UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Optično sledenje objektov s postopki računalniškega vida in mehanizma za usmerjanje kamere

(Optical objects tracking using computer vision methods and camera orientation mechanism)

Ime in priimek: Blaž Gombač
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, september 2016

Ključna dokumentacijska informacija

Ime in PRIIMEK: Blaž GOMBAČ

Naslov zaključne naloge: Optično sledenje objektov s postopki računalniškega vida in mehanizma za usmerjanje kamere.

Kraj: Koper

Leto: 2016

Število listov: 48 Število slik: 22
Število prilog: 1 Število strani prilog: 1 Število referenc: 40

Mentor: doc. dr. Peter Rogelj

Ključne besede: optično sledenje, računalniški vid, krmiljenje servomotorjev, Raspberry Pi, OpenCV, C, C++

Izvleček:
Zaključna naloga predstavlja sistem za optično sledenje objektov s postopki računalniškega vida in mehanizma za usmerjanje kamere. Cilj je narediti mehanizem, ki bo sledil objektu zanimanja tako, da bo le ta vedno v centru slike. Optično sledenje objektov zahteva uporabo računalniškega vida, s pomočjo katerega najprej identificiramo objekt, ki mu želimo slediti. Predpostavimo, da je naš objekt rdeča žogica, in tej želimo slediti. Sliko segmentiramo tako, da so na njej vidni samo rdeči objekti (primer: vse kar je na segmentirani sliki označeno z belo barvo, predstavlja rdečo barvo, vse kar je na sliki označeno z črno, pa predstavlja katerokoli drugo barvo razen rdeče). Ob zaznavi določenega objekta lahko z mehanizmom, ki krmili servomotorje, kamero usmerimo v smeri zaznanega objekta. Mehanizem je sestavljen iz mini računalnika Raspberry Pi ter mehanizma za usmerjanje kamere — pan-tilt enote, ki ga premikata dva servomotorja in na katerem je pritrjena kamera.

Key words documentation

Name and SURNAME: Blaž GOMBAČ

Title of the final project paper: Optical objects tracking using computer vision methods and camera orientation mechanism

Place: Koper

Year: 2016

Number of pages: 48 Number of figures: 22
Number of appendices: 1 Number of appendix pages: 1 Number of references: 40

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: optical tracking, computer vision, servomotor control, Raspberry Pi, OpenCv, C, C++

Abstract:
The thesis presents system for optical tracking objects with method of computer vision and mechanisms for directing camera. The purpose of the thesis is to create a mechanism that will be able to follow the object so that the object is always in the centre of the image. Optical tracking of objects requires the use of computer vision with which we first identify the object that we want to follow. We will assume that our object is a red ball that we want to follow. We have to rewrite our image in such type of picture on which only red objects are visible (for example everything on the segmented image that is marked with white colour presents red colour in reality and everything on the segmented image that is marked with black colour presents any other colour except for red). When we detect our object we can follow it with specific mechanism which controls the servomotors and with that we can direct the camera in the direction of the detected object. The mechanism consists of a mini-computer Raspberry Pi and mechanism for directing camera — pan-tilt unit, which is being moved by two servomotors, and on which we have our camera.

Zahvala

Zahvaljujem se mentorju doc. dr. Petru Roglju za pomoč pri izboru in stokovni pomoči pri izdelavi zaključne naloge.

Zahvaljujem se tudi družini, ki mi je omogočila študij, in dekletu za podporo ter pomoč med študijem in pisanjem zaključne naloge.

Kazalo vsebine

Kazalo slik

Kazalo prilog

Seznam kratic

itd.	in tako dalje
$1D$	enodimenzionalni prostor
$2D$	dvodimenzionalni prostor
$3D$	trodimenzionalni prostor
npr.	na primer
RGB	barvni prostor sestavljen iz rdeče, zelene in modre barve (ang. red, green and blue)
HSV	barvni prostor sestavljen iz odtenka, nasičenosti in vrednosti barve (ang. hue, saturation and value)
SoC.	sistem na čipu (ang. System on Chip)
$GPIO$	glavni namen vhodov in izhodov (ang. General Purpose of Input/Output)
A/D	analogno digitalni pretvornik (ang. Analog-to-digital converter)
A/D	digitalno analogni pretvornik (ang. Digital-to-analog converter)
$I2C$	integrirano vezje (ang. Inter-integrated circuit)
SDA	podatkovna linija (ang. Serial Data Line)
SCL	urina linija (ang. Serial Clock Line)
PWM	pulzna široka modulacija (ang. Pulse Width Modulation)
LED	svetleča dioda (ang. Light-emitting diode)
DC	enosmerni motor (ang. Direct Current Motor)
HD	visoka ločljivost videa (ang. High Definition)
USB	univerzalno serijsko vodilo (ang. Universal Serial Bus)
CSI	vmesnik za kamero (ang. Camera Serial Interface)
FTP	protokol za prenos datotek (ang. File transfer protocol)
IP	internetni protokol (ang. Interner Protocol)
RPI	Raspberry Pi

1 Uvod

Računalniški vid je tehnologija, ki poizkuša z mikroprocesorskimi elektronskimi napravami zaznavati in prepoznavati slike, oziroma povedano drugače videti in posledično tudi razumeti. Tehnologije se vedno bolj izboljšujejo, predvsem se izboljšujejo algoritmi, procesorska moč, itd. Tehnologija ne more nadomestiti človeškega vida, z njim jo težko primerjamo, saj ima vsaka svoje prednost in slabosti. Bistveni razliki med računalniškim in človeškim vidom sta prilagodljivost in natančnost. Človeški vid je neverjetno prilagodljiv (različna smer, jakost, barva osvetlitve, itd.), medtem ko je računalniški vid manj fleksibilen, a zato zelo natančen (lahko se ga npr. uporablja za zelo natančne in hitre meritve dimenzij barve, lokacije, itd.), česar človeško oko ni sposobno. Da tehnologije ne moremo primerjati s človeškim vidom, je razvidno iz kar nekaj primerov. Eden izmed njih je, da človeško oko lahko zazna, oziroma vidi samo barvno sliko, računalniški vid pa lahko s pomočjo določenih postopkov predela pridobljeno sliko in na njej prikaže samo objekte določeneice barve. Zaradi te prednosti se tehnologije poslužujemo za nadomeščanje človekovega zaznavanja. Taki primeri uporabe so pri nogometu, če sodnik zadetka ne vidi, oziroma zazna, lahko s pomočjo računalniškega vida prepozna, ali je bila žoga čez črto ali ne in tako določi, ali je zadetek veljaven ali ne. Računalniški vid se s pridom uporablja tudi za raznovrstne analize slik. Najbolj je to uporabno v medicini, kjer se pri izvajanju različnih slikovnih tehnik s pomočjo računalniškega vida slike pregleda ter določi, kje je prišlo do odstopanj. Lahko se uporablja tudi za kontrole izdelkov, tako da se narejeni izdelek slika in s pomočjo računalniškega vida preveri, ali je izdelek narejen po predpisih in se ga na podlagi tega sprejme ali zavrne. Iz teh primerov je razvidno, da se računalniški vid s pridom uporablja na veliko različnih področjih in prihaja vedno bolj v veljavo.

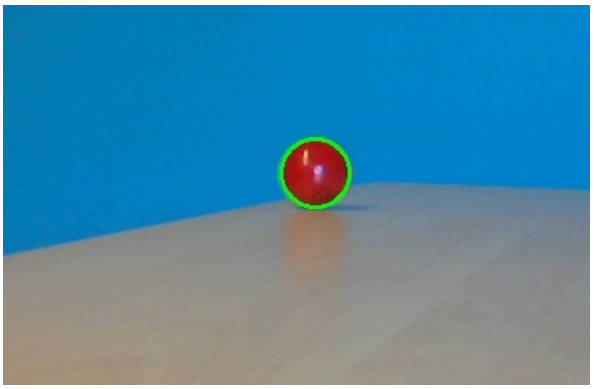
Cilj zaključne naloge je ustvariti sistem, ki bo z gibanjem mehanizma kamere v realnem času sledil objektu tako, da bo objekt vedno v središču slike. S pomočjo kamere zajemamo sliko, ki jo program, ki teče na mini računalniku razpozna in določi lokacijo objekta. Temu objektu sledimo s pomočjo mehanizma, z obračanjem kamere v horizontalni (levo in desno - pan) in vertikalni (gor in dol - tilt) smeri .

Zaključna naloga vsebuje pet poglavij, vključno z Uvodom. Uvodu sledi poglavje z naslovom Sledenje objektom, v katerem je predstavljena teorija za splošno razumevanje delovanja sistema. Za tem poglavjem je poglavje, ki opisuje sestavo sistema. V

četrtem poglavju je predstavljen sam program in njegovo delovanje, zadnje poglavje pa predstavlja zaključek oziroma ugotovitve zaključne naloge.

2 Uvod v računalniški vid in sledenje objektov

Če želimo nekemu objektu slediti, si moramo najprej razložiti, kako deluje računalniški vid, kako poteka prepoznavanje objekta s pomočjo segmentacije in kako objekte sledimo.



Slika 1: Posnetek sledenja žogice, kjer sistem dogajanje spremlja s kamero in označi žogico s krogom zelene barve.

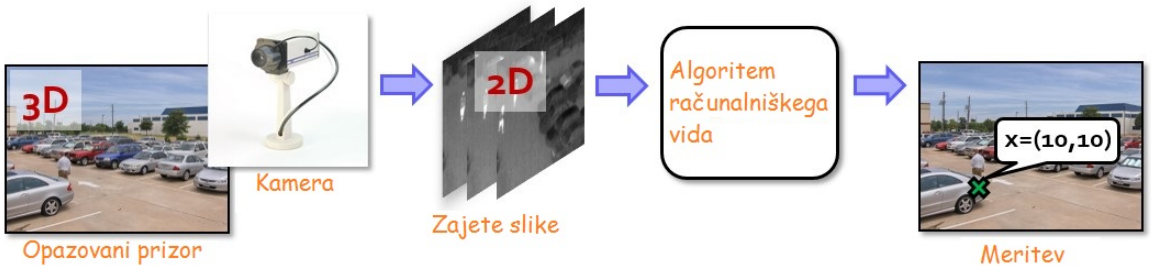
2.1 Računalniški vid

Računalniški vid je področje, ki preučuje kako iz 2D slik interpretirati, rekonstruirati in razumeti 3D prizor. [1] Omogoča računalnikom, da s pomočjo vida prepoznavajo osebe, predmete in našo neposredno okolico ter da se v njej lahko gibajo, zato lahko rečemo, da deluje podobno kot človeški vid. [2]

