robot v tuneli alebo v medzere na trati a ešte neprešiel na druhú stranu ako poloha robota sú z metódy posielané nuly.



Obr. 3.5 Nájdený robot vo videu pomocou metódy find_robot

3.2.4 Metóda find_template

Metóda find_template hľadá vo vstupnom obraze robota grafický tag. Má dva vstupné parametre a tými sú obraz v ktorom má byť nájdený grafický tag a potom absolútny uhol natočenia vozidla. Na začiatku metódy sa načíta predpripravený vzor grafického tagu. Tento vzor je však ešte potrebné upraviť. Z rozmerov tohto tagu sa spočíta stred tagu, ktorý sa spolu s uhlom natočenia posiela do funkcie *getRotationMatrix2D*, ktorá vypočíta maticu afinnej transformácie pre 2D rotáciu o zadaný uhol. Matica rotácie sa spolu so vzorom tagu a jeho rozmermi posiela do funkcie warpAffine. Táto funkcia vykoná na vzore afinnú transformáciu danú vstupnou maticou. Z takto upraveného vzoru sa ďalej spočíta histogram a pre zlepšenie vyhľadávania sa vo vzore pomocou funkcie *Canny*, vyhľadajú najvýraznejšie hrany. Až po týchto úpravách je získaný vzor, ktorý bude hľadaný vo vstupnom obraze. Vstupný obraz je ešte potrebné prahovať na binárny obraz rovnako ako je aj vzor. Pri vhodne zvolených parametroch adaptívneho prahovania je aj odstránená väčšina šumu z obrazu. Na takto upravený obraz sa aplikuje funkcia *Canny* s rovnakými parametrami ako pre vzor. Následne sa oba upravené obrazy hrán posielajú do funkcie *matchTemplate*, ktorá hľadá čo najväčšiu zhodu obrazu so vzorom. Nato používa jednu zo šiestich metód. V mojom prípade keď sa táto funkcia používa pre všetky videá nie je možné povedať, ktorá metóda podáva najlepšie výsledky a tak využívam postupne všetky okrem metód *TM_SQDIFF* a *TM_SQDIFF_NORMED,* ktorých výsledky boli správne len v minime prípadov. Z výsledkov následne vypočítam histogram a pomocou metódy *compareHist* porovnávam jednotlivé histogramy a hľadám najlepší z výsledkov. Poloha toho najlepšieho je výstupná hodnota tejto metódy.



Obr. 3.6 Obraz tagu nájdený metódou find_template

3.2.5 Motóda calc_angle

Táto metóda spočíta aktuálny uhol natočenia a absolútny uhol natočenia robota. Rozdiel medzi týmito uhlami vzniká z dôvodu aktualizovania vzoru pre algoritmus *SIFT*, pričom ten aktualizovaný vzor je vždy inak uhlovo natočený.
Vstupné parametre metódy sú poloha hľadaného objektu určená štyrmi bodmi, aktuálny uhol natočenia a hodnota absolútneho uhlu pri poslednom aktualizovaní vzoru pre *SIFT*. Začiatok metódy je vloženie polohy objektu do funkcie *minAreaRect()*. Táto funkcia nájde najmenší uhlovo natočený obdĺžnik, ktorý uzaviera tieto body. Z tejto funkcie si vezmem len uhol natočenia tohto obdĺžniku. Problém vzniká v tom, že táto metóda vracia uhol v rozmedzí 0 až -90 stupňov, tento uhol som nazval krátky. Preto som si kruh rozdelil na osem častí po 45 stupňoch. Následne zo vstupného aktuálneho uhla a krátkeho uhla zistím, v ktorej časti kruhu sa nachádzam a podľa toho spočítam nový aktuálny uhol v rozmedzí 0 až 360 stupňov. Nakoniec na základe absolútneho uhlu a nového aktuálneho spočítam nový absolútny uhol. Výstup z tejto metódy sú nové spočítané uhly.