2.1.1 Koncept računalniškega vida

Računalniški vid deluje po konceptu prikazanem na Sliki 2. Ta koncept deluje na podoben način kot človeški vid, s katerim s pomočjo očesa zaznamo opazovan prizor v tridimenzionalnem prostoru. Imamo neko opazovano okolico v tridimenzionalnem

prostoru , ki jo zazna kamera in nam jo generira v zaporedje dvodimenzionalnih slik. Čeprav lahko tudi pri človeškem vidu opravimo različne meritve, ne moremo biti tako natančni kot pri računalniškem vidu. S pomočjo kamere je zajet opazovan prizor, za primer vzamemo parkirišče prikazano na Sliki 2. Ta prizor se generira v zaporedje dvodimenzionalnih slik, ki so vhodni podatki za algoritem. V prikazanem primeru se izvede algoritem za prepoznavanje lokacije oziroma pozicije človeka na opazovanem prizoru. Tako pridemo do meritev, ki nam povedo, da se na koordinatnem sistemu dejanskega prostora, torej parkirišča, človek nahaja na koordinati x=(10,10). [3, 4]



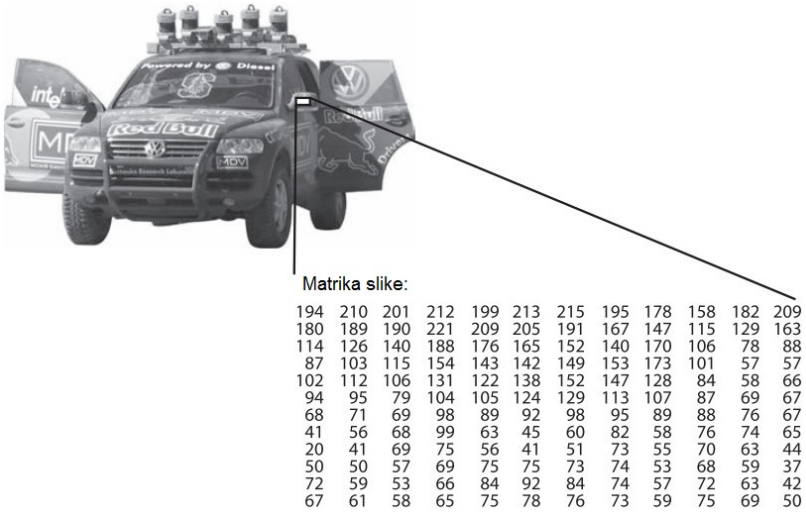
Slika 2: Primer koncepta računalniškega vida. [3]

V naši zaključni nalogi uporabljamo tak koncepta računalniškega vida, kot je na Sliki 2. Na dvodimenzionalnih slikah izvajamo algoritem za prepoznavanje objektov, ki nam nato na sliki vrne lokacijo prepoznanega objekta.

2.1.2 Delovanje računalniškega vida

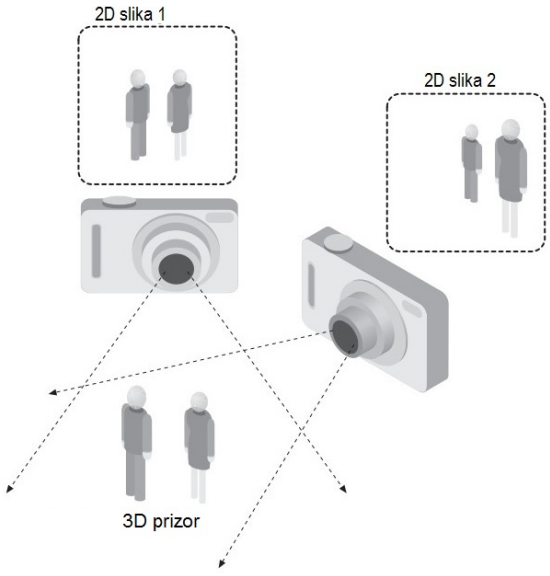
Za boljše razumevanje delovanja računalniškega vida si poglejmo, kako deluje človeški vid. Z njim upravljajo možgani, ki signal vida razdelijo na več kanalov, ki vsebujejo različne informacije. Naši možgani so sestavljeni iz različnih sistemov in eden izmed teh sistemov je sistem pozornosti. Ta sistem določa, na katere dele slike naj bodo možgani bolj pozorni in naj jih preučijo, ter kateri deli slike v tem trenutku niso pomembni in jih lahko izpustijo. Velik vpliv na percepcijo vida imajo tudi številni senzorji za nadzor mišičnih vlaken. Ti skupaj z vidno informacijo ter izkušnjami, ki smo jih že doživeli, ustvarijo v možganih skupno sliko. [4, 5]

Računalnik s pomočjo računalniškega vida prejme iz kamere matriko oziroma polje številk, ki prikazuje svetlost oziroma barve, kot je prikazano na Sliki 3. Ljudje na Sliki 3 vidimo stransko ogledalo na voznikovi strani, računalnik pa "vidi" samo matriko. Čeprav imajo števila znotraj matrike veliko šuma in nam dajejo malo informacij, lahko računalnik s pomočjo matrike "vidi" obliko ogledala. [4, 5]



Slika 3: Primer ogledala avtomobila, ki ga vidi računalnik kot matriko. [4]

Eno izmed težav pri računalniškem vidu nam predstavlja različen pogled na prizor, kar je prikazano na Sliki 4. Iz slike lahko vidimo, da nam dva različna zorna kota predstavljata dve različni sliki istega 3D prizora. Iz tega lahko ugotovimo, da nam en sam 2D pogled na 3D prizor ne more enovito rekonstruirati prizora. [4]



Slika 4: Primer različnega pogleda na isti prizor. [4]

2.1.3 Primeri uporabe računalniškega vida

Računalniški vid se v današnjem času uporablja v različne namene. Nekateri izmed njih so: [5]

- **Optično prepoznavanje znakov**

  Računalnik s pomočjo kamere slika ročno napisano črko oziroma besedo. Posamezno črko primerja s črkami, ki jih ima vnesene v svojem spominu, in ob ujemanju le-teh s slikano črko izpiše prepoznano črko ter tako celotno besedo. Primer uporabe je poštna pošiljka, ki prispe v center za razvrščanje le-teh. S pomočjo sistema prepoznajo naslov prejemnika in jo uvrstijo na določeno mesto. (Slika 5a)

- **Prepoznavanje objekta za nakup**

  Nekatere izdelke v trgovinah je mogoče kupiti posamično ali pa v paketu (npr. mleko, sokovi v tetrapakih, steklenice v paketih, itd.). Ker so ti izdelki navadno težki, se jih na vozičku odloži spodaj. Da ljudem ni potrebno dvigovati težjih izdelkov, imajo blagajne na višini spodnje police vozička vgrajen sistem za prepoznavanje izdelkov. Na skupni embalaži izdelkov so določeni znaki, ki so za vsak izdelek drugačni. Tako računalnik samodejno zazna izdelek, ki je na vozičku in ga samodejno prišteje trgovki v skupen račun. (Slika 5b)

- **Medicinsko slikanje**

  S pomočjo računalniškega vida se pregleduje slike organov, za katere imamo že v naprej določene referenčne vrednosti (npr. obliko, velikost, zgradbo, itd.). Glede na njihove značilnosti s pomočjo te tehnologije lahko pregledujemo posnetek organa in zaznavamo morebitna odstopanja od povprečnih značilnosti. Takšen primer je prepoznavanje tumorjev v možganih. (Slika 5c)

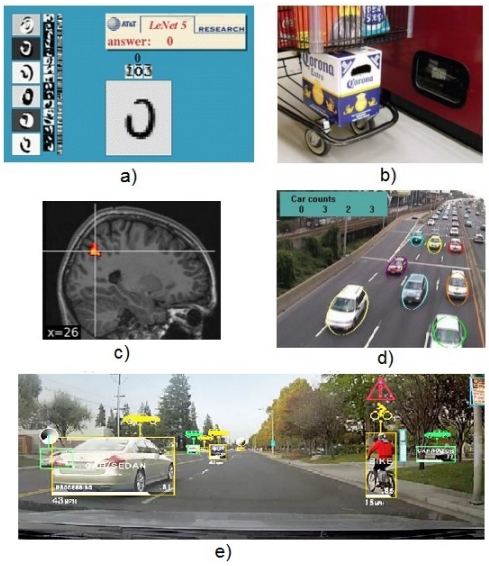
- **Sledenje objektov**

  Tehnologija prepoznava objekte ter jim sledi. Omogoča prepoznavanje in sledenje več objektom hkrati, kar omogoča tudi štetje le-teh. Primer uporabe te tehnologije je kamera na avtocesti, s katero lahko izračunamo količino prometa na njej. Računalniški vid prepozna avtomobile in jih prešteje. Ta sistem omogoča tudi prepoznavanje gneče na cesti, saj zazna, ali se avtomobili gibljejo ali ne. (Slika 5d)

- **Avtomobilska varnost**

  Računalniški vid s pomočjo kamer na avtomobilu odkriva nepričakovane ovire in omejitve. S kamer, ki zajemajo sliko pred, za in ob strani avtomobila prepozna

pešce, kolesarje in druga vozila na cesti in tako avtomatsko zavira, da ne bi prišlo do prometne nesreče. Ravno tako prepoznava table za omejitev na cestah in na avtomobilskem zaslonu prikaže omejitev hitrosti na cesti, po kateri se trenutno vozimo. (Slika 5e)



Slika 5: Primeri uporabe računalniškega vida. [5–7]

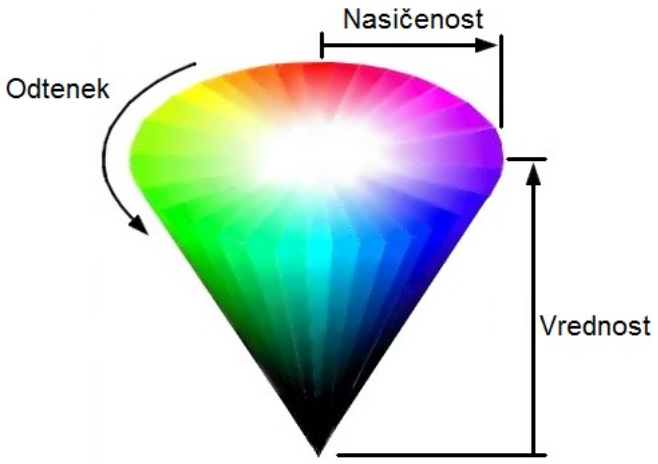
2.2 Segmentacija objekta

Segmentacija in prepoznavanje objekta je naloga računalniškega vida. Objekt si lahko predstavljamo kot področje na sliki z določenimi lastnostmi. Segmentacija slike je delitev slike na regije, ki ustrezajo določenim lastnostim. Vsak piksel na sliki je eden izmed številnih pikslov ustreznega področja. Segmentacija je eden izmed pomembnejših korakov pri analizi slike. Prepoznan objekt je sestavljen iz mnogih slikovnih pikslov. Ob dobri izvedbi segmentacije so vse druge faze sledenja objekta veliko enostavnejše. [8–10]

Segmentacije se razlikujejo glede na uporabljene lastnosti za segmentacijo, npr. obliko, velikost, robove, barvo, itd. Ena izmed možnosti prepoznavanja objekta je s pomočjo barvne segmentacije in to smo uporabili v naši zaključni nalogi. Barvno segmentacijo smo izbrali zato, ker nam barva pove veliko informacij. Kot primer si vzamemo objekt rdeče barve. Slika, na kateri se nahaja ta objekt, je v barvnem prostoru RGB (rdeča, zelena, modra, ang. red, green, blue). Izvedba barvne segmentacije v RGB prostoru je zelo občutljiva na spremembe svetlosti in zato lahko zelo vpliva na

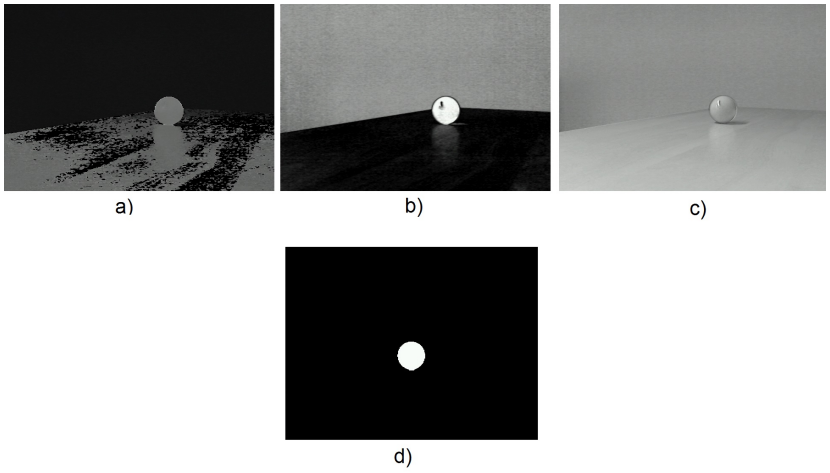
barvo komponente. Sliko je zato priporočeno pretvoriti v barvni prostor HSV (odtenek, nasičenost, vrednost, ang. hue, saturation, value), ki je bolj primeren za obdelavo slike po barvah. Barvni prostor HSV opisuje: [11]

- **odtenek** (ang. hue) opisuje številke, ki določajo položaj barve (npr. pri 240 se začne modra)
- **nasičenost** (ang. saturation) opisuje, kako izrazita je barva, oziroma kakšna je senca barve (modra barva je lahko polno zasičena in kot taka je označena s številko 255, z zmanjšanjem številke pa barvo manj zasičimo)
- **vrednost** (ang. value) opisuje svetlost oziroma intenzivnost barve, oziroma nam pove, kako temna je barva (številka 0 predstavlja črno barvo, z večanjem številke pa se vedno bolj odmikamo od črne barve)



Slika 6: Barvni prostor HSV. [12]

Z razumevanjem vseh treh komponent slike HSV lahko s filtriranjem iz slike odstranimo vse nepotrebne barve in izluščimo samo barvo (npr. rdečo), ki jo potrebujemo. Predpogoj za takšen način segmentacije je, da naš prizor ne vsebuje drugih objektov iste barve (v našem primeru rdeče). Ker prizor ne vsebuje drugih objektov iste barve, je mogoče zanesljivo ločiti iskan objekt od okolice. Tako dobimo segmentirano sliko, na kateri je predstavljena potrebna barva (npr. rdeča) z belo barvo, vse ostale pa s črno barvo. Tako dobimo izluščen objekt, ki mu želimo slediti.



Slika 7: Primer barvne segmentacije: Slika a) prikazuje kanal H barvnega prostora HSV, Slika b) prikazuje kanal S barvnega prostora HSV, Slika c) prikazuje kanal V barvnega prostora HSV in Slika d) prikazuje segmentirano sliko z izluščenim objektom rdeče barve.

2.3 Metode za sledenje objektov

Poznamo več vrst postopkov za sledenje objektov. Med najbolj poznane postopke uvrščamo Meanshift in Camshift, ki se osredotočata na iskanje največje gostote iskanih pikslov na sliki. Da si lažje predstavljamo sledenje objektov, bomo v nadaljevanju pobližje spoznali oba postopka, kljub temu, da ju v zaključni nalogi ne obravnavamo, saj bomo v nalogi predstavili lasten enostavnejši postopek za sledenje objektov, ki bo obravnavan v programskem delu naloge.

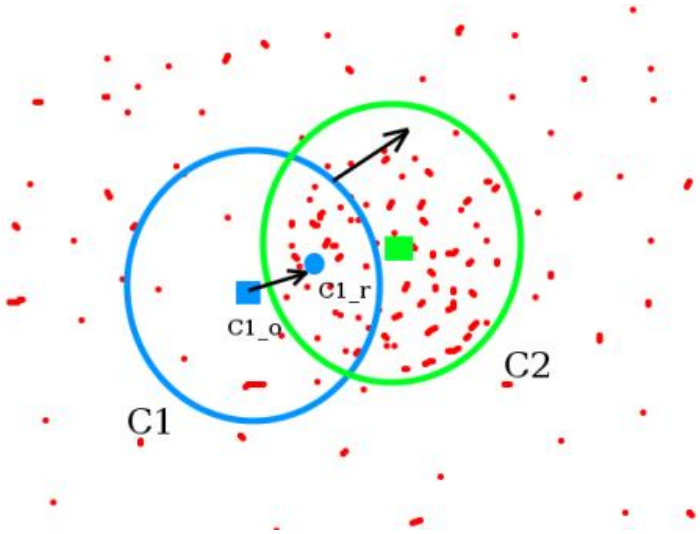
2.3.1 Metoda Meanshift

Algoritem Meanshift je neparametričen način iskanja lokalnega ekstrema v porazdelitvi gostote niza podatkov. Omogoča natančno lokalizacijo in računsko izvedljivost. Deluje tako, da vzame poljuben niz pikslov oziroma lokalno okno, v njem poišče najbolj gosto območje iskanih pikslov, ki nam predstavljajo piksle območja, ki mu želimo slediti (npr. piksle rdeče barve). Nato premaknemo okno tako, da je najbolj gosto območje iskanih pikslov središče okna. Ob tem upošteva samo piksle, ki se nahajajo v lokalnem oknu, ostalih pikslov ne upošteva. Deluje, dokler ne najde najgostejšega območja na sliki, ki nam predstavlja objekt, ki mu želimo slediti. Gostota slike nam predstavlja piksle željenega objekta, ki ga iščemo na sliki. Algoritem se ustavi, ko pridemo do najgostejšega območja iskanih pikslov. [4, 5]

Algoritem upošteva sledeče korake: [4, 5, 13]

- **1** izbere velikost, začetni položaj in obliko (krog, pravokotnik, simetričnost, nesimetričnost, itd.) iskalnega lokalnega okna
- **2** izračuna težišče iskanih pikslov
- **3** postavi okno na točko izračunano v 2. koraku
- **4** ponovitev 2. in 3. koraka do konvergence, oziroma dokler se premikanje okna ne ustavi (vedno se ustavi ob dosegu najgostejše točke)

Primer: [14] Vzamimo poljuben niz pikslov. Dobimo krog, ki ga moramo premakniti na območje z največjo gostoto iskanih pikslov na sliki, kakor je prikazano na Sliki 8.



Slika 8: Delovanje Meanshifta. [14]

Začetno okno je prikazano z modrim krogom in imenom $C1$, njegov center pa je označen z modrim kvadratom z imenom *C1_o*. Ob iskanju največje gostote iskanih pikslov dobimo točko označeno z modro piko in imenom *C1_r*, ki nam predstavlja novo sredino novega namišljenega kroga. Vektor od *C1_o* do *C1_r* nam pove, za koliko moramo premakniti krog, da bo točka največje gostote (*C1_r*) postala središče namišljenega kroga. Postopek ponavljamo, dokler ne dobimo kroga, ki vsebuje največjo gostoto točk na sliki. Končni krog na Sliki 8 je zelen krog z imenom $C2$.

2.3.2 Metoda Camshift

Camshift (ang. Continuously Adaptive Mean Shift) algoritem je adaptadcija Meanshift alforitma za sledenje objektom. Od Meanshifta se razlikuje po tem, da se velikost iskalnega okna prilagaja premikajočemu se objektu. Algoritem je namenjen predvsem sledenju glave ter obraza ljudi. [4, 15]

Algoritem deluje na temeljih Meanshift algoritma in se izračuna po naslednjih korakih: [13]

- **1** izbere začetno lokacijo iskalnega okna
- **2** več ponovitev Meanshift algoritma
- **3** s pomočjo 2. koraka nastavi velikost iskalnega okna
- **4** ponovitev koraka 2 in 3, dokler ne pride do konvergence

Za boljše razumevanje si lahko algoritem Camshift predstavljamo tako, da se iskano okno prilagodi velikosti iskane površine. Če imamo npr. obraz z jasnimi značilnostmi, algoritem samodejno prilagodi okno iskanja na velikost obraza. [14]

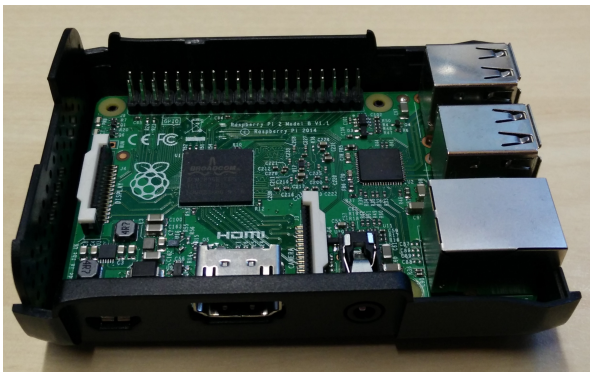
3 Predstavitev in zgradba sistema

Cilj izdelanega sistema je prepoznavanje enostavnih objektov in sledenje le-tem s pomočjo mehanizma za usmerjanje kamere v realnem času. S pomočjo kamere zajemamo sliko. Če želimo na sliki objekt prepoznati in mu slediti, moramo poznati njegove lastnosti, kot je na primer barva. Za prepoznavanje objekta izvedemo barvno segmentacijo v HSV barvnem prostoru. Na dobljeni segmentirani sliki je objekt prikazan z belo barvo, ozadje pa s črno barvo. V primeru, da je ta objekt rdeče barve, nam je rdeča barva na originalni sliki predstavljena z belo barvo na segmentirani sliki, ki predstavlja objekt. Ker za segmentacijo uporabljamo postopek, ki temelji izključno na osnovi zaznavanja barve pričakovanega objekta (v našem primeru rdečega) je pomembno, da na sliki ni drugih rdečih objektov razen željenega. Objektom želimo sledili tako, da vedno ko se le-ta premakne, se premakne s pomočjo servomotorjev tudi mehanizem, ki objekt ponovno postavi na sredino slike. Ob zaznavi objekta mehanizmu izračunamo potrebne krmilne signale za servomotorje, da se le ti premaknejo po vodoravni in navpični smeri za določen signal. Tako s pomočjo krmilnih signalov premikamo servomotorje v smeri gibanja našega objekta. Čeprav lahko objekt premikamo poljubno po prostoru, mu z našim sistemom ni mogoče slediti popolnoma, saj so motorji omejeni z obsegom rotacije za 180 °. Glavni del sistema predstavlja mini računalnik, na katerem se izvaja programska oprema. Servomotorji so na mini računalnik povezani preko PWM (PWM, ang. Pulse Width Modulation) krmilnika, ki z mini računalnikom komunicira preko I2C (ang. Inter-Intergrated Circuit) vodila po dveh temu namenjenih sponkah GPIO (GPIO, ang. general purpose of input/output) vmesnika mini računalnika.