3.2.6 Metóda find_centr

Metóda *find_centr* je zodpovedná za nájdenie stredu hnanej nápravy robota, ktorá sa pokladá za stred robota pre nájdenie jeho pozície na trati. Stred robota táto metóda vypočíta z pozície grafického tagu na robotovi danej štyrmi bodmi, absolútneho uhlu, x-ovej a y-ovej vzdialenosti tagu od stredu robota, ktorá je pre každý tím iná a konštanty na prevod medzi milimetrami a pixelmi v obraze. Ako prvý krok spočíta z polohy tagu jeho stred a následne vypočítam uhol alfa z absolútneho uhlu a pomocou goniometrických funkcií sin a cos spočítam polohu rohu ľavého horného rohu. V ďalšom kroku na základe x-ovej a y-ovej vzdialenosti a uhlu alfa spočítam uhol beta a následne znovu zo vzorcov rovnakých goniometrických funkcií spočítam bod kde sa nachádza stred hnanej nápravy robota. Čo je aj výstupnou hodnotou tejto metódy.

3.2.7 Metóda track_pos

Táto metóda vyhodnocuje podľa polohy robota, či sa stále nachádza na trati, či nevchádza do tunela alebo do medzery v trati alebo už nie je v cieli. Vstupné parametre sú pozícia tagu a pozícia stredu hnaného podvozku robota, objekt trať a pozičný parameter. Tento parameter sa mení len keď sa robot dostane na niektoré z dôležitých miest na trati a zaisťuje správnosť smeru pohybu robota. Na začiatku je potrebné previesť súradnice pozície stredu robota a pozíciu tagu na súradnice modelu trate.
Následne sa volá metóda *add_movement* na objekt triedy *Attempt* so súradnicami stredu robota. Tým sa pridá aktuálna pozícia robota do jeho celkovej trajektórie.
Ďalej metóda overuje či je robot na trati pomocou obrazu modelu trati, ktorý získa z objektu trať. Tento model je len binárny obraz, na ktorom je trať bielej farby a všetko ostatné čiernej. Z modelu trate je vytvorený výrez v pozícií stredu robota. Z hodnoty pixelov vo výreze je spočítaná suma, ak je nulová metóda vracia pole o dvoch prvkoch v ktorom je prvý nula a druhý nezmenená hodnota pozičného parametru. Znamená to, že sa robot nenachádza na trati. Ak je väčšia ako nula je na trati a v tom prípade prichádza aj bližšie zistenie či sa

nenachádza na jednom z dôležitých miest na trati. Týmito miestami sú vchod do tunela z jednej alebo z druhej strany, v tom prípade je výstupná hodnota dva alebo tri podľa strany a hodnota pozičného parametru. A ďalej to môže byť rozdelenie trate z jednej strany alebo z druhej, v tom prípade je výstup štyri alebo päť, znovu podľa smeru jazdy a hodnota pozičného parametru. Pri kontrole tunela a medzery je potrebné rovnakú kontrolu spraviť aj pre pozíciu tagu, z dôvodu možného prekrytia tagu v tuneli. Nakoniec ak je v cieli má pole na výstupe hodnotu šesť a hodnotu pozičného parametru. Ak nie je na žiadnom z týchto miest na trati v tom prípade je výstup rovný jedna a hodnote pozičného parametru.