3.1 Mini računalnik

Mini računalnik Raspberry Pi je nekajkrat manjši od osebnih računalnikov, je v velikosti kreditne kartice. Ima enake značilnosti in podobno tehnologijo kot mobilne naprave, namenska zabavna elektronika in osebni računalniki. Mini računalniki spadajo med mikroračunalnike, ki se jih uporablja predvsem v industriji. Večinoma so to računalniki v enem čipu in jih označujemo s kratico SoC (sistem na čipu, ang. System on Chip). Mini računalnik Raspberry Pi posnema zgradbo osebnega računalnika, vendar je vseeno manjši, ponuja okrnjeno funkcionalnost in posledično porabi malo energije. [16] Tako

ima torej vse značilnosti osebnih računalnikov, kot so priklop na monitor ali televizijo, uporaba standardne tipkovnice in miške, uporaba zvočnikov in povezave na internet. Mini računalnik je veliko manj zmogljiv kot osebni računalnik, ima pa eno veliko prednost, in sicer vhodno-izhodna vrata (GPIO), ki jih bomo še podrobneje spoznali v nadaljevanju. Razvit je bil z namenom pomoči ljudem pri raziskovanju računalništva in učenju programskih jezikov, kasneje pa se je začel uveljavljati tudi v industriji. [17]



Slika 9: Mini računalnik Raspberry Pi.

3.2 Vhodno-izhodna vrata (GPIO)

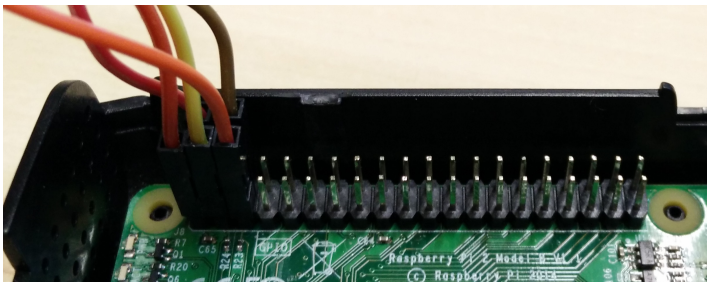
Kot smo že prej omenili, so ena za nas najpomembnejših lastnosti Raspberry Pi mini računalnika vhodno-izhodna vrata (GPIO). Te sponke predstavljajo vmesnik med mini računalnikom in zunanjim svetom. GPIO vmesnik ima 40 sponk, od tega jih je 26 vhodno-izhodnih, ostale pa so sponke za napajanje (ang. power) in ozemljitev (ang. ground). Vhodno-izhodne sponke so digitalne in omogočajo priklop digitalnih senzorjev in drugih naprav. V primeru potrebe analognih vhodnih ali izhodnih naprav lahko te pridobimo s priklopom ustreznih A/D (ang. Analog-to-digital converter) ali D/A (ang. Digital-to-analog converter) pretvornikov. Najenostavnejši način priklopa zunanjih naprav je z uporabo I2C (ang. Inter-Intergrated Circuit) vodila. [18]

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Slika 10: Sponke GPIO vmesnika mini računalnika Raspberry Pi. [19]

3.3 I2C vodilo

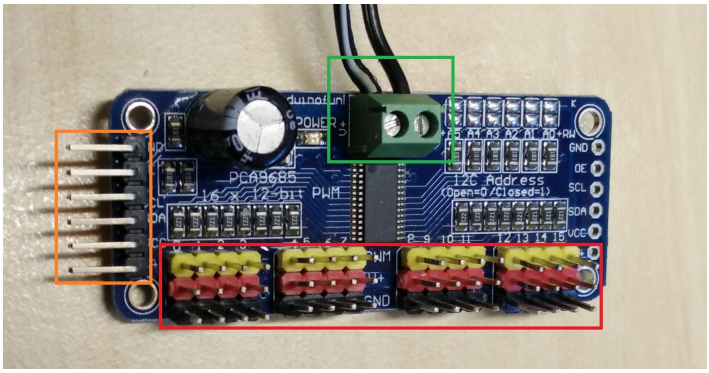
I2C (ang. Inter-Intergrated Circuit) komunikacijsko vodilo je razvil Philips. Uporablja se ga za komunikacijo pri nizko hitrostnih napravah. Je eno izmed najpogosteje uporabljenih standardov za komunikacijo enega čipa z drugim. Za takšno komunikacijo potrebujemo dve liniji. Ena je namenjena prenosu podatkov (SDA ang. Serial Data Line) druga pa urinemu signalu (SCL ang. Serial Clock Line). [20] Vodilo je namenjeno povezavi več naprav skupaj preko dveh linij na Raspberry Pi. Vsaka naprava povezana preko I2C vodila ima svoj unikaten naslov, ki se določi strojno. [21, 22]



Slika 11: I2C vodilo povezano na GPIO vmesnik.

3.4 Krmilnik za pulzno široko modulacijo

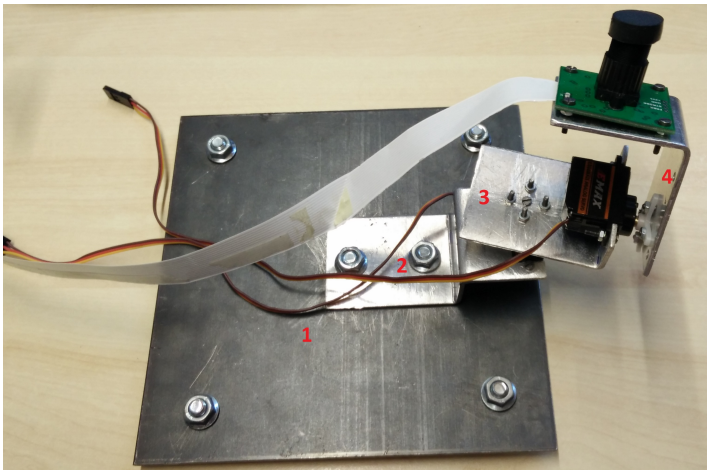
Krmilnik za pulzno široko modulacijo ali PWM (ang. Pulse Width Modulation) je na mini računalnik povezan s pomočjo I2C vodila in omogoča hkratno povezavo 16-ih naprav (v našem primeru povezavo servomotorjev). Delovanje krmilnika upravljamo programsko. Za krmiljenje vsakega motorja potrebujemo dve 12 bitni vrednosti, ki se hranita v štirih 8 bitnih registrih $LEDX\_ON\_L$ (nizki bajt 12-bitne vrednosti za prehod iz nizkega v visoko stanje za kanal števila x, kjer je x od 0 do 15), $LEDX\_ON\_H$ (visoki bajt 12-bitne vrednosti za prehod iz nizkega v visoko stanje za kanal števila x, kjer je x od 0 do 15), $LEDX\_OFF\_L$ (nizki bajt 12-bitne vrednosti za prehod iz visokega v nizko stanje za kanal števila x, kjer je x od 0 do 15) in $LEDX\_OFF\_H$ (visoki bajt 12-bitne vrednosti za prehod iz visokega v nizko stanje za kanal števila x, kjer je x od 0 do 15), oziroma z njimi določamo čas začetka in konca pulza. Ostali pomembni registri so MODE1 namenjeni nadzoru spanja, ponovnem zagonu, avtomatskem inkrementu in naslavljanju, MODE2 nadzoru inverzije, izhodno stanje in vrsta pogona. Pomembni so še registri za dodatne naslove $SUBADDR1$, $SUBADDR2$, $SUBADDR3$ in $ALLCALL$ ter register PRE_SCALE za nastavitev frekvence. Za olajšanje dela s PWM krmilnikom, do le tega lahko dostopamo preko I2C vodila, za katerega uporabljamo knjižnico $i2c-dev$. Prvotno so krmilnik uporabljali za krmiljenje LED (ang. Light-emitting diode) svetilk, vendar se ga danes uporablja v različne namene. [23–25]



Slika 12: Krmilnik PWM, na katerem je z zelenim pravokotnikom označen priklop za napajanje, z oranžnim pravokotnikom priklop na mini računalnik preko I2C vodila in z rdečim pravokotnikom priklopi za servomotorje.

3.5 Mehanizem za usmerjanje kamere

Mehanizem za usmerjanje kamere oziroma pan-tilt enota je sestavljena naprava, ki s premikanjem servomotorjev omogoča kameri, da je vedno usmerjena v objekt. S pomočjo servomotorjev omogoča mehanizem obračanje kamere po horizontalni (levo in desno - pan) in po vertikalni smeri (gor in dol - tilt) ter tako omogoča slednje gibanju nekega objekta.



Slika 13: Pan-tilt naprava.

Mehanizem za našo zaključno nalogo je sestavljen iz naslednjih delov (številke se ujemajo s številkami na Sliki 13):

- **1** Železne plošče, ki je nekoliko večja od drugih komponent. Iz železa je izdelana zato, ker potrebujemo težo, ki stabilizira mehanizem, kadar pride do gibanja, da se le ta ne prevrne. Na plošči so razvidni tudi štirje vijaki, ki služijo kot noge plošče, da je le ta bolj stabilna.

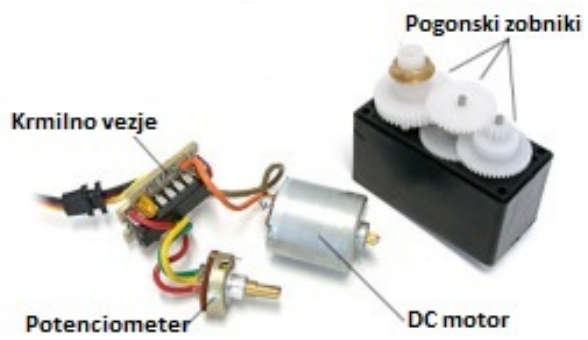
- **2** Aluminijastega Z-profila, ki predstavlja stojalo za servomotor in je na železno ploščo pritrjen z dvema vijakoma. Servomotor je na profil pritrjen tako, da se spodnji del motorja nahaja 1 cm od železne plošče. Ta servomotor služi za premikanje mehanizma v horizontalni osi.