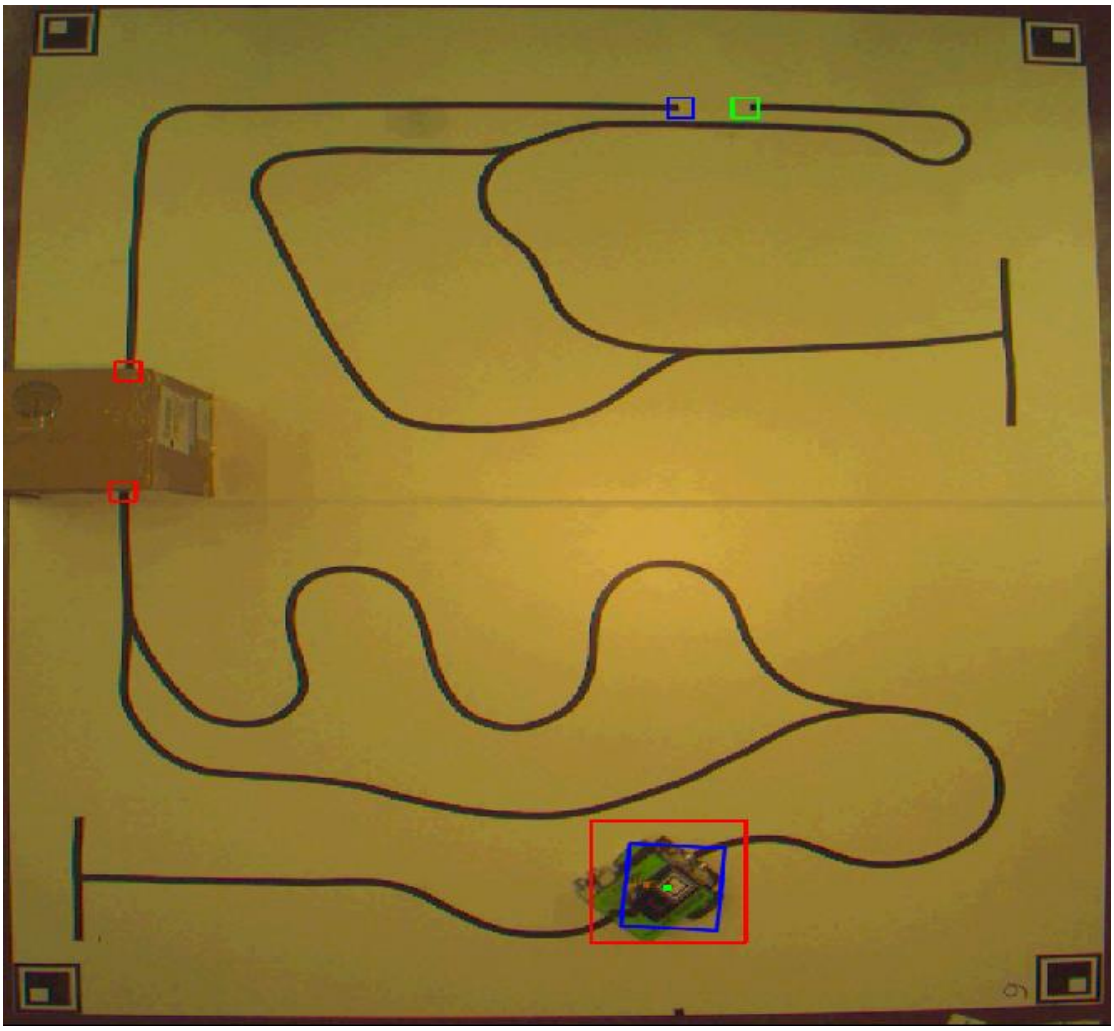
3.2.8 Metóda add_attempt

Metóda *add_attempt* pridáva objektu triedy *Team* súťažný pokus a to tak že vytvára objekt triedy *Attempt* z jej vstupných parametrov, ktorými sú objekt trať a hodnotu snímkovej frekvencie videa. Ďalej aj zvyšuje premennú počet pokusov konkrétneho objektu triedy *Team*.

3.2.9 Metóda Comp_attempt

Potom ako užívateľ pridá minimálne jeden tím je možné spustiť spracovanie videa so súťažnou jazdou. Tým sa volá metóda *Comp_attempt* a užívateľ je požiadaný aby zadal jej parametre, ktoré sú názov videa a názov jedného z tímov, ktorý už pridal v predošlom kroku. Ak tieto parametre zadá užívateľ správne tak sa otvorí okno s videom. A metóda *Comp_attempt* začne snímku po snímke hľadať presnú pozíciu a natočenie robota na trati. To tak, že načítanú farebnú snímku z videa prevedie na šedotónovú a hneď potom ju vhodne prahuje a následne filtruje mediánovým filtrom tak, aby sa odstránil šum vzniknutý osvetlením scény z jedného bodu zvrchu a okolitým osvetlením v miestnosti, ktoré nie je lineárne. Z toho dôvodu je v strede trate najväčšia intenzita osvetlenia a na okrajoch je najnižšia. Ďalšou vecou je aj rozdielnosť intenzity osvetlenia medzi jednotlivými videami, ktorá je značná. Následne sa pomocou rozdielových metód, ktoré boli popísané v predošlých kapitolách odčíta takto upravená snímka od snímky samotnej trate, ktorú užívateľ zadal na začiatku programu. Táto snímka pred odčítaním prešla rovnakými úpravami pre odstránenie šumu a rozdielnosti osvetlenia v programe slúži ďalej ako model pozadia.
Na rozdielovej snímke zostane už len samotný robot, jeho pozícia je nájdená pomocou vytvorených profilových histogramov. Na úplne prvom snímku videa sa hľadá pozícia na celej trati. Avšak v ďalších sa pre zrýchlenie behu používa metóda *find_robot*, ktorá ho nájde v pozícii kde bol predtým a tým urýchľuje beh programu.
Takto nájdená pozícia robota je však len približná a nie je možné z nej určiť jeho uhlové natočenie alebo stred hnanej nápravy, ktorá je v mojej práci považovaná za stred vozidla. Preto je hľadaný grafický tag, ktorý je na každom vozidle. A pre zjednodušenie a zrýchlenie nájdenia jeho pozície sa používa pozícia robota nájdená rozdielovými snímkami. Tá je posielaná spolu s aktuálnym uhlom, ktorý je na prvej snímke nulový do metódy *find_template*. Táto metóda sa používa len v prvej iterácii, keď sa uhol rovná nule, pretože sama nevie vyhľadávať bez zadaného uhla natočenia. Potom ako z metódy *find_template* dostanem presnú pozíciu grafického tagu, je tento obraz tagu použitý ako vzor pre

algoritmus SIFT, ktorý bol popísaný v predošlých kapitolách.
Algoritmus SIFT sa používa v každej ďalšej iterácii pre nájdenie pozície tagu, ako aj zistenie uhlu natočenia robota. Pre zrýchlenie a aj väčšiu pravdepodobnosť nájdenia tagu sa tag hľadá znovu len v časti obrazu, ktorá bola určená v predošlých krokoch pomocou rozdielových obrazov. Potom ako algoritmus SIFT nájde najdôležitejšie body ako vo vzore tak aj v obraze sa pre nájdenie zhôd medzi bodmi vzoru a bodmi obrazu požíva metóda *knnMatch,* ktorá nájde k najlepších zhôd pre každý bod zo vzoru v obraze . Ďalej sa pre vytriedenie správnych dvojíc bodov používa pomerový test popísaný v práci Davida Lowe. Ak po tomto teste nezostáva dostatočný počet zhôd pokračuje sa na ďalšiu iteráciu. Vďaka vhodne nastaveným parametrom algoritmu SIFT a vzoru, ktorý je zväčšený aby obsahoval väčší počet významných bodov tento problém nastáva len vo veľmi malom počte prípadov. Pri nájdení viac ako dvadsiatich správnych zhôd sa pozície bodov rozdelia na pozície vo vzore a pozície nájdené v obraze aby boli takto vložené ako parametre do funkcie *findHomography*, ktorá na základe týchto bodov a zvolenej metódy v mojom prípade LMEDS (*Least-Median robust method*) nájde perspektívnu transformáciu medzi dvomi plochami a roztriedi na základe toho či jej tieto dvojice vhodne zapadajú do spočítanej transformácie na tzv. „inliers and outliers“, ktoré vytvoria jeden z výstupných parametrov a to masku a druhým je homografická matica. Túto maticu spolu s rozmermi vzoru použijem ako vstupné parametre pre funkciu *perspectiveTransform*, ktorá nájde hľadaný objekt pri minimálne štyroch správnych dvojiciach bodov. Nájdenú pozíciu objektu ďalej posielam do funkcie *calc_angle* spolu s aktuálny uhlo a informáciu o tom aký bol uhol pri poslednej zmene hľadaného vzoru. Po spočítaní správneho uhlu od tejto funkcie je potrebné upraviť súradnice nájdeného tagu, pretože pozícia získaná predošlými algoritmami je posunutá o ofset, ktorý vznikol pretože významné body boli SIFT-om hľadané v oblasti nájdenej pomocou rozdielového obrazu.
Pozícia tagu na základe celého obrazu sa ďalej posiela spolu s uhlom natočenia vozidla a ďalšími parametrami každého konkrétneho tímu do metódy *find_centre*, ktorá nájde stred hnanej nápravy robota. Ďalej sa volá metóda track_pos, ktorá rozhoduje, či sa robot stále nachádza na trati. Ak sa nachádza tak ešte zisťuje či nie je v nejakom dôležitom bode ako je vchod do tunelu alebo cieľ. Ak je mimo trate viac ako jednu iteráciu program tento stav vyhodnotí, že robot je už mimo dráhy a ukončí jeho pokus a spustí metódu score_attempt a *print_score*, ktoré vytlačia prejdenú vzdialenosť a čas.
Ak nie je mimo trate, program pokračuje ďalej. Po prejdení viac ako štyroch iteráci je potrebné aktualizovať obraz vzoru pre algoritmus SIFT. To sa vykonáva znovu pomocou metódy find_template. Ak sa aktualizuje vzor tagu, je potrebné aj zaznamenať hodnotu absolútneho uhlu pred aktualizáciou. Táto hodnota uhlu sa používa v ďalších iteráciach pre spočítanie správneho absolútneho uhlu. Počas priebehu metódy je vždy spustené okno s videom súťažného robota. Ako aj okno s pohľadom na grafický tag a samotného robota. Táto metóda končí tromi spôsobmi prvý už bol povedaný a to ak vozidlo vyjde z trate, druhý je, že vozidlo dôjde do cieľa a tretím je zastavenie od užívateľa.
To môže vykonať stlačením tlačidla „q“. Tlačidlom „p“ video zastaví a ďalším stlačením sa video a aj algoritmus znovu spustí.