- **3** Aluminijastega L-profila, ki prav tako predstavlja stojalo za servomotor in na katerem je kolo za vrtenje, ki je pritrjeno na spodnjo stran stojala ter na servomotor. Za dodatno stabilnost je stojalo še z vijakom pritrjeno na rotor motorja. Na profil je pritrjen servomotor, ki služi premikanju mehanizma v vertikalni osi.

- **4** Še enega aluminijastega L-profila, ki predstavlja stojalo za kamero, na katerem je kolo za vrtenje, ki je pritrjeno na spodnjo stran stojala in na servomotor. Za dodatno stabilnost je stojalo z vijakom pritrjeno na rotor spodnjega motorja. Na L-profilu se nahaja tudi kamera, s pomočjo katere zajemamo slike.

3.6 Servomotorji

Servomotor je majhna električna naprava, ki je priklopljena na PWM krmilnik. Krmilimo ga s krmilnimi signali, ki se pokažejo z zasukom motorja. Servomotor potrebuje en priklop s tremi sponkami in sicer +5V in ničelna spoka (GND) ter priklop za krmilni (PWM) signal. Pulzno širinski signal določa, v katero smer je gred motorja obrnjena. Za take motorje je značilno, da se lahko gred motorja od sredinske lege obrne le za kot 90° v obe smeri oziroma v našem primeru za 180° gibanja servomotorjev. [26] Obstajajo tudi servomotorji, ki omogočajo premik okrog celotne svoje osi za 360°. Ko je servomotor v srednji legi, ima enak kot vrtenja v obe smeri, čemur pravimo nevtralni položaj servomotorja. Pulzno širinski signal poslan motorju določa lego gredi motorja in glede na dolžino poslanega signala se motor obrne na željeni položaj. Motorji tipično zahtevajo PWM signal ferkvence 50Hz, kar pomeni periodo 20ms. Če pride do spremembe dolžine pulza krmilnega signala se motorji premaknejo. Ko se gred motorja nahaja v željenem položaju, se gibanje ustavi. [27]



Slika 14: Komponente servomotorja. [27]

Servomotor je v bistvu enosmerni motor (DC motor, ang. Direct Current Motor). Za boljše razumevanje delovanja servomotorja, ga moramo razstaviti in pogledati njegove komponente, ki so:

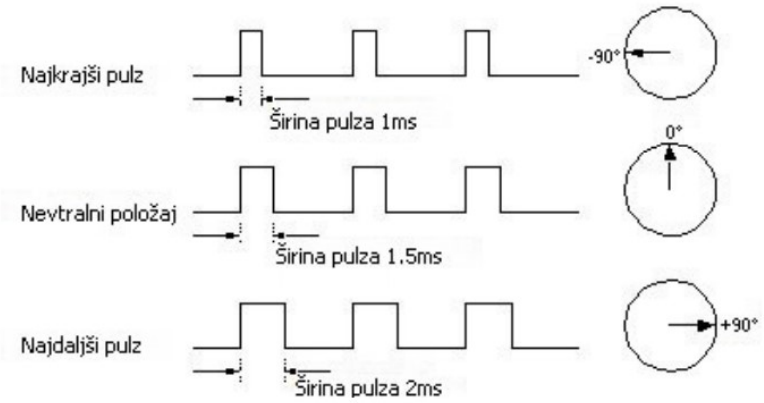
- krmilno vezje

- potenciometer
- enosmerni motor
- pogonski zobniki (mehanizem za krmiljenje)

Krmilno vezje, na katerega je priključen potenciometer, ustvari servomotor tak, da se vrti v skladu z našimi željami. Določa smer vrtenja in med vrtenjem motorja potenciometer spremeni upornost in tako lahko krmilno vezje natančno regulira, koliko gibanja potrebuje in v katero smer. Pogonski zobniki oziroma mehanizem za krmiljenje služi za bolj natančno vrtenje oziroma premikanje. [27]

3.6.1 Krmiljenje servomotorjev s pulzno širinsko modulacijo

Servomotorje krmilimo s pulznimi širinskimi signali. Tipični servomotorji pričakujejo posodabljanje pulznih širinskih signalov vsakih 20 ms z impulzno širino med 1 ms in 2 ms (med 0 ° in 180 °), pri čemer je osnovna ferkvenca 50Hz. Če je impulzna širina singnala 1.5 ms, se ta nahaja v nevtralnem položaju (90 °), v katerem ima servomotor enako možnost vrtenja v smeri urinega kazalca ali proti njemu. Pulz je najkrajši takrat, ko je impulzna širina signala 1 ms in je pozicija servomotorja na 0 °, najdaljši pa takrat, ko je impulzna širina signala 2 ms ter pozicija servomotorja 180 °. [26, 28]

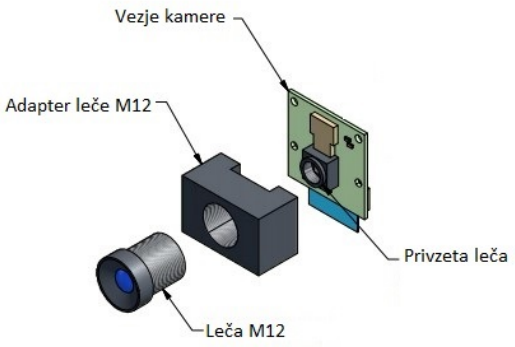


Slika 15: Različne pozicije glede na širino pulza krmilnega signala. [26]

3.7 Kamera

V naši zaključni nalogi uporabljamo Raspberry Pi kamero (RPI), ki je namenjena temu mini računalniku. Kamera deluje z objektivi M12, možnostjo full HD (ang.

High Definition) resolucije 1080p in senzorjem s 5 milijonov slikovnih pikslov (ang. megapixel).



Slika 16: Komponente kamere. [29]

Kamera je sestavljena iz: [29]

- vezja kamere (ploščice z vezjem)
- privzete leče
- adapterja leče M12
- leče M12

V zaključni nalogi uporabljamo ultra širokokotno lečo, poznano tudi pod imenom ribje oko, (ang. fisheye) ki je namenjena zajemanju široke panoramske slike.

Kot alternativno rešitev lahko uporabimo tudi spletno kamero. Vendar uporaba le-te sistemu prinaša drugačne lastnosti.

Razlike med kamero Raspberry Pi in spletno kamero (za primerjavo smo uporabili spletno kamero Logitech C270): [30–34]

- RPI kamera se na mini računalnik poveže preko vmesnika za kamero (CSI, ang. Camera Serial Interface), medtem ko se spletna kamera poveže preko univerzalnega serijskega vodila (USB, ang. Universal Serial Bus).
- Spletna kamera zajema slike z ločljivosjto 3 milijonov slikovnih pikslov, kar predstavlja sliko velikosti 2048 x 1536 slikovnih pikslov, RPI kamera pa z ločljivostjo 5 milijonov slikovnih pikslov, kar predstavlja sliko velikosti 2592 x 1944 slikovnih pikslov.

- RPI kamera zajema video z ločljivostjo full HD (1080p), spletna kamera pa zajema video z ločljivostjo HD (720p).
- Zorni kot spletne kamere je 60 °, medtem ko je zorni kot RPI kamere 65 °. RPI kamera ima za razliko od spletne kamere možnost menjave leč (M12). Ker smo v naši zaključni nalogi uporabljali ultra širikokotno lečo, smo si tako zorni kot povečali na 180 °.
- Za delovanje RPI kamera potrebuje knjižnico *raspicam*, medtem ko spletna kamera ne potrebuje nobenih dodatnih knjižnic.

4 Izvedba programskega dela

V tem poglavju je predstavljen programski del sistema in njegovo delovanje. Celoten programski del je napisan v programskem jeziku C in C++. Za lažje pisanje oziroma programiranje in razumevanje smo uporabili programski okolji Eclipse in Microsoft Visual Studio 2015. Zaradi lažjega pisanja in uporabe programskih okolij smo program pisali na računalniku in ne na mini računalniku Raspberry Pi. Zaradi tega je bilo potrebno datoteke prenesti iz računalnika na Raspberry Pi, za kar smo uporabili internetno povezavo, in sicer protokol za prenos datotek (FTP, ang. File transfer protocol) oziroma program, ki ta protokol omogoča FileZilla. Za delo s slikami smo uporabili knjižnico OpenCV. To je odprtokodna knjižnica napisana v C/C++ jeziku, namenjena računalniškemu vidu in strojnemu učenju. [35]

4.1 Programski del sistema

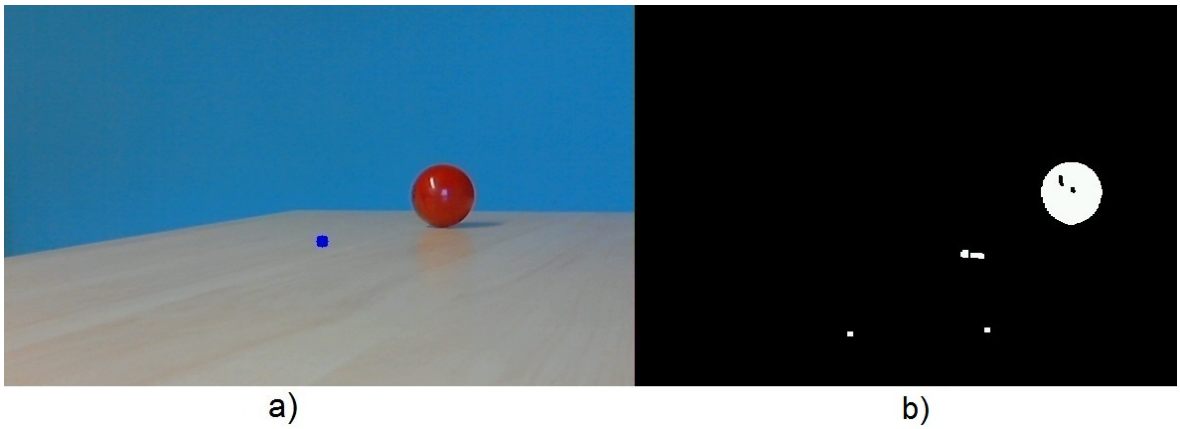
Program je sestavljen iz dveh komponent, ki bodo v nadaljevanju razdeljene na podkomponente. Komponenti sta v medsebojni relaciji odjemalec - strežnik. Prva komponenta služi za prepoznavanje objekta, določanje njegove lokacije in kot odjemalec, ki se poveže oziroma pošilja podatke drugi komponenti oziroma strežniku. Strežnik preko odjemalca dobi lokacijo objekta in izračuna vrednost za premik servomotorjev.

Slika 17: Komponentni diagram sistema.

4.1.1 Prepoznavanje objekta

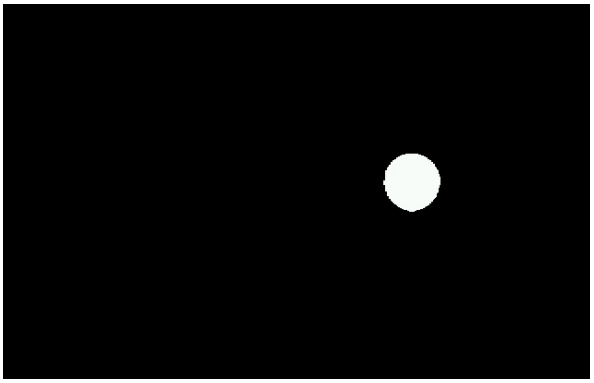
S kamere na mini računalnik Raspberry Pi zajamemo zaporedje slik. Zajem slik in ostale funkcije za delo s slikami so realizirane s pomočjo knjižnice OpenCV. Slika, ki jo zajame kamera, je v barvnem prostoru RGB (prikazano na Sliki 18a). Barvno segmentacijo je lažje izvesti v barvnem prostoru HSV, ker so vse komponente v RGB barvnem

prostoru odvisne od osvetlitve objekta, medtem, ko sta v HSV le dve komponenti in je osvetlitev ločena v komponenti vrednost (ang. value), ki za nas ni pomembna. S tem preidemo iz segmentacije v trirazsežnem prostoru na segmentacijo v dvorazsežnem prostoru, kar je seveda enostavnejše. Zato pretvorimo sliko z barvnega prostora RGB v HSV, kar naredimo s pomočjo funkcije *cvtColor*, ki je namenjen pretvarjanju enega barvnega prostora v drugega. Nato moramo določiti, katere barve želimo segmentirati. V našem primeru je to rdeča barva, ki ima mejne vrednosti: $h1Low = 116$, $h2High = 179$, $s1Low = 77$, $s2High = 255$, $v1Low = 70$ in $v2High = 255$. Mejne vrednosti *h1Low, s1Low, v1Low* imajo vrednost nižjo od mejne vrednosti barvnega prostora HSV in jih zaradi tega štejemo kot ozadje. Mejne vrednosti *h2High, s2High, v2High* pa imajo vrednosti enake ali večje od mejnih vrednosti barvnega prostora HSV in jih prav tako štejemo kot ozadje. S pomočjo vrednosti potrebnih za prepoznavo rdečega objekta in funkcije *inRange* dobimo segmentirano sliko, ki jo predstavimo tako, da je področje segmentiranega objekta obarvano belo in območje ozadja črno (prikazano na Sliki 18b). [36]



Slika 18: Na sliki sta prikazani dve sliki; slika a) predstavlja barvni prostor RGB in slika b) predstavlja prepoznan objekt s pomočjo segmentacije.

Iz Slike 18c) je razvidnega kar nekaj šuma na samem objektu in okrog njega. Šum na objektu prepoznamo kot črne piksle znotraj bele površine žogice na segmentirani sliki. Ta šum je posledica močnejše osvetlitve nekaterih področij objekta (na to vpliva luč, ki je postavljena nad objektom) in zaradi odbleska jo segmentiramo kakor ozadje. Podoben razlog je tudi za nastanek šuma okrog objekta, ki zaradi različnih osvetlitev nekatere delčke površine zazna kot odtenke rdeče barve. Ta dva problema lahko rešimo s pomočjo morfoloških funkcij, ki jih omogoča knjižnica OpenCV. Za odstranitev šuma iz okolice (okrog objekta) uporabimo funkcijo *opening*, za dopolnitev šuma v objektu pa uporabimo funkcijo *closing*. [36–38]



Slika 19: Segmentirana slika brez šuma.

4.1.2 Lokacija objekta

V prejšnjem poglavju smo objekt prepoznali in pridobili segmentirano sliko v črnobelem prostoru, na katerem je z belo barvo prikazan prepoznani objekt. Lokacijo objekta lažje izračunamo, če prej odstranimo ves šum, saj nam le ta vpliva na izračun lokacije (več šuma pomeni večjo napako). Za izračun lokacije objekta bomo uporabili središče oziroma težišče segmentiranega področja, kar bomo ponazorili z momenti ničtega in prvega reda, ki se uporabljajo za opis binarnih območjih. Najprej se posvetimo binarnemu območju (označen z B), za katerega definiramo (številka 0 predstavlja piksle ozadja, številka 1 pa piklse segmentiranega območja): [39]

$$I_B(x,y) = \begin{cases} 0 \\ 1 \end{cases}$$

Moment ničtega reda (M_{00}) podaja površino binarnega območja: [39]

$$M_{00} = \sum_x \sum_y I_B(x,y)$$

Momenta prvega reda (M_{10}, M_{01}) pa nimata neposrednega pomena in sta odvisna od pozicije in velikosti področja: [39]

$$M_{10} = \sum_x \sum_y x I_B(x,y)$$

$$M_{01} = \sum_x \sum_y y I_B(x,y)$$

Momenti segmentirane slike so poimenovani z $oMoments$. Moment ničtega reda $oMoments.m00$ izračuna površino segmentiranega objekta, momenta prvega reda pa

nam izračunata širino ($oMoments.m01$) in dolžino ($oMoments.m10$) segmentiranega objekta. [39]

$area = oMoments.m00$; *//površina*
$m01 = oMoments.m01$; *//širina*
$m10 = oMoments.m10$; *//dolžina*

S pomočjo momentov ničtega in prvega reda izračunamo težišče objekta. Predstavljajmo si, da naša slika predstavlja koordinatni sistem. X koordinato središča objekta izračunamo tako, da dobljen moment x osi (dolžina, $m10$) delimo s površino segmentiranega objekta.

$x = m10/area$;

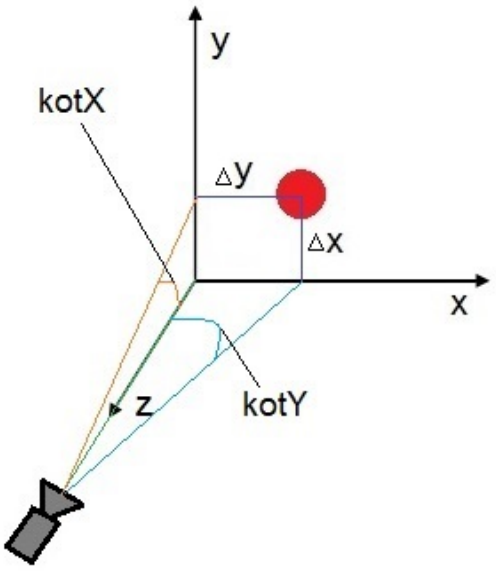
Y koordinato središča objekta izračunamo na enak način kakor x koordinato, vendar uporabimo moment osi y (širina, $m01$).

$y = m01/area$;

Tako dobimo oceno pozicije središča našega objekta. Ker imamo izračunano površino objekta, si lahko za nadaljnje izračune omejimo velikost objekta na želeno površino, kar pripomore k temu, da postopek ne bo prepoznal pozicije manjših objektov od določene velikosti. Paziti moramo, da nimamo na sliki dveh ali več objektov enake barve, saj potem postopek ne more izračunati središča enega objekta, ampak izračuna središče vseh objektov skupaj in tako se središče postavi med njiju. Posledično sistem ne deluje več pravilno.

4.1.3 Določitev kotov med osjo kamere in objekta

Sedaj, ko je lokacija objekta poznana, je potreben izračun kotov odmika objekta od osi kamere, ki bodo kasneje služili za premik servomotorjev. Naprej si poglejmo, kakšna je relacija med pozicijo objekta in kotom.



Slika 20: Nahajanje objekta v realnem koordinatnem prostoru.

Kot je prikazano na Sliki 20 lokacijo objekta (rdeče žogice) pridobimo s pomočjo kotne funkcije *tangens* (nasprotna kateta/priležna kateta). Razdaljo objekta od kamere nam predstavlja vrednost z, Δx nam predstavlja odmik po horizontalni osi in Δy odmik po vertikalni osi. Kot α, ki nam predstavlja odmik po horizontalni osi, izračunamo:

$$\alpha = arctan\frac{\Delta x}{z}$$

Če želimo izračunati odmik po vertikalni osi, pa v enačbo namesto Δx zapišemo Δy. Za majhne kote lahko odvisnost aproksimiramo z linearno fukcijo (z oddaljevanjem objekta kot linearno narašča) odvisnosti pozicije objekta na sliki in kota. Izračun linearno narašča in zato je ta funkcija približek linearne funkcije, zaradi česar lahko namesto kotnih funkcij uporabimo konstanto. To je mogoče, saj je linearna funkcija izračunana s konstanto skoraj enaka linearni funkciji izračunani s kotnimi funkcijami. Z uporabo konstante in ne kotnih funkcij si poenostavimo izračun kotov za odmik objekta od osi. Konstanto za izračun kotov smo pridobili tako, da smo kote, ki smo jih izračunali s pomočjo kotne funkcije, primerjali z uporabo različnih vrednosti s katerimi smo nadomestili kotne funkcije in dobili približno enake vrednosti. Da smo prišli do teh vrednosti je bilo potrebno opraviti več poskusov primerjav iz katerih smo dobili konstanto, ki je v našem primeru 60.

$$\alpha = kx * \Delta x$$

V enačbi nam vrednost kx predstavlja konstanto po x osi (v našem primeru 1/60), Δx pa premik po x osi.

Vrednost $dIzpisX$ predstavlja vrednost x, ki je izračunana v predhodnjem poglavju. Kot po x osi ($kotX$) izračunamo tako, da pridobljeno koordinato nahajanja objekta na x osi ($dIzpisX$), delimo z dobljeno konstanto.

$kotX = dIzpisX/60;$

Na enak način izračunamo kot po y osi ($kotY$), vendar vzamemo koordinato nahajanja objekta na y osi ($dIzpisY$).