Obr. 3.7 Otvorené okno počas jazdy robota. Modrý štvorec označuje nájdený graf. tag, červený označuje robota nájdeného pomocou rozdielových obrazov, zelená bodka je stred hnanej nápravy. Ostatné malé štvorčeky označujú medzery v trati.

3.2.10 Metóda print_teams

Je to jediná triedna metóda triedy Team, bez vstupných parametrov a má za úlohu vytlačiť všetky vytvorené objekty tejto triedy.

3.3 Trieda Attempt

Objekt triedy Attempt sa vytvára pri každom súťažnom pokuse o prejdenie trati. Objekt tejto triedy nesie informácie o pohybe robota po trati. Jeho čas, trajektóriu a aj prejdenú vzdialenosť.

3.3.1 Metóda __init__

Metóda *__init__* má tri vstupné parametre a to masku, ktorá je obraz súťažnej trate s hrúbkou jedného pixelu, prevodnú konštantu z pixelu na milimeter a snímkovú frekvenciu videa. Týmito hodnotami inicializuje atribúty objektu a vytvorí si ďalšie nulové atribúty pre prejdenú vzdialenosť, čas, trajektóriu a obraz pohybu.

3.3.2 Metóda score_attempt

Táto metóda na základe vstupných parametrov, ktorými sú číslo snímky štartu pokusu a číslo snímky konca pokusu a atribútov objektu spočíta čas tohto pokusu. Následne pomocou aplikovania bitovej operácie AND na masku, čo je model trate a obraz pohybu robota metóda zistí prejdenú trajektóriu a prepočíta ju z pixelov na milimetre.

3.3.3 Metóda add_movement

Metóda add_movement na základe vstupu, ktorým sú rohy štvorca vytvoreného okolo stredu robota pridáva pohyb do obrazu pohybu robota.

3.3.4 Metóda print_score

Táto metóda je volaná po skončení súťažného pokusu bez vstupných parametrov a vytlačí prejdenú vzdialenosť a čas, za ktorý ju robot prešiel.

3.4 Trieda Results

Objekt tejto triedy je v programe vždy len jeden s názvom res a je vytvorí sa keď užívateľ chce zistiť výsledky súťažných jázd. Objekt má dva atribúty a tie sú list všetkých mien tímov a list ich najlepších pokusov.

3.4.1 Metóda __init__

Metóda __init__ vytvorí oba prázdne atribúty objektu team-best a team_names.