$kotY = dIzpisY/60;$

4.1.4 Komunikacija med programskima komponentoma

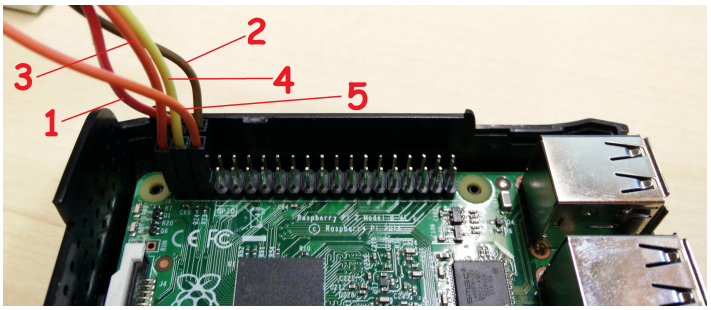
Ob izračunu kotov za odmik objekta od osi kamere nastane problem sočasne izvedbe premika servomotorjev in lociranja objekta na istem mini računalniku. Ta problem smo pričakovali in smo ga rešili že na začetku, in sicer tako, da smo programski del razdelili na dve komponenti. Prva komponenta (odjemalec) pošilja lokacijo premika drugi komponenti (strežnik), ta pa vrednosti sprejme in premika servomotorje. Da smo omogočili komunikacijo med obema komponentama v realnem času, smo ju povezali z UDP spletno vtičnico. To je protokol za prenašanje podatkov, pri katerem odjemalec pošilja pakete strežniku brez preverjanja, ali jih je le ta dobil. Paketi se lahko tudi izgubijo ali pa lahko strežnik prejme pokvarjene pakete, vendar nam to ne predstavlja problema, saj te pakete neprestano pošiljamo. Poleg tega tudi pakete pošiljamo lokalno, tako da med pošiljateljem in prejemnikom ni omrežja. Odjemalcu moramo določiti, na kateri IP (ang. Internet Protocol) naslov (v našem primeru lokalni naslov 127.0.0.1) in vrata (ang. port) (v našem primeru 8080) naj pošilja pakete. Enako deluje tudi strežnik, ki ima prav tako nastavljeni isti dve lastnosti, da lahko ve, kje naj pričakuje pakete. [40] Odjemalec preko UDP vtičnice pošlje strežniku seznam fi, ki je seznam kotov za odmik objekta po horizontalni ($kotX$) in vertikalni ($kotY$) smeri:

$float\ fi[] = \{kotX, kotY\};$

4.1.5 Krmiljenje servomotorjev

Če želimo krmiliti servomotorje, jih moramo najprej povezati na Raspberry Pi. Servomotorja sta povezana na PWM krmilnik, ki je preko I2C vodila povezan na mini računalnik.I2C, vodilo je na mini računalnik povezano preko petih sponk, kot je razvidno s Slike 21. Uporabljamo sponke za napajanje 5V (žica 1), ozemljitev (žica 2),

napajanje 3,3V (žica 3), linijo za prenos podatkov - SDA (žica 4) in linijo za urin signal - SCL (žica 5).



Slika 21: GPIO vmesnik mini računalnika Raspberry Pi z I2C vmesnikom za našo zaključno nalogo.

Preden začnemo krmiliti motorje, je potrebna še inicializacija I2C vodila in krmilnika PWM. Sedaj, ko razumemo, kako so servomotorji priklopljeni na mini računalnik in kako sta I2C vodilo in krmilnik PWM inicializirana, lahko preidemo na krmiljenje servomotorjev.

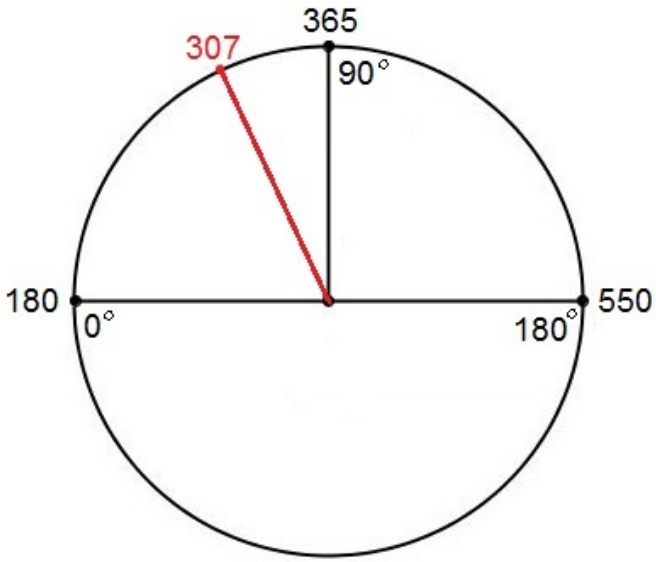
Za boljše razumevanje bomo tukaj še enkrat ponovili, kako servomotorji pridobijo signal za krmiljenje. Mini računalnik generira krmilne signale in jih preko I2C vodila pošilja PWM krmilniku. PWM krmilnik prejme 12-bitne podatke, ki jih nato generira v krmilna signala za servomotorja. Tipični servomotorji pričakujejo nove impulze širine vsakih 20 ms z impulzno širino med 1 ms in 2 ms, pri čemer je osnovna ferkvenca 50Hz. Če bo impulzna širina servomotorja 1.5 ms, bo le-ta na 90 °, če bo 1 ms, bo na 0 ° če bo pa 2 ms, bo na 180 °. Upoštevati pa moramo, da so to servomotorji, ki se lahko premikajo le od 0 ° do 180 °. [28]

Če želimo servomotorje krmiliti, moramo najprej določiti krmilni signal, ki nam predstavlja 90 °. Po različnih ugotovitvah je razvidno, da se motor zasuka za 90 ° s širino pulza 1,5 ms. Da bi preverili, ali to drži tudi v našem primeru servomotorjev, smo morali krmilno vrednost, s katero dosežemo periodo 1.5 ms, izračunati na ta način:

$$pwm\_vrednost = \frac{1,5ms}{20ms} * 2^{12} = 307$$

V enačbi smo poleg poznanih 1,5 ms upoštevali še posodabljanje širine impulza vsakih 20 ms in 12 bit-ov veliko vodilo I2C (2^{12}). Dobili smo krmilno vrednost 307, ki smo jo preizkusili. Pokazalo se je, da v našem primeru ta vrednost ne premakne servomotorja na 90 °. Po preizkusu ugotovimo, da je naša krmilna vrednost za zasuk 90° enaka 365, kar je prikazano na Sliki 22. Tako odstopanje se zgodi zaradi odstopanja

motorja. Glede na krmilni signal smo določili minimum (180) in maksimum krmilne vrednosti (550) premika motorja po horizontalni ter vertikalni smeri.



Slika 22: Krog predstavlja premikanje servomotorja. Številke na notranji strani kroga nam predstavljajo premikanje motorja v stopinjah, številke na zunanji strani kroga pa krmilne vrednosti premika servomotorja.

Za lažji začetek računanja in sledenja objektom postavimo oba motorja na neko v naprej poznano pozicijo. Odločili smo se, da motor na horizontalni osi nastavimo na maksimum (550), motor na vertikalni osi pa na minimum (180). Mini računalnik preko UDP protokola prejema kota, za katera se morata servomotorja premakniti, da lahko sledita objektu. Pomembno je, da vemo, kakšna je bila predhodna krmilna vrednost ($star\_pwm1$). Zaradi začetne postavitve servomotorja na horizontalni osi na maksimum je krmilna vrednost za ta servomotor enaka 550. Vrednost $fi1$ je prva vrednost s seznama, ki smo ga pridobili od odjemalca in je namenjena prvemu motorju. Za boljše razumevanje si lahko v predhodnem poglavju pogledamo seznam fi in razberemo, na katerem mestu se nahaja kateri kot. Ker pa vrednosti $pwm1$ in $fi1$ nista enakovredni potrebujemo še dodatno konstanto k_fi1, ki preslika kot v razliko krmilne vrednosti (v našem primeru približno dve enoti (bita) na kotno stopinjo). Zaradi zakasnitve sistema (predvsem obdelave slike) krmilimo motorje na podlagi slik, ki so bile zajete nekoliko pred tem. Ta razlika lahko privede do odstopanj. Da se temu izognemo, je priporočljivo nekoliko zmanjšati ojačanje našega sistema (torej zmanjšati konstanto k_fi1) in omejiti spremembo krmilnih signalov v enem koraku (v našem primeru smo se odločili za omejitev na 10 enot dolžine PWM signala).

Naša enačba za izračun krmilne vrednosti:

$pwm1 = star\_pwm1 - k\_fi1;$

Krmilno vrednost za premik servomotorja po horizontalni osi ($pwm1$) izračunamo s pomočjo predhodne krmilne vrednosti ($stari\_pwm1$), ki nam pove, kje se motor trenutno nahaja. Tej vrednosti nato odštejemo prispevek kota (konstanto k_fi1) za odmik objekta od horizontalne osi. Nato preverimo, ali je krmilna vrednost manjša od minimuma ali večja od maksimuma. Če je krmilna vrednost večja od maksimuma, jo spremenimo v maksimum, če je manjša od minimuma, jo spremenimo v minimum. Do te omejitve pride zaradi zakasnitve slike, saj motor vedno krmilimo z vrednostmi, ki so posledica slike izpred nekaj trenutkov (ko smo sliko zajeli), medtem pa smo morali sliko še obdelati. Zaradi tega se dejansko stanje razlikuje od ocenjenega. Da do tega ne pride, omejimo krmilno vrednost na minimum 180 ter maksimum 550 krmilne vrednosti.

Pri računanju krmilne vrednosti za vertikalno os se uporablja druga vrednost $fi2$ s seznama fi, ki jo iz stopinj v signale pretvorimo na enak način, kakor smo to storili za konstanto k_fi1. Tako dobimo konstanto k_fi2. Tudi izračun krmilne vrednosti $pwm2$ je izveden skoraj na enak način kakor izračun krmilne vrednosti $pwm1$, razlikuje se samo v začetni postavitvi motorja. Servomotor je na vertikalni osi na začetku postavljen na svoj minimum (180). Ker je servomotor na horizontalni osi na začetku na maksimumu, mu je potrebno dobljene vrednosti odštevati, na vertikalni osi pa je servomotor na začetku na minimumu in mu je zato potrebno dobljene vrednosti prištevati.

$pwm2 = star\_pwm2 + k\_fi2;$

Krmilno vrednost za premik servomotorja po vertikalni osi ($pwm2$) ravno tako izračunamo iz predhodne krmilne vrednosti ($star\_pwm2$), ki ji prištejemo prispevek kota (konstanta k_fi2) za odmik objekta od vertikalne osi. Nato tudi vrednost $pwm2$, enako kot smo to storili za vrednost $pwm1$, preverimo ali je večja od maksimuma in ali je manjša od minimuma.

Servomotorja se nato s krmilno vrednostjo premakneta na želeno pozicijo.

4.2 Delovanje sistema

Ob zagonu sistema oziroma zagonu programske opreme sistema se nam na zaslonu, ki je priklopljen na mini računalnik Raspberry Pi, prikažeta dve sliki. Sliki sta namenjeni boljšemu razumevanju barvne segmentacije ter izračunu krmilnih vrednosti objekta, ki mu želimo slediti. Prva slika je prikazana v barvnem prostoru RGB in služi za boljšo

predstavitev prostora. Druga slika pa je segmentirana slika v črno-belem prostoru, na kateri je lepo razviden objekt želene barve.

Program dobro segmentira izbrani objekt (v našem primeru rdečo žogico) iz okolice. Velik vpliv na segmentacijo ima svetlost prostora. Če je prostor bolj temen, slabše segmentira izbrani objekt, zaradi česar je potrebno barvo objekta spremeniti na bolj temno. Tudi če je prostor presvetel, slabše segmentira izbrani objekt in mu je zato potrebno spremeniti barvo na bolj svetlo. Iz tega je razvidno, da moramo poiskati čimbolj idealen prostor, da bi bila barva objekta res prava. Ker je to težko doseči, se na sliki pojavlja nekaj šuma. Kot je že v prejšnjih poglavjih opisano, segmentacijo objekta zelo motijo drugi objekti iste barve, zato se jim moramo izogibati. Zaradi zakasnitve obdelave slike smo krmilne signale po horizontalni in vertikalni osi omejili na velikost premika in sicer na največ +/-10 premika naenkrat od predhodne krmilne vrednosti. Kar pomeni, da če je krmilna vrednost manjša od 10, se motor premakne samo enkrat, če pa je večja, se motor premakne večkrat po 10 krmilnih vrednosti, dokler ne pride do dobljene krmilne vrednosti.

5 Zaključek

V zaključni nalogi je opisan sistem za sledenje objektom s postopki računalniškega vida in mehanizma za premikanje kamere. Izdelava sistema mi je predstavljala kar velik iziv, saj sem se prvič srečal z mini računalnikom Raspberry Pi, C in C++ jezikoma, krmilnikom za pulzno-širino modulacije in večino preostalih komponent. Takoj po ideji za izdelavo zaključne naloge sem iskal rešitev za implementacijo le-te, da bi bila cenovno ugodna in dovolj natančna. Rešitev, ki sem jo opisal, je cenovno dokaj ugodna, ampak cenovno ugodne komponente imajo bolj izrazite pomanjkljivosti (npr. mini računalnik je manj zmogljiv, motorji zagotavljajo le manjšo natančnost, itd.), zaradi česar prihaja tudi do odstopanj oziroma napak. Ena izmed izboljšav bi bila v boljši strojni opremi. Stojalo pan-tilt, ki sem ga izdelal sam, bi bilo mogoče kupiti oziroma izdelati z veliko boljšimi pripomočki kakovostnejše, da pri tem ne bi prihajalo do odstopanj. Možnost izboljšave bi bila tudi v programski opremi, saj tudi tam naletimo na nekaj odstopanj. Težava se pojavi pri prepoznavanju objekta v okolici, kjer je veliko šuma. Šum se spreminja od prostora do prostora (npr. z različno osvetlitvijo prostora) in že nahajanje npr. v prostoru, ki je v celoti bele barve, bi omogočalo boljše prepoznavanje objekta. Šum bi lahko odpravili tudi z izboljšanjem kamere. Ker sta postopka za odstranjevanje odvečnega šuma in dopolnjevanje objekta potratna in zamudna, se to pozna na delovanju sistema. Problem se pojavi tudi pri našem načinu sledenja objektu, saj ta ravno tako upočasnjuje delovanje sistema. Z drugačnim postopkom (npr. Meanshift) za sledenje objektom bi pridobili precej boljši sistem za sledenje kompleksnejšim objektom, saj postopek išče najbolj gosto območje na sliki in ne bi bilo potrebe po odpravljanju šuma in dopolnjevanju segmentiranega področja. Zaradi večje časovne zahtevnosti postopka, bi ravno tako oziroma morda še bolj upočasnil delovanje sistema. Zaradi vseh naštetih problemov prihaja pri uporabi sistema do odstopanj in nekoliko počasnejšega sledenja objektov.

Do odstopanj prihaja tudi zaradi zakasnitve obdelave slik. Problem smo rešili tako, da smo krmilno vrednost omejili na največ 10 krmilnih vrednosti premika naenkrat od predhodne krmilne vrednosti. Odstopanje se opazi pri velikih premikih, ko se motor premika po 10 krmilnih vrednosti, dokler ne pride do dobljene krmilne vrednosti. Pri manjših premikih od 10 krmilnih signalov odstopanje ni tako izrazito, saj se motor premakne le enkrat do dobljene krmilne vrednosti.

Čeprav nobena rešitev zaenkrat še ni idealna, bi bilo potrebno sistem nadgraditi. Za takšno rešitev pa bi bilo potrebno uporabiti druge sisteme, ki delujejo dobro v realnem času. Čeprav so taki sistemi že dolgo znani in njihovi postopki mogoči, nam oviro za tako implementacijo predstavlja zmogljivost opreme ter njen cenovni razred.

6 Literatura in viri

[1] *What is computer vision?*, Esce. `https://www.ecse.rpi.edu/Homepages/qji/CV/3dvision_intro.pdf`. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 3.)*

[2] F. Solina, *Računalniški vid*, `http://videolectures.net/promo_franc_solina_slo/`. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 3.)*

[3] M. Kristan in Laboratorij za strojni vid, Fakulteta za elektrotehniko, Univerza v Ljubljani, Kako računalniki vidijo: računalniški vid in njegova uporaba v praksi. 2010, poslano v objavo. *(Citirano na strani 4.)*

[4] G. Bradski and A. Kaehler, *Learning OpenCV*, 2008. *(Citirano na straneh 4, 5, 9 in 11.)*

[5] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2010. *(Citirano na straneh 4, 6, 7 in 9.)*

[6] E. Brown, *Nvidia aims octa-core, 256-GPU, Tegra X1 SoC at smart cars*, `http://hackerboards.com/nvidia-aims-octa-core-256-gpu-tegra-x1-soc-at-smart-cars/`. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 7.)*

[7] B. Thirion, *A-Brain*, `http://www.msr-inria.fr/projects/a-brain/`. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 7.)*

[8] S. Sural, G. Qian and S. Pramanik, *Segmentation and histogram generation using the HSV color space for image retrival*, 2010. *(Citirano na strani 7.)*

[9] C. A. Glasbey and G. W. Horgan, *Image analysis for the biological sciences*, 1995. *(Citirano na strani 7.)*

[10] Computer Science & Engineering, University of Washington, *Computer vision*, 2015. *(Citirano na strani 7.)*

[11] New Mexico Tech Computer Center, *Introduction to color theory*, http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 8.)*

[12] D. Radić, *Korekcija boja monitora*, http://www.informatika.buzdo.com/s357-korekcija-boja.htm. (Datum ogleda: 23. 8. 2016.) *(Citirano na strani 8.)*

[13] C. Zhang, Y. Qiao, E. Fallon and X. Xu, *An Improved CamShift Algorithm for Target Tracking in Video Surveillance*, 2010. *(Citirano na straneh 9 in 11.)*

[14] OpenCv, *Meanshift and Camshift*, http://docs.opencv.org/3.1.0/db/df8/tutorial_py_meanshift.html#gsc.tab=0. (Datum ogleda: 14. 8. 2016.) *(Citirano na straneh 10 in 11.)*

[15] J. G. Allen, R. Y. D. Xu, J. S. Jin, *Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces*, 2006. *(Citirano na strani 11.)*

[16] S. P. Vavpotič, *Kakšni so današnji mikrokrmilniki?*, http://www.monitor.si/clanek/kaksni-so-danasnji-mikroracunalniki/153251/. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 12.)*

[17] Raspberry Pi Foundation, *What is Raspberry Pi?*, https://www.raspberrypi.org/help/videos/. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 13.)*

[18] Raspberry Pi Foundation, *Gpio: Raspberry Pi models A and B*, https://www.raspberrypi.org/documentation/usage/gpio/. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 13.)*

[19] Sparfun, *Raspberry gPIo*, https://learn.sparkfun.com/tutorials/raspberry-gpio. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 14.)*

[20] P. Krkoč, *Programiranje z Arduino (6) - I2C vodilo*, http://www.svet-el.si/o-reviji/programiranje/2716-216-43. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 14.)*

[21] S. Monk, *Configuring I2C*, https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 14.)*

[22] i2c.info, *I2C Info – I2C Bus, Interface and Protocol*, http://i2c.info/. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 14.)*

[23] Sparfun Electronics, *Pulse-width Modulation*, https://learn.sparkfun.com/tutorials/pulse-width-modulation. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 15.)*

[24] EMLID LIMITED, *PWM generator*, https://docs.emlid.com/navio/Navio-dev/servo-and-rgb-led/. (Datum ogleda: 24. 8. 2016.) *(Citirano na strani 15.)*

[25] Adafruit, *Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685*, https://www.adafruit.com/product/815. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 15.)*

[26] T. Kušar, D. Rihtaršič in S. Kocijančič, *Predelava servo-motorja v pogonski motor, krmiljen s servo-impulzi* , http://www2.arnes.si/∼tkusar/downloads/servo_2.pdf. (Datum ogleda: 14. 8. 2016.) *(Citirano na straneh 17 in 18.)*

[27] Electical4u, *Servomechanism — Theory and Working Principle of Servo Motor*, http://www.electrical4u.com/servo-motor-servo-mechanism-theory-and-working-principle/. (Datum ogleda: 14. 8. 2016.) *(Citirano na straneh 17 in 18.)*

[28] F. Reed, *How Do Servo Motors Work*, https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html. (Datum ogleda: 14. 8. 2016.) *(Citirano na straneh 18 in 27.)*

[29] R. J. Kinch, *Raspberry Pi Camera Board*, http://www.truetex.com/raspberry_pi_m12_lens_adapter.pdf. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 19.)*

[30] Raspberry Pi Foundation, *RASPBERRY PI 2 MODEL B*, https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 19.)*

[31] Logitech, *HD WEBCAM C270*, https://secure.logitech.com/en-us/product/hd-webcam-c270. (Datum ogleda: 22. 8. 2016.) *(Citirano na strani 19.)*

[32] R. J. Kinch, *Raspberry Pi*, http://www.truetex.com/raspberrypi. (Datum ogleda: 22. 8. 2016.) *(Citirano na strani 19.)*

[33] J. Pinkney, *Webcam Field of Views*, https://buntworthy.github.io/Wecam-field-of-views/. (Datum ogleda: 22. 8. 2016.) *(Citirano na strani 19.)*

[34] Wikipedia, *Fisheye lens*, https://en.wikipedia.org/wiki/Fisheye_lens. (Datum ogleda: 22. 8. 2016.) *(Citirano na strani 19.)*

[35] OpenCV, *About*, http://opencv.org/about.html. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 21.)*

[36] OpenCV, *Operations on Arrays:inRange*, http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 22.)*

[37] OpenCV, *Miscellaneous Image Transformations:cvtColor*, http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 22.)*

[38] OpenCV, *Morphological Transformations*, http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. (Datum ogleda: 19. 8. 2016.) *(Citirano na strani 22.)*

[39] P. Rogelj, *Računalniški vid: Slikovna področja*, 2014. *(Citirano na straneh 23 in 24.)*

[40] M. Rouse, *UDP (User Datagram Protocol)*, http://searchsoa.techtarget.com/definition/UDP. (Datum ogleda: 14. 8. 2016.) *(Citirano na strani 26.)*

Priloge

A Izvorna koda sistema

- Na zgoščenki je priložena izvorna koda za sistem optično sledenje objektov s postopki računalniškega vida in mehanizma za usmerjanje kamere.