3.4.2 Metóda best_attempts

Metóda best_attempts má parameter dictonary so všetkými objektmi triedy Team. Tieto objekty postupne prejde a vyberie si z každého objektu jeho jednotlivé súťažné pokusy, ktoré

porovná a vždy ten najlepší uloží do listu team_best. Ak tím nemá žiadny pokus tak ho nepridáva do tohto listu.

3.4.3 Metóda make_res

Metóda make_res na základe vstupného parametru, ktorý je dictionary so všetkými objektmi triedy Team vytvorí poradie tímov od najlepšieho po najhorší. Na začiatku si do listu team_names vyberie mená všetkých tímov, ktoré majú aspoň jeden súťažný pokus o prejdenie dráhy. Potom zavolá metódu best_attempts, ktorá nájde všetky tie najlepšie pokusy jednotlivých tímov a naplní nimi list team_best. Ďalej tento list zoradí tak aby prvý bol tím, ktorý prešiel do cieľa s najlepším časom alebo ktorý prešiel najväčšiu vzdialenosť a postupne podľa rovnakého kľúča zoradí aj ostatné tímy.

3.4.4 Metóda print_res

Metóda *print_res* je bez vstupných parametrov a na základe atribútov objektu triedy *Results* vytlačí konečné poradie tímov zoradené od najlepšieho po najhorší , vytlačí aj čas a prejdenú vzdialenosť.

4 Záver

Po testovaní programu som prišiel na to, že vyhodnocovať pohyb vozidiel po čiare na základe stredu hnanej nápravy funguje hlavne v prípade, ak robot má prednú hnanú nápravu. V tom prípade je schopný ňou omnoho lepšie kopírovať trasu, po ktorej má ísť, ako v prípade zadnej hnanej nápravy. Najväčším problémom pri vyhodnocovaní toho, či sa robot nachádza na trati je, keď sa snaží skrátiť si zákrutu a ide do nej mierne šikmo. V tom prípade sa stred hnanej nápravy nemusí nachádzať na trati aj viacero iterácií po sebe. To program vyhodnotí ako skutočnosť, že je vozidlo mimo dráhy a ukončí pokus. V týchto prípadoch som mohol zväčšiť okolie, v ktorom program kontroloval, či sa stále nachádza na trati, alebo zväčšiť model trate. Avšak v tom prípade by sa zhoršilo presné určenie miesta, v ktorom vypadol z trate, poprípade by nebolo možné rozoznať, či vybehol z trate alebo nie, v úsekoch, kde sú veľmi blízko pri sebe dva pruhy dráhy. Z toho dôvodu som volil radšej užší model trate a zároveň menšie okolie pre rozpoznávanie polohy robota, čo však viedlo k častejšiemu vypadávaniu robotov zo súťažnej trate. Podľa viacerých pokusov, ktoré som spravil, najlepšie kopíroval trať tím číslo šesť. Poradie bolo nasledovné:

Tím č.	prejdená vzdialenosť [cm]	Čas [s]
6	528.7	14.96
5	422.1	16.3
3	224.8	22.86
4	2172.2	15.3
2	1833.3	6.83

Tab. č 2: Najlepšie pokusy jednotlivých tímov.

Z výsledného poradia a dodaných videí v prílohe možno vidieť, že detekcia a vyhodnotenie jednotlivých jázd nie je úplne presná. Hlavne vo vrchnej časti obrazu, aj keď ide vozidlo po trati, nastávajú problémy s detekciou, a to pravdepodobne v dôsledku skreslenia obrazu v hornej časti. Toto skreslenie som sa pokúšal odstrániť geometrickými transformáciami, čo sa mi však bez možnosti skalibrovať kameru, ktorou boli tieto videá natáčané, nepodarilo. Niektoré videá museli byť zas odobraté z detekcie, a to napríklad z dôvodu zlého štartu, ako napríklad video team4_1.avi alebo video team2_1NN.avi z dôvodu zlého postavenia na štarte.

Literatúra

[1] YOUNG, Ian T., Jan J. GERBRANDS a Lucas J. VAN VLIET. Fundamentals of Image Processing[online]. 2. vydanie. The Netherlands: Delft University of Technology, 1997[cit. 2017-11-10]. ISBN 9075691017. Dostupné z: https://www.researchgate.net/profile/Ian_Young3/publication/2890160_Fundamentals_Of_Image_Processing/links/09e4150b48158c78b2000000/Fundamentals-Of-Image-Processing.pdf

[2] HLAVÁČ, Václav a Miloš SEDLÁČEK. *Zpracování signálů a obrazů.*[online]. Praha: Vydavatelství ČVUT, 1999 [cit. 2017-11-22]. Dostupné z: http://www.ugn.cas.cz/~kolcun/G/azro/txt/cvut_Hlavac_Sedlacek.pdf

[3] VERNON, David. *Machine Vision: Automated Visual Inspection and Robot Vision* [online]. Prentice Hall International, 1991 [cit. 2017-11-25]. ISBN 0-13-543398-3. Dostupné z: http://homepages.inf.ed.ac.uk/rbf/BOOKS/VERNON/toc.htm

[4] TOMORI, Zoltán a Matej NIKOROVIČ. Počítačové videnie v praxi [online]. Košice: Ústav experimentálnej fyziky SAV, 2016 [cit. 2017-11-26]. Dostupné z: http://home.saske.sk/~tomori/Downloads/Poc_videnie/PV_2016.pdf

[5] SONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image Processing, Analysis, and Machine Vision*[online]. Cambridge: Champman and Hall Computing, 2008[cit. 2017-11-23]. ISBN 978-0-945-24428-7. Dostupné z: http://libgen.io/_ads/F9CA8B507D2EB59C665847C0D575B83D

[6] LIM, Jae S. Two-dimensional signal and image processing. Englewood Cliffs, N.J.: Prentice Hall, c1990. ISBN 0139353224.

[7]PARKER, James R. *Algorithms for image processing and computer vision*. New York: John Wiley, c1997. ISBN 0-471-14056-2.

[8]Canny, John, "A Computational Approach to Edge Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence,Vol. PAMI-8, No. 6, 1986, pp. 679-698.

[9] OpenCV: Canny edge detector [online]. [cit. 2017-12-10] Dostupné z: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html#result

[10] SOLEM, Jan E. Programming Computer Vision with Python. O'Reilly Media, 2012. ISBN 1449316549.

[11] OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform) [online]. [cit. 2018-14-5] Dostupné z: https://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html

[12] ŠIKUDOVÁ, Elena. *Počítačové videnie: detekcia a rozpoznávanie objektov*.[online] Praha: Wikina, 2014.[cit. 2018-14-5] ISBN 978-80-87925-06-5. Dostupné z: http://sccg.sk/~cernekova/Pocitacove_videnie.pdf

[13] HORÁK, Karel, Ilona KALOVÁ, Petr PETYOVSKÝ a Miloslav RICHTER. Počítačové videní[online]. Brno, 2008[cit. 2017-11-29]. Dostupné z: http://www.uamtold.feec.vutbr.cz/vision/TEACHING/MPOV/Pocitacove_videni_S.pdf

[14] Python: What is Python? Executive Summary [online]. [cit. 2017-12-26] Dostupné z: https://www.python.org/doc/essays/blurb/

[15] Python: Comparing Python to Other Languages [online]. [cit. 2017-12-26] Dostupné z: https://www.python.org/doc/essays/comparisons/

[16] OpenCV: About [online]. [cit. 2017-12-26] Dostupné z: https://opencv.org/about.html

[17] JANÁKOVÁ, Ilona. *Prednáška z predmetu BZVS: Filtrace šumu a poruch* [online]. In: . Brno [cit. 2017-12-28]. Dostupné z: http://midas.uamt.feec.vutbr.cz/ZVS/zvs_cz.php

[18] OpenCV: Canny edge detector [online]. [cit. 2017-12-28] Dostupné z: https://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[19] OpenCV: Harris corner detector [online]. [cit. 2017-12-28] Dostupné z: https://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html

Zoznam elektronických príloh

Elektronická príloha CD obsahuje :

- Kompletný zdrojový kód programu pre sledovanie čiary
- Video nahrávky robotov sledujúcich čiaru
- Všetky obrazy vzorov potrebné pre chod programu
- Dokument PDF s textom bakalárskej práce